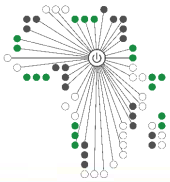


Universidade Virtual Africana

INFORMÁTICA APLICADA: CSI 4301

PROGRAMAÇÃO DE SISTEMAS

Eloy Mendes

Prefácio

A Universidade Virtual Africana (AVU) orgulha-se de participar do aumento do acesso à educação nos países africanos através da produção de materiais de aprendizagem de qualidade. Também estamos orgulhosos de contribuir com o conhecimento global, pois nossos Recursos Educacionais Abertos são acessados principalmente de fora do continente africano.

Este módulo foi desenvolvido como parte de um diploma e programa de graduação em Ciências da Computação Aplicada, em colaboração com 18 instituições parceiras africanas

de 16 países. Um total de 156 módulos foram desenvolvidos ou traduzidos para garantir disponibilidade em inglês, francês e português. Esses módulos também foram disponibilizados como recursos de educação aberta (OER) em oer.avu.org.

Em nome da Universidade Virtual Africana e nosso patrono, nossas instituições parceiras, o Banco Africano de Desenvolvimento, convido você a usar este módulo em sua instituição, para sua própria educação, compartilhá-lo o mais amplamente possível e participar ativamente da AVU Comunidades de prática de seu interesse. Estamos empenhados em estar na linha de frente do desenvolvimento e compartilhamento de recursos educacionais abertos.

A Universidade Virtual Africana (UVA) é uma Organização Pan-Africana Intergovernamental criada por carta com o mandato de aumentar signi cativamente o acesso a educação e treinamento superior de qualidade através do uso inovador de tecnologias de comunicação de informação. Uma Carta, que estabelece a UVA como Organização Intergovernamental, foi assinada até agora por dezenove (19) Governos Africanos - Quênia, Senegal, Mauritânia, Mali, Costa do Mar m, Tanzânia, Moçambique, República Democrática do Congo, Benin, Gana, República da Guiné, Burkina Faso, Níger, Sudão do Sul, Sudão, Gâmbia, Guiné-Bissau, Etiópia e Cabo Verde.

As seguintes instituições participaram do Programa de Informática Aplicada: (1) Université d'Abomey Calavi em Benin; (2) Université de Ougagadougou em Burkina Faso; (3) Université Lumière de Bujumbura no Burundi; (4) Universidade de Douala nos Camarões; (5) Universidade de Nouakchott na Mauritânia; (6) Université Gaston Berger no Senegal; (7) Universidade

das Ciências, Técnicas e Tecnologias de Bamako no Mali (8) Instituto de Administração e Administração Pública do Gana; (9) Universidade de Ciência e Tecnologia Kwame Nkrumah em Gana; (10) Universidade Kenyatta no Quênia; (11) Universidade Egerton no Quênia; (12) Universidade de Addis Abeba na Etiópia (13) Universidade do Ruanda; (14) Universidade de Dar es Salaam na Tanzânia; (15) Universite Abdou Moumouni de Niamey no Níger; (16) Université Cheikh Anta Diop no Senegal; (17) Universidade Pedagógica em Moçambique; E (18) A Universidade da Gâmbia na Gâmbia.

Bakary Diallo

O Reitor

Universidade Virtual Africana

Créditos de Produção

Autor

Eloy Mendes

Par revisor(a)

Celestino Barros

UVA - Coordenação Académica

Dr. Marilena Cabral

Coordenador Geral Programa de Informática Aplicada

Prof Tim Mwololo Waema

Coordenador do módulo

Jules Degila

Designers Instrucionais

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipa Multimédia

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Direitos de Autor

Apoiado por

Projeto Multinacional II da UVA financiado pelo Banco Africano de Desenvolvimento.

Índice

Descrição Geral do Curso

Bem-vindo(a) ao curso Programação de Sistemas

Programação de Sistemas (PS) é o estudo avançado e métodos com os quais se constrói sistemas informáticos e sistemas - Servidores (do tipo cliente versus servidor) especializados, usando linguagem de programação C, e C ANSI numa perspectiva tecnológica e comercial. A linguagem de programação C, apesar de ser uma das mais antigas, é ainda uma referência para implementação de sistemas eficazes nos domínios das ciências e investigação tecnológica, à nível industrial e em algumas situações à nível comercial.

Durante a última década, o surgimento de novas linguagens de programação, nomeadamente (os de alto - níveis) como Java, Pyton, etc, tomaram em parte a posição da linguagem C, pelo menos a níveis comerciais, C ANSI é entretanto, uma das melhores opções pela sua forma clássica de funcionalidades na industria de eletrônica, automação, e demais aplicações industriais, pela fácil aderência em ser implementada em micro-circuitos,e este goza de robustez de sintaxe e semântica, enquanto linguagem de programação.

Este curso permite que você inicie as aulas de teoria e prática necessária para o desenvolvimento dos sistemas e servidores de uso comercial (para qualquer usuário), bem como servidores que agregam serviços especializados para empresas. Práticas baseados em tutoriais (hands-on) serão equilibradas com discussão de literatura relevante no campo da ciência da computação, por exemplo, as melhores práticas de desenvolvimento de algoritmia em C, tratamento de erro, técnicas avançadas de implementação de sistemas especializados e servidores de redes, tratamento de processos, memória RAM, tratamento de sinais em sistemas Linux, e manipulação de arquivos, dentre outras metodologias.

Pré-requisitos

- Introdução aos sistemas operacionais
- Introdução à programação estruturada
- Básico sistema operacional UNIX / Linux
- Programação em C

Materiais

Os materiais necessários para completar este curso são:

- Um PC instalado com o sistema operacional Linux de preferência com acesso à internet.

Objetivos do Curso

Após a conclusão deste curso, você deve ser capaz de:

- Usar bibliotecas de programação de baixo nível para desenvolver ferramentas e software relacionado sistema;
- Escrever arquivos de comando básicos do sistema operacional;
- Programa em baixo nível Unix / Linux para estender / melhorar as funcionalidades do sistema operacional;

Unidades

Unidade 0: Pré-avaliação

Esta unidade fornece uma visão geral para o módulo de programação de sistemas. Esta analisa os conceitos básicos de programação com linguagem C e destaca a organização dos conteúdos dos módulos, atividades e avaliações.

Unidade 1: Bibliotecas C e as chamadas de I/O de sistema

Esta unidade fornece uma visão geral da Biblioteca C GNU, incluindo as bibliotecas padrões e portabilidade, os fundamentos do uso da biblioteca, e o caso de funções de biblioteca C para a alocação de armazenamento para dados do programa. A unidade explora ainda mais as funções da biblioteca GNU C para a manipulação de arquivos e diretórios. A unidade também discute o sistema de entrada / saída de arquivo chama GNU plataforma / LINUX e destaca do kernel I / O chama relação ao Funções padrão de E / S para as bibliotecas C;

Unidade 2: Programação em ShellScript e Assembly C

Programa em shell fornecido pelos sistemas operacionais aceita instruções do usuário ou comandos, e passá-los ao kernel, de forma semelhante, os programas escritos em Línguas Higherlevel (alto nivel) tais como C também fornecem uma maneira para instruir o hardware para executar alguma tarefa do usuário. Para ocasiões em alto nível os programadores precisam usar instruções de montagem nos seus programas, a coleção do compilador GNU permite aos programadores adicionar instruções em linguagem e arquitetura assembly dependente para seus programas. Esta unidade de programação explora tanto contexto, ou seja, o invólucro programação e montagem inlining (linha de comando) em C.

Unidade 3: Processos, threads e gerenciamento de memória

O Programa deve ser trazido para a memória e colocado dentro de um processo para que ela seja executado. Uma instância em execução de um programa é chamado de processo e threads como subdivisão de processos, são um mecanismo para permitir que um programa possa fazer mais do que uma coisa (processo) de cada vez. Nesta unidade, vamos descrever o processo, linha e funções de manipulação de memória em sistemas Linux cuja maioria são semelhantes aos de outros sistemas UNIX.

Unidade 4: Comunicações interprocessos

Interprocessos entre comunicação é a transferência de dados entre processos. Nesta unidade, será apresentado várias formas de comunicação entre processos pais e filhos, entre os processos "independentes", e até mesmo entre os processos em máquinas diferentes. Três tipos de comunicação enterprocessos são discutidos : Pipes; FIFO;Sockets .

Avaliação

Em cada unidade encontram-se incluídos instrumentos de avaliação formativa a fim de verificar o progresso do(a)s aluno(a)s.

No final de cada módulo são apresentados instrumentos de avaliação sumativa, tais como testes e trabalhos finais, que compreendem os conhecimentos e as competências estudadas no módulo.

A implementação dos instrumentos de avaliação sumativa fica ao critério da instituição que oferece o curso. A estratégia de avaliação sugerida é a seguinte:

1 e 2	Unidade 1: Bibliotecas C e as chamadas de I/O de sistema e, Unidade 2: Programação em ShellScript e Assembly C	45%
3	Unidade 3: Processos, threads e gerenciamento de memória	25%
4	Unidade 4: Comunicações interprocessos	30%

Calendarização

Unidade	Temas e Atividades	Estimativa do tempo
0 e1	Estudos Independentes, aulas práticas, tutoriais, Tarefas de casa	40 hr
2	Estudos Independentes, aulas práticas, tutoriais, Tarefas de casa	30 hr
3	Estudos Independentes, aulas práticas, tutoriais, Tarefas de casa	25 hr
4	Estudos Independentes, aulas práticas, tutoriais, Tarefas de casa	25 hr

Leituras e outros Recursos

As leituras e outros recursos deste curso são:

Unidade 0

Leituras e outros recursos obrigatórios:

- 1. Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001, pp 3-15

Otimas leituras e outros recursos

- 2. Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love
- 3. UNIX Systems Programming: Communication, Concurrency, and Threads. By Kay A. Robbins, Steven RobbinsLinguagem C, Luís Manuel
- Dias Damas, FCA - Editora de Informática ISBN: 972-722-156-4
- Elementos de Programação com C, Pedro Guerreiro, 3ª Edição, FCA – Editora de Informática ISBN:972-722-300-1

Unidade 1

Leituras e outros recursos obrigatórios:

- The GNU C Library Reference Manual, Sandra Loosemore With Richard M. Stallman, Roland McGrath, Andrew Oram, and Ulrich Drepper for version 2.21, Copyright c 1993–2014 Free Software Foundation, Inc. (http://www.gnu.org/software/libc/manual/pdf/libc.pdf)
- 2. Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001, pp 281-300

Otimas leituras e outros recursos

- 1. Linux System os Programming: Talking Directly to the Kernel and C Library By Robert Love
- 2. UNIX Systems Programming: Communication, Concurrency, and Threads. By Kay A. Robbins, Steven Robbins
- 3. http://www.acm.uiuc.edu/webmonkeys/book/c_guide/: The C Library Reference Guide
- 4. http://www.delorie.com/gnu/docs/glibc/libc_toc.html: The GNU C Library

Unidade 2

Leituras e outros recursos obrigatórios:

- 1. Learning the bash Shell: Unix Shell Programming By Cameron Newham and Bill
- Rosenblatt,Copyright 2005, O'Reilly Media, USA.

Otimas leituras e outros recursos opcionais:

- Learning the bash Shell: Unix Shell Programming (In a Nutshell (O'Reilly)),
- Kindle Edition, Cameron Newham (Author)
- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex
- Samuel, New Riders Publishing, FIRST EDITION: June, 2001

Unidade 3

Leituras e outros recursos obrigatórios:

- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001, pp 3-15

Leituras e outros recursos opcionais:

- Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love
- Linux Kernel Development, Robert Love, Pearson Education, 22 Jun 2010
- UNIX Systems Programming: Communication, Concurrency, and Threads. By Kay A. Robbins, Steven Robbins

Unit 4

Leituras e outros recursos obrigatórios:

- Advanced Linux Programming, by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, New Riders Publishing, FIRST EDITION: June, 2001

Leituras e outros recursos opcionais:

- UNIX Systems Programming: Communication, Concurrency, and Threads, By Kay A. Robbins, Steven Robbins, Prentice Hall Professional, 2003
- Interprocessos Communications in Linux, By John Shapley Gray, Prentice Hall Professional, 2003

- UNIX Network Programming: Interprocess communications, Volume 2 , W.Richard Stevens Prentice Hall PTR, 1999
- Linux System Programming: Talking Directly to the Kernel and C Library By Robert Love

Unidade 0. Pré-Avaliação

Introdução à Unidade

O propósito desta unidade é verificar a compreensão dos conhecimento que você possui relacionados a este curso. O módulo de Programação do Sistema trata dos tópicos avançados em programação C, assim, assume-se que você já está familiarizado com a linguagem de programação C e que sabe como usar as funções da biblioteca padrão do C em seus programas. A linguagem C é a línguagem mais utilizada para o desenvolvimento de Sistemas; a maioria dos comandos e bibliotecas que discutimos neste módulo, e a maior parte do kernel do Linux em si, são escritos em C.

As informações contidas neste módulo é igualmente aplicável aos programas em C ++, porque essa linguagem foi desenvolvida a partir de um super conjunto C. Para quem já programou em outra plataforma de sistema como o UNIX antes, as chances são boas em se entender tudo. Ademais conhecer as funções do Linux de baixo nível, funções I/O open / read/ stat, e assim por diante). Eles são diferentes de funções de I /O da biblioteca padrão C (fopen, fprintf, fscanf, e assim por diante). Ambos são úteis na programação do sistema, e vamos aprender, e usar ambos os conjuntos de funções de I/O neste módulo. Caso você não esteja familiarizado com as funções de baixo nível I/O ("LowLevel I / O) , aconselha-se a manter calma e dedicar, e terá a certeza de estar em breve familiarizado com o assunto uma vez que na Unidade 1 trataremos "LowLevel I / O".

Este módulo não fornece uma introdução geral aos sistemas GNU / Linux. É assumido que já se tem conhecimento básico de como interagir com um sistema GNU / Linux e executar operações básicas em ambientes gráficos e de linha de comando (Para quem está a se perguntar como pode usar GCC no Windows, pode apenas baixar Cygwin a partir www. cygwin.com). Deve seguir as convenções de base que será utilizado.

- Quando mostramos interações com um comando shell, usamos $ como a janela de comandos (o seu shell provavelmente está configurado para utilizar uma linha diferente). Tudo após o alerta é o que se digita, enquanto outras linhas de texto são a resposta do sistema. Por exemplo, neste interação $uname Linux o sistema usou o simbolo US $. Foi digitado o comando uname. O sistema respondeu a "impressão" Linux que apresentará a versão do Kernel.
- Os exemplos de código fonte inclui um nome de arquivo entre aspas duplas. Se você copiar o exemplo de código, salve-o em um arquivo com esse nome. Os exemplos de código foram escritos e testado usando a distribuição Red Hat 6.2 do GNU / Linux. Esta distribuição incorpora as versões 2.2.14 do kernel do Linux, versão 2.1.3 da biblioteca GNU C, e versão 1.1.2 EGC do compilador GNU C. A informação e os programas fornecidos neste módulo devem ser geralmente aplicáveis a outras versões e distribuições de GNU / Linux bem como incluindo também as versões 2,4 do kernel do Linux e as versões 2,2 da biblioteca GNU C.

Objectivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

- Criar e abrir o arquivo fontes C/C++;
- Compilar com GCC os códigos fontes únicos/ múltiplas em C/C ++
- Link de arquivos fontes;
- Automatizar o processo de compilação em link GNU Make;
  Debug C / C ++ programas com GNU Debugger

- Trabalhar com arquivos de cabeçalho nos programas C / C ++
- Procurar informações ajuda a partir de código fontes de Linux

TERMOS-CHAVE

Compiler / Compilador: Código fonte legível por humanos transformados objetos código fonte legível por máquinas (computadores) de forma que possam ser realmente funcionais.

Linker: Um programa de computador que tem um ou mais arquivos de objectos gerados por um compilador e as combina em um único arquivo executável, arquivo de biblioteca, ou outro arquivo objeto.

Debugger: O programa que se usa para descobrir por qual motivo seu programa não está comportando da maneira que você acha que deveria.

Pipes Permite a comunicação sequencial de um processo para outro processo;

FIFO são semelhantes aos pipes, excepto que os processos não relacionados pode comunicar, porque o tubo é dado um nome no sistema de arquivos;

Sockets permite comunicação de suporte entre os processos não relacionados mesmo e computadores diferentes

GCC: Colecção de compiladores GNU

GNU: GNU não é linux

Atividade 0.1 - Introdução a Programação C

0.1.1 Introdução

Este módulo revê os passos básicos necessários para criar um programa C ++ ou C Linux . Em particular, descreve como criar e modificar código fonte C e C ++, como compilar esse código, e depurar o resultado.

Detalhes de atividade

Criar / Abrir um código fonte C ou C ++

Um editor é um programa que se usa para editar o código-fonte. Lotes de diferentes editores estão disponíveis para Linux, mas o editor mais popular e completamente estruturado é provavelmente o GNU Emacs. Pode-se começar Emacs ou qualquer um de seu editor favorito digitando seu nome em sua janela de terminal e pressionar a tecla Return, por exemplo, do tipo emacs para invocar o editor Emacs. Quando seu editor for iniciado, pode-se usar os menus no topo para criar um novo código fonte. Se desejar criar um código fonte C, use um nome de arquivo que termina em .c ou .h. Se quiser criar um C ++ código fonte, use um nome de arquivo que termina em .cpp, .hpp, .cxx, .hxx, .C ou .H. Quando o arquivo é aberto, pode escrever como faria em qualquer programa de processamento de texto comum.

Compilando com GCC

Um compilador transforma o código-fonte legível por humanos em código objeto legíveis por máquinas de forma que possam ser executados. Os compiladores de escolha em sistemas Linux são todos parte da coleção dos compiladores GNU , normalmente conhecido como GCC. GCC também incluem compiladores para C, C ++, Java, ObjectiveC, Fortran. Este módulo se concentra principalmente em programação C. Suponha que você tenha um projeto com um código fonte C ++ "reciprocal.cpp" e código fonte C "main.c", como mostrado a seguir. Esses dois arquivos devem ser compilados e, em seguida, ligados entre si para produzir um programa chamado reciprocal (Nota: No Windows, executáveis geralmente têm nomes que terminam em .exe. Programas Linux, por outro lado, geralmente não têm extensão. Então, o equivalente Windows deste programa provavelmente seria chamado reciprocal.exe, a versão Linux é simplesmente reciprocal.). Este programa calcula o reciprocal de um número inteiro.

```
//Programa "main.c" - C Codigo fonte—main.c

#include <stdio.h>

#include "reciprocal.hpp"

int main (int argc, char **argv)

{

int i;

i = atoi (argv[1]);

printf ("The reciprocal of %d is %g\n", i, reciprocal (i));

return 0;

}

//Program "reciprocal.cpp" - C++ source file—reciprocal.cpp

#include <cassert>

#include "reciprocal.hpp"
```

```
double reciprocal (int i) {

// I should be non-zero.

assert (i != 0);

return 1.0/i;

}
```

Há também um arquivo de cabeçalho chamado "reciprocal.hpp", tal como indicado a seguir:

```
// Arquivo de cabeçalho "reciprocal.hpp" - Arquivo de cabeçalho—
reciprocal.hpp

#ifdef __cplusplus

extern "C" {

#endif

extern double reciprocal (int i);

#ifdef __cplusplus

}

#endif
```

O primeiro passo é transformar o código-fonte C e C ++ em código objeto.

A. Compilar um único arquivo de código- fonte

O nome do compilador C é gcc .Para compilar um código fonte C, você pode usar a opção -c. Assim, por exemplo, inserir este no prompt de comando compila o arquivo código fonte main.c:

```
$ gcc c main.c
```

O arquivo objeto resultante é nomeado main.o.

Compilador do C ++ é chamado g++ .Sua operação é muito semelhante ao gcc;

Compilando o reciprocal.cpp :

```
$ G++ -c reciprocal.cpp
```

A opção -c diz ao G++ para compilar o programa com apenas um arquivo objeto, sem ela, G++ tentará vincular o programa para produzir um executável. Depois de digitar esse comando, você terá um objeto chamado reciprocal.o. Provavelmente precisa-se de um par de outras opções para construir qualquer programa razoavelmente grande. A opção - I é usado para dizer GCC onde procurar arquivos de cabeçalho. Por padrão, o GCC procura no diretório atual e nos diretórios onde cabeçalhos para as bibliotecas padrão estão instalados. Se precisar incluir arquivos de cabeçalho de algum outro lugar, você vai precisar da opção -I. Por exemplo, suponha que o projeto tem um diretório chamado src, para arquivos de codigo fonte, e outro

chamado include. Você deverá compilar reciprocal.cpp como este para indicar que g++ deve usar o diretório ../include adicionalmente para encontrar reciprocal.hpp:

```
$ g++ -c  -I ../include reciprocal.cpp
```

Às vezes queria-se definir macros na linha de comando. Por exemplo, no código de produção, você não quer que haja sobrecarga da verificação de declaração presente em reciprocal.cpp; isto é, só para ajudá-lo a depurar o programa. Você desliga a verificação pela definição do macro NDEBUG. Você poderia adicionar um #define explícita para reciprocal.cpp, mas isso exigiria mudar a própria fonte. Fica fácil simplesmente definir NDEBUG na linha de comando, como este:

```
$ g++ -c  -D NDEBUG reciprocal.cpp
```

Se você quer definir NDEBUG para algum valor específico, faz-se:

```
$ g++ -c -D NDEBUG = 3 reciprocal.cpp
```

Para quem realmente queira construir o código de produção, provavelmente deve querer ter GCC de código otimizado, para que ele seja executado tão rapidamente quanto possível. Usa-se a opção de linha de comando O2. (GCC tem vários níveis diferentes de otimização; o segundo nível é apropriado para a maioria dos programas).

Por exemplo, a linha seguinte compila reciprocal.cpp com a otimização ligado:

```
$ g++ -c -O2 reciprocal.cpp
```

Note-se que a compilação com otimização pode fazer seu programa mais difícil para depurar com um depurador. Além disso, em certos casos, a compilação com otimização pode descobrir erros em seu programa que não se manifestam anteriormente. Você pode passar muitas outras opções para o gcc e g++. A melhor maneira de obter uma lista completa está para visualizar a documentação on-line. Você pode fazer isso digitando o seguinte comando no seu prompt de comando:

```
$ Info gcc
```

B. Arquivos Object Linking

Agora tendo já compilado main.c e reciprocal.cpp, procedemos o processo de ligá-los. Deve-se usar sempre g++ para vincular um programa que contém o código C ++, mesmo que também contém código C. Se o programa contém somente o código C, devemos usar o gcc no lugar. Porque este programa contém C e C ++, você deve usar g ++, como este:

$ g++ -o reciprocal main.o recíprocal.o

A opção o dá o nome do arquivo para gerar como saída por etapa. Agora podemos executar recíprocal da seguinte forma:

```
$ ./reciprocal 7
```

O recíprocal de 7 é 0,142857

Como você pode ver, g ++ tem ligadas automaticamente na biblioteca de tempo de execução C padrão que contém a implementação de printf .Se você precisava para ligar em outra biblioteca (como um usuário interface dgráfica do kit de ferramentas), você teria especificado a biblioteca com a opção -l. No Linux, nomes de biblioteca quase sempre começa por exemplo libpam.a. O módulo de autenticação conectável (PAM) biblioteca é chamado libpam.a. Para fazer o link em libpam.a, usa-se um comando do tipo:

```
$g++ -o recíprocal main.o reciprocal.o -lpam
```

O compilador adiciona automaticamente o prefixo lib e, o sufixo .a. Tal como acontece com arquivos de cabeçalho, o vinculador (linkers) localiza as bibliotecas em algum lugar padrão, incluindo o / lib e / usr / lib que contêm as bibliotecas de sistema padrão. Se quiser que o linkers procura outros diretórios, deve-se usar a opção G, em paralelo com a opção I discutido anteriormente.Podemos usar esta linha para instruir o linkers a procurar bibliotecas no diretório pam / usr / local / lib /, antes de pesquisar em locais habituais:

```
$ g++ -o reciprocal main.o reciprocal.o -L/usr/local/lib/pam –lpam
```

Embora não seja obrigatório usar a opção -I para obter o pré-processador para pesquisar a corrente diretório, deve-se usar a opção -L para obter o vinculador para procurar o diretório atual. Pode-se em particular usar o seguinte comando para instruir o vinculador a encontrar a biblioteca de teste na atual diretório:

```
$ gcc -o app app.o -L. -ltest
```

Automatizando o processo com o GNU make

Para quem está acostumado em programação para o sistema operacional Windows, provavelmente está acostumado a trabalhar com um ambiente de desenvolvimento integrado (IDE). Adiciona-se códigos fontes para o seu projeto e, em seguida, o IDE constrói seu projeto automaticamente. Embora IDEs estejam disponíveis para Linux, este módulo não vai discutí-los. Em vez disso, este módulo mostra como usar o GNU make para recompilar automaticamente o código, que é o que a maioria dos programadores Linux realmente fazem.

A idéia básica por trás da marca é simples. Devemos dizer o que há para se fazer, e depois atribuímos regras que explicam como construí-los. Também é possível especificar as dependências que indicam quando um alvo particular deve ser reconstruída. Em nossa amostra do projeto recíproca, há três metas óbvias: reciprocal.o, main.o, e o próprio recíproco. Tem-se as regras em mente para construção dessas metas em forma de linhas de comando exemplificando anteriormente. As dependências exigem um pouco de pensamento. Claramente, recíprocal depende do reciprocal.o e main.o porque não é possível vincular o programa completo até que você construa cada um dos arquivos de objeto. Os arquivos de objeto devem ser reconstruídos sempre que os arquivos de origem correspondentes mudarem. Há mais uma torção no que uma mudança para reciprocal.hpp também deve causar ambos os arquivos de objeto a ser reconstruído, pois ambos arquivos de origem incluem o arquivo de cabeçalho.

Além dos alvos óbvios, deve haver sempre um alvo limpo. Esta meta remove todos os arquivos e programas de objetos gerados de modo que se pode começar de novo. A regra para este destino usa o comando rm para remover os arquivos.

Pode-se transmitir toda essa informação a ser executado colocando as informações em um arquivo chamado Makefile.

Aqui está o conteúdo do Makefile:

reciprocal: main.o reciprocal.o

g++ $(CFLAGS) -o reciprocal main.o reciprocal.o

main.o: main.c reciprocal.hpp

gcc $(CFLAGS) -c main.c

reciprocal.o: reciprocal.cpp reciprocal.hpp

g++ $(CFLAGS) -c reciprocal.cpp

clean:

rm -f *.o reciprocal

Pode-se ver que os alvos são listados à esquerda, seguido de dois pontos e, em seguida, todas as dependências. A regra para construir esse alvo é na linha seguinte (ignorar o bit $ (CFLAGS) para o momento). Em consonância com a regra sobre ele deve começar com um caractere Tab, ou faze-lo ficará confuso. Se editarmos o nosso Makefile no Emacs, o Emacs irá ajudar-nos com a formatação. Se tivermos removido os arquivos objeto já construído, basta digitar

```
$ make
```

na linha de comando, veremos o seguinte:

```
$ make

gcc -c main.c

g ++ -c reciprocal.cpp

g ++ -o recíprocal reciprocal.o main.o
```

Podemos ver que make construiu automaticamente os arquivos de objetos e, em seguida, ligou-os. Se agora alterarmos main.c de forma trivial e digite make novamente, teremos o seguinte:

```
$ make

gcc c main.c

g ++ -o recíprocal reciprocal.o main.o
```

Podemos ver que make reconstruiu o main.o e para voltar a ligar o programa, mas não se deu ao trabalho de recompilar reciprocal.cpp porque nenhuma das dependências para reciprocal.o tinha mudado. O $ (CFLAGS) é uma variável make. Você pode definir esta variável tanto no Makefile ou sobre a linha de comando. GNU make irá substituir o valor da variável quando ele executa a regra. Assim, por exemplo, para recompilar com a otimização habilitada, você faria isso:

```
 $ make clean

rm -f *.o reciprocal

$ make CFLAGS=-O2

gcc -O2 -c main.c

g++ -O2 -c reciprocal.cpp

g++ -O2 -o reciprocal main.o reciprocal.o.
```

Note-se que o flag -O2 foi inserido no lugar do $ (CFLAGS) nas regras. Esta secção, tem apresentado apenas os recursos mais básicos de make. Você pode descobrir mais, digitando o seguinte:

```
$ Info make
```

Nesse manual, você encontrará informações sobre como fazer a manutenção de um Makefile mais fácil, como reduzir o número de regras que você precisa para escrever, e como calcular automaticamente dependências. Você também pode encontrar mais informações no GNU, Autoconf, Automake, e Libtool por Gary V.Vaughan, Ben Elliston, Tom tromey, e Ian Lance Taylor (New Riders Publishing,2000).

Depuração - GNU Debugger (GDB)

O depurador é um programa que você usa para descobrir por que seu programa não está se comportando da maneira você acha que deveria. O GNU Debugger (GDB) é o depurador usado pela maioria dos programadores Linux. Podemos usar GDB para percorrer o código, definir pontos de interrupção, e examinar o valor de variáveis locais.

A. Compilando com informações de depuração

Para usar o GDB, você terá que compilar com informações de depuração habilitada. Faça isso adicionando o -g interruptor na linha de comando de compilação. Se você estiver usando um Makefile, como descrito anteriormente, você pode apenas definir CFLAGS igual a -g, quando você executar o make, como mostrado aqui:

```
$ make CFLAGS=-g

gcc -g -c main.c

g++ -g -c reciprocal.cpp

g++ -g -o reciprocal main.o reciprocal.o
```

Quando você compilar com -g, o compilador inclui informação extra nos arquivos de objeto e executáveis. O depurador usa essas informações para descobrir quais endereços correspondem a que as linhas em que os arquivos de origem, como imprimir as variáveis locais, e assim por diante.

B. Rodando GDB

Você pode iniciar-se gdb digitando:

```
$ gdb reciprocal
```

Quando o gdb se inicia, você deve ver o prompt do GDB:

(gdb)

O primeiro passo é executar o seu programa dentro do depurador. Basta digitar o comando de marcha e de qualquer argumentos do programa. Tente executar o programa sem argumentos, como este:

```
((gdb) run

Starting program: reciprocal

Program received signal SIGSEGV, Segmentation fault.

 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287

287 strtol.c: No such file or directory.

(gdb)
```

O problema é que não existe um error-checking no código principal. O programa prevê um argumento, mas, neste caso, o programa foi executado sem argumentos. A mensagem indica um SIGSEGV falha de programa. GDB sabe que a queda real aconteceu em uma função chamada __strtol_internal. Essa função é na biblioteca padrão, e se a fonte não estiver instalado, o que explica a mensagem “No such as file or directory”. Você pode ver a pilha usando o comando em que:

```
(gdb) where

#0 __strtol_internal (nptr=0x0, endptr=0x0, base=10, group=0)

at strtol.c:287

#1 0x40096fb6 in atoi (nptr=0x0) at ../stdlib/stdlib.h:251

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8
```

Você pode ver a partir desta exibição que principal chamado a função atoi com um ponteiro NULL, que é a fonte do problema. Você pode ir até dois níveis na pilha até chegar a principal usando-se o comando:

```
(gdb) up 2

#2 0x804863e in main (argc=1, argv=0xbffff5e4) at main.c:8

8 i = atoi (argv[1]);
```

Note-se que o gdb é capaz de encontrar a fonte para main.c, e mostra a linha onde a chamada de função errada ocorreu. Você pode ver o valor de variáveis usando o comando de impressão:

```
(gdb) print argv[1]

$2 = 0x0
```

Isso confirma que o problema é realmente um ponteiro NULL passou para atoi. Você pode definir um breakpoint usando o comando pausa:

```
((gdb) break main

Breakpoint 1 at 0x804862e: file main.c, line 8.
```

Este comando define um ponto de interrupção na primeira linha da principal. Agora tente executar novamente o programa com um argumento, como este:

```
(gdb) run 7

Starting program: reciprocal 7

Breakpoint 1, main (argc=2, argv=0xbffff5e4) at main.c:8

8 i = atoi (argv[1]);
```

Você pode ver que o depurador parou no ponto de interrupção. Você pode passar por cima da chamada para ATOI usando o seguinte comando:

```
(gdb) next

9 printf ("The reciprocal of %d is %g\n", i, reciprocal (i));
```

Se você quiser ver o que está acontecendo dentro de reciprocidade, use o comando passo como este:

```
(gdb) step

reciprocal (i=7) at reciprocal.cpp:6

6 assert (i != 0);
```

Você está agora no corpo da função recíproco. Podemos achar que é mais conveniente para executar gdb dentro do Emacs ao invés de usar gdb diretamente da linha de comando. Use o comando M-x gdb para iniciar gdb em uma janela do Emacs. Se você estiver parado em um ponto de interrupção, Emacs automaticamente puxa para cima o arquivo fonte apropriado. É mais fácil de descobrir o que está acontecendo quando você está acessando o arquivo inteiro em vez de apenas uma linha de texto.

Como obter mais informações

Quase todas as distribuições Linux vem com uma grande quantidade de documentação útil. Você pode aprender a maioria do que foi apresentado neste módulo por documentação lendo na sua Distribuição Linux (embora ele provavelmente iria demorar muito mais tempo). A documentação não é sempre bem organizado, porém, assim que a parte difícil é encontrar o que você precisa. A documentação também às vezes está fora de data (outofdate), de modo a ter tudo o que você lê de forma clara. Se o sistema não se comportam da maneira como uma página do manual (páginas de manual) pode ser que por exemplo, o manual esteja desatualizado.

Para ajudá-lo a navegar, aqui estão as fontes mais úteis de informação sobre programação Linux avançada.

A. Páginas de manuais (man)

Distribuições Linux incluem páginas do manual para a maioria dos comandos padrão, chamadas de sistema e padrão funções de biblioteca. As páginas de manual são divididos em seções numeradas; para os programadores, os mais importantes são estes:

1. Os comandos do usuário
2. As chamadas ao sistema
3. Funções da biblioteca padrão
4. Comandos / sistemas administrativos

Os números denotam seções da página homem. Páginas de manual do Linux vem instalado em seu sistema, o uso do comando man para acessá-los. Para procurar uma página do manual, basta invocar o nome do homem, onde nome é um nome de comando ou função. Em alguns casos, o mesmo nome ocorre em mais do que uma seção, você pode especificar a seção explicitamente, colocando o número da seção antes do nome. Por exemplo, se você digitar o seguinte, você vai ter a página do manual para o comando do sono (em seção 1 das páginas man Linux):

```
$ man sleep
```

Para ver a página do manual para a função de biblioteca sleep, use este comando:

```
$ man 3 sleep
```

Cada página do manual inclui um resumo online do comando ou função. O comando whatis exibe todas as páginas man (em todas as seções) para um nome correspondente comando ou função. E se você não tem certeza qual comando ou função que deseja, você pode realizar uma busca de palavras-chave nas linhas de resumo, usando o manual -k palavra-chave.

Manuais de páginas incluem uma grande quantidade de informação muito útil e deve ser o primeiro lugar que você ligar para ajuda. A página do manual para um comando descreve as opções e argumentos de linha de comando, de entrada e de saída, os códigos de erro,

configuração, e semelhantes. A página man para uma chamada de sistema ou biblioteca função descreve parâmetros e valores de retorno, listas de códigos de erro e efeitos colaterais, e especifica que incluem arquivo para usar se você chamar a função.

B. Informações

As Informações da documentação de sistema contém a documentação mais detalhada para muitos componentes de núcleo do sistema GNU / Linux, além de vários outros programas. Páginas de informação tem documentos hipertexto, semelhante às páginas da Web. Para iniciar o navegador Informações text-based, basta digitar informações em modo shell do windows. Você será apresentado com um menu de documentos Info instalados em seu sistema. (Pressione Control + H para visualizar as teclas para navegar um documento de Informações).

Entre os documentos de informação mais úteis são estes:

- libc—The GNU C library, including many system calls
- gdb—The GNU debugger
- gcc—The gcc compiler
- emacs—The Emacs text editor
- info—The Info system itself

Quase todas as ferramentas padrão de programação do Linux (incluindo ld , o vinculador; como , o montador; e gprof , o Profiler) vêm com páginas de informações úteis. Você pode pular diretamente para um determinado documento especificando o nome da página na linha de comando:

```
$ info libc
```

Se você faz a maioria de sua programação em Emacs, você pode acessar o navegador embutido Informações pelo digitando M-x info ou C -h i.

C. Arquivos de cabeçalho

Você pode aprender muito sobre as funções do sistema que estão disponíveis e como usá-los por acessar os arquivos de cabeçalho do sistema. Estes residem em / usr / include e / usr / include / sys .Se você está recebendo erros de compilação de usar uma chamada de sistema, por exemplo, dê uma olhada no arquivo de cabeçalho correspondente para verificar se a assinatura da função é o mesmo que o que está listado na página do manual.

Em sistemas Linux, um monte de detalhes nitty-gritty de como o sistema chama o trabalho, se refletem em arquivos de cabeçalho no diretórios / usr / include / bits , / usr / include / asm e / usr / include / linux .Por exemplo, os valores numéricos dos sinais são definidos no /usr/ include/bits/signum.h cabeçalho .Estes arquivos fazem uma boa leitura para abrir mentes. Não incluí-los diretamente em seus programas, porém, sempre use os arquivos de cabeçalho em / usr / include ou como mencionado na página do manual para a função que você está usando.

D. Código Fonte (Source Code)

Estes são materiais Open Source, certo? O árbitro final de como o sistema funciona é o próprio sistema de código fonte, e felizmente para programadores Linux, que o código fonte está disponível gratuitamente. As possibilidades são, a sua distribuição Linux inclui o código fonte completo para todo o sistema e todos programas incluídos com ele, se não, você tem direito nos termos da GNU-General Public License Licença para solicitá-lo a partir do distribuidor (O código fonte não pode ser instalado em seu disco. Consulte a documentação da sua distribuição para obter instruções sobre como instalá-lo).

O código fonte para o kernel Linux em si é normalmente armazenado em / usr / src / linux. Se este módulo deixa-lhe com mais vontade por detalhes de como os processos funcionam, memória compartilhada, e dispositivos de sistema funciona, você sempre pode aprender directamente em linha do código-fonte. A maior parte das funções do sistema descrito neste módulo são implementados na biblioteca GNU C; verifique a documentação da sua distribuição para a localização do código fonte da biblioteca C.

Conclusão

Nesta unidade, foram apresentados os conceitos básicos de programação com a linguagem C / C ++.

Avaliação

1. Pratique os códigos exemplos ilustrados ao longo desta unidade.

Resumo da Unidade

Esta unidade apresentou os princípios fundamentais de como trabalhar com arquivos fontes C++ / C, compilação, ligação e depuração.

Avaliação da Unidade

Verifique a sua compreensão!

Instruções

1. Qual é o propósito de um Makefile?
2. Verifique o código fonte dado na pasta sistema de arquivos Linux / usr / src / linux
3. Explique de forma técnica, qual propósito do uso das bibliotecas nos codigos fontes C/ C++
4. Explique a finalidade dos argumentos argc , argv no trecho do codigo a seguir: int main (int argc, char **argv)
5. Quais as características de uma linguagem de baixo nivel
6. Qual a diferença e a importância de criar um programa C cujo código fonte termina com extensão .c / .h

Critérios de Avaliação

Como guiado pelos Regulamentos Instituição de classificação da Oferta

Leituras e Outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de "Leituras e Outros Recursos do curso".

1. Mark Mitchell, Jeffrey Oldham, e Alex Samuel; Linux Programação Avançada;

Unidade 1. As bibliotecas do C, Sistemas I/O e chamadas ao Sistema

Introdução à Unidade

A biblioteca padrão C oferece macros definições de tipo e funções para tarefas como manipulação de strings, cálculos matemáticos, processamentos de entrada / saída, alocação de memória e vários outros serviços do sistema operacional (SO). Esta unidade irá fazer uso da biblioteca GNU C, e assumir que você esteja pelo menos um pouco familiarizado com a linguagem de programação C e conceitos básicos de programação. Especificamente, a familiaridade com o padrão ISO, em vez de "tradicionais" pré-ISO linguagem C é considerado.

A biblioteca GNU C inclui vários arquivos de cabeçalho, cada um dos quais fornece definições e declarações para um grupo de instalações conexas; esta informação é utilizada pelo compilador C ao processar seu programa. Por exemplo, o arquivo de cabeçalho `stdio.h' declara instalações para a realização de entrada e saída, e o arquivo de cabeçalho ` string.h' declara utilitários de processamento de string. Há um grande número de funções na biblioteca GNU C e não é realista esperar que você será capaz de se lembrar exatamente como usar cada um deles. É mais importante para se tornar geralmente familiarizado com os tipos de instalações que a biblioteca oferece, de modo que quando você estiver escrevendo seus programas possa reconhecer quando fazer uso de funções de biblioteca. A finalidade desta unidade é instruir-nos como usar as facilidades da biblioteca GNU. Especificamente, a unidade deve discutir o arquivo comumente usados pelas funções de I/O e chamadas ao sistema e facilidades para arquivo, diretório e links manipulações de arquivos.

Objectivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

- Aplicar diferentes instalações da biblioteca C em programas em C
- Usar as funções de I/O e chamadas ao sistema
- Manipular arquivos, diretórios e links em programas em C

TERMOS-CHAVE

ISO: Organização dos Padrões Internacionais

GNU: GNU Não é Unix

POSIX: Extensão das Interface para Sistemas Operacionais Portaveis

BIBLIOTECAS C: Coleção de rotinas de programas pré-compilados disponíveis para reutilização em um programa C comum.

DIRETÓRIO / PASTA: Estrutura de sistema de arquivo no qual se armazena arquivos do computador

LINK / SIMI LINK : Um tipo especial de arquivo que contém uma referência a outro arquivo ou diretório sob a forma de um caminho absoluto ou relativo e que afeta resolução do caminho.

ARQUIVO: É uma unidade lógica de armazenamento de dados em sistemas operacionais

E/S: ENTRADA / SAÍDA: De entrada / saída (também "I/O" ou dita de "E/S") refere-se à actividade de ligação a um dispositivo de entrada ou de saída para a leitura ou a escrita de dados, respectivamente.

CHAMADA AO SISTEMA: É um pedido para o sistema operacional para fazer algo em nome do programa do usuário

Actividades de Aprendizagem

Actividade 1.1 - Visão Geral sobre as Bibliotecas C GNU

Introdução

A linguagem C não oferece facilidades embutidas para a realização de tais operações comuns como entrada / saída, gerenciamento de memória, a manipulação de string, e assim por diante. Em vez disso, estas facilidades são definidos em uma biblioteca padrão, que pode compilar e vincular com os seus programas .A biblioteca GNU C, descrito nesta unidade, define todas as funções de biblioteca que são especificadas pelo padrão ISO C, bem como recursos adicionais específicos para POSIX e outros derivados do sistema operacional Unix, e extensões específicas para o sistema GNU.

Detalhes da actividade

Padrão e Portabilidade

Esta seção aborda os vários padrões e outras fontes que a biblioteca GNU C está baseada. Estas fontes incluem o ISO C e padrões POSIX, e as implementações do System V e Berkeley Unix.

Esta secção dá-lhe uma visão geral desses padrões, de modo que você saiba o que eles são quando eles são mencionados em outras partes da unidade

A. ISO C - O padrão internacional para a programação C

A biblioteca GNU C é compatível com o padrão C adotado pelo Instituto Nacional Americano de Normas (ANSI): Padrão American National X3.159-1989 - "ANSI C" e, posteriormente, pela Organização das Normas Internacionais (ISO): ISO / IEC 9899: 1990, "linguagens de programação - C". Neste documento referimo-nos aqui a norma ISO C como um padrão que é genérico de ratificação. Os arquivos de cabeçalho e instalações da biblioteca que compõem a biblioteca GNU são um super conjunto especificados pela norma ISO C.

Se nós estamos preocupados com a estrita observância da norma ISO C, devemos usar a opção `ANSI' quando compilarmos os nossos programas com o compilador GNU C. Isso informa o compilador para definir apenas características padrão ISO a partir de arquivos de cabeçalho da biblioteca, a menos que você explicitamente peça recursos adicionais.

Ser capaz de restringir a biblioteca de modo a incluir apenas as características ISO C é importante porque ISO C coloca limitações em quais nomes podem ser definidos pela implementação da biblioteca, e as extensões GNU não se encaixam estas limitações.

Esta unidade não tenta dar-lhe detalhes completos sobre as diferenças entre a ISO C e linguagens mais velhas. Ele dá conselhos sobre como escrever programas para trabalhar provavelmente sob várias especificações C, mas não tem como objetivo ser documento completo sobre C.

B. POSIX (Extensão das Interface para Sistemas Operacionais Portaveis) - A ISO / IEC 9945 (aka IEEE 1003) normas para os sistemas operacionais

A biblioteca GNU também é compatível com a família ISO POSIX de padrões, conhecida formalmente como a Extensão das Interface para Sistemas Operacionais Portáveis para ambientes de computador (ISO / IEC 9945). Eles também foram publicados como ANSI / IEEE Std 1003. POSIX é derivada principalmente de várias versões do sistema operacional Unix.

As instalações da biblioteca especificadas pelos padrões POSIX são um super conjunto exigidos pela ISO C; POSIX especifica características adicionais das funções ISO C, bem como especificando novas funções adicionais. Em geral, os requisitos e funcionalidades adicionais definidas pelos padrões POSIX destinadas a proporcionar apoio a nível mais baixo para um determinado tipo de ambiente de sistema operacional, em vez de apoio geral a linguagem de programação que pode ser executado em muitos ambientes de sistemas operacionais diferentes.

A biblioteca GNU C implementa todas as funções especificadas na norma ISO / IEC 9945-1: 1996, o POSIX comumente referido como POSIX.1. As extensões primárias para as instalações ISO C especificadas por esta norma incluem arquivo de interface do sistema primitivos, funções de controle de terminais específicos de dispositivos e funções de controle de processo.

Algumas instalações de ISO / IEC 9945-2: 1993, a Shell POSIX e utilitários padrão (POSIX.2) também são implementadas na biblioteca GNU. Estes incluem utilitários para lidar com expressões regulares e outras instalações de correspondência de padrões.

C. Berkeley Unix - BSD e SunOS

A biblioteca GNU C define instalações de algumas versões do Unix, que não são formalmente padronizadas, especificamente a partir do 4.2 BSD, 4,3 BSD, e 4.4 Sistemas BSD Unix (também conhecido como Berkeley Unix) e de SunOS (um popular derivado 4,2 BSD que inclui alguns Sistema Unix funcionalidade V). Estes sistemas suportam a maioria das instalações ISO C e POSIX, e 4.4 BSD e mais recentes versões do SunOS, de facto, apoiá-los todos. As instalações BSD incluem links simbólicos, a função de seleção, as funções de sinal BSD e soquetes

D. SVID - Descrição da Interface do System V

A SVID é um documento que descreve o sistema operacional AT & T Unix System V. É até certo ponto, um super-conjunto do padrão POSIX.

A biblioteca GNU C define a maioria das instalações exigidas pelo SVID que não são também exigidos pelas normas ISO C ou POSIX, para compatibilidade com outros sistemas Unix (tais como o SunOS), que incluem estas instalações System V Unix. No entanto, muitas das instalações mais obscuras e menos, geralmente úteis exigidas pelo SVID não estão incluídos. (Na verdade, Unix-se System V não fornece todos eles). As instalações apoiadas de System V incluem os métodos de comunicação entre processos e memória compartilhada, a busca e funções da familia drand48 , fmtmsg e várias outras funções matemáticas.

E. XPG (A Guia de portabilidade X/Open)

O XPG, publicado pela companhia X/Open, Ltd., é um padrão mais geral do que POSIX. X/Open detém os direitos autorais do Unix e o XPG especifica os requisitos para sistemas que se destinam a ser um sistema Unix.A biblioteca GNU C obedece a X/Open - Guia de Portabilidade, Edição 4.2, com todas as extensões comuns a XSI (X / Open System Interface) sistemas compatíveis e também todas as extensões X / Open UNIX. As adições no topo de POSIX são derivados principalmente da funcionalidade disponível em sistemas System V e BSD. Alguns dos muitos erros ruins em sistemas System V foram no entanto, corrigidos. Desde o cumprimento da norma XPG, as extensões Unix tem sido uma condição prévia para obter as chances de marca Unix são boas para a funcionalidade e está disponível em sistemas comerciais.

Usando a Biblioteca

Esta seção descreve algumas das questões práticas envolvidas no uso da biblioteca GNU C.

A. Arquivos de cabeçalho

Bibliotecas para uso em programas C, realmente, consistem de duas partes: arquivos de cabeçalho que definem tipos e macros, declarar variáveis e funções; e a biblioteca ou arquivo real que contém as definições das variáveis e funções.

(Lembre-se que em C, a declaração apenas fornece informações que uma função ou variável

tem e dá o seu tipo. Para uma declaração de função, informações sobre os tipos de seus argumentos podem ser fornecidos também. O propósito das declarações é permitir que o compilador processa corretamente referências para as variáveis e funções declaradas. Uma definição, por outro lado, na verdade aloca armazenamento para uma variável ou diz o quê uma função faz.)

A fim de utilizar as instalações da biblioteca GNU C, você deve ter certeza que seu código fonte de programa inclui os arquivos de cabeçalho. Isso é para que o compilador tenha declarações destas instalações disponíveis e podem processar corretamente as referências a ele. Uma vez que seu programa tem sido compilado, o vinculador resolve estas referências às definições reais fornecidas no arquivo.

Arquivos de cabeçalho estão incluídas em um arquivo fonte do programa pela directiva de pré-processador "#include". A Linguagem C suporta duas formas desta directiva; o primeiro,

```
#include "header"
```

É normalmente usada para incluir 'header' um arquivo de cabeçalho que você escreva sozinho; este conterá as definições e declarações que descrevem as interfaces entre as diferentes partes da sua aplicação particular. Por exemplo,

```
#include <file.h>
```

É normalmente usado para incluir um arquivo de cabeçalho `file.h' que contém definições e declarações para uma biblioteca padrão. Este arquivo normalmente será instalada em um local padrão pelo seu administrador de sistema. Você deve usar essa segunda forma para os arquivos de biblioteca de cabeçalho C.

Normalmente, `directivas #include' são colocados na parte superior do código fonte C, antes de qualquer outro código. E se você começa seus arquivos de origem com alguns comentários explicando o que o código fonte faz (a boa idéia), coloque as directivas `# include' imediatamente depois, após o macro teste de definição de recursão.

A biblioteca GNU C fornece vários arquivos de cabeçalho, cada uma das quais contém o tipo e macro definições e declarações de variáveis e função para um grupo de instalações relacionadas. Isso significa que os programas podem precisar incluir vários arquivos de cabeçalho, dependendo exatamente o que facilidades que você está usando.

Alguns arquivos de cabeçalho da biblioteca incluem outros arquivos de cabeçalho biblioteca automaticamente. No entanto, como uma questão de estilo de programação, você não deve contar com isso; é melhor incluir explicitamente todo o cabeçalho arquivos necessários para as instalações da biblioteca que você está usando. Os arquivos de cabeçalho da biblioteca GNU C tem sido escrito de tal maneira que não interessa se um arquivo de cabeçalho é incluído mais do que acidentalmente uma vez; incluindo um arquivo de cabeçalho uma segunda vez não tem qualquer efeito. Da mesma forma, se o seu programa precisa incluir vários arquivos de cabeçalho, a ordem na qual estão incluídos não importa.

Nota de compatibilidade: Inclusão de arquivos de cabeçalho padrão em qualquer ordem e qualquer número de vezes funciona em qualquer aplicação ISO C. No entanto, este não tem sido tradicionalmente o caso em muitos implementações mais antigas C.

Estritamente falando, você não tem de incluir um arquivo de cabeçalho para usar uma função que ele declara; você poderia declarar a função explicitamente a si mesmo. Mas geralmente é melhor incluir o arquivo de cabeçalho porque pode definir os tipos e macros que não estão disponíveis de outra forma e porque pode definir mais eficientes substituições macro para algumas funções. É também uma maneira de ter a correta declaração.

B. Definições Macro de Funções

Se nós descrevemos algo como uma função, que pode ter uma definição de macro bem. Isto não tem normalmente nenhum efeito sobre a forma como o programa é executado - a definição de macro faz a mesma coisa que a função seria. Em particular, equivalentes macro para funções de biblioteca avaliar argumentos exatamente uma vez, em da mesma maneira que o faria uma chamada de função. A principal razão para estas definições de macro é que às vezes eles podem produzir uma expansão em linha que é consideravelmente mais rápido do que uma real chamada de função.

Tomando o endereço de uma função ou biblioteca funciona mesmo se ele também é definido como um macro. Isto é porque, neste contexto, o nome da função não é seguido pelo parêntese esquerdo que é sintaticamente necessário reconhecer uma chamada de macro. Ocasionalmente, você poderá querer evitar utilizar a definição de macro de uma função - talvez para fazer seu programa mais fácil de depurar. Há duas maneiras que você pode fazer isso:

- É possível evitar uma definição de macro de uma utilização específica colocando o nome da função em parênteses. Isso funciona porque o nome da função não aparece em uma sintática contexto em que é reconhecível como uma chamada de macro.
- Você pode suprimir qualquer definição de macro para um código fonte inteiro usando a diretiva do pré-processador `#undef ', salvo indicação em contrário explicitamente na descrição dessa instalação.

Por exemplo, suponha que o arquivo de cabeçalho `stdlib.h' declara uma função chamada abs como :

abs extern int (int);

e também fornece uma definição de macro para abs. Em seguida:

```
#include <stdlib.h>

int f (int * i) {return abs (++ * i); }
```

a referência ao abs pode se referir a uma macro ou uma função. Por outro lado, em cada um dos seguintes exemplos a referência é a uma função e não uma macro.

```
#include <stdlib.h>

int g (int * i) {return (abs) (++ * i); }

abs #undef

int h (int * i) {abs retorno (++ * i); }
```

Desde definições de macro que dobram para uma função se comportam exatamente da mesma forma como a função de versão real, normalmente não há necessidade de qualquer um desses métodos. De facto, a remoção de macro definições geralmente apenas faz o seu programa mais lento.

C. Nomes Reservados

Os nomes de todos os tipos de bibliotecas, macros, variáveis e funções que vêm do ISO C padrão são reservados incondicionalmente; seu programa não pode redefinir esses nomes. Todos os outros nomes da biblioteca são reservados se o seu programa inclui explicitamente o arquivo de cabeçalho que define ou declara-los. Há várias razões para essas restrições:

- Outras pessoas que lêem o seu código poderiam ficar muito confusos se você estivesse usando uma função saída chamado para fazer algo completamente diferente do que a função de saída padrão faz, por exemplo. Prevenir esta situação contribui para tornar seus programas mais fáceis de entender e contribui para a modularidade e facilidade de manutenção.
- Isso evita a possibilidade de um utilizador acidentalmente redefinindo uma função da biblioteca que é chamado por outras funções de biblioteca. Se redefinição for autorizada, essas outras funções não trabalhariam corretamente.
- Ele permite que o compilador faça otimizações especiais que agrada as chamadas para estas funções, sem a possibilidade de que eles podem ter sido redefinidos pelo utilizador. Algumas instalações da biblioteca, tais como aquelas para lidar com argumentos variáveis e saídas não locais, realmente necessitam de uma quantidade considerável de cooperação por parte do compilador C, e no que diz respeito à implementação, pode ser mais fácil para o compilador para tratar estes como partes embutidas na linguagem.

Além dos nomes documentados Biblioteca GNU, nomes reservados incluem todos identificadores externos (funções e variáveis globais) que começam com um sublinhado ('_') e todos os identificadores independentemente do uso que começam com duas sublinhados ou um sublinhado, seguido de um nomenclatura própria - são nomes reservados. Isso é para que os arquivos de biblioteca e cabeçalho pode definir funções, variáveis, e macros para fins internos, sem risco de conflito com nomes de programas do usuário.

Algumas classes adicionais de nomes de identificadores são reservados para futuras extensões para a linguagem C ou para o ambiente POSIX.1. Embora a utilização destes nomes para seus próprios fins agora pode não causar um problema, eles levantam a possibilidade de conflito com as futuras versões do C ou Padrões POSIX, assim que você deve evitar esses nomes.

- Nomes que começam com 'E' seguido um dígito ou letra maiúscula podem ser usados para nomes de código de erro adicionais.
- Nomes que começam com `LC_ ' seguido de uma letra maiúscula podem ser usados para suplementar macros especificando atributos localidade.
- Nomes de todas as funções matemáticas existentes seguida de `f ' ou 'l' são reservados para funções que operam em argumentos floats e longos duplos, correspondendo respectivamente.
- Nomes que começam com 'SIG' seguidos por uma letra maiúscula são reservados para suplementar nomes de sinal.
- Nomes que começam com `SIG_ 'seguido de uma letra maiúscula são reservados para suplementar ações de sinal.
- Nomes que começam com 'str', ou 'wcs' seguidos por uma letra minúscula são reservados para funções de string e de arrays adicionais.
- Nomes que terminam com '_t' são reservados para nomes de tipos adicionais.
- Além disso, alguns arquivos de cabeçalho individuais reservam nomes além daqueles que eles realmente definem. Você só precisa se preocupar com essas restrições se o seu programa inclui o cabeçalho de arquivo especial.
- O arquivo de cabeçalho `dirent.h ' reserva nomes prefixados com` d_'.
- O arquivo de cabeçalho `fcntl.h 'reserva nomes prefixados com` l_', `F_ ',` O_', e `S_ '.
- O arquivo de cabeçalho `grp.h 'nomes de reservas com o prefixo` gr_'.
- O arquivo de cabeçalho `limits.h 'nomes reservas seguida de` _MAX'.
- O arquivo de cabeçalho `pwd.h 'reserva nomes prefixados com` pw_'.
- O arquivo de cabeçalho `signal.h 'reserva nomes prefixados com` sa_' e `sa_ '.
- O arquivo de cabeçalho `sys / stat.h 'reserva nomes prefixados com` st_' e `S_ '.
- O arquivo de cabeçalho `sys / times.h 'reserva nomes prefixados com` tms_'.
- Os nomes das reservas prefixados com `c_ 'arquivo de cabeçalho` termios.h, `V',` I ',' O ', e `TC'; e nomes prefixados com 'B' seguido de um dígito.

D. Característica de testes de Macros

O conjunto exato de recursos disponíveis quando se compila um código fonte é controlado pela característica de teste de macros que se define. Se compilarmos os programas usando o `gcc -ANSI ', obtém apenas os recursos de biblioteca ISO C, a menos que se solicita explicitamente recursos adicionais através da definição de uma ou mais recursos de macros. Veja seção `GNU GCC Opções de comando" no GNU Manual do CC/GCC, para mais informações sobre GCC opções

Deve-se definir esses macros usando `directivas de pré-processador #define' no topo dos seus arquivos de código-fonte. Estas directivas devem vir antes de qualquer #include de um arquivo de cabeçalho do sistema. Isto é melhor para torná-los a primeira coisa no arquivo, precedido apenas por comentários. Você também pode usar a opção `D' para GCC, mas é melhor se fizer os arquivos antes de indicar o seu próprio significado em uma forma auto-suficiente. Este

sistema existe para permitir que a biblioteca esteja em conformidade com vários padrões. Embora em diferentes normas são muitas vezes descritas como supersets um do outro, eles são geralmente incompatíveis, porque padrões maiores requerem funções com nomes menores reservados para programa do usuário. Este tem sido um problema na prática. Por exemplo, algunsprogramas não-GNU definem funções nomeadas getline que não têm nada a ver com getline desta biblioteca. Eles não iriam ser compiláveis se todos os recursos foram habilitados indiscriminadamente. Isto não deve ser utilizado para verificar se um programa corresponde a uma norma limitada. É insuficiente para esta finalidade, pois não irá protegê-lo de incluir arquivos de cabeçalho fora do padrão, ou contando com a semântica indefinidas dentro do padrão.

Macro: _POSIX_SOURCE

Se você definir essa macro, em seguida, a funcionalidade do padrão POSIX.1 (IEEE Padrão 1003.1) está disponível, bem como todos os equipamentos ISO C.

O estado de _POSIX_SOURCE é irrelevante se você definir a macro _POSIX_C_SOURCE A um número inteiro positivo.

Macro: _POSIX_C_SOURCE

Definir essa macro para um número inteiro positivo para controlar quais funcionalidade POSIX é feita disponível. Quanto maior for o valor deste macro, mais funcionalidade serão disponibilizadas.

Se definirmos essa macro para um valor maior ou igual a 1, então a funcionalidade da edição da norma POSIX.1 (IEEE Padrão 1003.1-1990) 1990 estará disponível. Se definirmos essa macro para um valor maior ou igual a 2, em seguida, a funcionalidade de a edição da norma POSIX.2 (IEEE Padrão 1.003,2-1992) 1992 será disponível.

Se definir esta macro para um valor maior do que ou igual a 199309L, em seguida, o funcionalidade a partir da edição da norma POSIX.1b de 1993 (IEEE Padrão 1003.1b- 1993) será disponibilizada.

Valores maiores para _POSIX_C_SOURCE irá permitir futuras ampliações. O processo da norma do POSIX vai definir esses valores conforme necessário, e a Biblioteca C do GNU deve apoiá-los algum tempo depois eles se tornam padronizados. A edição de POSIX.1 1996 (ISO / IEC 9945-1: 1996) afirma que, se você definir _POSIX_C_SOURCE para um valor maior que ou igual a 199506L, em seguida, a funcionalidade da edição de 1996 será disponibilizado.

Macro: _BSD_SE

Essa macro definirá funcionalidade derivada do BSD4.3, Unix está incluído, bem como o ISO C, POSIX.1, POSIX.2 e materiais.

Algumas das características derivadas de BSD4.3 Unix conflituam com as características correspondentes especificadas pelo padrão POSIX.1. Se este macro é definido, as definições do 4,3 BSD tomam precedência sobre as definições POSIX.

Devido à natureza de alguns dos conflitos entre BSD4.3 e POSIX.1, precisa-se usar uma biblioteca de compatibilidade BSD especial ao vincular programas compilados para BSD

compatíveis. Isso ocorre porque algumas funções devem ser definidas de duas maneiras diferentes, um deles na biblioteca C normal, e um deles na biblioteca compatíveis. Se o seu programa define _BSD_SOURCE, deve dar a opção `-lbsd-compat ' para o compilador ou linker ao vincular o programa, para dizer-lhe para encontrar funções nesta Biblioteca especial de compatibilidade antes de verificá-los na biblioteca C normal.

Macro: _SVID_SOURCE

Se definirmos essa macro, funcionalidade derivada de SVID estará incluído, bem como a ISO C, POSIX.1, POSIX.2, e X / Open - Macros.

Macro: _XOPEN_SOURCE

Macro: _XOPEN_SOURCE_EXTENDED

Se você definir essa macro, funcionalidade descrita na X/Open Portability Guide é incluído. Este é um super conjunto de funcionalidades POSIX.1 e POSIX.2 e de fato _POSIX_SOURCE e _POSIX_C_SOURCE são automaticamente definidos. Se a macro _XOPEN_SOURCE_ EXTENDED também estiver definida, mais funcionalidade estarão igualmente disponíveis. As funções extra vão fazer todas as funções disponíveis que são necessários para a macro X/Open Unix.

Se a macro _XOPEN_SOURCE tiver o valor de 500 o que inclui todas as funcionalidades descrito até agora além de algumas novas definições do Single UNIX Specification, versão 2.

Macro: _LARGEFILE_SOURCE

Se esta macro for definida algumas funções extras estão disponíveis para corrigir algumas deficiências em todos os padrões anteriores. Especificamente, as funções fseek e ftell são disponíveis. Sem estas funções a diferença entre a interface ISO C (fseek, ftell) e o baixo nível da interface POSIX (lseek) levaria a problemas. Esta macro foi introduzida como parte da extensão para suporte de arquivo Grandes dito de LFS.

Macro: _LARGEFILE64_SOURCE

Se definirmos esse macro um conjunto adicional de funções fica disponível permitindo sistemas de 32 bit para usar arquivos de tamanhos além do limite normal de 2GB. Esta interface não está disponível se o sistema não suporta arquivos grandes. Em sistemas onde os arquivos naturais de limite de tamanho for maior do que 2B (ou seja, em sistemas de 64 bits) as novas funções são idênticas às funções substituídas.

A nova funcionalidade será disponibilizada por um novo conjunto de tipos e funções que substituem os já existentes. Os nomes desses novos objetos contêm 64 para indicar as suas variações, por exemplo, off_t vs. off64_t e fseek vs. fseek64.

Esta macro foi introduzida como parte da extensão do arquivo Grande (LFS). É uma interface de transição para o período em 64 deslocamentos de bits não são geralmente utilizados (ver _FILE_OFFSET_BITS).

Macro: _FILE_OFFSET_BITS

Esta macro determina qual a interface do sistema de arquivos deve ser usada, uma substituindo a outra.

Considerando _LARGEFILE64_SOURCE torna a interface 64bits disponível como uma interface adicional, _FILE_OFFSET_BITS permite que a interface de 64 bits substitua a velha interface.

Se _FILE_OFFSET_BITS estiver indefinido, ou se for definido para o valor de 32, nada altera. A interface de 32bits, off_t é utilizada como tipos que têm um tamanho de 32 bits em sistemas de 32 bits.

Se a macro for definida com o valor 64, a interface para arquivo grandes substitui a interface antiga. Ou seja, as funções não são disponibilizadas sob nomes diferentes (como eles estão com _LARGEFILE64_SOURCE). Em vez dos antigos nomes de função, agora referenciar as novas funções, por exemplo, uma chamada para fseek agora, na verdade chama fseek64.

Esta macro deve ser selecionada apenas se o sistema fornece mecanismos para lidar com arquivos grandes. Em sistemas de 64 bits esta macro não tem efeito desde que os * 64 funções são idênticas às funções normais. Esta macro foi introduzida como parte da LFS.

Macro: _ISOC99_SOURCE

Até que o padrão ISO C revista, for amplamente adotada os novos recursos não são automaticamente activados. A libc GNU, no entanto, tem uma implementação completa do novo padrão e para permitir que os novos recursos do macro _ISOC99_SOURCE devem ser definidos.

Macro: _GNU_SOURCE

Se definirmos essa macro, deve ficar incluído: ISO C89, C99 ISO, POSIX.1, POSIX.2, BSD, SVID, extensões X / Open, LFS, e GNU. Nos casos onde os conflitos POSIX.1 com BSD, as definições POSIX têm precedência. Se desejarmos obter o efeito completo de _GNU_SOURCE mas fazer as definições BSD tomar precedência sobre as definições POSIX, use esta seqüência de definições:

```
#define _GNU_SOURCE

#define _BSD_SOURCE

#define _SVID_SOURCE
```

Note que se fizer isso, deve-se vincular o programa com a biblioteca de compatibilidade BSD passando a opção `-lbsd-compat ' para o compilador ou vinculador (Linker) Nota: Se esquecermos de fazer isso, pode-se obter erros muito estranhos em tempo de execução.

Macro: _REENTRANT

Macro: _THREAD_SAFE

Se definirmos uma destas macros, versões de reentrada de várias funções serão declaradas. Algumas das funções são especificadas no POSIX.1c mas muitos outros estão disponíveis apenas em alguns outros sistemas ou são exclusivos para GNU libc. O problema é o atraso na padronização da interface de biblioteca de segurança de thread C.

Ao contrário de outros sistemas, nenhuma versão especial da biblioteca C deve ser utilizada para linking. Há apenas uma versão, mas ao compilar este deve ter sido especificado para compilar como thread de segurança.

Recomendamos que se use _GNU_SOURCE em novos programas. Se não especificarmos a opção de GCC 'ANSI' e não definirmos qualquer uma destas macros explicitamente, o efeito é o mesmo que define _POSIX_C_SOURCE para 2 e _POSIX_SOURCE, _SVID_SOURCE, e _BSD_SOURCE para 1.

Quando definirmos uma macro de teste de uma função para solicitar uma classe maior de recursos, ele é "inofensivo" para definir além de uma macro de teste de recurso para um subconjunto desses recursos. Por exemplo, se definirmos _POSIX_C_SOURCE, Em seguida, definir _POSIX_SOURCE assim não tem efeito. Da mesma forma, se você definir _GNU_SOURCE, em seguida, definir tanto _POSIX_SOURCE ou _POSIX_C_SOURCE ou _SVID_SOURCE assim não terá nenhum efeito.

Note-se, contudo, que as características de _BSD_SOURCE não são um subconjunto de qualquer um dos outros recursos macros de teste suportados. Isso é porque ele define características BSD que têm precedência sobre o Recursos POSIX que são solicitados pelas outras macros. Por esta razão, definindo _BSD_SOURCE para além das outras macros de teste de dispositivos tem um efeito: o que faz com que características BSD tenham prioridade sobre as características POSIX.

A Biblioteca C de Aplicação de Armazenamento de Dados do Programa

A linguagem C suporta dois tipos de alocação de memória através das variáveis em programas em C:

- atribuição estática é o que acontece quando você declara uma variável estática ou global. Cada variável estática ou global define um bloco de espaço, de um tamanho fixo. O espaço é alocado uma vez, quando o programa for iniciado (parte da operação exec), e nunca é liberada.

- Alocação automática acontece quando você declarar uma variável automática, como uma função argumento ou uma variável local. O espaço para uma variável automática é atribuída quando a instrução composta que contém a declaração for inserida, e é liberada quando essa instrução composta é encerrada. Um terceiro tipo importante de alocação de memória, alocação dinâmica, não é suportado por variáveis C mas está disponível através de funções de biblioteca GNU C.

A. Alocação Dinâmica da Memória

Alocação dinâmica de memória é uma técnica na qual os programas determinam como eles estão executando e aonde armazenar algumas informações. Precisa-se de alocação dinâmica quando a quantidade de memória que se precisa, depende de fatores que não são conhecidos antes do programa ser executado.

Por exemplo, pode precisar de um bloco para armazenar uma linha de leitura a partir de um arquivo de entrada; uma vez que não há nenhum limite de quanto tempo a linha pode ocupar, deve alocar a memória dinamicamente e torná-lo dinamicamente aumentada conforme se lê mais linha. Ainda pode-se precisar de um bloco para cada registo ou de cada definição nos dados de entrada; desde que não pode saber de antemão quantos haverá, deve alocar um novo bloco para cada registo ou definição de como lê-lo.

Quando usa a alocação dinâmica, a atribuição de um bloco de memória é uma ação que o programa solicita explicitamente. Chama-se uma função ou macro quando deseja alocar espaço e especificar o tamanho com um argumento. Se quisermos liberar o espaço, faz-se chamando outra função ou macro. Pode-se fazer essas coisas quando quiser, quantas vezes quisermos.

Alocação dinâmica não é suportada pelo variáveis C; não há nenhuma classe de armazenamento dinâmica. A única maneira de alocação de memória dinamicamente é através de uma chamada ao sistema (que é geralmente através de uma Biblioteca GNU C chamada de função), e a única maneira de se referir a espaço alocado dinamicamente é através de um ponteiro.

B. Alocação básica da memória

A função de alocação dinâmica mais geral é malloc. Ela permite alocar blocos de memória de qualquer tamanho, em qualquer momento, torná-los maiores ou menores, a qualquer momento, e libertar os blocos individualmente, em qualquer momento (ou nunca). Para alocar um bloco de memória, usa-se malloc. O protótipo para esta função está em 'stdlib.h'.

Porque é menos conveniente, e porque o processo real de alocação dinâmica requer mais tempo de computação; Programadores geralmente usam alocação dinâmica somente quando alocação estática automática não funciona. Por exemplo, se quiser alocar dinamicamente algum espaço para realizar um foobar struct, não pode-se declarar uma variável do tipo foobar struct cujo conteúdo do espaço é alocado dinamicamente. Mas pode declarar uma variável do tipo ponteiro foobar struct * e atribuir-lhe o endereço do espaço. Então você pode usar os operadores de '*' e '->' sobre este variável - ponteiro para fazer referência ao conteúdo do espaço:

```
{
struct foobar * ptr = (foobar struct *) malloc (sizeof (struct
foobar));
ptr-> name = x;
ptr-> next = current_foobar;
current_foobar = ptr;
}
void * malloc (size size_t) function
```

Esta função retorna um ponteiro para recém-alocados bytes de tamanho de bloco longo ou um ponteiro nulo se o bloco não pode ser alocado. Os conteúdos do bloco são indefinidos; deve-se inicializa-lo manualmente. Normalmente convertemos o valor como um ponteiro para o tipo de objeto que se deseja armazenar no bloco. Veja um exemplo de fazê-lo, e de inicialização do espaço com zeros usando o função de biblioteca memset , abaixo.

```
struct foo * ptr;
...
ptr = (struct foo *) malloc (sizeof (foo struct));
if (ptr == 0) abort ();
memset (PTR, 0, sizeof (struct foo));
```

Pode-se armazenar o resultado de malloc em qualquer variável de ponteiro, sem um elenco, porque ISO converte automaticamente o tipo void * a outro tipo de ponteiro quando necessário. Mas o elenco é necessário que não sejam operadores de atribuição ou se contextos você pode querer o seu código para ser executado em tradicional C.

Lembre-se que ao alocar espaço para uma string, o argumento para malloc deve ser um mais a comprimento da corda. Isso ocorre porque a cadeia é terminado com um caractere nulo que não conta no "comprimento" da corda, mas não precisa de espaço. Por exemplo:

```
char * ptr;
...
ptr = (char *) malloc (comprimento + 1);
```

O bloco que malloc dá-lhe é garantido para ser alinhados de forma que ele pode armazenar qualquer tipo de dados.

No sistema GNU, o endereço é sempre um múltiplo de oito na maioria dos sistemas, e um múltiplo de dezasseis anos em sistemas de 64 bits.

C. liberando memória alocada com malloc

Quando você já não precisa de um bloco que você tem com malloc, use a função livre para fazer o bloco disponível para ser atribuído novamente. O protótipo para esta função é em 'stdlib.h'.

```
void free (void * ptr) function
```

A função free desaloca o bloco de memória apontado para por ptr.

```
void cfree (void * ptr) function
```

Esta função faz a mesma coisa como livre. É fornecida para compatibilidade com SunOS; você deve usar livre em vez.

Libertar um bloco altera o conteúdo do bloco. Não espere para encontrar quaisquer dados (como um ponteiro para o próximo bloco em uma cadeia de blocos) no bloco depois de libertá-lo. Copiar o que você precisa para fora do bloco antes de libertá-lo! Aqui está um exemplo da maneira correta de libertar todos os blocos em uma chain, e as cordas que eles apontam para:

```
struct chain
  {
struct chain *next;
char *name;
  }
   void free_chain (struct chain *chain)
{
while (chain != 0)
{
struct chain *next = chain->next;
free (chain->name);
free (chain);
chain = next;
    }
}
```

Ocasionalmente, gratuito pode realmente voltar a memória para o sistema operacional e tornar o processo menor. Geralmente, tudo o que pode fazer é permitir que uma chamada mais tarde para malloc reutilizar o espaço. Enquanto isso, o espaço permanece em seu programa como parte de uma lista de livre usado internamente pelo malloc.

Não há nenhum ponto em libertar blocos no final de um programa, porque todo o espaço do programa é dado de volta para o sistema quando o processo termina.

Instalações D. Outros relacionados

```
void * realloc (void * ptr, newSize size_t) function
```

A função realloc altera o tamanho do bloco cujo endereço é ptr ser newSize.

```
void * calloc (size_t contagem, eltsize size_t) função
```

Essa função aloca um bloco de tempo suficiente para conter um vetor de elementos de contagem, cada um de tamanho eltsize. Seu conteúdo é zerado antes calloc retornos.

Conclusão

Esta unidade forneceu uma visão geral da Biblioteca C GNU. Uma série de questões foram apresentados, incluindo as Normas de biblioteca e portabilidade, os fundamentos do uso do biblioteca, e no caso de funções de biblioteca C para a alocação de armazenamento para os dados do programa.

Avaliação

1. Se não houver mais espaço disponível, malloc retorna um ponteiro nulo. Você deve verificar o valor de cada chamada para malloc. Em vez disso, é útil para escrever um sub-rotina que chama malloc e relata um erro se o valor for um valor nulo ponteiro, retornando somente se o valor for diferente de zero. Esta função é

   convencionou chamar xmalloc (veja Unit_2_Demos \ xmalloc.c). Estude este programa.

2. Demonstrar o uso de malloc, escrevendo um programa completo C com base no função fornecida Unit_2_Demos \ savestring.c que copiar uma sequência

   de caracteres em uma sequência nullterminated recém-alocado.

3. Escreva uma sub-rotina, chamada realloc, que cuida da mensagem de erro como xmalloc faz para malloc (REFERÊ a questionar No. 1 acima)

4. Definir a função calloc usando malloc

Actividade 1.2 -Arquivos, dorectórios e Links

Introdução

Nesta seção, apresentamos as funções da biblioteca GNU C para manipular arquivos. Estas funções estão preocupadas com o operar sobre os ficheiros, em vez de no seu conteúdo. Entre as facilidades descritas nesta seção são funções para examinar ou modificar directórios, funções para renomear e excluir arquivos e funções de análise e definição atributos de arquivo como permissões de acesso e tempos de modificação.

Detalhes da atividade

Examinando ou modificar o Directório

A. Manipulando o diretório de trabalho

Cada processo tem associado a ele um diretório, chamado de diretório de trabalho atual ou simplesmente directório de trabalho que é usado na resolução de nomes de arquivos relativos. Quando você login e iniciar uma nova sessão, seu diretório de trabalho é definido inicialmente para o diretório associado com sua conta de login no banco de dados do usuário do sistema. Você pode encontrar inicial de qualquer usuário usando o funções getpwuid ou getpwnam.

Os usuários podem alterar o diretório de trabalho usando comandos shell como cd. As funções descritas nesta secção são os primitivos usados por esses comandos e por outros programas para o exame e mudar o diretório de trabalho.

Protótipos para estas funções são declaradas no arquivo de cabeçalho `unistd.h '.

```
Função: char * getcwd (char * buffer, o tamanho size_t)
```

A função getcwd retorna um nome de arquivo absoluto representando o trabalho atual diretório, armazená-lo no array de caracteres de buffer que você fornecer. O tamanho argumento é como você informar ao sistema o tamanho de alocação de memória intermédia . A versão da biblioteca GNU desta função também permite que você especifique um ponteiro nulo para o buffer argumento. Então getcwd atribui uma memória intermédia automaticamente, como com malloc. Se o tamanho é maior do que zero, então o buffer é muito grande; de outra forma, a memória intermédia é tão grande quanto necessária para manter o resultado.

O valor de retorno é buffer em caso de sucesso e um ponteiro nulo em caso de falha.

A Tabela 1 define o condições de erro errno para a função getcwd.

Erros da função getcwd	Descrição do erro
EINVAL	O tamanho argumento é zero e buffer não é um ponteiro nulo.
ERANGE	O tamanho argumento é menor do que o comprimento do directório de trabalho nome. Você precisa alocar uma matriz maior e tente novamente
EACCES	Permissão para ler ou pesquisar um componente do nome do arquivo era negado.

Função: char * getwd (char * buffer)

Isto é semelhante ao getcwd, mas não tem qualquer maneira de especificar o tamanho da memória intermédia. O GNU biblioteca fornece getwd apenas para compatibilidade com BSD.

O argumento buffer deve ser um ponteiro para uma matriz de pelo menos PATH_MAX bytes. Dentro do sistema GNU, não há limite para o tamanho de um nome de arquivo, de modo que este não é necessariamente espaço suficiente para conter o nome do diretório. É por isso que esta função está obsoleta.

Função: int chdir (const char * filename)

Esta função é utilizada para definir o diretório de trabalho do processo para filename .

O valor, retorno bem sucedido normal a partir chdir é 0. Um valor de -1 é retornado para indicar um erro. As condições de erro errno definidos para esta função é o nome do arquivo de costume erros de sintaxe, além de ENOTDIR se o arquivo nome do arquivo não é um diretório.

B. Acessando Diretórios

Esta seção apresenta as facilidades que permitem ler o conteúdo de um arquivo de diretório. Isto é útil se você quer que seu programa para listar todos os arquivos em um diretório, talvez como parte de um menu. A função opendir abre um fluxo de diretório cujos elementos são entradas de diretório. Usa a função readdir no fluxo de diretório para recuperar essas entradas, representada como dirent struct objetos. O nome do arquivo para cada entrada é armazenado no membro d_name desta estrutura. A Tabela 2 apresenta o que encontramos em uma única entrada de diretório, como você pode obtê-lo a partir de um diretório

corrente. Todos os símbolos são declarados no arquivo de cabeçalho `dirent.h '.

Tabela 2: Formato de uma entrada de diretório como obtido a partir de um fluxo de diretório.

Tipo de dados: dirent struct Este é um tipo de estrutura utilizada para retornar informações sobre entradas de diretório. Ele contém o seguintes campos:	
Nome de campos	Significados
Char d_name []	Este é o componente nome do arquivo terminada em nulo. Este é o único campo que você pode contar em todos os sistemas POSIX
d_fileno ino_t	Este é o número de série do ficheiro. Para compatibilidade BSD, você também pode se referir a este membro como d_ino. No sistema GNU ea maioria dos sistemas POSIX, por a maioria dos arquivos Este é o mesmo como o membro st_ino que STAT voltar para o Arquivo.
unsigned char d_namlen	Este é o comprimento do nome do arquivo, não incluindo o nulo de terminação personagem. Seu tipo é unsigned char porque esse é o tipo inteiro do tamanho adequado.

<table>
<tr><td>unsigned char d_type</td><td>Este é o tipo de ficheiro, possivelmente, desconhecido. As seguintes constantes são definido para o seu valor:<table><tr><td>CONSTANTES</td><td>SIGNIFICADO</td></tr><tr><td>DT_UNKNOW</td><td>O tipo é desconhecido. Em alguns sistemas, esta é a única;</td></tr><tr><td>N</td><td>Valor devolvido</td></tr><tr><td>DT_REG</td><td>Um arquivo regular.</td></tr><tr><td>DT_DIR</td><td>Um diretório.</td></tr><tr><td>DT_FIFO</td><td>Um pipe nomeado, ou FIFO</td></tr><tr><td>DT_SOCK</td><td>Um socket do domínio local.</td></tr><tr><td>DT_CHR</td><td>Um dispositivo de caracteres.</td></tr><tr><td>DT_BLK</td><td>Um dispositivo de bloco.</td></tr></table></td></tr>
<tr><td></td><td>Esse membro é uma extensão do BSD. Cada valor, exceto DT_UNKNOWN corresponde aos bits de tipo de arquivo no membro st_mode de statbuf struct. Estas duas macros converter entre valores d_type e valores st_mode:

Função: int IFTODT (modo mode_t) : retorna o valor d_type correspondente ao modo .

Função: mode_t DTTOIF (int dirtype) : retorna o valor st_mode correspondente a dirtype .</td></tr>
<tr><td colspan="2">Esta estrutura pode conter membros adicionais no futuro. Quando um arquivo tem vários nomes, cada nome tem a sua própria entrada de diretório. A única maneira que você pode dizer que as entradas de diretório pertencem a um único arquivo é que eles têm o mesmo valor para o campo d_fileno. atributos de arquivo como o tamanho, os tempos de modificação, e similares são parte do arquivo em si, e não qualquer especial entrada de diretório. qualquer especial entrada de diretório.</td></tr>
</table>

C. Abrir um fluxo Diretório

Esta seção descreve como abrir um fluxo de diretório. Todos os símbolos são declarados no cabeçalho arquivo `dirent.h '.

Tipo de dados: DIR

O tipo de dados DIR representa um fluxo de diretório.

Você não deve nunca alocar objetos do dirent struct ou tipos de dados DIR, uma vez que o diretório funções de acesso fazer isso por você. Em vez disso, você se refere a estes objetos usando os ponteiros retornados para a função especificada na Tabela 3.

Tabela 3: As funções opendir

Função: DIR * opendir (const char * dirname) A função opendir abre e retorna um fluxo de diretório para a leitura do diretório cujo nome de arquivo é dirname . O fluxo tem o tipo DIR *. Se não tiver êxito, opendir retorna um ponteiro nulo. Além da sintaxe do nome de arquivo habitual erros, as seguintes condições de erro errno são definidas por esta função:	
Condição de erro	Significado
EACCES	permissão de leitura é negado para o diretório nomeado por dirname.
EMFILE	O processo tem muitos arquivos aberto.
ENFILE	O sistema inteiro, ou talvez o sistema de arquivos que contém o diretório, não pode apoiar todos os arquivos abertos adicionais na momento. (Este problema não pode acontecer no sistema GNU.)

O tipo DIR é tipicamente implementado usando um descritor de arquivo, ea função opendir em termos da função aberta. fluxos de diretório e os descritores de arquivos subjacentes estão fechados no exec.

D. Leitura e fechar Corrente Diretório

A Tabela 4 apresenta as funções utilizadas para ler entradas de diretório a partir de um fluxo de diretório, e perto o fluxo de quando você é feito com ele. Todos os símbolos são declarados no arquivo de cabeçalho `dirent.h.

A Tabela 4: Funções usado para ler as entradas de diretório a partir de um fluxo de diretório

<table>
<tr><td colspan="3">Função: dirent struct * readdir (DIR * dirstream) Esta função lê a próxima entrada a partir do directório. É normalmente retorna um ponteiro para uma estrutura que contém informações sobre o arquivo. Esta estrutura é alocada estaticamente e pode ser reescrita por uma chamada subseqüente.

Portabilidade Nota: Em alguns sistemas, readdir pode não retornar entradas para ` '. e `.. ', mesmo embora estes são sempre os nomes de arquivos válidos em qualquer diretório.

Se não houver mais entradas no diretório ou um erro é detectado, readdir retorna um valor nulo ponteiro. As seguintes condições de erro errno são definidos para esta função:</td></tr>
<tr><td></td><td>Condição de erro</td><td>Segnificado</td></tr>
<tr><td></td><td>EBAD

F</td><td>O dirstream argumento não é válido</td></tr>
<tr><td colspan="3">Função: int closedir (DIR * dirstream)

Esta função fecha o fluxo de diretório dirstream . Ele retorna 0 em caso de sucesso e -1 em caso de falha.

As seguintes condições de erro errno são definidos para esta função:</td></tr>
<tr><td></td><td>Condição de Erro</td><td>Significado</td></tr>
<tr><td></td><td>Erro</td><td></td></tr>
<tr><td></td><td>EBADF</td><td>O dirstream argumento não é válido.</td></tr>
</table>

E. Random Access em um córrego Diretório

A Tabela 5 apresenta as funções utilizadas para reler partes de um diretório que você já tenha lido a partir um fluxo de diretório aberto. Todos os símbolos são declarados no arquivo de cabeçalho `dirent.h '.

Tabela 5: Funções utilizadas para reler partes de um diretório que você já leu de uma aberta fluxo de diretório.

Função: void rewinddir (* dirstream DIR) A função rewinddir é usado para reinicializar o fluxo diretório dirstream , de modo que se você chama readdir retorna informações sobre a primeira entrada no diretório novamente. este função também percebe se os arquivos foram adicionados ou removidos para o diretório desde que foi aberto com opendir. (As entradas para esses arquivos podem ou não ser devolvido por readdir se eles foram adicionados ou removidos desde a última chamada opendir ou rewinddir.)
Função: off_t telldir (DIR * dirstream) A função telldir retorna a posição do fluxo de diretório de arquivos dirstream . Você pode utilizar este valor com seekdir para restaurar o fluxo de diretório para essa posição.
Função: void seekdir (DIR * dirstream, off_t pos) A função seekdir define a posição do fluxo de diretório de arquivos dirstream para pos . O valor pos deve ser o resultado de uma chamada anterior para telldir neste fluxo particular; encerramento e reabrir o diretório pode invalidar os valores retornados por telldir

Trabalhando com vários nomes de arquivo

Em sistemas POSIX, um arquivo pode ter vários nomes ao mesmo tempo. O nome adicional ao

nome (nomes) existente é chamado de um hard link para o arquivo. Todos os nomes são igualmente reais, e nenhum um deles é preferido para os outros. Um arquivo pode ter nomes em vários diretórios, de modo que o organização do sistema de arquivos não é uma estrita hierarquia ou árvore. Na maioria das implementações, que não está possível ter ligações diretas para o mesmo arquivo em vários sistemas de arquivos.

O sistema GNU suporta soft links ou links simbólicos . Esta é uma espécie de "arquivo", que é essencialmente um ponteiro para outro nome de arquivo. Ao contrário de hard links, links simbólicos podem ser feitas aos diretórios ou todo arquivo sistemas sem restrições. Você também pode fazer um link simbólico para um nome que é não o nome de qualquer arquivo (Abrir esta ligação irá falhar até que um arquivo com esse nome é criado).

Da mesma forma, se o link simbólico aponta para um arquivo existente que posteriormente é excluído, o link simbólico continua a apontar para o mesmo nome de arquivo mesmo que o nome nomes mais nenhum arquivo.

A razão links simbólicos funcionam da maneira que eles fazem é que as coisas especiais acontecem quando você tentar abrir o link. A função de abertura percebe que você tenha especificado o nome de um link, lê o arquivo nome contido no link, e abre o nome do arquivo em vez disso. A função de estatísticas explora igualmente no arquivo que o link simbólico aponta para, em vez de no próprio link.

Por outro lado, outras operações como apagar ou renomear o arquivo operar no próprio link. A sfunções readlink e lstat também abster-se de seguir links simbólicos, porque o seu objectivo é para se obter informação sobre a ligação. O mesmo acontece com link, a função que faz um hard link - ele faz um hard link para o link simbólico, que raramente quer.

A. Criando Hard Links

Para adicionar um nome para um arquivo, use a função de ligação. Criando um novo link para um arquivo não copiar o conteúdo do arquivo; faz simplesmente um novo nome pelo qual o arquivopode ser conhecida, para além nome existente do arquivo ou nomes.

O link para a função relata um erro se você tentar fazer um hard link para o arquivo de outro arquivo Quando este sistema não pode ser feito. O protótipo para a função de ligação é declarado no cabeçalho arquivo `unistd.h '. Quadro 6 fornece o formato da função de ligação .

Tabela 6: O formato da função de link

Função: Int link (const char * oldname, char const * newname) A função link faz com que um novo link para o arquivo existente chamado por oldname , no âmbito do novo nomear newname . Esta função retorna um valor de 0 se for bem sucedido e -1 em caso de falha. Em adição aos erros habituais nome do arquivo de sintaxe para ambos oldname e newname , o seguinte erro errno condições são definidas para esta função:		
	Condição de erro	significado / descrição
	EACCES	O diretório no qual o novo link é para ser escrito não é gravável.
	EEXIST	Já existe um arquivo chamado newname . Se você quisersubstituir esta ligação com um novo link, você deve remover o antigo vincular explicitamente pela primeira vez

	EMLINK	Há já muitos links para o arquivo nomeado por oldname . (O número máximo de links para um arquivo é LINK_MAX; sistemas de arquivos bem desenhados nunca relatório deste erro, porque permitem mais links do que o disco poderia possivelmente segurar. No entanto, você ainda deve ter em conta a possibilidade de este erro, uma vez que poderia resultar da rede acesso a um sistema de arquivos em outra máquina.
	ENOENT	O arquivo chamado por oldname não existe. Você não pode fazer uma link para um arquivo que não existe
	ENOSPC	O sistema de diretório ou arquivo que contém o novo link é "cheio" e não pode ser prorrogado
	EROFS	O diretório que contém o novo link não pode ser modificado porque é em um sistema de arquivos somente leitura.
	EXDEV	O diretório especificado no newname está em um arquivo diferente sistema que o arquivo existente.

B. Criação de Links Simbólicos

As funções link simbólico e readlink são usados para manupular link simbólico em arquivos. protótipos para essas funções estão em `unistd.h '. A Tabela 7 descreve o formato da função de ligação simbólica .

Tabela 7: A descrição da função symlink.

Função Int: symlink (const char * oldname, const char * newname) A função symlink faz um link simbólico para oldname chamado newname O valor de retorno normal a partir do link simbólico é 0. Um valor de retorno -1 indica um erro. Além das habituais erros de nome de arquivo de sintaxe, as seguintes condições de erro são errno definido para esta função:

	condição de erro	Já existe um arquivo existente com o nome newname . .
	EEXIST	Já existe um arquivo existente com o nome newname .
	ENOSPC	O sistema de diretório ou arquivo não pode ser estendido para tornar o novo link
	EIO	Ocorreu um erro de hardware durante a leitura ou escrever dados no disco
	EROFS	O arquivo newname existiria em uma leitura único sistema de arquivos

Função: int readlink (const char * filename, char * buffer, o tamanho size_t)

A função readlink obtém o valor do link simbólico nome de arquivo . O nome do arquivo que o link aponta é copiado para buffer . Esta cadeia de nome de arquivo é não null-rescindido; readlink normalmente retorna o número de caracteres copiados. O tamanho argumento especifica o número máximo de caracteres para copiar, geralmente o tamanho de alocação de memória intermédia .

Um valor de -1 é devolvido em caso de erro. Para além dos erros de nome de arquivo de sintaxe habitual, as seguintes condições de erro errno são definidos para esta função:

	condição de erro	Significado / descrição
	EINVAL	O arquivo nomeado não é um link simbólico.
	EIO	Ocorreu um erro de hardware durante a leitura ou escrever dados no disco.

Conclusão

Nesta actividade, descrever as funções da biblioteca GNU C para a manipulação de arquivos e diretórios. Especificamente, funções para examinar ou modificar diretórios e trabalhando com vários nomes de arquivos conforme foram apresentados.

Avaliação

eia os exemplos de código fornecidos abaixo e aplicá-los a implementar um programa de trabalho (I) Aqui está um exemplo mostrando como você poderia implementar o comportamento de getcwd do GNU (NULL, 0) usando apenas o comportamento padrão de getcwd

```
char *

gnu_getcwd ()

{

int size = 100;

char *buffer = (char *) xmalloc (size);

while (1)

{

char *value = getcwd (buffer, size);

if (value != 0)

return buffer;

size *= 2;

free (buffer);

buffer = (char *) xmalloc (size);

 }

}
```

Nota : Consulte a seção Atividade 1.1 acima, para obter informações sobre xmalloc, que não é uma função de biblioteca, mas é um nome habitual utilizado na maioria dos softwares GNU.

(II) Aqui está um programa simples que imprime os nomes dos arquivos no trabalho atual diretório.

```
#include <stddef.h>

#include <stdio.h>

#include <sys/types.h>
```

```
#include <dirent.h>

int

main (void)

{

DIR *dp;

struct dirent *ep;

dp = opendir (“./”);

if (dp != NULL)

{

while (ep = readdir (dp))

puts (ep->d_name);

(void) closedir (dp);

}

else

puts (“Couldn’t open the directory.”);

return 0;

}
```

A ordem em que os arquivos aparecem em um diretório tende a ser bastante aleatória. A mais útil programa seria classificar as entradas (talvez por alfabetizar-los) antes de imprimi-los.

(III) Se o valor de retorno é igual tamanho , você não pode dizer se ou não havia espaço para voltar o nome inteiro. Então, faça um buffer maior e chamar readlink novamente. Aqui está um exemplo

```
char*

readlink_malloc (char *filename)

{

int size = 100;

while (1)

{

char *buffer = (char *) xmalloc (size);

int nchars = readlink (filename, buffer, size);

if (nchars < size)
```

```
return buffer;

free (buffer);

size *= 2;

}

}
```

Actividade 1.3 -File chamadas Input / Output System

Introdução

Programadores C em GNU / Linux têm dois conjuntos de funções de entrada / saída à sua disposição. A biblioteca padrão C fornece funções de I / O: printf, fopen, e assim por diante. O próprio kernel do Linux fornece outro conjunto de operações de I / O, que operam a um nível mais baixo do que as funções de biblioteca C.

Porque este módulo é para pessoas que já conhecem a linguagem C, vamos supor que você tem

encontrado e sabe como usar a biblioteca C funções de I / O. Muitas vezes, há boas razões para

utilizar as funções de I / O de baixo nível do Linux. Muitos destes são chamadas do sistema kernel, e fornecer o mais acesso direto aos recursos do sistema subjacente que está disponível para programas de aplicação. De fato, a biblioteca C padrão I / O rotinas são implementadas em cima do baixo nível Linux sistema de I / O chama. Usando esta última é geralmente o modo mais eficiente para executar a entrada e saída de Operações- e é, por vezes, mais conveniente, também.

Detalhes da atividade

Leitura e Escrita de Dados

A função primeira de I / O que você provavelmente encontrou quando você primeiro aprendeu a linguagem C foi printf. Isto formata uma cadeia de texto e, em seguida, imprime-lo para a saída padrão. A generalizada versão, fprintf, pode imprimir o texto para um fluxo diferente de saída padrão. Um fluxo é representado por um ponteiro FILE *. Você obter um * ponteiro arquivo abrindo um arquivo com fopen.

Quando estiver pronto, você pode fechá-lo com fclose. Além fprintf, você pode usar tais funciona como fputc, fputs e fwrite para gravar dados para o fluxo, ou fscanf, fgetc, fgets e fread para ler os dados.

Com o baixo nível Linux I / O operações, você deve usar um identificador de chamada um descritor de arquivo em vez de um ponteiro FILE *. Um descritor de arquivo é um valor inteiro que se refere a um caso particular de um arquivo aberto em um único processo. Ele pode ser aberto para leitura, para a escrita, ou para ambos

leitura e escrita. Um descritor de arquivo não tem para se referir a um arquivo aberto; pode representar uma ligação com um outro componente do sistema que é capaz de enviar ou

recebendo dados. Por exemplo, uma ligação a um dispositivo de hardware é representado por um arquivo descritor, como é uma tomada aberta ou uma extremidade de um tubo. Inclui os arquivo de cabeçalho <fcntl.h>,Sys / types.h>, <sys / stat.h> e <unistd.h> se você usar qualquer um dos de baixo nível funções de I / O descrito aqui.

A. Abrindo um arquivo

Para abrir um arquivo e produzir um descritor de arquivo que pode acessar esse arquivo, utilize o aberto chamada. isto toma como argumentos o nome do caminho do arquivo a ser aberto, como uma cadeia de caracteres, e bandeiras especificando como abrí-lo. Você pode usar aberto para criar um novo arquivo; se o fizer, passar um terço argumento de que especifica as permissões de acesso para definir para o novo arquivo. Se o segundo argumento é O_ RDONLY, o arquivo é aberto somente para leitura; ocorrerá um erro se você posteriormente tentar escrever para o descritor de arquivo resultante. Do mesmo modo, faz com que O_ WRONLY o descritor de arquivo para ser somente gravação. Especificando O_RDWR produz um descritor de arquivo que pode ser utilizado tanto para a leitura e para a escrita. Note que nem todos os arquivos podem ser abertos em todos três modos. Por exemplo, as permissões em um arquivo podem proibir um determinado processo a partir de abrí-lo para ler ou para escrever; um arquivo em um dispositivo só de leitura, como um CD- ROM não pode ser aberto para gravação. Você pode especificar opções adicionais usando o bit a bit ou desse valor com um ou mais bandeiras. Estes são os valores mais comumente utilizados:

- Especifique O_TRUNC para truncar o arquivo aberto, se existia anteriormente. Dados escritos para o descritor de arquivo irá substituir o conteúdo anterior do arquivo.
- Especifique O_APPEND para anexar a um arquivo existente. Dados escritos para o arquivo Descritor vai ser adicionado ao fim do ficheiro.
- Especifique O_CREAT para criar um novo arquivo. Se o nome do arquivo que você fornece a não aberto não existir, um novo arquivo será criado, desde que o diretório contendo ele existe e que o processo tem permissão para criar arquivos em esse diretório. Se o arquivo já existe, ele é aberto em vez disso.
- Especifique O_EXCL com O_CREAT para forçar a criação de um novo arquivo. Se o arquivo já existe, a chamada aberta falhará.

Se você chamar aberto com O_CREAT, fornecer um terceiro argumento adicional especificando o permissões para o novo arquivo. Por exemplo, o programa dado em questão No. 1 do presente seção de avaliação da atividade cria um novo arquivo com o nome especificado na linha de comando. Ele usa a bandeira O_EXCL com abertas, por isso, se o arquivo já existe, um erro ocorre. O novo arquivo recebe permissões de leitura e gravação para o proprietário e grupo proprietário, e permissões de leitura apenas para os outros. (Se o seu umask é definida como um valor diferente de zero, o real Permissões podem ser mais restritivos.)

B. Descritores Fechamento de Arquivo

Quando você é feito com um descritor de arquivo, feche-o com perto. Em alguns casos, não é necessário chamar explicitamente fechar porque o Linux fecha todos os descritores de arquivos abertos quando um termina processo (isto é, quando o programa termina). Quando fecha um descritor de arquivo, você não deve usá-lo. Fechando um descritor de arquivo pode causar Linux para tomar uma acção específica, dependendo da natureza do descritor de ficheiro. Por exemplo, quando você fechar um descritor de arquivo para uma tomada de rede, Linux fecha a conexão de rede entre os dois computadores se comunicando através da tomada.

Linux limita o número de descritores de arquivos abertos que um processo pode ter aberto ao mesmo tempo. Um limite típico é 1.024 descritores de arquivos por processo. Você pode ajustar esse limite com a setrlimit chamada de sistema.

C. Escrita de Dados

Gravar dados em um descritor de arquivo usando a gravação de chamadas. Fornecer o descritor de arquivo, um ponteiro para um buffer de dados e o número de bytes para escrever. O descritor de arquivo deve ser aberto para escrevendo. Os dados gravados no arquivo não precisa ser uma cadeia de caracteres; escrever cópias arbitrárias bytes do buffer para o descritor de arquivo. O programa dado em questão No. 2 da presente seção de avaliação da atividade anexa o tempo atual para o arquivo especificado no comando linha. Se o arquivo não existir, ele será criado. Este programa também usa o tempo , localtime e asctime funções para obter e formatar a hora atual; verificar a sua respectiva homem

páginas para obter mais informações.

A gravação chamada retorna o número de bytes que foram realmente escritas ou -1 se acusar erro. Para certos tipos de descritores de arquivos, o número de bytes realmente escrito pode ser menos do que o número de bytes solicitados. Neste caso, é até você para chamar escrever de novo para escrever o resto dos dados. A função dada na pergunta número 3 desta atividade seção de avaliação demonstra como você pode fazer isso. Note-se que para algumas aplicações,

você pode ter que verificar a existência de condições especiais no meio da operação de escrita. Para exemplo, se você está escrevendo a uma tomada de rede, você terá que aumentar essa função para detectar se a ligação de rede foi fechada no meio da operação de gravação, e se tiver, para reagir de forma adequada.

D. Dados Reading

A chamada correspondente para leitura de dados é lido . Como escrever , é preciso um descritor de arquivo, um Ponteiro para uma reserva, e uma contagem. A contagem especifica quantos bytes são lidos a partir do o descritor de arquivo para o buffer. A chamada para read retorna -1 em caso de erro ou o número de bytes realmente ler. Isto pode ser menor do que o número de bytes solicitadas, por exemplo, se não há bytes suficientes no arquivo.

O programa dado em questão No. 4 desta seção de avaliação da atividade fornece uma demonstração de leitura. O programa imprime um despejo hexadecimal do conteúdo do

arquivo especificado na linha de comando. Cada linha exibe o deslocamento no arquivo e no próximo 16 bytes. É mostrado imprimindo um despejo de seu próprio arquivo executável.

E. Movimentação um arquivo

Um descritor de arquivo lembra a sua posição em um arquivo. Como você ler ou escrever para o arquivo descritor, a sua posição adiantamentos correspondentes ao número de bytes que você ler ou escrever.

Às vezes, porém, você vai precisar para se mover um arquivo sem ler ou gravar dados. Por exemplo, você pode querer escrever sobre o meio de um arquivo sem modificar o começo, ou você pode querer voltar para o início de um arquivo e re-lê-lo sem reabri-lo.

A chamada lseek permite reposicionar um descritor de arquivo em um arquivo. Passe-o arquivo descritor e dois argumentos adicionais especificando a nova posição.

- Se o terceiro argumento é SEEK_SET, lseek interpreta o segundo argumento como um posição, em bytes, desde o início do processo.
- Se o terceiro argumento é SEEK_CUR, lseek interpreta o segundo argumento como um compensado, o que pode ser positivo ou negativo, a partir da posição actual.
- Se o terceiro argumento é SEEK_END, lseek interpreta o segundo argumento como um deslocamento a partir do fim do ficheiro. Um valor positivo indica uma posição para além da final do arquivo.

A chamada para lseek retorna a nova posição, como um deslocamento a partir do início do arquivo.

O tipo de deslocamento é off_t . Se ocorrer um erro, lseek retorna -1.You não pode usar lseek com alguns tipos de descritores de arquivos, tais como descritores de arquivos soquete. Se você quiser encontrar o posição de um descritor de arquivo em um arquivo sem alterá-la, especifique um 0 deslocamento do

posição-corrente para exemplo:

```
off_t position = lseek (file_descriptor, 0, SEEK_CUR);
```

Linux permite que você use lseek para posicionar um descritor de arquivo para além do final do arquivo.

Normalmente, se um descritor de arquivo é posicionado no final de um arquivo e você escrever para o arquivo descritor, Linux expande automaticamente o arquivo para abrir espaço para os novos dados. Se vocês posicionar um descritor de arquivo para além do final de um arquivo e, em seguida, escrever para ele, Linux primeiros expande o arquivo para acomodar o "gap" que você criou com o lseek funcionamento e, em seguida, escreve para o final do mesmo. Esta diferença, no entanto, na verdade, não ocupam espaço no disco;em vez disso, o Linux faz uma nota de quanto tempo ele é. Se, posteriormente, tentar ler a partir do arquivo, ele aparece ao seu programa que a lacuna é preenchida com 0 bytes. Usando este comportamento de lseek, é possível criar arquivos extremamente grandes que ocupam quase

nenhum espaço em disco. o programa lseek-enorme dada em questão No. 5 desta seção de avaliação da atividade faz isso.

Ele toma como argumentos de linha de comando um nome de arquivo e um tamanho de arquivo de destino, em megabytes. O programa abre um novo arquivo, avança além do fim do arquivo usando lseek , e depois escreve uma 0 byte único antes de fechar o arquivo.

Note-se que estas lacunas mágicos em arquivos são uma característica especial do ext2 sistema de arquivos que é normalmente usado para discos GNU / Linux. Se você tentar usar lseek-grande para criar um arquivo em algum outro tipo de sistema de arquivos, tais como os de gordura ou vfat sistemas de arquivos usados para montar DOS e partições do Windows, você verá que o arquivo resultante faz realmente ocupar o pleno quantidade de espaço em disco. O Linux não permitem que você rebobinar antes do início de um arquivo com lseek.

informações do arquivo Extraindo

Usando aberto e ler , você pode extrair o conteúdo de um arquivo. Mas como sobre outro arquivo em formação? Por exemplo, invocando ls -l exibe, para os arquivos no diretório atual informações como o tamanho do arquivo, a última modificação, permissões e o proprietário.

O status de chamada é usado para obter esta informação sobre um arquivo.

Chamada de estatísticas com o caminho para o arquivo que você está interessado e um ponteiro para uma variável do tipo Stat struct . Se a chamada para Stat for bem sucedido, ele retorna 0 e preenche os campos da estrutura com informações sobre esse arquivo; caso contrário, retorna -1.

Estes são os campos mais úteis em estatísticas struct :

- st_mode contém permissões de acesso do arquivo.
- Além as permissões de acesso, o campo st_mode codifica o tipo de arquivo em bits de ordem superior. Veja o texto imediatamente a seguir esta lista com marcadores para instruções sobre como decodificar esta informação.
- st_uid e st_gid conter os IDs do usuário e grupo, respectivamente, para o qual o arquivo pertence.
- st_size contém o tamanho do arquivo, em bytes.
- st_atime contém o tempo em que este arquivo foi acessado pela última vez (leitura ou escrita).
- st_mtime contém o tempo em que este arquivo foi modificado pela última vez.

Estas macros verificar o valor do valor do campo st_mode para descobrir que tipo de arquivo você invocado status diante. A macro é avaliada como verdadeira se o arquivo é desse tipo.

```
S_ISBLK (modo) dispositivo de bloco

S_ISCHR (modo) de dispositivo de caractere

S_ISDIR (modo) Diretorio

S_ISFIFO (modo) FIFO (pipe nomeado)

S_ISLNK (modo) link simbólico

S_ISREG (modo) arquivo regular

S_ISSOCK (modo) Socket
```

O campo st_dev contém o número do dispositivo principal e secundário do dispositivo de hardware no qual este arquivo reside. O número maior de dispositivo é deslocado 8 bits de esquerda; o dispositivo menor número ocupa os significativos 8 bits menos. O campo st_ino contém o número inode deste arquivo. Esta localiza o arquivo no sistema de arquivos.

Se você chamar de estatísticas em um link simbólico, estatísticas segue o link e você pode obter o informações sobre o arquivo que o link aponta, não sobre a própria ligação simbólica.

Isto implica que S_ISLNK nunca será verdade para o resultado de estatística . Use o lstat função se você não quiser seguir links simbólicos; Esta função obtém informações sobre o associar-se ao invés de destino do link. Se você chamar lstat em um arquivo que não é um link simbólico, é equivalente a estatística . Chamando de estatísticas sobre um link quebrado (um link que aponta para um inexistente ou -alvo inacessível) resulta em um erro, ao chamar lstat em uma tal ligação não. Se você já tem um arquivo aberto para leitura ou escrita, chamar fstat vez de estatísticas . Isso leva um descritor de arquivo como seu primeiro argumento, em vez de um caminho. Um programa dado em questão No. 1 desta seção de avaliação da atividade apresenta uma função que aloca um buffer grande o suficiente para armazenar o conteúdo de um arquivo e, em seguida, lê o arquivo para o buffer. A função usa fstat

para determinar o tamanho do buffer de que necessita para alocar e também para verificar que o arquivoestá na verdade, um arquivo regular.

Leitura e escrita de Vector

O writer tomado como argumentos de um apontador para o início de um buffer de dados e o comprimento de que o buffer. Ele escreve uma região contígua de memória para o descritor de arquivo.

No entanto, um programa, muitas vezes, precisa escrever vários itens de dados, cada um residente em um diferentes partes da memória. Para usar gravação , o programa quer terá que copiar os itens em uma região de memória única, o que obviamente faz uso ineficiente dos ciclos de CPU e memória, ou terá que fazer várias chamadas para escrever.

Para algumas aplicações, várias chamadas para escrever são ineficientes ou indesejável. Por exemplo, ao escrever a uma tomada de rede, duas chamadas para escrever pode causar dois pacotes a serem enviados através da rede, enquanto que os mesmos dados podem ser enviados num único pacote, se uma única chamada para escrever eram possíveis.

O writev chamada permite que você escrever várias regiões contíguas de memória para um descritor de arquivo em uma única operação. Isso é chamado de gravação vector . O custo de utilização writev é que você deve criar uma estrutura de dados especificando o início e duração de cada região de memória. Esta estrutura de dados é uma matriz de iovec struct elementos. cada elemento especifica uma região da memória para escrever; os campos iov_base e iov_len especificar o endereço do início da região e do comprimento da região, respectivamente.

Se você sabe de antemão quantos regiões que você precisa, você pode simplesmente declarar uma struct iovec variável de matriz; se o número de regiões pode variar, você deve alocar a matriz dinamicamente. Chamada writev passar um descritor de arquivo para gravar, a matriz iovec struct , e o número de elementos na matriz. O valor de retorno é o número total de bytes gravados.

O programa fornecido na pergunta número 1 desta seção de avaliação da atividade escreve sua argumentos de linha de comando para um arquivo usando uma única chamada writev . O primeiro argumento é o nome do arquivo; o segundo e subseqüentes argumentos são gravadas no arquivo desse nome, um em cada linha. O programa aloca uma matriz de iovec struct elementos que é duas vezes, enquanto o número de argumentos que está escrevendo- para cada argumento que escreve o texto do argumento em si, bem como uma nova linha de caracteres. Porque não sabemos o número de argumentos de antecedência, a matriz é alocado usando malloc. Linux fornece uma função correspondente readv que lê em uma única operação em várias regiões contíguas de memória. Semelhante a writev , uma matriz de

iovec struct elementos especifica as regiões de memória na qual os dados serão lidos a partir de o descritor de arquivo.

Relação com funções padrão C Biblioteca de I / O

Mencionamos anteriormente que a biblioteca C padrão funções de I / O são executados em cima deles funções de baixo nível I / O. Às vezes, porém, é útil para usar funções da biblioteca padrão com o arquivo descritores, ou utilizar funções de baixo nível de I / O em um * stream biblioteca de arquivos padrão. GNU / Linux permite-lhe fazer as duas coisas.

Detalhes da atividade

Se você abriu um arquivo usando fopen , você pode obter o descritor de arquivo subjacente usando a função fileno. Isso leva a FILE* argumento do arquivo e retorna o descritor de arquivo. Por exemplo, para abrir um arquivo com a biblioteca padrão fopen chamada, mas escrever para ele com writev , você pode usar este código:

```
FILE * stream = fopen (filename, "w");

int file_descriptor = fileno (stream);

writev (file_descriptor, vetor, vector_length);
```

Note-se que fluxo e file_descriptor correspondem ao mesmo arquivo aberto. Se você chamar essa linha, você já não pode escrever para file_descriptor:

```
fclose (stream);
```

Da mesma forma, se você chamar essa linha, você não poderá mais gravar stream:

```
close (file_descriptor);
```

Para ir para o outro lado, de um descritor de arquivo para um fluxo, use o fdopen função. Isso constrói um FILE * ponteiro de fluxo que corresponde a um descritor de arquivo. O fdopen função recebe um arquivo argumento descritor e um argumento de cadeia que especifica o modo em que para criar o fluxo.

A sintaxe do argumento modo, é a mesma que a do segundo argumento para fopen , e ele devem ser compatíveis com o descritor de ficheiro. Por exemplo, especificar um modo de r para um arquivo de leitura descritor ou w para um descritor de arquivo de gravação. Tal como acontece com fileno , o fluxo e arquivo descritor referem-se a o mesmo arquivo aberto, por isso, se você fechar um, você não pode, posteriormente, usar o outro.

Conclusão

Essa atividade apresentou como realizar as seguintes operações de I / O usando operações de baixa nível (chamadas de sistema):

- Abra um arquivo e criar um descritor de arquivo
- Fechar descritores de arquivos
- dados gravar no arquivo descritores
- Lê dados de descritores de arquivos
- Movendo-se em torno de um arquivo em um descritor de arquivo

Além disso, a chamada do sistema estatística foi discutido para fornecer um meio para extrair outra informações associadas a um arquivo, juntamente com o writev chamada usado para escrever várias regiões contíguas de memória para um descritor de arquivo em um único operação. Notou-se que de uma maneira semelhante, a função correspondente readv pode ser usado para ler numa única operação em várias regiões descontínuas de memória.

Finalmente, a semelhança ea conversão do baixo nível I / O chamadas para o C

funções de E / S Biblioteca I e vice-versa foi demonstrada

Avaliação

1. Considere o programa “criar-file.c” fornecido abaixo

```
//create-file.c - Create a New File

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>
```

```
#include <unistd.h>

int main (int argc, char* argv[])

{

/* The path at which to create the new file. */

char* path = argv[1];

/* The permissions for the new file. */

mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;

/* Create the file. */

int fd = open (path, O_WRONLY | O_EXCL | O_CREAT, mode);

if (fd == -1) {

/* An error occurred. Print an error message and bail. */

perror (“open”);

return 1;

}

return 0;

}
```

Anote a saída observada em cada um dos seguintes

a. Copiar e executar o programa (em plataforma GNU / Linux) para criar

um arquivo chamado “demo_tsfile””

b. Use o Linux -l ls comando para exibir o arquivo “demofile”

atributos. O que é o comprimento de “demofile”? Por quê?

c. Repita a operação em (a) supra

2. Considere o programa “timestamp.c” fornecido abaixo

```
//timestamp.c - Append a Timestamp to a File

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/types.h>
```

```
#include <time.h>

#include <unistd.h>

/* Return a character string representing the current date and
time. */

char* get_timestamp ()

{

time_t now = time (NULL);

return asctime (localtime (&now));

}

int main (int argc, char* argv[])

{

/* The file to which to append the timestamp. */

char* filename = argv[1];

/* Get the current timestamp. */

char* timestamp = get_timestamp ();

/* Open the file for writing. If it exists, append to it;

otherwise, create a new file. */

int fd = open (filename, O_WRONLY | O_CREAT | O_APPEND, 0666);

/* Compute the length of the timestamp string. */

size_t length = strlen (timestamp);

/* Write the timestamp to the file. */

write (fd, timestamp, length);

/* All done. */

close (fd);

return 0;

}
```

Anote a saída observada em cada um dos seguintes

uma. Copiar e executar o programa (em plataforma GNU / Linux) para criar

um arquivo chamado “demo_tsfile”

b. Use o Linux gato comando para exibir o conteúdo do

"Demo_tsfile"

c. Repita as operações em (a) e (b) acima. Dê razões para a

resultados observados durante esta etapa.

3. Leia o programa "write-all" fornecido abaixo e dar uma detalhada

explicação sobre o que se passa no programa

```
//write-all.c

ssize_t write_all (int fd, const void* buffer, size_t count)

{

size_t left_to_write = count;

while (left_to_write > 0) {

size_t written = write (fd, buffer, count);

if (written == -1)

/* An error occurred; bail. */

return -1;

else

/* Keep count of how much more we need to write. */

left_to_write -= written;

}

/* We should have written no more than COUNT bytes! */

assert (left_to_write == 0);

/* The number of bytes written is exactly COUNT. */

return count;

}
```

4. Considere o programa "hexdump.c" fornecido abaixo

```
//hexdump.c - Print a Hexadecimal Dump of a File

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>
```

```
#include <unistd.h>

int main (int argc, char* argv[])

{

unsigned char buffer[16];

size_t offset = 0;

size_t bytes_read;

int i;

/* Open the file for reading. */

int fd = open (argv[1], O_RDONLY);

/* Read from the file, one chunk at a time. Continue until read

“comes up short”, that is, reads less than we asked for.

This indicates that we’ve hit the end of the file. */

do {

/* Read the next line’s worth of bytes. */

bytes_read = read (fd, buffer, sizeof (buffer));

/* Print the offset in the file, followed by the bytes themselves.

*/

printf (“0x%06x : “, offset);

for (i = 0; i < bytes_read; ++i)

printf (“%02x “, buffer[i]);

printf (“\n”);

/* Keep count of our position in the file. */

offset += bytes_read;

}

while (bytes_read == sizeof (buffer));

/* All done. */

close (fd);

return 0;

}
```

Executar o programa, passando o arquivo executável do programa como um parâmetro de linha de comando. Qual é a impressão observado fora?

5. Utilizar o programa "lseek-huge" fornecido abaixo fazer um arquivo de 1GB (1024MB)

```
//lseek-huge.c - Create Large Files with lseek
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
int zero = 0;
const int megabyte = 1024 * 1024;
char* filename = argv[1];
size_t length = (size_t) atoi (argv[2]) * megabyte;
/* Open a new file. */
int fd = open (filename, O_WRONLY | O_CREAT | O_EXCL, 0666);
/* Jump to 1 byte short of where we want the file to end. */
lseek (fd, length - 1, SEEK_SET);
/* Write a single 0 byte. */
write (fd, &zero, 1);
/* All done. */
close (fd);
return 0;
}
```

Observe o espaco livre na unidade antes e depois da operação.

uma. Use o Linux df -h. Comando e observar a saída

a. Copiar e executar o programa (em plataforma GNU / Linux) para criar um arquivo chamado "demo_bigfile" do tamanho de 1024MB

b. Use o Linux -l ls comando para listar os atributos do criado "Demo_bigfile"

c. Use o Linux df -h. Comando mais uma vez e comparar o disco

espaço consumido em relação à observada em (a) acima. Justifique

em suas observações.

d. Examinar o conteúdo do arquivo "demo_bigfile" usando o

hexdump.c programa fornecido em questão No. 4, acima

(% ./hexdump demo_bigfile | head-10). Por favor, note que você, provavelmente

Resumo da Unidade

Nesta unidade, forneceu-se uma visão geral da Biblioteca GNU C, juntamente com uma série de questões relativas a Biblioteca C do GNU, incluindo as Normas de biblioteca e Mobilidade, o conceitos básicos do uso da biblioteca, e no caso de funções de biblioteca C para a alocação de armazenamento para dados do Programa.

Além disso, descrevemos as funções da biblioteca GNU C para manipular arquivos e

diretórios. Especificamente, as funções para examinar ou modificar diretórios e trabalhando com vários nomes de arquivo foram apresentados.

Por último, a unidade discutido arquivo de entrada / saída sistema chama em GNU plataforma / LINUX. isto explorada chamadas para ler e gravar dados, chama para extrair informações do arquivo, exige escrevendo várias regiões contíguas de memória para um descritor de arquivo em um único operação de leitura ou numa única operação em várias regiões não adjacentes de memória, e destacou as operações de entrada e saida, as funções do kernel e a relação das funções padrão das bibliotecas C padrão.

Critétios de Avaliação

Será guiado conforme os criterios da cada instituição.

Avaliação

Verifique a sua compreensão!

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

1. Mark Mitchell, Jeffrey Oldham, e Alex Samuel; Programação Avançada Linux;Copyright © 2001 pela New Riders Publishing; Primeira edição: junho de 2001
2. http://www.acm.uiuc.edu/webmonkeys/book/c_guide/ : O Guia de Referência C Library
3. http://www.delorie.com/gnu/docs/glibc/libc_toc.html : O GNU C Library

Unidade 2. Shell que programa e que encaixa o conjunto em C

Introdução à Unidade

O Kernel é o coração de um sistema operacional (OS). Controla o recurso, que são as facilidades disponíveis no ósmio, tal como a facilidade para armazenar dados, dados da cópia na impressora, memória, gerência de arquivos, etc. A semente decide-se quem usará este recurso, para quanto tempo e quando. Funciona seus programas (ou jogo execute até arquivoss binárias). A semente age como um intermediario entre o Hardware do computador e os vários programas/aplicação/shell como mostrado em figura 1.

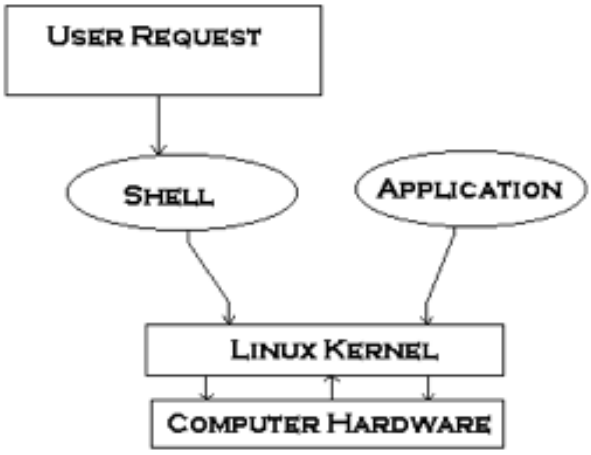


Figura 1: Kernel e o Hardware entre os vários usuários

A semente é uma parcela residente da memória do ósmio. Executa a seguinte tarefa:-

- Gerência do I/O
- Gerência Process
- Gerência de dispositivo
- Gerência de arquivos
- Gerência da memória

O Hardware de computador compreende a língua de 0 e de 1 chamados língua binária. Em dias adiantados de computar, as instruções ao computador foram fornecidas usando a língua binária, que é difícil de ler e escrever.

Sistemas Operativos fornecem um programa especial chamado Shell. Shell aceita a instrução ou os comandos em inglês (na maior parte) e se seu um comando válido, ele for passagem à semente. Shell é um programa de usuário ou o ambiente do ósmio fornecido para a interação do usuário. Shell é um intérprete da língua de comando que execute os comandos lidos do

dispositivo de entrada padrão (teclado) ou de um arquivo. Shell não é parte da semente do sistema, mas os usos a semente do sistema executar programas, críam os arquivos etc.

As línguas de programação fornecem também uma maneira de instruir a ferragem para executar alguma tarefa do usuário com o desenvolvimento da aplicação. Os programas escritos em línguas Higher-level tais como o funcionamento de C e de C++ em quase todas as arquiteturas rendem uma produtividade mais elevada ao escrever e ao manter o código. Para ocasiões quando os programadores necessitam usar instruções de conjunto em seus programas, a coleção do compilador do GNU permite programadores adicionar instruções arquitetura-dependentes da língua de conjunto a seus programas.

As instruções da língua de conjunto são arquitetura-dependentes, assim, por exemplo, os programas que usam as instruções x86 não podem ser compilados em computadores de PowerPC. Para usá-los, você requererá uma facilidade na língua de conjunto para sua arquitetura. Entretanto, as indicações inline do conjunto permitem-no alcançar diretamente a ferragem e podem-nas também render um código mais rápido. [[An]] ASM a instrução permite que você introduza instruções de conjunto em programas de C e de C++. Observe isso ao contrário das instruções ordinárias do código do conjunto, ASM as indicações permitem-no especificar operandos da entrada e da saída usando a sintaxe de C

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

- Definir os conceitos inlining do shell e do conjunto
- Empregar o shell para executar a tarefa diferente de user/OS
- Explicar o processo de escrever programas do shell e construções de programação do shell
- Explicar as técnicas de misturar instruções de conjunto com o código de C
- Empregar a programação do shell/shell que scripting para automatizar tarefas diferentes do sistema.
- Escrever os programas de C simples encaixados com instruções de conjunto para apressar-se acima de processar das instruções

Shell	Um intérprete da língua de comando que execute os comandos lidos do dispositivo de entrada padrão (teclado) ou de uma arquivos.
Programa/certificado de Shell	O shell script comandos do usuário , e que são inseridos diretamente pelo usuário , ou que podem ser lidos a partir de um Arquivo chamado o shell script ou programa shell. shell scripts são interpretados , não compilado . O shell lê comandos a partir da linha de script por linha e pesquisas para esses comandos no sistema, enquanto um compilador converte um programa em forma legível por máquina, um Arquivo executável - o que pode então ser usado em um script shell.
Assembly language	A linguagem assembly é uma linguagem de programação de baixo nível para um computador ou outro dispositivo programável específico para uma arquitetura de computador específico em contraste com a maioria das linguagens de programação de alto nível , que são geralmente múltiplos sistemas em todo portáteis.
High-level language	A linguagem de programação como C / C ++ que suporta o desenvolvimento do sistema em um alto nível de abstração , liberando o desenvolvedor de manter em seus lotes cabeça de detalhes que são irrelevantes para o problema na mão .
Kernel (Nucleo)	O núcleo é o módulo central de um sistema operativo . É a parte do sistema operacional que carrega em primeiro lugar, e permanece na memória principal
Operating System	Software que gerencia os recursos de hardware e software de computador e fornece serviços comuns de programas de computador

Actividades de Aprendizagem

Atividade 1.1 - Princípios de Shell

Introdução

Diversos shells estão disponíveis no ósmio de Linux/Unix including o BASH, o CSH, o KSH e o TCSH como detalhado na tabela 1. Para encontrar todos os shells disponíveis em seu sistema datilografe seguinte comando:

gato /etc/shells de $. Nota que cada shell faz o mesmo trabalho, mas cada um compreende a sintaxe de comando diferente e fornece funções internas diferentes.

Tabela 1: Alguns shells da terra comum ambiente no ósmio de Linux/Unix

Nome de Shell	Tornado perto	Onde	Observação
BASH (Bourne-Outra vez Shell)	Raposa de Brian e Chet Ramey	Fundação livre do software	A maioria de shell comum em Linux. É shell do Freeware.
CSH (C Shell)	Alegria da conta	Universidade de Califórnia (para o DEB)	A sintaxe e o uso do shell de C são muito similares a a língua de programação de C.
KSH (Korn Shell)	David Korn	EM & Laboratórios de T Bell	--
TCSH	Veja a página do homem.		
Tipo $ do tcsh do homem	--	TCSH é uma versão realçada mas completamente compatível do shell do UNIX C de Berkeley (CSH).	TCSH é uma versão realçada mas completamente compatível do shell do UNIX C de Berkeley (CSH).

O MS-DOS fornece um shell nomeado COMMAND.COM que é usado também para a mesma finalidade, mas não é tão poderoso quanto os shells de Linux/Unix!

Todo o shell lê o comando do usuário (através do teclado ou do rato) e diz a ósmio o que os usuários querem. Se nós estivermos dando comandos de keyboard é chamado linha relação do comando (geralmente em-dianteira do alerta de $ para o ósmio de Linux/Unix. O símbolo alerta, entretanto, depende em cima de seus shell e ambiente que é ajustado pelo administrador de sistema, conseqüentemente você pode começar o alerta diferente).

Para encontrar a corrente shell no tipo de Linux/Unix depois do comando: eco $SHELL de $ Normalmente os shells são interativos. Isso significa que um shell aceita o comando do usuário (através do teclado) e o executa.

Mas se você necessitar fornecer uma seqüência de dois ou mais comandos, você pode armazenar esta seqüência dos comandos a um arquivo de texto e dizer o shell para executar a arquivos de texto em vez de incorporar um comando de cada vez. Isto é sabido como programa/certificado do shell. Um programa/certificado de Shell é a série do comando escrito na arquivos de texto lisa. Os Scripts de Shell são justos como arquivoss de grupo no MS-DOS mas têm mais poder do que a arquivos de grupo do MS-DOS.

Um certificado de Shell pode fazer exame da entrada do usuário, arquivos e output os na tela. É útil criar para possuir comandos e excepto o tempo, automatiza alguma tarefa da vida do dia hoje, também administração que do sistema a parte pode também ser automatizada.

Nesta unidade nós usaremos bash shell.

Detalhes da atividade

o que faz o shell

Em Linux/Unix, o shell é separado do ósmio (olhar e sensação da mudança). O shell lê e executa comandos

- Alguns segurados pelo shell próprio (pwd, eco,…)
- alguns são programas armazenados em algum diretório (olhar nos diretórios no PATH). Comece um subshell executar estes

O shell fornece a sustentação para a interação melhor com o computer/OS (história do comando, edição, configuração). Do shell as sustentações também que scripting (é uma língua de programação)

A. Executando um comando

Após ter lido um comando, o shell pode fazer algum que processa (veja wildcards, etc. na descrição da sintaxe que segue), então ele deve encontrar um programa para executar o comando. Alguns comandos são executados diretamente pelo shell. Outros comandos são executados por programas separados. Estes são encontrados olhando em uma lista dos diretórios para programas com o nome apropriado. O shell procurara diretórios na variável do PATH. Uma mistura - a tabela é usada fazer a busca rápida. Você pode adicionar comandos novos simplesmente adicionando programas novos (um programa pode ser toda a arquivos executável including Scripts - consulte às permissões de Linux/Unix) aos diretórios no PATH. Você pode modificar/adiciona diretórios no PATH.

B. Pesquisando a cerca dos Commands

Type – diz-lhe se um comando for um built-in, um pseudônimo, ou um programa (externo ao shell)

which – diz em que diretório uma utilidade é encontrada

help - informação de exposições sobre comandos internos (é um builtin próprio)

info bash - um lugar bom a ler sobre o shell do BASH

Por examplo:

```
$ which echo

/bin/echo

$ type -a echo

echo is a shell builtin

echo is /bin/echo

$ help echo # help provides information on builtin commands

echo: echo [-neE] [arg ...]
```

Output the ARGs. If -n is specified, the trailing newline is suppressed. If the -e option is given, interpretation of the following backslash-escaped characters is turned on:

```
\a alert (bell)

\b backspace

\c suppress trailing newline

\E escape character

\f form feed

\n new line

\r carriage return

\t horizontal tab

\v vertical tab

\\ backslash

\num the character whose ASCII code is NUM (octal).
```

Você pode explicitamente desligar a interpretação dos caráteres acima com - a opção de E.

C. Arquivos de iniciação

Os comandos e as variáveis colocados em arquivos da iniciação são lidos quando o shell é começado. Se as variáveis forem colocadas dentro system-wide o init arquiva, ele é feito disponível a cada shell. Os Customizations devem ser exportados para estar disponíveis. Os comandos e os pseudônimos não podem ser exportados assim que devem ser colocados em arquivos usuário-específicos do init.

i. /etc/proarquivo - System-Wide arquivos da iniciação

Lido primeiramente pelo shell. As tarefas que realiza inclui:

- Jogos e variáveis das exportações: PATH, TERM, PS1, etc.
- Indica índices de /etc/motd (os msg do dia)
- Permissões da criação da arquivos do defeito dos jogos (umask)

ii. iniciação Usuário-específica

Se o shell for um shell do início de uma sessão, procura uma das seguintes arquivos (em ordem)

- ~/.bash_proarquivo
- ~/.bash_login
- ~/.proarquivo

Se for um shell interativo do non-início de uma sessão, lê a arquivos

- ~/.bashrc

iii. Outros customizations

- PS1 - alerta
- PATH - adicione ao PATH de busca
- Ajuste opções do shell
- noclobber
- ignoreeof
- comando-linha que edita a opção (vi ou emacs)

Veja .bashrc arquivos para exemplos.

Variáveis de 1.1.2.2 Shell

O shell mantem-se a par de um jogo de nomes e de valores do parâmetro. Alguns destes parâmetros determinam o comportamento do shell. Nós podemos alcançar estas variáveis a

- valores novos ajustados para que alguns customize o shell.
- encontre para fora o valor de algum para ajudar realizar uma tarefa

Exemplos de variáveis do shell dentro sh / ksh / bash:

Variável de Shell	Finalidade
PWD	diretório de funcionamento atual
PATH	lista dos lugares para procurar comandos
HOME	diretório home do usuário
MAIL	onde seu email é armazenado
TERM	que tipo do terminal você tem
HISTARQUIVO	onde sua história do comando é conservada

1. Indicando variáveis de Shell

Prefixe o nome de uma variável do shell com o "$" para dereference. [[The]] eco o comando fará:

```
eco $HOME

eco $PATH

Você pode usar estas variáveis em toda a linha de comando:

ls - al $HOME
```

2. Ajustando variáveis de Shell

Você pode mudar o valor de uma variável do shell com um comando da atribuição (este é um comando do builtin do shell):

```
HOME=/etc

PATH=/usr/bin: /usr/etc: /sbin

NEWVAR= do " avu do avu avu "

Anote a falta dos espaços em torno do "="
```

a. exportação

Usado a exportação uma variável e faz disponível aos subshells. O valor passado dentro e as mudanças feitas a um valor no subshell não persistem no chamador. Subshell é criado sempre

que um certificado ou um programa são funcionados e herdam variáveis exportadas do shell do pai.

b. PATH

Cada vez que você dá ao shell uma linha de comando faz o seguinte:

- Verifica para ver se o comando for um built-in do shell.
- Se não - tentativas para encontrar um programa cujo nome (o nome de arquivo) seja o mesmo que o comando.

A variável do PATH diz ao shell onde procurar programas (comandos non internos).

Por exemplo:

```
eco $PATH de $

~/bin: /usr/local/sbin: /usr/local/bin: /usr/sbin: /usr/bin: /sbin:
/bin: /usr/games:.

$ PATH=$ {PATH}: /home/avu

eco $PATH de $

~/bin: /usr/local/sbin: /usr/local/bin: /usr/sbin: /usr/bin: /sbin:
/bin: /usr/games:.:/home/avu
```

O PATH é uma lista de ": " diretórios limitados. O PATH é uma lista e uma ordem da busca. Você pode adicionar o material a seu PATH mudando a arquivos da partida do shell.

Notas sobre o PATH:

Se você não tiver "." em seu PATH, comandos em seu diretório atual não será encontrado. Você pode ou não pode querer adicionar "." a seu PATH. Se você não e para querer executar um comando no diretório atual

./command

O PATH é procurarado sequencialmente; o primeiro programa combinando é executado. Beware de nomear seus executables teste porque há um outro programa chamado teste e isto pode ser executado quando você entra

teste

c. ajuste o comando

O comando do jogo (builtin do escudo) com nenhuns parâmetros imprimirá para fora uma lista de todas as variáveis do escudo.

Algumas opções comuns:

- noclobber - sustentos milivolt, PC, redirection de suprimir uma pipe existente
- - o vi (ou - o emacs) - comando-linha dos jogos que edita a modalidade
- ignoreeof - o ctrl-D não retirará o escudo

d. $PS1 e $PS2

A variável do escudo PS1 é sua linha de comando alerta. A variável do escudo PS2 está usada como um alerta quando o escudo necessita mais entrada (no meio de processar um comando). Mudando PS1 e/ou PS2 você pode mudar o alerta. O Bash suporta algum material extravagante na string alerta:

```
\ t é substituem pelo tempo atual

\ w é substituído pelo diretório atual

\ h é substituído pelo hostname

\ u é substituído pelo username

\n é substituído por um newline
```

Alerta do bash do exemplo:

eco $PS1 do ~

======= [- do =======] \ h \ t \ n \ w

Você pode mudar seu alerta mudando PS1:

Mestre de PS1= " sim? "

Para fazer mudanças fure, isto é, se você quiser dizer o escudo (bash) para usar sempre sim o mestre do alerta "? ", você necessita armazenar a mudança em uma pipe da partida do escudo. Para o bash - mude a pipe ~/.bashrc.

e. SHELL

A variável de SHELL prende o PATH do escudo do início de uma sessão.

f. HOME

A variável HOME contem o PATH a seu diretório home. Quando você se usar Cd comande com nenhuns argumentos, os usos do comando o valor da variável HOME como o argumento.

eco $HOME

/home/avu

3. Metacharacters

Shell metacharacters são os caráteres que são segurados especialmente pelo escudo. Abaixo está uma lista (incompleta) de metacharacters do escudo:

- > >> < << | Redirection do comando, tubulação
- * [] ? {} Especificação da pipe
- ; & || arranjar em seqüência do comando do && ()
- “, ‘agrupando o texto
- As citações dobro e as únicas citações indicam que o texto incluido deve ser tratado como uma unidade, não como um grupo das palavras.
- # comentando
- O descanso da linha após estes caráteres é ignorado
- \ (backslash) - o caráter do escape
- Indica que o caráter imediatamente depois do backslash deve ser tratado literalmente. Remover uma pipe nomeada o “#bogus” lhe pode usar o backslash rm \ #bogus

4. Alias

Os comandos podem ser aliased (rebatizado) para alterar seu comportamento. Um uso comum dos pseudônimos é adicionar opções ao comportamento do defeito de determinados comandos (por exemplo. ls, rm, milivolt, …). É prática comum a aliás rm para alertar o usuário certificar-se de que as pipes especificadas devem ser removidas. Isto é especialmente importante desde que você possa remover todas as pipes com (veja wildcards) rm *. Para ver pseudônimos atuais, use aliás comando. Isto pode também ser usado ajustar pseudônimos novos.

Críe um pseudônimo usando o comando do builtin aliás =value[conhecido]

- Você deve geralmente incluir o valor nas citações.
- nenhum espaço em torno do “=”
- Sem um valor, aliás cópias - para fora o pseudônimo associou com o nome

Por exemplo:

```
dir= “ls” de $alias

ls= “ls de $alias - CF”

$dir

      Output mesmos que para “ls - CF”
```

5. Substituição variável

Os escudos suportam o uso das variáveis. Uma variável é um nome que seja limitado a um valor. Para ajustar uma variável (no Bash) incorpore apenas o name=value (nenhuns espaços em torno "="). Para ver apenas os ajustes de todas as variáveis entrar "jogo". A matança uma variável: unset nome.

Ao processar um comando, a substituição variável ocorre. Uma variável em um comando é embandeirada por um prefixo "$" do sinal do dólar.

o eco de $ meu escudo é $SHELL

Meu escudo é /bin/bash

eco escreve para fora seus argumentos depois que a substituição é executada.

A substituição variável ocorrerá dentro das citações dobro

o eco de $ "meu escudo é $SHELL"

Meu escudo é /usr/local/bin/tcsh

A substituição não ocorre dentro das únicas citações.

o eco de $ "meu escudo é $SHELL"

Meu escudo é $SHELL

Quando o uso de um nome variável não está desobstruído, inclua-o dentro das cintas $ {nome}

```
$ prefix=cs333

$ suffix=.pdf

eco $prefix03$suffix de $

.pdf

eco $ de $ {prefixo} 03$ {sufixo}

Cs33303.pdf
```

Isto ocorre ao construir nomes da pipe no certificado arquiva.

Muitas variáveis do escudo do uso dos programas (chamadas também ambiental variáveis) para começar a configuração

informação.

Por exemplo:

- A IMPRESSORA é usada por comandos imprimindo determinar a impressora do defeito.
- O TERMO é usado pelos programas (por exemplo, vi, pinho) determinar que tipo de terminal está sendo usado.

- O VISUAL é usado por programas da notícia da rede, etc., determinar que editor a se usar.

7. Substituição do comando

A substituição do comando permite saída (stdout) de um comando substituir o nome do comando. Há dois formulários:

O Bourne original: `do comando do `

A extensão do Bash (e o Korn): $ (comando)

A saída do comando é substituída em:

```
eco $ de $ (ls)
```

pipe da taxa do foo?

o eco de $ "é hoje $ (data "+%A %d %B %Y")"

Hoje é quinta-feira 30 setembro 2014

8. Citar forte - únicas citações

Inibe toda a substituição, e o meaning especial dos metacharacters:

```
o eco "$USER de $ é $USER"

$USER é $USER

o eco de $ 'é hoje da "`data"

hoje é `da data do `

eco de $ "nenhum background&"

Nenhum background&

o eco "I de $ disse, "rádio! ""

Eu disse, "rádio!"
```

9. Citar fraco - citações dobro

Permite o comando e a substituição variável. Inibe o meaning especial de todos metacharacters restantes:

```
o eco de $ "meu nome é $USER &"

Meu nome é avu &

"do eco de $ \ $2.00 diz o `da data do `"

$2.00 dizem o sol janeiro 15 01:43: 32 EST 200
```

10. Execução do comando

Às vezes nós necessitamos combinar diversos comandos. Há quatro formatos para comandos combinando em uma linha: arranjado em seqüência, agrupado, acorrentado, e condicional.

Uma seqüência dos comandos pode ser incorporada em uma linha. Cada comando deve ser separado de seu predecessor pelo semicolon. Não há nenhum relacionamento direto entre os comandos.

command1; command2; command3

Usando comandos agrupados nós podemos aplicar a mesma operação ao grupo. Os comandos são agrupados colocando os em parênteses e funcionados em um subshell.

Por exemplo:

```
eco "mês"  > pipe; cal 10 2000  >> pipe

(eco "mês"; cal 10 2000)  > pipe
```

Com os comandos condicionais nós podemos combinar dois ou mais comandos usando relacionamentos condicionais E (o &&) e OU (ll). Se nós E dois comandos, o segundo formos executados

somente se o primeiro é bem sucedido. Se nós OU dois comandos, o segundo formos executados somente do primeiro falharmos.

Por exemplo:

```
eco "copy do && do PC arquivo1 arquivo2 bem sucedido"

PC arquivo1 arquivo2 || o eco "copy falha"
```

Sintaxe de 1.1.2.3 Shell

1. Comentários -

```
# este é um comentário

ls # lista as pipes no diretório atual
```

2. Linha continuação - \

```
$echo A long \

> linha
```

3 Separador de comando -;

Você pode alistar mais de um comando por a linha separada perto;

ls; quem

4. Do separador do Pathname/

Cd /home/avu

5. Wildcards (globbing), e expansão do pathname

* - combine toda a string (vazios including)

? - combine todo o único caráter

[jogo] - caráteres do fósforo alistados no jogo (pode ser a escala)

[! jogo] - combine todo o caráter não dado no jogo

Por exemplo:

```
ls *.c

ls *.?

ls *. [Hh][ Tt][ Ll]

a-z [do ls]
```

6. Redirection e tubulações da pipe

< - dirija de novo a entrada da fonte especificada

> - dirija de novo a saída à fonte especificada

>> - dirija de novo a saída e adicione-a à fonte especificada

| - conduza a saída de um comando à entrada ao seguinte

Por exemplo:

```
palavra do grep  < /usr/dict/words

ls  > lista

ls  >> lista

ls - l| wc - l
```

7. stderr

Anote esse redirection da pipe da saída padrão [stdout] não inclui as mensagens de erro, que vão ao erro padrão [stderr] (quando você estiver em uma sessão terminal, ambos stdout e stderr vá à tela; entretanto, quando você dirigir de novo stdout a uma pipe, stderr vai ainda à tela). stdout é designado pelo descriptor de pipe 1 e stderr por 2 (a entrada padrão é 0). Para dirigir de novo o uso do erro padrão 2>

arquivonothere do ls > alistando 2> erro

arquivonothere 2 &1> do ls > lista # ambos stdout e stderr dirigido de novo à lista

8. Trabalhos de background

[[The]] & o operador funciona o comando no fundo.

grep "we.*"< /usr/word/dict > wewords &

Isto funciona grep comando no fundo - você começa imediatamente um alerta novo e pode continuar seu trabalho quando o comando for funcionado.

trabalhos é um comando do builtin. Alista os trabalhos ativos (parados, ou funcionando no fundo). Veja também o comando picosegundo. matança fará exame de um PID (veja picosegundo) ou jobspec (veja trabalhos)

Conclusão

Nesta atividade nós apresentamos os princípios do escudo do Unix/Linux para familiarizá-lo com o ambiente do escudo.

Avaliação

Pratique todos os exemplos usados nesta atividade.

Atividade 1.2 - Programação de Shell

Introdução

1. Que é um script?

Um certificado é um programa pequeno que seja executado pelo escudo. O certificado é uma pipe de texto que contenha:

- Shell comanda-o normalmente usar-se.
- Construções do controle de fluxo de Shell (por exemplo, se-então-outro, etc.)
- Um uso mais pesado das variáveis do que você normalmente usar-se-ia da linha de comando.

2. Por que escreva Scripts?

Toda a tarefa você (à mão) mais de duas vezes deve provavelmente ser envolvido em um certificado. As seqüências das operações que você executa frequentemente podem ser colocadas em um arquivo do certificado e então ser executadas como um único comando. Por exemplo, rebatizando todas os arquivos do formulário

Cs333l*.ppt a cs333l*n.ppt requer a milivolt comando para cada arquivo.

Os escudos do Unix/Linux são muito poderosos, mas há algumas coisas que não fazem bem.

3. Shell scripting - por que não?

Tarefas Recurso-intensive, especialmente onde a velocidade é um fator, aplicações complexas, onde a programação estruturada seja uma necessidade, as aplicações missão-críticas em cima de que você estão apostando o rancho, ou o futuro da companhia, situações onde a segurança

é importante, onde você necessita proteger de encontro a cortar. O projeto consiste em subcomponents com dependências bloqueando. Operações extensivas da arquivo requeridas (o Bash é limitado ao acesso de série da arquivo, e àquele somente em uma forma line-by-line particularmente desajeitada e ineficiente). Necessite gerar ou manipular gráficos ou GUIs. Acesso direto da necessidade à ferragem do sistema. Porto da necessidade ou I/o. do soquete. Necessite usar bibliotecas ou relação com código legado.

Detalhes da atividade

Começar de 1.2.2.1 começou com Scripting

1. Criando um certificado

Vamos criar um certificado de escudo para dar-nos a informação sobre o sistema. Nós criamos o certificado usando um editor de texto. Vamos chamar a arquivo "status".

```
#! /bin/bash

uptime

usuários
```

Seja certo introduzir um "entrada" após a última linha antes de você saída o editor.

2. Funcionando um certificado

Para executar o escudo que nós podemos fazer

$ do status do bash

10:37 acima de 23 dias, 23:54, 14 usuários,

média da carga…

billc do afjhj…

Nós podemos também executar a arquivo como um comando se o apropriados executarem o acesso forem concedidos.

```
$ ./status

bash: ./status: Permissão negada

$ do status do chmod +x

$ ./status # trabalha corretamente.
```

3. #! - "sha-estrondo"

Nao needed se o bash retroceder fora do certificado, mas…

Os escudos procuram "#!" no começo muito da arquivo. É seguido pelo programa

(intérprete) que lerá esta arquivo:

```
#! /bin/bash

#! /usr/bin/python

# este é um programa do Python. O Bash não o lê. Dá-o direito

# ao intérprete do Python.
```

Expressões condicionais de 1.2.2.2

Para executar ifs e passa o tempo nós necessitamos construir expressões condicionais.
Uma expressão condicional é uma que avalía para rectificar ou falso dependendo de seus operandos. ' Um valor do retorno process de 0 é feito exame para ser verdadeiro; todo o valor nonzero é falso.

1. teste - Expressões condicionais

Realmente teste é uma utilidade de disco. [ ] é seu shorthand. Fornece para um grande muitos testes e está disponível a todos os escudos.

teste expressão

ou

[expressão] - expressão separada dos espaços dos suportes

teste retorna um status de saída de zero (sucesso) se a expressão avaliar para rectificar. teste usa uma variedade dos operadores. Os operadores Unary da arquivo podem testar várias propriedades de arquivo. São aqui justos alguns:

- e Verdadeiro se a arquivo existir

- f Verdadeiro se a arquivo for uma arquivo regular

- d Verdadeiro se a arquivo for um diretório

- w Verdadeiro se a arquivo existir e for writable

- O Verdadeiro se I possuir a arquivo

Por exemplo:

```
se [ - e ~avu/public_html ] ; então

                o eco "Avu tem um diretório público da correia
fotorreceptora"

fi
```

2. [] - operadores da arquivo e da string

- Operadores "arquivo1 da arquivo binária op arquivo2"

- NT Verdadeiro é arquivo1 é mais novo do que arquivo2

- ot Verdadeiro é arquivo1 é mais velho do que arquivo2

- ef Verdadeiro se arquivo1 e arquivo2 consultarem ao mesmo inode

- Operadores Unary “string op” da string

- z Verdadeiro se a string for do comprimento zero

- n Verdadeiro se a string não for do comprimento zero

- l Retorna o comprimento da string

Por exemplo:

```
se [ - z “$myVar” ] ; então
“do eco \ $myVar tem o comprimento nulo”
fi
```

3. [] - operadores da string

Estes comparam a ordem lexical

==! = < > <= >=

Nota, < > é o redirection da arquivo. Escape d

Por exemplo:

```
se [ “ABC”! = “ABC” ] ; então
o eco ‘vê. Matérias do caso. ‘; fi
se [ 12 \< 2 ] ; então
o eco “12 é menos de 2?” ; fi
```

4 [] - operadores aritméticos

Trabalho somente para inteiros.

Operadores binários:

- tenente - gt - le - ge - eq - ne

Por exemplo:

se [2 - le 3] ; então; o eco “esfria!” ; fi

x=5

se [ne 12 de “$x” -] ; então

o eco “esfria ainda”; fi

5. [] - Operadores lógicos

Ferramentas lógicas da expressão

- ! expressão - lógica não (isto é, sentido das mudanças da expressão)
- e1 - um e2 rectifica se ambas as expressões forem verdadeiras.

- e1 - o e2 rectifica se e1 ou e2 forem verdadeiro.
- \ (expressão \) trabalha como parênteses normais para expressões; use espaços em torno da expressão.

Por exemplo:

```
teste - escaninho de e - a - d /bin é verdadeiro

[  - e ~/.bashrc - a! - eco do && ] de d ~/.bashrc

Verdadeiro
```

6. [[teste]]

O Bash adicionou [[]] para mais C-como o uso:

se [[- && de e ~/.bashrc! - d ~/.bashrc]]

então

o eco "vamos analisar gramaticalmente esse filhote de cachorro"

fi

se [[- z "$myArquivo" || ! - r $myArquivo]]

…

É um built-in. Cite por que às vezes $myArquivo, às vezes não (ele é geralmente uma idéia boa fazer assim)?

Expressões aritméticas de 1.2.2.3

O Bash trata geralmente variáveis como strings. Você pode mudar aquele usando a sintaxe aritmética da expansão: ((arithmeticExpr)). A notação (()) é um shorthand para deixado indicação do builtin.

```
$ x=1

$ x=x+1 # "x+1" é justo uma string

eco $x

x+1
```

A nota, $[] é deprecada

```
$ x=1

$ x=$x+1 # ainda apenas uma string

eco $x de $

1+1
```

Mais perto, mas ainda nao direito.

```
$ x=1

$ ((x=x+1))

eco $x de $
```

Finalmente!

Condição IF

1. Amostra: Condicional básico if. then

```
#! /bin/bash

if [ "$1" = "foo" ] ; then

expressão do eco \

avaliado como verdadeiro

fi
```

2. Amostra: Condicional básico if. then... else

```
#! /bin/bash

if [ "$1" = "foo"  ]

then

ecoe o "primeiro argumento é "foo""

else

ecoe o "primeiro arg não é "foo""

fi
```

3. Amostra: Condicionais com variáveis

```
#! /bin/bash

T1= " foo "

T2= " barra "

if [ "$T1" == "$T2" ] ; then

expressão do eco avaliada como verdadeira

else

expressão do eco avaliada como falsa

fi
```

Sempre variáveis das citações nos scripts!

4. Verificando o returno do valor do comando

```
if diff "$arquivoA" "$arquivoB" > /dev/null
then
echo "Arquivos are identical"
else
echo "Arquivos are different"
fi
```

Condição Case

```
case $opt in
        a ) echo "Opção a";;
        b ) echo " Opção b";;
        c ) echo " Opção c";;
\? ) echo \
'usage: alice [-a] [-b] [-c] args...'
exit 1;;
esac
```

Variáveis especiais de

```
$# o número dos argumentos
$* todos os argumentos
$@ todos os argumentos (citados individualmente)
$? valor do retorno do último comando executado
$$ identificação process do escudo
$HOME, $IFS, $PATH, $PS1, $PS2
```

Certificados e argumentos de

Os certificados podem ser começados com parâmetros, apenas como comandos

aScript arg1 arg2…

Os certificados podem alcançar estes argumentos com as variáveis do escudo:

```
"$n" É o valor do nth parâmetro.  O comando é o parâmetro zero
"$#" É o número dos parâmetros incorporados.
```

```
"$*" expande como uma lista de todos os parâmetros incorporados
exceto o comando.

Let rapidamente escrever um certificado para ver isto:

(esta primeira linha é uma maneira rápida e suja escrever uma lima)

gato de $  > xx # gato lêem do stdin se nenhuma lima especificar

eco $0

eco $#

eco $1 $2

eco $*
```

O Cd # o Controle-d são a extremidade do caráter de lima.

a lima xx do #The do chmod +x xx de $ é agora um certificado de escudo executável.

$ ./xx um b c certificado do #Execute com três parâmetros

./xx

3

um b

um b c

$ xx certificado do #Execute com nenhum parâmetro

./xx

0

Os parâmetros Unspecified expandem como strings vazias (isto é., como nada)

Laços nos scripts

O laço for é um pouco diferente de outras línguas de programação. Bàsicamente, let você iterar sobre uma série de "palavras" dentro de uma string. O while executa uma parte de código se a expressão do controle for verdadeira, e para-a somente quando é falso (ou uma ruptura explícita está encontrado dentro do código executado. O laço until for quase equivalente ao laço do quando, exceto que o código está executado quando a expressão do controle avaliar a falso.

1. For Loop

i. For loop : Examplo 1

```
$ for x in 1 2 a; do

> echo $x
```

```
> done
1
2
a
```

ii. For loop : Example 2

```
$ for x in *; do
> echo $x
> done
bin
mail
public_html
...
```

iii. For loop : Example 3

```
#!/bin/bash
for i in $(cat list.txt) ; do
echo item: $i
done
```

iv. For loop : Example 4

```
#!/bin/bash
for (( i=0; i<10; ++i )) ; do
echo item: $i
done
```

2. while loop

i. while loop: Example 1

```
COUNTER=0
while [ $COUNTER -lt 10 ] ; do
echo The counter is $COUNTER
let COUNTER=COUNTER+1
done
```

ii. while loop: Example 2

```
COUNTER=0

while (( COUNTER < 10 )) ; do

echo The counter is $COUNTER

(( COUNTER = COUNTER+1 ))

Done
```

3. until loop

until loop: Example 1

```
#!/bin/bash

COUNTER=20

until [ $COUNTER -lt 10 ]

do

echo COUNTER $COUNTER

let COUNTER-=1

done
```

4. Exemplo: usando laços a rebatizar arquivos

Objetivo: Rebatize todas as arquivos do formulário cs333l*.ppt a cs333l*.exe onde * devem estar 02, 03… 10, se usar para o laço.

Solução 1:

para n em 02 03 04 05 06 07 08 09 10; [[do]]

milivolt cs265l$n.ppt cs265l$n.exe

feito

Solução 2:

para ((n=2; n< 10; ++i)) ; [[do]]

milivolt cs333l0$n.ppt cs333l0$n.exe

feito

milivolt cs333l10.ppt cs333l10.exe

Rebatiza cs333l02.ppt a cs333l02.exe, a cs333l03.ppt a cs333l03.exe, etc.

5. Condições de controle do laço

break - termina o loop

continue - causas um salto à iteração seguinte do laço

6. Debbugando os erros

Se quiser prestar atenção aos comandos realmente que estão sendo executados em uma lima do certificado, introduza a linha

```
"set - x" no script.
set - x
for  n in *; do
eco $n
done
```

Indicará o comando expandido antes de executá-lo.

```
+ echo bin
bin
+ echo mail
mail
...
```

Funções

Como em quase toda a língua de programação, você pode usar funções agrupar partes de código em uma maneira mais lógica ou praticar a arte divine do recursion. Declarar uma função é apenas uma matéria da função da escrita:

my_func {my_code}.

Chamar uma função é justa como a chamada de um outro programa, você escreve apenas seu nome.

1. Exemplo das variáveis de escopo

```
#! /bin/bash
HELLO=Hello          HELLO #variable com espaço do programa
função hello {
HELLO=World local       HELLO #variable com espaço local
eco $HELLO
```

```
}

eco $HELLO de $

$ hello

eco $HELLO de $
```

2. Funções com parâmetros: Exemplo

```
#! /bin/bash

function quit {

echo “adeus!”

exit

}

function hello {

echo “Hello $1”

}

For  name em Godfrey Justo;

do

hello $name

done

quit
```

Output:

```
Hello Godfrey

Hello Justo

Adeus
```

3. Expansão do parâmetro

```
$ {parâmetro: - palavra} - use valores de defeito.

$ {parâmetro: =word} - atribua valores de defeito.

$ {parâmetro:? a palavra} - erro da exposição se zero ou Unset.

$ {parâmetro: +word} - use o valor alterno.
```

4. Mais expansão do parâmetro

Nós podemos remover as partes de um valor:

Expansão do parâmetro	Finalidade
$ {param#pattern}	remove (##) o teste padrão principal o mais curto (#) ou o mais longo, se houver um fósforo
$ {param##pattern}	
$ {param%pattern}	remove (%) o teste padrão arrastando o mais curto (%) ou o mais longo, se fósforo
$ {param%%pattern}	

o teste padrão é expandido apenas como na expansão do pathname (globbing) - *? , []

Além disso nós podemos

encontre o comprimento de uma string: eco $ {#foo}

substrings do extrato: eco $ {foo: 2: 3}

execute a busca de Regex e substitua-a.

Para mais detalhes veja o manpage do Bash.

5. Expansão do parâmetro - exemplo

```
$ foo=j.i.c

eco $ de $ {foo#*.}

i.c

eco $ de $ {foo##*.}

c

eco $ de $ {foo%.*}

j.i

eco $ de $ {foo%%.*}

j
```

Conclusão

Esta atividade apresentou o processo da escrita e de executar certificados de escudo, construções da língua e destacou as características diferentes encontradas programação no escudo do Unix/Linux/que scripting.

Avaliação

1. Pratique os exemplos usados nesta atividade

Atividade 1.3 - Código Inline do conjunto

Introdução

Línguas de alto nível tais como o funcionamento de C e de C++ em quase todas as arquiteturas e produtividade mais elevada do rendimento ao escrever e ao manter o código. Para ocasiões quando os programadores necessitam usar instruções de conjunto em seus programas, a coleção do compilador do GNU permite programadores adicionar instruções arquitetura-dependentes da língua de conjunto a seus programas. As indicações inline do conjunto do GCC não devem ser usadas indiscriminately. As instruções da língua de conjunto são arquitetura-dependentes, assim, por exemplo, os programas que usam as instruções x86 não podem ser compilados em computadores de PowerPC. Para usá-los, você requererá uma facilidade na língua de conjunto para sua arquitetura. Entretanto, as indicações inline do conjunto permitem-no alcançar diretamente a ferragem e podem-nas também render um código mais rápido.

[[An]] ASM a instrução permite que você introduza instruções de conjunto em programas de C e de C++. Por exemplo, esta instrução

ASM ("fsin": "=t" (resposta): "0" (ângulo));

é uma maneira de x86-specific de codificar esta indicação de C:

```
resposta = sin (ângulo);
```

A expressão sin (ângulo) está executado geralmente como uma ligação de controle na biblioteca de math, mas se você especificar - O1 ou uma bandeira mais elevada do optimization, GCC é esperta bastante substituir a ligação de controle com um único fsin instrução de conjunto.

Observe isso ao contrário das instruções ordinárias do código do conjunto, ASM as indicações permitem-no especificar operandos da entrada e da saída usando a sintaxe de C. Para ler mais sobre o jogo de instrução x86, que nós usaremos nesta seção, veja o HTTP: /developer.intel.com/design/pentiumii/manuals/ e http://www.x86-64.org/documentation.

Embora ASM as indicações podem ser abusadas, permitem que seus programas alcancem a ferragem de computador diretamente, e podem produzir os programas que executam rapidamente. Você pode usá-los ao escrever o código de sistema operando-se que diretamente necessidades interagir com a ferragem. Por exemplo, /usr/include/asm/io.h contem instruções de conjunto para alcançar diretamente a entrada - portos de saída. A lima do código de fonte de Linux /usr/src/linux/arch/i386/kernel/process.s fornece um outro exemplo, usando-se hlt no código do laço inativo. Veja outras arquivos do código de fonte de Linux dentro /usr/src/linux/arch/ e /usr/src/linux/drivers/.

As instruções de conjunto podem também apressar o laço innermost de programas de computador. Por exemplo, se a maioria de tempo running de um programa computasse o seno e o cosine dos mesmos ângulos, este laço innermost poderia ser usar-se recoded fsincos instruções x86 (as mudanças algorítmica ou de dados da estrutura podem ser mais eficazes em reduzir tempo running de um programa do que usando instruções de conjunto). Veja, por exemplo, /usr/include/bits/mathinline.h, que envolve acima em macros algumas seqüências de conjunto inline que apressam a computação transcendental da função. Você deve usar o conjunto inline apressar-se acima do código somente como um último recurso. Os compiladores atuais são completamente sofisticados e sabem muito sobre os detalhes dos processadores para que geram o código. Conseqüentemente, os compiladores podem frequentemente escolher as seqüências do código que podem parecer unintuitive ou o roundabout mas que execute realmente mais rapidamente do que outras seqüências de instrução. A menos que você compreender o jogo de instrução e os atributos programando de seu processador do alvo muito bem, você é provavelmente melhor fora de deixar os optimizers do compilador gera o código do conjunto para você para a maioria de operações.

Ocasionalmente, uma ou dois instruções de conjunto podem substituir diversas linhas do código da língua higher-level. Por exemplo, determinar a posição do bocado nonzero o mais significativo de um inteiro nonzero que usa as línguas de programação de C requer um laço ou umas computações floating-point. Muitas arquiteturas, including o x86, têm uma única instrução de conjunto (bsr) para computar esta posição de bocado.

Detalhes da atividade

Conjunto Inline simples de 1.3.2.1

Aqui nós introduzimos a sintaxe de ASM instruções de ajuntador com um exemplo x86 para deslocar bocados de um valor 8 à direita:

ASM (“shrl $8, %0”: “=r” (resposta): “r” (operando): “centímetro cúbico”);

O keyword ASM é seguido pelas seções consistindo de uma expressão parenthetic separadas por dois pontos. A primeira seção contém uma instrução de ajuntador e seus operandos. Neste exemplo, shrl direito-desloca os bocados em seu primeiro operando. Seu primeiro operando é representado perto %0. Seu segundo operando é a constante imediata $8. A segunda seção especifica as saídas. A instrução uma output será colocada na resposta variável de C, que deve ser lvalue. A string “=r” contém um sinal de iguais que indica um operando da saída e r indicando que a resposta está armazenada em um registo. A terceira seção especifica as entradas. O operando variável de C especifica o valor para deslocar. A string “r” indica que está armazenado em um registo mas omite um sinal de iguais porque é um operando da entrada, não um operando da saída. A quarta seção indica que a instrução muda o valor no registo do centímetro cúbico do código de circunstância.

1. Converter-se ASM às instruções de conjunto

Tratamento do GCC de ASM as indicações são muito simples. Produz instruções de conjunto para tratar do ASM’ os operandos de s, e substituem ASM indicação com a instrução que você

especifica. Não analisa a instrução em nenhuma maneira. Por exemplo, o GCC converte este fragmento do programa

foo dobro, barra;

```
ASM ("mycool_asm %1, %0": "=r" (barra): "r" (foo));

a estas instruções de conjunto x86:

movl -8 (%ebp), %edx

movl -4 (%ebp), %ecx

#APP

mycool_asm %edx, %edx

#NO_APP

movl %edx, - 16 (%ebp)

movl %ecx, - 12 (%ebp)
```

Recorde isso foo e barra cada um requer duas palavras do armazenamento da pilha 32 em uma arquitetura do bocado x86. O registo ebp pontos aos dados na pilha. A primeira cópia de duas instruções foo em registos EDX e ECX em qual mycool_asm opera-se. O compilador decide-se usar os mesmos registos armazenar a resposta, em que é copí barra pelas duas instruções finais. Escolhe registos apropriados, mesmo reusing o mesmo regista, e copía operandos a e das posições apropriadas automaticamente.

Sintaxe prolongada do conjunto de 1.3.2.2

Nas subseções que seguem, nós descrevemos as réguas da sintaxe para ASM indicações. Suas seções são separadas por dois pontos. Nós consultaremos a este ilustrativo ASM indicação, que computa a expressão booleana x > y:

ASM ("fucomip %%st (1), %%st; seta %%al":

"=a" (resultado): "u" (y), "t" (x): "centímetro cúbico", "st");

Primeiramente, fucomip compara seus dois operandos x e y, e valores das lojas que indicam o resultado no registo do código de circunstância. Então seta converte estes valores em uns 0 ou 1 resultados.

1. Instruções de assembler

A primeira seção contém as instruções de ajuntador, incluídas na citação - marcas. O exemplo ASM contém duas instruções de conjunto, fucomip e seta, separado por semicolons. Se o ajuntador não permitir semicolons, use caráteres do newline (\ n) separar instruções. O compilador ignora os índices desta primeira seção, a não ser que esse um nível de sinais da porcentagem seja removido, assim %% mudanças a %. O meaning de %%st (1) e outros tais termos são arquitetura-dependentes.

O GCC queixar-se-á se você especificar - tradicional opção ou - ansi opção ao compilar conter do programa ASM indicações. Para evitar de produzir estes erros, como em arquivos de encabeçamento, use o keyword alternativo __asm__.

2. Saídas

A segunda seção especifica operandos da saída das instruções' usando a sintaxe de C. Cada operando é especificado por uma string do confinamente do operando seguida pela expressão da A. A. nos parênteses. Para os operandos da saída, que devem ser lvalues, a string do confinamente deve começar com um sinal de iguais. O compilador certifica-se de que a expressão de C para cada operando da saída esteja no fato lvalue. As letras que especificam registos para uma arquitetura particular podem ser encontradas no código de fonte do GCC, no REG_CLASS_FROM_LETTER macro. Por exemplo, GCC/config/i386/i386.h a lima da configuração no GCC alista as letras do registo para a arquitetura x86 (você necessitará ter alguma familiaridade com os internals do GCC para fazer o sentido desta lima). A tabela 1 sumaria abaixo estes.

Tabela 1: Letras do registo para a arquitetura de Intel x86

Letras de registro	Regista que o GCC pode se usar
R	Registo geral (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP)
Q	Registo geral para os dados (EAX, EBX, ECX, EDX)
F	Registo Floating-point
T	Registo floating-point superior
U	registo floating-point do Segundo--alto
A	Registo de EAX
B	Registo de EBX
C	Registo de ECX
D	Registo de EDX
X	Registo de SSE (registo fluindo da extensão de SIMD)
Y	Registos dos multimedia MMX
A	Um valor de 8 byte deu forma de EAX e de EDX
D	Ponteiro do destino para as operações da corda (EDI)

S	Ponteiro da fonte para as operações da corda (ESI)

Operandos múltiplos no ASM a indicação, cada uma especificadas por uma corda do confinamente e a expressão da A.A., são separadas por vírgulas, como ilustrado no exemplo ASM' seção de entrada de s. Você pode especificar até 10 operandos, denotados %0, %1,…, %9, nas seções da saída e de entrada. Se não houver nenhum operando da saída mas há uns operandos input ou clobbered os registos, saem da seção da saída vazia ou marcam-na com um comentário como

```
/* nenhumas saídas */.
```

3. Entradas

A terceira seção especifica os operandos da entrada para as instruções de ajuntador. A corda do confinamente para um operando da entrada não deve ter um sinal de iguais, que indique lvalue. Se não, a sintaxe de um operando da entrada é a mesma que para operandos da saída. Para indicar que um registo é ambos leia de e escrito ao mesmo ASM, use uma corda do confinamente da entrada do número do operando da saída. Por exemplo, para indicar que um registo da entrada é o mesmo como o primeiro número do registo da saída, uso 0. Os operandos da saída são numerados à esquerda à direita, começando com 0. Meramente especificar a mesma expressão de C para um operando da saída e um operando da entrada não garante que os dois valores estarão colocados no mesmo registo. Esta seção de entrada pode ser omitida se não houver nenhum operando da entrada e a seção subseqüente do clobber estiver vazia.

4. Clobbers

Se uma instrução modificar os valores de um ou mais registar como um efeito lateral, especifique clobbered registos no ASM' seção de s quarto. Por exemplo, fucomip a instrução modifica o registo do código de circunstância, que é denotado centímetro cúbico. A representação separada das cordas clobbered registos com vírgulas. Se a instrução puder modificar uma posição de memória arbitrária, especifique a memória.

Usando a informação do clobber, o compilador determina que valores devem ser recarregados após ASM executa. Se você não especificar esta informação corretamente, o GCC pode supor incorretamente que os registos contêm ainda os valores que, no fato, overwritten, que afetará a exatidão do seu programa.

Examplo

A arquitetura x86 inclui as instruções que determinam as posições de menos jogo significativo mordido e o jogo o mais significativo mordido em uma palavra. O processador pode executar estas instruções completamente eficientemente. No contraste, executar a mesma operação em C requer um laço e um deslocamento do bocado. Por exemplo, bsrl instrução de conjunto computa a posição do bocado o mais significativo ajustado em seu primeiro operando, e coloca a posição de bocado (que conta de 0, menos bocado significativo) em seu segundo

operando. Para colocar a posição de bocado para o número na posição, nós poderíamos usar este ASM indicação:

ASM ("bsrl %1, %0": "=r" (posição): "r" (número));

O one-way que você poderia executar a mesma operação em C está usando este laço:

long i;

for (i = (número >> 1), posição = 0; i! = 0; ++ posição)

i >>= 1;

Para testar as velocidades relativas destas duas versões, nós colocá-las-emos em um laço que compute as posições de bocado para um grande número valores. O programa "bit-pos-loop.c" dado abaixo faz isto que usa a execução do laço de C. O programa dá laços em inteiros excedentes, de 1 até o valor especificado na linha de comando. Para cada valor do número, computa o bocado o mais significativo que é ajustado.

O programa subseqüente "bit-pos-asm.c" faz a mesma coisa usando a instrução de conjunto inline. Anote que em ambas as versões, nós atribuímos a posição de bocado computada a a temporário resultado variável. Este é coerce o optimizer do compilador de modo que não elimine a computação inteira da posição de bocado; se o resultado não for usado nem não for armazenado na memória, o optimizer elimina a computação como "o código inoperante."

//Program "bit-pos-loop.c" - Find Bit Position Using a Loop

```
#include <stdio.h>

#include <stdlib.h>

int main (int argc, char* argv[])

{

long max = atoi (argv[1]);

long number;

long i;

unsigned position;

volatile unsigned result;

/* Repeat the operation for a large number of values. */

for (number = 1; number <= max; ++number) {

/* Repeatedly shift the number to the right, until the result is

zero. Keep count of the number of shifts this requires. */

for (i = (number >> 1), position = 0; i != 0; ++position)
```

```
    i >>= 1;
    /* The position of the most significant set bit is the number of
    shifts we needed after the first one. */
    result = position;
    }
    return 0;
    }
    //Program "bit-pos-asm.c" - Find Bit Position Using bsrl
    #include <stdio.h>
    #include <stdlib.h>
    int main (int argc, char* argv[])
    {
    long max = atoi (argv[1]);
    long number;
```

unsigned position;

volatile unsigned result;

```
    /* Repeat the operation for a large number of values. */
    for (number = 1; number <= max; ++number) {
    /* Compute the position of the most significant set bit using the
    bsrl assembly instruction. */
    asm ("bsrl %1, %0" : "=r" (position) : "r" (number));
    result = position;
    }
    return 0;
    }
```

Nós compilaremos ambas as versões com optimization cheio:

```
$ cc -O2 -o bit-pos-loop bit-pos-loop.c
$ cc -O2 -o bit-pos-asm bit-pos-asm.c
```

Agora vamos funcionar cada um que usa o comando do tempo medir o tempo de execução. Nós especificaremos um valor grande como a comando-linha argumento, para certificar-se de

que cada versão faz exame pelo menos de alguns segundos ao funcionamento.

```
$ time ./bit-pos-loop 250000000
```

19.51user 0.00system 0:20.40elapsed 95%CPU (0avgtext+0avgdata

0maxresident)k0inputs+0outputs (73major+11minor)pagefaults 0swaps

$ time ./bit-pos-asm 250000000

3.19user 0.00system 0:03.32elapsed 95%CPU (0avgtext+0avgdata

0maxresident)k0inputs+0outputs (73major+11minor)pagefaults 0swaps

Observe que a versão que usa o conjunto inline executa bastante mais rapidamente (seus resultados para este exemplo podem variar).

Considerações de Optimization

O optimizer do GCC tenta rearranjar e reescrever código dos programas' para minimizar o tempo de execução mesmo na presença de ASM expressões. Se o optimizer determinar que ASM' s os valores da saída não são usados, a instrução serão omitidos a menos que o keyword temporário ocorre no meio ASM e seus argumentos. (Como um caso especial, o GCC não moverá ASM sem alguns operandos fora de um laço.) alguns da saída ASM pode ser movido nas maneiras que são difíceis de predizer, mesmo através dos saltos. A única maneira garantir requisitar particular da instrução de conjunto é incluir todas as instruções no mesmo ASM. Usar-se asms pode restringir a eficácia do optimizer porque o compilador não sabe asms' semântica. O GCC é forçado para fazer as suposições conservadoras que podem impedir alguns optimizations. Emptor do Caveat!

Manutenção e considerações de Portability

Se você se decidir se usar nonportable, arquitetura-dependente ASM as indicações, encapsulating estas indicações dentro dos macros ou das funções podem ajudar na manutenção e em mover. Colocando todos estes macros em uma lima e documentando os facilitará mover a uma arquitetura diferente, algo que ocorre com freqüência surpreendendo mesmo para programas "throwaway". Assim, o programador necessitará reescrever somente uma lima para a arquitetura diferente. Por exemplo, a maioria ASM as indicações no código de fonte de Linux são agrupadas em /usr/src/linux/include/asm and usr/src/linux/include/asm-i386 header files, and /usr/src/linux/arch/i386/ and /usr/src/linux/drivers/ codigo fonte.

Conclusão

Nesta atividade nós apresentamos os princípios de inling código do conjunto no programa da A.A. /C++.

Avaliação

Pratique os exemplos dados nesta atividade, para ajudar-lhe começar mais introspecções do conjunto inlining.

Sumário da unidade

Esta unidade apresentou três aspectos de programar um sistema computatorizado, a saber, usando o escudo, as línguas high-level tais como C/C++, e a língua de conjunto. Especificamente, o escudo do Unix/Linux foi introduzido ao lado do processo da escrita e das construções de um programa do escudo. Nós apresentamos os escudos geralmente usados estamos disponíveis no ósmio de Linux/Unix, e explicamos o processo de escrever um certificado de escudo e de trabalhar com variáveis nos escudos finalmente, as técnicas de inlining o código do conjunto no programa da A.A. /C++ foi apresentado.

Avaliação da unidade

Verifique sua compreensão!

Variado Exercícios

Instruções

Escreva o seguinte certificado de escudo, excepto ele, executam-no e anotam-no para baixo é saída.

```
# certificado para imprimir atualmente a informação do usuário que
início de uma sessão, data atual & hora

#

espaço livre

eco "Hello $USER"

é do eco "hoje \ c"; data

echo "Number of user login : \c" ; who |  wc -l

echo "Calendar"

cal

exit 0
```

1. Tenta o comando seguinte :

a. $ expr 1 + 3 $ echo $?

b. $ echo Welcome $ echo $?

c. $ wildwest canwork? $ echo $?

d. $ date $ echo $?

e. $ echon $? $ echo $?

2. Se você quiser imprimir a sua localização inicial do então você dá o comando

a. $ echo $HOME or

b. $ echo HOME

3. Qual dos comandos acima é a correta? Por quê?

4. Explique brevemente como você pode realizar cada uma das operações:

a. definir variável x com o valor 10 e imprimi-lo na tela?

b. definir xn variável com valor Rani e imprimi-lo na tela?

c. soma de impressão de dois números, digamos 6 e 3?

d. definir duas variáveis x = 20, y = 5 e, em seguida, divisão de impressão de x e y (isto é, X / Y)?

e. Modificar em (d) acima e divisão da loja de x e y para uma variável chamada z.

5. Apontar erros se houver no script a seguir:

```
# Script to test MY knowledge about variables!
#
myname=Vivek
myos = TroubleOS
myno=5
echo “My name is $myname”
echo “My os is $myos”
echo “My number is myno, can you see this number”
```

6 Escrever um script, pad, essa pad vontade do lado esquerdo. Contagem directa de 2 a 10.

7. Faça o seguinte programa de montagem inline, salvá-lo, executá-lo e anotar a sua saída (Os exemplos faz uso de declarações de montagem em linha prolongados. Ele executa operações aritméticas simples sobre operandos inteiros e exibe o resultado

```
#include <stdio.h>
int main() {
   int arg1, arg2, add, sub, mul, quo, rem ;
```

```
    printf( "Enter two integer numbers : " );

    scanf( "%d%d", &arg1, &arg2 );

    /* Perform Addition, Subtraction, Multiplication & Division */

    __asm__ ( "addl %%ebx, %%eax;" : "=a" (add) : "a" (arg1) , "b"
(arg2) );

    __asm__ ( "subl %%ebx, %%eax;" : "=a" (sub) : "a" (arg1) , "b"
(arg2) );

    __asm__ ( "imull %%ebx, %%eax;" : "=a" (mul) : "a" (arg1) , "b"
(arg2) );


    __asm__ ( "movl $0x0, %%edx;"

              "movl %2, %%eax;"

              "movl %3, %%ebx;"

               "idivl %%ebx;" : "=a" (quo), "=d" (rem) : "g" (arg1),
"g" (arg2) );

    printf( "%d + %d = %d\n", arg1, arg2, add );

    printf( "%d - %d = %d\n", arg1, arg2, sub );

    printf( "%d * %d = %d\n", arg1, arg2, mul );

    printf( "%d / %d = %d\n", arg1, arg2, quo );

    printf( "%d %% %d = %d\n", arg1, arg2, rem );


    return 0 ;

}
```

8 Escreva o seguinte programa assembly inline, salvá-lo, executá-lo e anotar a sua saída (O exemplo a apontar para calcular o máximo divisor comum utilizando bem conhecido algoritmo de Euclides).

```
#include <stdio.h>

int gcd( int a, int b ) {

    int result ;

    /* Compute Greatest Common Divisor using Euclid's Algorithm */

    __asm__ __volatile__ ( "movl %1, %%eax;"
```

```
                        "movl %2, %%ebx;"
                        "CONTD: cmpl $0, %%ebx;"
                        "je DONE;"
                        "xorl %%edx, %%edx;"
                        "idivl %%ebx;"
                        "movl %%ebx, %%eax;"
                        "movl %%edx, %%ebx;"
                        "jmp CONTD;"
                        "DONE: movl %%eax, %0;" : "=g" (result) :
"g" (a), "g" (b)
    );
    return result ;
}
int main() {
    int first, second ;
    printf( "Enter two integers : " ) ;
    scanf( "%d%d", &first, &second );

    printf( "GCD of %d & %d is %d\n", first, second, gcd(first,
second) ) ;

    return 0 ;
}
```

9. Re-escrever os programas assembly em linhas dadas exercícios 7 e 8 totalmente em linguagem C e comparar a velocidade de execução do programa de montagem versão em linha com a versão do programa C correspondente.

Leituras e outros recursos

1. Learning the bash Shell: Unix Shell Programming (In a Nutshell (O'Reilly)), Kindle Edition, Cameron Newham (Author)
2. bash Cookbook: Solutions and Examples for bash Users (Cookbooks (O'Reilly)), Kindle Edition, Carl Albing (Author), JP Vossen (Author), Cameron Newham (Author)

3. Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001

4. Professional Assembly Language, John Wiley & Sons, 22 Feb 2005, By Richard Blumhttp://www.codeproject.com/Articles/15971/Using-Inline-Assembly-in-C-C

Unidade 3. Processos threads e Gestão de Memória

Introdução à Unidade

Uma instância em execução de um programa é chamada de processo. Por exemplo, se você tem dois terminais janelas mostrando na tela, então provavelmente você está executando o mesmo programa de terminal duas vezes, ou seja, você tem dois processos do terminal. Cada janela de terminal provavelmente está executando um shell; cada shell em execução é um outro processo. Quando você invocar um comando de um shell, o correspondente programa é executado num processo novo; o processo shell recomeça quando isso processo seja concluído. Programadores avançados costumam usar vários processos cooperantes em um único aplicativo para permitir que o aplicativo faça mais de uma coisa ao mesmo tempo, para aumentar a aplicação

robustez, e fazer uso de programas já existentes.

Threads, como processos, são um mecanismo para permitir que um programa faça r mais de uma coisa de cada vez. Tal como acontece com os processos, threads aparecem para executar simultaneamente; Escalas do Kernel do linux de forma assíncrona, interrompendo cada thread de vez em quando para dar aos outros a oportunidade de executar.

Conceptualmente, um thread existe dentro de um processo. Threads são uma unidade de granulação mais fina-da execução de processos. Quando você chamar um programa, Linux cria um novo processo e nesse processo cria um único thread, que executa o programa sequencialmente. Esse thread pode criar Threads adicionais; todos estes thread executa o mesmo programa no mesmo processo, mas cada thread pode ser uma execução diferente parte do programa a qualquer momento.

Programa deve ser trazida para a memória e colocada dentro de um processo para ser executado. A memória é a área de armazenamento de dados primário para computadores. Chamamos a unidade de memória básica um pouco. Um bit pode conter dois valores diferentes: 0 ou 1. A memória é composta de um número de células que podem armazenar algum número de bits. A memória é apenas um array de bytes. Cada célula tem um número para identificá-lo,

chamou o seu endereço. Programas referem-se endereços de memória para atingir. Um sistema de gestão de chamadas memória virtual organiza memória no “páginas”, que são unidades de memória geral, uma pequena Kbytes de tamanho. CPU tem uma unidade chamada Unidade de gerenciamento de memória que é responsável pela operando de memória virtual.

Nesta unidade, descrevem funções de manipulação de processo, thread e memória em sistemas Linux cuja maioria dos são semelhantes aos de outros sistemas UNIX. A maioria das funções são descritas declarado no arquivo de cabeçalho <unistd.h>; verifique a página do homem para cada função para ter certeza.

Porque um processo e todos os seus threads podem ser executar apenas um programa por vez, se qualquer thread dentro de um processo chama uma das funções exec, todos os outros threads estão terminou (o novo programa pode, é claro, criar novos Threads). GNU / Linux implementa o POSIX API thread padrão (conhecido como pthreads). Todas as funções de linha e tipos de dados são declarados no arquivo de cabeçalho <Pthread.h> .As funções pthread não estão incluídos na biblioteca padrão C. Em vez disso, eles estão em libpthread, então você deve adicionar -lpthread à linha de comando quando você ligar o seu programa

Objetivos da Unidade

Após a conclusão desta unidade, você deve ser capaz de:

- Explicar os conceitos de gerenciamento de processo, thread e memória
- Criar processos utilizando as funções do sistema, fork e exec
- Gerenciar e manipular processos usando vários sinais do sistema
- Criar e gerir threads
- Comparar e contrastar os processos de threads
- Demonstrar entendimento de gerenciamento de memória em programas C

Processo	Um processo é um exemplo de um programa em execução em um computador. Isto é perto de significado para a tarefa, um termo usado em alguns sistemas operacionais. Em alguns outros sistemas operacionais UNIX / Linux e, um processo é iniciado quando um programa é iniciado (quer por um utilizador entrar numa shell de comando ou por outro programa).
Thread	Um thread de execução é a menor seqüência de programação instruções que podem ser gerenciados de forma independente por um programador, que é, tipicamente, uma parte do sistema operacional. o implementação de threads e processos difere entre os sistemas operativos, mas na maioria dos casos, um thread é um componente de um processo. Vários threads podem existir dentro do mesmo processo e partilhar recursos, como memória, enquanto os processos diferentes não compartilhar esses recursos. Em particular, os threads de um processo compartilham suas instruções (código executável) e seu contexto (os valores de seus variáveis em qualquer momento).

Gerenciamento de memória	Gerenciamento de memória é o ato de memória do computador gestão no nível do sistema. O requisito essencial da memória gestão é proporcionar maneiras de alocar dinamicamente porções de memória para programas a seu pedido, e libertá-lo para reutilização quando não é mais necessário. Isto é crítico para qualquer computador avançado sistema onde mais de um único processo em curso pode ser qualquer momento.

Actividades de Aprendizagem

Actividade 1.1 -Processos

Introdução

Mesmo quando você se senta em seu computador, existem processos em execução. Cada programa em execução utiliza um ou mais processos. Vamos começar dando uma olhada nos processos já em seu computador.

Detalhes da actividade

Trabalhando com processos

1. Processo Ids

Cada processo em um sistema Linux é identificado pelo seu ID único processo, por vezes referido como pid. IDs de processo são números de 16 bits que são atribuídos sequencialmente pelo Linux como novos processos são criado. Cada processo também tem um processo pai (exceto o processo primeiro processo especial de init em Sistemas Linux.

Assim, você pode pensar dos processos em um sistema Linux como organizados em uma árvore, com o processo de inicialização em sua raiz. O ID do processo pai, ou ppid, é simplesmente o ID do processo do

pai do processo. Quando se refere a processar IDs em um programa C ou C ++, use sempre o pid_t typedef, que é definida em <sys / types.h>. Um programa pode obter o ID do processo do processo ele está correndo com a chamada de sistema getpid (), e pode obter o ID do processo do seu processo pai com a chamada de sistema getppid (). Por exemplo, o programa "print-pid.c" fornecido abaixo imprime seu ID do processo e ID do processo do seu pai.

// Programa de impressão de pid.c - Imprimindo a identificação de processo

```
#include <stdio.h>

#include <unistd.h>

int main ()

  {

printf ( "A identificação do processo é% d \ n", (int) getpid ());

printf ( "O ID do processo pai é% d \ n", (int) getppid ());

return 0;                    }
```

Observe que, se você invocar este programa várias vezes, um ID de processo diferente é relatado porque cada chamada está em um novo processo. No entanto, se você chamá-lo toda vez que a partir do mesmo shell, o ID do processo pai (isto é, a identificação do processo do processo de shell) é o mesmo.

2. Visualizando Processos ativos

O comando ps exibe os processos que estão em execução no sistema. O GNU / Linux versão do ps tem muitas opções porque tenta ser compatível com versões de ps em várias outras variantes de UNIX. Estas opções controlam os processos que estão listadas e quais informações sobre cada é mostrado. Por padrão, invocando ps exibe os processos controlados pelo terminal ou janela de terminal em que ps é invocado. Por exemplo:

```
$ ps
```

PID TTY TEMPO CMD

21693 pts / 8 00:00:00 festança

21694 pts / 8 00:00:00 ps

Esta invocação de ps mostra dois processos. O primeiro, festança, é o shell em execução neste terminal. A segunda é a instância em execução do próprio programa ps. A primeira coluna, marcado PID, exibe o ID do processo de cada um. Para uma visão mais detalhada sobre o que está sendo executado em seu GNU / Linux sistema, invocar esta:

```
$ Ps -e -o pid, ppid, comando
```

A opção -e instrui ps para exibir todos os processos em execução no sistema. O -o pid, ppid, opção de comando diz ps quais informações para mostrar sobre cada processo - neste caso, o processo ID, o ID do processo pai, e o comando em execução neste processo. Abaixo estão as primeiras linhas e últimas linhas de saída deste comando no sistema típico. Você pode ver uma saída diferente, dependendo do que está sendo executado no sistema.

```
$ ps -e -o pid,ppid,command
```

PID	PPID	COMMAND
1	0	init [5]
2	1	[kflushd]
3	1	[kupdate]
	...	
21725	21693	Xterm
21727	21725	Bash
21728	21727	ps -e -o pid,ppid,command

Note que o ID do processo pai do comando ps, 21727, é o ID do processo bash, o shell a partir do qual eu invoquei ps. O ID do processo pai do bash por sua vez é 21725, o ID do processo do programa xtrem em que o shell está em execução.

3. Matar um processo

Pode matar um processo em execução com o comando kill. Basta especificar na linha de comando do ID de processo do processo a ser eliminado. Por padrão, o comando funciona matar enviando o processo um SIGTERM ou sinal de término. Isso faz com que o encerramento do processo, a menos que a execução programa controla ou máscaras explicitamente o sinal SIGTERM.

Criação Processos -

Duas técnicas comuns são usados para a criação de um novo processo, a função do system e a função fork() juntamente com as funções exec. O primeiro é relativamente simples, mas devem ser usados com moderação porque é ineficiente e consideravelmente tem riscos de segurança. A segunda técnica é mais complexo, mas oferece maior flexibilidade, rapidez e segurança.

1. Utilizando da função system

A função do sistema na biblioteca padrão C fornece uma maneira fácil de executar um comando a partir de dentro de um programa, tanto quanto se o comando foi digitado em um shell. Na verdade, o sistema cria um subprocess executando o shell padrão Bourne (/ bin / sh) e, em seguida, passa o comando para que shell para execução. Por exemplo, o “System.c” programa fornecido abaixo invoca os ls

comando para exibir o conteúdo do diretório raiz, como se você digitou ls -l / dentro de uma concha.

// Programa “System.c” - usando a chamada de sistema

```
#include <stdlib.h>

int main ()
```

```
{
 int return_value;
 RETURN_VALUE = system( "ls -l /");
 return return_value;
}
```

A função system retorna o status de saída do comando shell. Se o próprio reservatório não pode ser executado, sistema retorna 127; se ocorrer outro erro, o sistema retorna -1. Porque a função sistema usa uma concha para invocar o seu comando, é sujeita às características, limitações e falhas da segurança shell do sistema. Você não pode contar com a disponibilidade de qualquer versão específica do shell Bourne.

Em muitos sistemas UNIX, / bin / sh é um link simbólico para outro shell. Por exemplo, na maioria Sistemas GNU / Linux, / bin / sh pontos para bater (o B ourne- A ell ganho SH), e diferente GNU / Linux distribuições usam diferentes versões do bash. Invocando um programa com privilégios de root com o função do sistema, por exemplo, pode ter resultados diferentes em diferentes sistemas GNU / Linux. Portanto, é preferível usar o garfo e o método exec para a criação de processos.

2. Usando fork e exec

A API DOS e Windows contêm a família desova de funções. Estas funções tomar como argumento o nome de um programa para executar e criar uma nova instância do processo do programa. Linux não contém uma única função que faz tudo isso em um passo. Em vez disso, o Linux fornece um função, garfo, o que torna um processo filho que é uma cópia exata do seu processo pai. Linux fornece um outro conjunto de funções, a família exec, o que provoca um processo particular para cessar sendo um exemplo de um programa e, em vez disso se tornar uma instância de outro programa. Para gerar um novo processo, primeiro você usar garfo para fazer uma cópia do processo atual. Então você usa exec para transformar um desses processos em uma instância do programa que pretende gerar.

a. fork chamando

Quando um programa chama fork, um processo duplicado, o chamado processo de filho, é criado. o pai processo continua a executar o programa a partir do ponto em que foi chamado garfo. O processo filho, também, executa o mesmo programa a partir do mesmo lugar. Então como é que os dois processos são diferentes? Primeiro, o processo filho é um processo novo e, portanto, tem um novo ID processo, diferente de seu pai de ID do processo. Uma maneira para um programa de distinguir se é no processo dos pais ou filho é chamar getpid. No entanto, a função fork fornece diferentes valores de retorno ao processos de um pai e filho processo "vai em" ao chamado garfo, e dois processos "vem fora ", com diferentes valores de retorno. O valor de retorno no processo pai é o ID do processo do filho. O valor de retorno no processo de filho é zero. Porque nenhum processo já tem um ID de processo zero, isso torna mais fácil para o programa se ele está funcionando agora como o pai ou filho processo.

Programa "fork.c" fornecido abaixo é um exemplo do uso da forquilha para duplicar processo de um programa.

Observe que o primeiro bloco do "if" é executada apenas no processo pai, enquanto o Cláusula de "else" é executado no processo filho.

// Fork.c Programa - Usando garfo para duplicar o processo de um Programa

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main ()

{

  pid_t child_pid;

  printf ("the main program process ID is %d\n", (int) getpid ());

  child_pid = fork ();

  if (child_pid != 0) {

printf ("this is the parent process, with id %d\n", (int) getpid
());

printf ("the child's process ID is %d\n", (int) child_pid);

  }

  else

printf ("this is the child process, with id %d\n", (int) getpid
());

              return 0;

}
```

b. Usando a família exec

As funções exec substituir o programa em execução em um processo com outro programa. Quando um programa chama uma função exec, esse processo será encerrado imediatamente executar esse programa e começa a execução de um novo programa desde o início, assumindo que a chamada exec não faz encontrar um erro. Dentro da família exec, existem funções que variam ligeiramente no seu capacidades e como eles são chamados. Funções que contêm a letra p em seus nomes (execvp e execlp) aceitar um nome do programa e procurar um programa com esse nome na atual caminho de execução; funções que não contêm a p deve ser dado o caminho completo do programa em executado. Funções que contêm a letra v em seus nomes (xecv, execvp e execve) aceitar a lista de argumentos para o novo programa como uma matriz terminada em NULL de ponteiros para strings. Funções que contêm a letra l (execl, execlp e execle) aceitar a lista de argumentos usando os mecanismo da linguagem C varargs. Funções que contêm a letra e em seus nomes (execve e execle) aceitam um argumento adicional, um conjunto de variáveis de ambiente. O argumento deve ser um NULL

- terminado matriz de ponteiros para string de caracteres. Cada cadeia de caracteres deve ser da forma "VARIÁVEL = valor".

Porque exec substitui o programa de chamada com outro, ele nunca retorna a menos que um erro ocorre. A lista de argumentos passado para o programa é análogo aos argumentos de linha de comando que você especificar para um programa quando você executá-lo a partir do shell. Eles estão disponíveis através do argc e parâmetros argv a principal. Lembre-se, quando um programa é chamado a partir do shell, o shell define o primeiro elemento da lista de argumentos argv [0] para o nome do programa, o segundo elemento da lista de argumentos argv [1] para o primeiro argumento de linha de comando, e assim por diante. Quando você usa um função exec em seus programas, você também deve passar o nome da função como o primeiro elemento da lista de argumentos.

c. Usando fork e exec Juntos

Um padrão comum para executar um subprograma dentro de um programa, primeiro é o processo fork() e, em seguida subprograma exec. Isso permite que o programa de chamada para continuar a execução no processo pai enquanto o programa de chamada é substituído pelo subprograma no processo filho. O programa "Fork-exec.c" fornecido abaixo, como "System.c" programa dado acima lista o conteúdo da raiz diretório usando o comando ls. Ao contrário do "System.c" do programa, no entanto, ele chama os ls comando diretamente, passando os argumentos de linha de comando -l e / invés de invocá-lo através de um shell.

// Programa "fork-exec.c" - Usando fork e exec Juntos

```
#include <stdio.h>

#include <stdlib.h>

#include <sys / types.h>

#include <unistd.h>

/ * Gerar um processo filho a execução de um novo programa. PROGRAMA
é o nome

do programa a ser executado; o caminho será procurado para este
programa.
```

ARG_LIST é uma lista terminada em NULL de string de caracteres para ser

passado como lista de argumentos do programa. Retorna o ID do processo

o processo gerado. * /

int spawn (char* program, char** arg_list)

```
{

pid_t child_pid;

/* Duplicate this process. */
```

```
child_pid = fork ();
if (child_pid != 0)
  /* This is the parent process. */
  return child_pid;
else {
  /* Now execute PROGRAM, searching for it in the path. */
  execvp (program, arg_list);
  /* The execvp function returns only if an error occurs. */
  fprintf (stderr, "an error occurred in execvp\n");
  abort ();
        }
}
int main ()
{
  /* The argument list to pass to the "ls" command. */
  char* arg_list[] = {
  "ls", /* argv[0], the name of the program. */
  "-l",
  "/",
  NULL /* The argument list must end with a NULL. */
  };
  /* Spawn a child process running the "ls" command. Ignore the
    returned child process ID. */
  spawn ("ls", arg_list);
  printf ("done with main program\n");
return 0;
}
```

3. Processos Scheduling

Linux Scheduling o processo pai e o filho de forma independente; não há nenhuma garantia de que um será executado em primeiro lugar, ou quanto tempo ele será executado antes do Linux interrompe-lo e permite que o outro processo (ou algum

outro processo no sistema) prazo. Em particular, nenhuma, parte ou a totalidade do comando ls pode ser executado em o processo filho antes do pai concluída. Linux promete que cada processo será executado eventualmente,-nenhum processo será completamente carente de recursos de execução.

Você pode especificar que um processo é menos importante e deve ser dada uma prioridade menor -por atribuindo-lhe um valor niceness maior. Por padrão, cada processo tem uma niceness de zero. A maior valor niceness significa que o processo é dada uma prioridade de execução menor; Por outro lado, um processo de com um menor (ou seja, negativo) niceness fica mais tempo de execução. Para executar um programa com um niceness diferente de zero, use o comando nice, especificando o valor niceness com a opção -n. Para exemplo, é assim que você pode chamar o comando "sort input.txt> output.txt", uma longa triagem operação, com prioridade reduzida de modo que ele não torne o sistema muito lento:

```
$ nice -n 10 tipo input.txt> output.txt
```

Você pode usar o comando renice para mudar a niceness de um processo em execução a partir do linha de comando. Para alterar a niceness de um processo em execução por meio de programação, use a função nice. Estes argumento é um valor de incremento, que é adicionado ao valor de niceness do processo que chama.

Lembre-se que um valor positivo eleva o valor niceness e reduz, assim, o processo de prioridade de execução.

Note que apenas um processo com privilégios de administrador pode executar um processo com um valor negativo ou niceness reduzir o valor niceness de um processo em execução. Isso significa que você pode especificar valores negativos ao nice e renice comandos somente quando logado como root, e somente um processo a correr como root pode passar um valor negativo para a função nice. Isso impede que usuários comuns de agarrar prioridade de execução longe de outras pessoas que utilizam o sistema.

Sinais

Os sinais são mecanismos de comunicação com e manipular processos em Linux. Nisso seção apresentamos alguns dos sinais e técnicas mais importantes que são usados para processos de controle.

Um sinal é enviado uma mensagem especial para um processo. Sinais são assíncronas; quando um processo de recebe um sinal, que processa o sinal imediatamente, sem terminar a função actual ou até mesmo a linha de código atual. Existem várias dezenas de diferentes sinais, cada um com uma diferente significado. Cada tipo de sinal é indicada pelo seu número de sinal, mas em programas, você geralmente se referem a um sinal pelo seu nome. No Linux, estes são definidos no /usr/include/bits/signum.h. Não deve incluir este arquivo de cabeçalho diretamente em seus programas, no entanto, em vez disso, use <signal.h>.

Quando um processo recebe um sinal, que pode fazer uma de várias coisas, em função do sinal de default disposition. Para cada sinal, não é uma disposição padrão, o que determina o que acontece com o processar se o programa não especifica algum outro comportamento.

Para a maioria dos tipos de sinais, um programa pode especificar algum outro comportamento, seja para ignorar o sinal ou para chamar um sinal de manipulador especial função para responder ao sinal. Se um manipulador de sinal é usado, o programa atualmente em execução é pausa, o manipulador de sinal é executado, e, quando o manipulador de sinal retorna, o programa recomeça.

O sistema Linux envia sinais para os processos em resposta às condições específicas. Por exemplo, SIGBUS (erro de bus), SIGSEGV (violação de segmentação) e SIGFPE (exceção de ponto flutuante) pode ser enviado para um processo que tenta executar uma operação ilegal. A disposição padrão para estes sinais para encerrar o processo e produzir um arquivo de núcleo. Um processo pode também enviar um sinal

com um outro processo. Um uso comum deste mecanismo é acabar com outro processo, enviando-lhe um SIGTERM ou sinal SIGKILL. Outro uso comum é enviar um comando para a execução programa. Dois sinais "UserDefined" são reservados para essa finalidade: SIGUSR1 e SIGUSR2 . O Sinal SIGHUP é por vezes utilizado para este fim, bem como, comumente para acordar uma marcha lenta programa ou causar um programa para reler seus arquivos de configuração.

A função sigaction pode ser usado para definir uma disposição de sinal. O primeiro parâmetro de sinal é a número. Os próximos dois parâmetros são ponteiros para estruturas sigaction; A primeira delas contém a disposição desejada para esse número do sinal, enquanto que a segunda recebe a disposição anterior.

O campo mais importante na primeira ou segunda estrutura sigaction é sa_handler. Pode demorar um de três valores:

- SIG_DFL, que especifica a disposição padrão para o sinal.
- SIG_IGN, que especifica que o sinal deve ser ignorado
- Um ponteiro para uma função de sinal de manipulador. A função deve ter um parâmetro, o número de sinal e de retorno void.

Dado que os sinais são assíncronas, o programa principal pode estar num estado muito frágil quando um sinal é processada e, portanto, enquanto uma função manipulador de sinal é executado. Portanto, você deve evitar realizar qualquer operação de I / O ou chamando a maioria das funções de biblioteca e sistema de manipuladores de sinal.

Um manipulador de sinal deve executar o trabalho mínimo necessário para responder ao sinal, e em seguida controle para o programa principal retornar (ou terminar o programa). Na maioria dos casos, este consiste simplesmente de gravar o facto de que ocorreu um sinal. O programa principal, em seguida, verifica periodicamente se um sinal de que ocorreu e reage em conformidade.

É possível para um processador de sinal a ser interrompida pela entrega de um outro sinal. enquanto este pode soar como uma ocorrência rara, se ocorrer, será muito difícil de diagnosticar e depurar o problema. Portanto, você deve ser muito cuidadoso sobre o seu programa faz em um sinal manipulador.

Mesmo atribuir um valor a uma variável global pode ser perigoso, porque a atribuição pode na verdade, ser levada a cabo em duas ou mais instruções de máquina, e pode ocorrer um segundo sinal entre eles, deixando a variável num estado corrompido. Se você usar uma variável global para sinalizar um sinal a partir de uma função de sinal de manipulador, que deve ser do tipo sig_atomic_t especial. Linux garantias de que as atribuições para as variáveis deste tipo são realizados em uma única instrução e portanto, não pode ser interrompido a meio caminho. No Linux, sig_atomic_t é um int ordinário; de fato, atribuição para tipos inteiros do tamanho de int ou menor, ou para ponteiros, são atômicas. Se você quiser escrever um programa que é portátil para qualquer sistema UNIX padrão, no entanto, usar sig_atomic_t para estes variáveis globais. O esqueleto programa "sigusr1.c" abaixo, por exemplo, utiliza um sinal de manipulado tocount função do número de vezes que o programa recebe SIGUSR1, um dos sinais reservados para uso do aplicativo.

// Programa Sigusr1.c - Usando um manipulador de sinal

```
#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

sig_atomic_t sigusr1_count = 0;

void handler (int signal_number)

{

++sigusr1_count;

}

int main ()

{

struct sigaction sa;

memset (&sa, 0, sizeof (sa));

sa.sa_handler = &handler;

sigaction (SIGUSR1, &sa, NULL);

/* Do some lengthy stuff here. */

/* ... */

printf ("SIGUSR1 was raised %d times\n", sigusr1_count);

return 0;

}
```

Terminação de Processo

Normalmente, um processo termina em uma de duas maneiras. Ou o programa de execução chama da saída função ou função principal retorna do programa. Cada processo tem um código de saída: um número queo processo retorna para a sua mãe. O código de saída é o argumento passado para a função de saída, ou a valor retornado do principal.

Um processo pode também terminar de forma anormal, em resposta a um sinal. Por exemplo, o SIGBUS, SIGSEGV e SIGFPE sinais mencionados anteriormente causar o encerramento do processo. De outros sinais são usados para encerrar um processo explicitamente. O sinal SIGINT é enviado para um processo quando

o usuário tenta terminá-la digitando Ctrl + C no seu terminal. O sinal SIGTERM é enviado pela matar comando. A disposição padrão para ambos destes é para finalizar o processo. Chamando ofunção abortar, um processo de se auto-envia o sinal SIGABRT, que encerra o processo e produz um arquivo de núcleo. O sinal de terminação mais poderoso é SIGKILL, que termina um processo imediatamente e não pode ser bloqueada ou tratado por um programa.

Qualquer um destes sinais podem ser enviados utilizando o comando kill, especificando uma linha de comando adicional bandeira; por exemplo, para terminar um processo problemático, enviando-lhe um SIGKILL, invoque o seguinte, onde pid é o ID do processo:

```
$ kill -kill pid
```

Para enviar um sinal a partir de um programa, use a função de matar. O primeiro parâmetro é o processo de destino ID. O segundo parâmetro é o número de sinal; usar SIGTERM para simular o comportamento padrão o comando kill. Por exemplo, onde pid filho contém a identificação do processo do processo de criança, pode-se utilizar a função kill para encerrar um proceso filho do pai da seguinte forma:

```
kill (child_pid, SIGTERM);
```

Incluir o <sys / types.h> e <> signal.h cabeçalhos se você usar a função de matar. Por convenção, o código de saída é usada para indicar se o programa executado correctamente. Um código de saída zero indica execução correta, enquanto um código de saída diferente de zero indica que ocorreu um erro. No neste último caso, o valor particular retornada pode dar uma indicação da natureza do erro. É um boa idéia para ficar com esta convenção em seus programas porque os outros componentes do Sistema GNU / Linux assumir esse comportamento. Por exemplo, conchas assumir essa convenção quando você conectar vários programas com o && (AND lógico) e ll operadores (OR lógico). Portanto, você deve retornar explicitamente zero, a partir de sua função principal, a menos que ocorra um erro.

Com a maioria das conchas, é possível obter o código de saída do programa executado mais recentemente usando o $ especial? variável. Aqui está um exemplo em que o comando ls é invocado duas vezes e seu código de saída é exibida após cada chamada. No primeiro caso, ls executado corretamente e regresso o código de saída zero. No segundo caso, sl encontra um erro (porque o nome especificado no a linha de comando não existe) e, portanto, retorna um código de saída diferente de zero.

$$ ls /

bin coda etc lib misc nfs proc sbin usr

boot dev home lost+found mnt opt root tmp var

$ echo $?

0

$ ls bogusfile

ls: bogusfile: No such file or directory

$ echo $?

1

Note-se que mesmo que o tipo de parâmetro da função de saída é a função principal retorna int e um int, Linux não preserva o total de 32 bits do código de retorno. Na verdade, você deve usar a saída códigos só entre zero e 127. Os códigos de saída acima de 128 têm um significado especial, quando um processo é terminado por um sinal; seu código de saída é 128 mais o número de sinais.

1. Esperando o encerramento do processo

Se você digitou e correu o garfo e programa de exemplo exec "fork-exec.c" na Seção 1.1.1.2, você deve ter notado que a saída do programa ls muitas vezes aparece após o "programa principal" tem já completado. Isso porque o processo de criança, em que ls é executado, está prevista independentemente do processo pai.

Como o Linux é um sistema operacional multitarefa, ambos os processos aparecem para executar ao mesmo tempo, e você não pode prever se o programa ls terá a chance de correr antes ou após o processo pai é executado. Em algumas situações, contudo, é desejável que o processo pai esperar até que um ou mais processos filho ter completado. Isto pode ser feito com a família de espera chamadas de sistema. Estas funções permitem que esperar por um processo para concluir a execução, e permitir que o processo pai para recuperar informações sobre rescisão de seu filho. Existem quatro diferentes sistema chama na família espera; você pode optar por ficar um pouco ou um monte de informações sobre o

processo que saiu, e você pode escolher se você se preocupa com o que processo filho terminado.

O mais simples tal função é chamado simplesmente esperar. Ele bloqueia o processo de chamada até que um dos seus

processos filho saídas (ou ocorre um erro). Ele retorna um código de status através de um argumento de ponteiro inteiro,

a partir do qual você pode extrair informações sobre como o processo filho saiu. Por exemplo, a WEXITSTATUS macro extrai o código de saída do processo filho.

Você pode usar a macro WIFEXITED determinar a partir de status de saída de um processo

filho se essa processo terminado normalmente (através da função de saída ou o retorno do principal) ou morreu de uma não processada sinal. Neste último caso, utilizar a macro WTERMSIG para extrair a partir do seu estado de saída do sinal número pelo qual morreu. Abaixo é a função principal do garfo e exemplo exec novamente. Neste momento, as chamadas processo pai esperar para esperar até que o processo de criança, em que o comando ls executa, está terminado.

```
int main ()
 {
int child_status;
/ * A lista de argumentos para passar para o comando "ls". * /
char * arg_list [] = {
"Ls", / * argv [0], o nome do programa. * /
"-l",
"/",
NULL * A lista / argumento deve terminar com um NULL. * /
};
/ * Gerar um processo filho de executar o comando "ls". ignorar o
retornado ID processo filho. * /
spawn ( "ls", arg_list);
/ * Aguarde até que o processo filho para ser concluído. * /
wait (&child_status);
if(WIFEXITED (child_status))
printf ( "o processo filho saiu normalmente, com o código de saída%
d \ n",
WEXITSTATUS (child_status));
else
printf ( "o processo filho terminou de forma anormal \ n");
return 0;
}
```

Várias chamadas de sistema similar estão disponíveis no Linux, que são mais flexíveis ou fornecer mais informações sobre o processo filho sair. A função waitpid pode ser usado para esperar por uma processo filho específico para sair em vez de qualquer processo filho. A função wait3 retorna uso da CPU estatísticas sobre o processo filho sair, ea função wait4 permite especificar adicional opções sobre quais processos para esperar.

2 . Processos Zombie

Se um processo filho termina enquanto seu pai está chamando uma função de espera, o processo filho desaparece e seu status de terminação é passado para seu pai através da chamada espera. Mas o que acontece quando uma criança

processo termina e o pai não está chamando esperar? Será que simplesmente desaparecem? Não, porque então informações sobre o seu término, tais como se ele saiu normalmente e, em caso afirmativo, qual a sua saída estado é-se perderia. Em vez disso, quando um processo filho termina, torna-se um processo zumbi.

Um processo zumbi é um processo que foi finalizado, mas não foi feita ainda. É o

responsabilidade do processo pai para limpar suas crianças zumbi. As funções de espera fazer isso, também, por isso não é necessário controlar se o seu processo de processo filhoainda está em execução antes de aguardar isto. Suponha, por exemplo, que um programa bifurca um processo filho, executa alguma outra cálculos e, em seguida, chamadas de esperar. Se o processo de processo filhonão tenha terminado naquele momento, o pai processo irá bloquear na chamada espera até que o processo filho terminar. Se o processo filho termina antes as chamadas processo pai espera, o processo de processo filhose torna um zumbi. Quando o pai chamadas de processo esperar , o zombie status de terminação da processo filhoé extraído, o processo filho é excluído e a chamada espera retorna imediatamente.

O que acontece se o pai não limpar os seus filhos? Eles ficam em torno do sistema, zumbis processos. O “programa zombie.c ” fornecido abaixo garfos um processo filho, que termina imediatamente e depois vai dormir por um minuto, sem nunca limpar a processo filho.

//ProgramA zombie.c - fazendo um processo Zombie

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main ()
{
   pid_t child_pid;
 /* Create a child process. */
   child_pid = fork ();
    if (child_pid > 0) {
    /* This is the parent process. Sleep for a minute. */
    sleep (60);
   }
   else {
```

```
            /* This is the child process. Exit immediately. */
      exit (0);
       }
        return 0;
   }
```

Tente compilar este arquivo para executavel chamando o programa acima. Executá-lo, e enquanto ele ainda está em execução, listar os processos no sistema, invocando o comando a seguir em outra janela.

```
            $ ps -e -o pid,ppid,stat,cmd
```

Esta lista o ID do processo, pai ID do processo, estado do processo, e linha de comando processo. Observa que, para além do processo de make-zombie-mãe, há um outro processo de make-zombie listados.

É o processo de criança; note que o seu ID do processo pai é o ID do processo do principal make-zombie processo. O processo filho é marcado como <extinta>, e seu código de status é Z , por zumbis.

O que acontece quando o programa make-zombie principal termina quando o processo pai sai, sem nunca chamar espera ? O processo zumbi ficar em torno de? No-tente executar ps novamente e observe que ambos os processos make-zumbis sumiram. Quando um programa sai, seus filhos são herdadas por um processo especial, a inicialização do programa, que sempre é executado com o ID de processo de 1 (que é o primeiro processo começou quando do arranque do Linux) .O inicialização processo limpa automaticamente quaisquer processos filho zombie que herda.

3. Limpando os processos Filhos de forma assíncrona

Se você estiver usando um processo filho simplesmente para exec outro programa, não há problema em chamar esperar imediatamente no processo pai, que irá bloquear até que o processo de processo filhocompleta. Mas, muitas vezes, você vai querer o processo pai para continuar em execução, como um ou mais filhos executar de forma síncrona. Como pode você ter certeza que você limpar processos filhos que tenham concluído para que você não deixe zombie processos, que consomem recursos do sistema, em torno de mentir?

Uma abordagem seria para o processo pai para chamar wait3 ou wait4 periodicamente, para limpar crianças zumbi. Chamada espera para este fim não funciona bem porque, se há crianças têm rescindido, a chamada será bloqueado até que a pessoa faz. No entanto, wait3 e wait4 tomar um sinalizador adicional

parâmetro, para o qual você pode passar o valor da bandeira WNOHANG . Com este sinalizador, a função é executado em de não bloqueio modo -é irá limpar um processo filho termina se houver um, ou simplesmente voltar se não há. O valor de retorno da chamada é o ID do processo da processo filhoencerrado na antiga caso, ou zero, no último caso.

Uma solução mais elegante é para notificar o processo pai quando uma processo filhotermina. Existem vários maneiras de fazer isso usando os métodos de " Comunicação entre ", mas felizmente Linux faz isso por você, usando sinais. Quando um processo filho termina, Linux envia o processo pai do SIGCHLD sinal. A disposição padrão deste sinal é não fazer nada, razão pela qual você pode

não tenha notado antes. Assim, uma maneira fácil de limpar processos filho é pela manipulação SIGCHLD . É claro que, ao limpar o processo filho, é importante para armazenar a sua cessação estado se essa informação for necessária, porque uma vez que o processo é limpo usando espera, que informação não está mais disponível. O programa "sigchld.c" fornecido abaixo é o que parece para um programa para usar um SIGCHLD manipulador para limpar os processos filho.

// Programa "sigchld.c" - Limpando crianças por Handling SIGCHLD

```
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
sig_atomic_t child_exit_status;
void clean_up_child_process (int signal_number)
{
   /* Clean up the child process. */
   int status;
   wait (&status);
    /* Store its exit status in a global variable. */
    child_exit_status = status;
}
int main ()
{
    /* Handle SIGCHLD by calling clean_up_child_process. */
    struct sigaction sigchld_action;
    memset (&sigchld_action, 0, sizeof (sigchld_action));
    sigchld_action.sa_handler = &clean_up_child_process;
    sigaction (SIGCHLD, &sigchld_action, NULL);
```

```
        /* Now do things, including forking a child process. */

         /* ... */

      return 0;

   }
```

Note como o manipulador de sinal armazena estado de saída do processo filho em uma variável global, a partir do qual o principal programa pode acessá-lo. Porque a variável é atribuído em um manipulador de sinal, seu tipo é sig_atomic_t.

Conclusão

Nesta atividade foram apresentados os conceitos de processo e explicou várias técnicas e ferramentas para criar e trabalhar com processos. Em particular, a criação do processo utilizando o sistema , da forquilha e exec funções foi apresentado. Além disso, a gestão e manipulação dos processos usando

vários sinais do sistema foi destacada.

Avaliação

- Pratique o código de exemplo fornecido nesta actividade.

Atividade 1.2 - Threads

Introdução

Temos visto como um programa pode desembolsar um processo filho. O processo filho é inicialmente executando o seu programa do pai, com a memória virtual do seu pai, o arquivo descritores, e assim por diante copiado processo filho que pode modificar a sua memória, descritores de arquivos mais próximos, e assim por diante, sem afetar seu pai, e vice versa. Quando um programa cria outro thread, no entanto, nada é copiado. a criação eo thread criado compartilham o mesmo espaço de memória, descritores de arquivos, e outros recursos do sistema como o original. Se um thread muda o valor de uma variável, por exemplo, o outro thread posteriormente, verá o valor modificado. Da mesma forma, se uma thread fecha um descritor de arquivo, outra threads não podem ler ou escrever para que descritor de arquivo.

Cada thread em um processo é identificado por um ID de thread. Ao se referir a enfiar IDs em C ou C ++ programas, use o tipo pthread_t . Durante a criação, cada thread executa uma função de thread. Isto é apenas uma função comum e contém o código que o thread deve ser executado. Quando a função retornam , as saídas de thread. No GNU / Linux, as funções de thread tomar um único parâmetro, do tipo void *, e ter um tipo de retorno void *. O parâmetro é o argumento de thread: GNU / Linux passa o valor juntamente com o thread sem olhar para ele. Seu programa pode usar esse parâmetro para passar dados para um novo thread. Da mesma forma, o programa pode usar o valor de retorno para transmitir dados a partir de uma lista de thread de sair de volta para seu criador.

O pthread_create função cria um novo thread. Você fornecê-lo com o seguinte:

I. Um ponteiro para uma pthread_t variável, em que o identificador do thread do novo thread é armazenado.

II. Um ponteiro para um objeto de atributo de thread. Esse objeto controla detalhes de como o thread interage com o resto do programa. Se você passar NULL como o atributo da linha, um thread será criado com os atributos da linha padrão. atributos de thread são apresentados na tarde seção, "Thread Atributos."

III. Um ponteiro para a função de thread. Este é um ponteiro de função normal, deste tipo:

void * (*) (void *)

IV. Um valor argumento do tipo void *. O que quer que você passa é simplesmente passado como o argumento para a função thread quando o thread começa a executar.

Uma chamada para pthread_create retorna imediatamente, e o thread original continua executando o instruções a seguir a chamada. Enquanto isso, o novo thread começa a executar a função de thread.

Schedules Linux ambos os thread de forma assíncrona, e seu programa não deve contar com o parente ordem em que as instruções são executadas nos dois threads.

O programa " thread-create.c " fornecido abaixo cria um thread que imprime x do continuamente para erro padrão. Depois de chamar pthread_create , a principal impressão de thread de o continuamente a norma erro.

// Programa "thread-create.c" - criar um thread

```
#include <pthread.h>

#include <stdio.h>

/* Prints x's to stderr. The parameter is unused. Does not return.
*/

void* print_xs (void* unused)

{

while (1)

fputc ('x', stderr);

return NULL;

}

/* The main program. */

int main ()

{
```

```
pthread_t thread_id;

/* Create a new thread. The new thread will run the print_xs

function. */

pthread_create (&thread_id, NULL, &print_xs, NULL);

/* Print o's continuously to stderr. */

while (1)

fputc ('o', stderr);

return 0;

}
```

Compilar e vincular este programa usando o seguinte código:

```
$ Cc -o thread-criar thread-create.c -lpthread
```

Tente executá-lo para ver o que acontece. Observe o padrão imprevisível de xeo de como Linux horários alternadamente os dois threads. Sob circunstâncias normais, um thread sai em um dos dois maneiras. Uma maneira, como ilustrado anteriormente, é de retornar da função de thread. O retorno valor da função thread é considerado como sendo o valor de retorno do thread. Alternativamente, um thread pode sair explicitamente chamando pthread_exit . Esta função pode ser chamado de dentro do thread função ou de alguma outra função chamada direta ou indiretamente pela função de thread. O argumento para pthread_exit é valor de retorno do thread.

Detalhes Atividade

Passando dados para Threads

O argumento thread proporciona um método conveniente de passar os dados para threads. Por causa do tipo do argumento é void *, porém, você não pode passar uma grande quantidade de dados diretamente através do argumento. Ao invés, use o argumento da linha para passar um ponteiro para uma estrutura ou matriz de dados. Uma utilizada técnica é para definir uma estrutura para cada função de thread, que contém os " parâmetros " que a função thread espera.

Usando o argumento de thread, é fácil de reutilizar a mesma função thread para muitos threads. Todosestes threads executar o mesmo código, mas em dados diferentes. O programa " thread-create2 " dado a seguir é semelhante ao do exemplo anterior. Este cria duas novas linhas, uma para impressão x do e o outro para imprimir o da . Em vez de imprimir infinitamente, no entanto, cada thread imprime um número fixo de caracteres e depois sai, retornando a partir da função de thread. A mesma função thread, char_print , é usado por ambos os threads, mas cada um está configurado de maneira diferente usando struct char_ print_parms .

// Programa "thread-create2" - Criar dois threads

```
#include <pthread.h>
#include <stdio.h>
/* Parameters to print_function. */
struct char_print_parms
{
/* The character to print. */
char character;
/* The number of times to print it. */
int count;
};
/* Prints a number of characters to stderr, as given by PARAMETERS,
which is a pointer to a struct char_print_parms. */
void* char_print (void* parameters)
{
/* Cast the cookie pointer to the right type. */
struct char_print_parms* p = (struct char_print_parms*)
parameters;
int i;
for (i = 0; i < p->count; ++i)
fputc (p->character, stderr);
return NULL;
}
/* The main program. */
int main ()
{
pthread_t thread1_id;
pthread_t thread2_id;
struct char_print_parms thread1_args;
```

```
struct char_print_parms thread2_args;

/* Create a new thread to print 30,000 'x's. */

thread1_args.character = 'x';

thread1_args.count = 30000;

pthread_create (&thread1_id, NULL, &char_print, &thread1_args);

/* Create a new thread to print 20,000 o's. */

thread2_args.character = 'o';

thread2_args.count = 20000;

pthread_create (&thread2_id, NULL, &char_print, &thread2_args);

return 0;

}
```

Mas espere! O programa " thread-create2" tem um bug sério nele. O thread principal (que executa ofunção principal) cria as estruturas de parâmetros thread (thread1_args e thread2_ args) como locais variáveis e, em seguida passa ponteiros para essas estruturas para os threads que ele cria. O que pode impedir Linux a partir de agendamento das três thread de tal forma que os principais termina a execução antes de qualquer um dos outros dois threads são feitas? Nada! Mas, se isso acontece, a memória contendo a estruturas de parâmetros thread será desalocada enquanto os outros dois threads ainda estão acessando-o !!

Juntando as Threads

Uma solução é a força principal que esperar até que os outros dois threads são feitas. O que nós precisamos é de uma função semelhante de esperar que aguarda uma thread para terminar, em vez de um processo. Essa função é pthread_join , que leva dois argumentos: o ID de thread do thread para esperar, e um ponteiro para um void * variável que irá receber o valor de retorno do thread acabado. Se você não se preocupam com o valor de retorno da linha, passar NULL como o segundo argumento.

A revista principal programa fornecido abaixo mostra a função principal corrigido para o buggy exemplo de programa " thread-create2.c" . Nesta versão, a principal não sair até que ambos os thread imprimindo x e o quando tem concluído, para que eles não estão mais usando as estruturas de argumento.

// Função principal revista para o programa "thread-create2.c"

```
  int main ()

{

pthread_t thread1_id;

pthread_t thread2_id;
```

```
struct char_print_parms thread1_args;

struct char_print_parms thread2_args;

/* Create a new thread to print 30,000 x's. */

thread1_args.character = 'x';

thread1_args.count = 30000;

pthread_create (&thread1_id, NULL, &char_print, &thread1_args);

/* Create a new thread to print 20,000 o's. */

thread2_args.character = 'o';

thread2_args.count = 20000;

pthread_create (&thread2_id, NULL, &char_print, &thread2_args);

/* Make sure the first thread has finished. */

pthread_join (thread1_id, NULL);

/* Make sure the second thread has finished. */

pthread_join (thread2_id, NULL);

/* Now we can safely return. */

return 0;

}
```

A moral da história: Certifique-se de que todos os dados que você passar para um thread por referência não é desalocada, mesmo por um thread diferente, até ter certeza de que o thread é feito com ele. Isto é verdade tanto para as variáveis locais, que são desalocadas quando estes estiverem fora do escopo, e para heap-variáveis atribuídas, que se desalocar quando for chamando free (ou usando delete em C ++).

Valores de retorno de threads

Se o segundo argumento que você passa para pthread_join não é nulo, o valor de retorno do thread será colocado no local apontado por esse argumento. O valor de retorno da thread, como o argumento, da thread, é do tipo void * . Se você quer passar para trás um único int número pequeno ou outra, você pode fazer isso facilmente, lançando o valor para void * e, em seguida, lançando de volta para o tipo apropriado depois chamando pthread_join .

O programa "primes.c " dada a seguir calcula o n primo número em um thread separado quethread retorna o número primo desejado como seu valor de retorno de thread. O thread principal, entretanto, é livre para executar outro código. Note-se que o algoritmo de divisão sucessiva utilizado em compute_prime função é bastante ineficiente; consultar um livro sobre algoritmos numéricos, se você precisa para calcular muitos números primos em seus programas.

// Programa "primes.c" - Números Compute Prime em um thread

```
#include <pthread.h>
#include <stdio.h>
/* Compute successive prime numbers (very inefficiently). Return the
Nth prime number, where N is the value pointed to by *ARG. */
void* compute_prime (void* arg)
{
int candidate = 2;
int n = *((int*) arg);
while (1) {
int factor;
int is_prime = 1;
/* Test primality by successive division. */
for (factor = 2; factor < candidate; ++factor)
if (candidate % factor == 0) {
is_prime = 0;
break;
}
/* Is this the prime number we're looking for? */
if (is_prime) {
if (--n == 0)
/* Return the desired prime number as the thread return value. */
return (void*) candidate;
}
++candidate;
}
return NULL;
}
int main ()
{
```

```
pthread_t thread;

int which_prime = 5000;

int prime;

/* Start the computing thread, up to the 5,000th prime number. */

pthread_create (&thread, NULL, &compute_prime, &which_prime);

/* Do some other work here... */

/* Wait for the prime number thread to complete, and get the
result. */

pthread_join (thread, (void*) &prime);

/* Print the largest prime it computed. */

printf("The %dth prime number is %d.\n", which_prime, prime);

return 0;

       }
```

Mais Ids em um thead

Ocasionalmente, é útil para uma sequência de código de thread para determinar qual é executá-la. O thread_self função retorna o thread ID da thread em que ele é chamado. Este ID thread pode ser comparado com outro ID thread usando o pthread_equal função. Essas funções podem ser útil para determinar se um determinado ID thread corresponde ao thread atual. Para exemplo, é um erro para um thread para chamar pthread_join para se juntar a si. (Neste caso, pthread_join retornaria o código de erro EDEADLK .) Para verificar isso de antemão, pode-se usar um código como a seguir:

if (! pthread_equal (pthread_self (), other_thread))

pthread_join (other_thread, NULL);

Atributos de uma threads

Atributos de thread fornecem um mecanismo para afinar o comportamento de threads individuais. Relembre-se que pthread_create aceita um argumento que é um ponteiro para um objeto de atributo de thread. Se passarmos um ponteiro nulo, os atributos da thread padrão são usados para configurar o novo thread. No entanto, pode-se criar e personalizar um atributo de objeto de thread para especificar outros valores para os atributos.

Para especificar atributos de thread personalizadas, você deve seguir estes passos:

I. Criar um objeto pthread_attr_t. A maneira mais fácil é simplesmente para declarar uma variável automática deste tipo.

II. Chamada pthread_attr_init , passando um ponteiro para este objeto. Isso inicializa os atributos para seus valores padrão.

III. Modificar o objeto de atributo para conter os valores de atributos desejados.

IV. Passar um ponteiro para o objeto de atributo ao chamar pthread_create .

v. Chamada pthread_attr_destroy para liberar o objeto de atributo. O pthread_attr_t variável em si não é desalocada; ele pode ser reinicializado com pthread_attr_init .

Um único objecto atributo thread pode ser utilizado para iniciar vários thread. Não é necessário para manter o thread atributo de objeto em torno após os threads foram criados. Para a maioria GNU / Linux tarefas de programação de aplicativos, apenas um atributo da linha é tipicamente de interesse (o outro atributos disponíveis são principalmente para especialidade em tempo real de programação). Este atributo é o thread de destacar estado. Um thread pode ser criado como um thread acopláveis (o padrão) ou como uma destacada thread. Uma thread acopláveis, como um processo, não é limpo automaticamente pelo GNU / Linux quando se termina. Em vez disso, o estado de saída do thread trava em torno do sistema (como uma espécie de processo zombie) até que outro thread chama pthread_join para obter seu valor de retorno. Só então são a sua recursos liberados. Um thread individual, em contraste, é limpa automaticamente quando termina. Porque um thread individual é imediatamente limpa, outro thread não pode sincronizar a sua conclusão usando pthread_join ou obter seu valor de retorno.

Para definir o estado de desconexão em um objeto de atributo thread, use pthread_attr_ setdetachstate . O primeiro argumento é um ponteiro para o objeto de atributo da linha, eo segundo é o estado de desconexão desejado.

Porque o estado acopláveis é o padrão, é necessário chamar isto só para criar isolada threads; passar PTHREAD_CREATE_DETACHED como o segundo argumento. O programa" Detached.c " dada a seguir cria um thread individual, definindo o atributo thread estado desanexar para a thread.

// Programa "detached.c" - Programa de esqueleto que cria um thread separada

```
#include <pthread.h>

void* thread_function (void* thread_arg)

{

/* Do work here... */

}

int main ()

{

pthread_attr_t attr;

pthread_t thread;

pthread_attr_init (&attr);

pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
```

```
pthread_create (&thread, &attr, &thread_function, NULL);

pthread_attr_destroy (&attr);

/* Do work here... */

/* No need to join the second thread. */

return 0;

}
```

Mesmo que um thread é criado em um estado acopláveis, pode depois ser transformado em um thread individual. Façam isso, chamar pthread_detach . Uma vez que uma linha é individual, não pode ser feita acopláveis novamente.

Conclusão

Nesta atividade foram apresentados os conceitos de thread e explicou várias técnicas e ferramentas para criar e gerenciar a execução de thread. Em particular, a criação de thread usando pthread_create função foi apresentado. Além disso, técnicas de gestão de vários thread foram destacados.

Avaliação

- Pratique o código de exemplo fornecido nesta actividade

Atividade 1.3 - Gerenciamento de memória

Introdução

A memória é a área de armazenamento de dados primários para computadores. Chamamos a unidade de memória básica um pouco . Um bit pode conter dois valores diferentes: 0 ou 1. Por que os computadores usam aritmética binária? Isto é a maneira mais confiável e eficiente para expressar dados. Porque, a informação digital pode ser armazenada como tensão. Temos que distinguir cada valor diferente. Se tivermos mais valores (como decimal sistema), que faz com que menos de separação entre valores adjacentes. Mas com dois valores (sistema binário: 0 e 1) valores diferentes serão distinguidos com distância máxima. Como uma regra: Para obter mais distância devemos usar dois pontos. A distância será a distância da regra. Memória consiste de um número de células que pode armazenar um número de bits. A memória é apenas um array de bytes. Cada célula tem um número para identificá-lo, chamou o seu endereço . Programas para referir endereços alcançar memória. Células adjacentes têm endereços consecutivos. Se a memória tem m células, as células tem endereços de 0 a M-1 . Se CPU suporta n bit, pode referir endereços 0 para 2 ^ n - 1 . Por exemplo, Intel Pentium II é um processador de 32 bits e pode endereçar 4 GBytes de memória. Cada célula armazena um inteiro. Um inteiro é o número de n bits. É de 32 bits (4 bytes) se você tiver Intel Pentium II. Portanto, o tamanho máximo de memória endereçável = n * 2 ^ n. Nos últimos anos, quase todos os fabricantes têm padronizado sobre uma célula de 8 bits, que é chamado byte Os bytes são agrupados em palavras. Uma CPU de 32 bits tem 4 bytes / palavra. A CPU com registros de 32 bits pode detém 32-bits de cada vez.

Por essa razão, os registos que o acesso à memória também são de 32 bits. Por isso, pode apontar máxima ao 11111111111111111111111111111111 em formato binário (0xFFFFFFFF em forma HEXAL). Isso é mesmo com a 2 ^ n - 1.

Nos primeiros anos, memórias de computador eram pequenos e mais caro. Os programadores estavam usando um tamanho de memória total de apenas 4096 palavras de 18 bits para ambos os programas do usuário e sistema operacional em PDP-1. Assim, o programador tinha que encaixar seu programa nesta pequena memória. Atualmente, os computadores ter alguns gigabytes de memória, mas os programas modernos precisam de muito mais memória. Resolver este problema, os sistemas operacionais utilizam memórias secundárias, tais como disco como memória principal. No primeira técnica, o programador dividiu o programa em um número de pedaços chamados sobreposições .

No início do programa, primeira sobreposição foi carregado para a memória. Quando terminou, cargas próximos sobreposição. Os programadores devem gerenciar sobreposições entre memória e disco. programador foi responsável para encontrá-lo a partir do disco e carregá-lo para a memória. Esta foi uma tarefa difícil para programadores.

Em 1961, um grupo de pesquisadores de Manchester estabeleceu gestão automática de sobreposição sistema chamado de memória virtual . A memória virtual é organizado em " páginas ". Uma página é uma memória unidade tipicamente alguns Kbytes de tamanho. É, sobretudo, de 4 Kbytes. Você pode aprender tamanho da página, digitando PAGESIZE comando no Linux / Unix. Quando um programa referências a um endereço de uma página não presente na memória principal, uma falha de página ocorre. Após uma falha de página, o sistema operacional procura para a página correspondente no disco e carrega-lo para a memória principal usando uma substituição de página algoritmo tais como LRU (Least Recently Used (LRU) - trabalha na ideia de que as páginas que têm foram mais utilizados nos últimos instruções são mais susceptíveis de ser muito usado na próxima algumas instruções também).

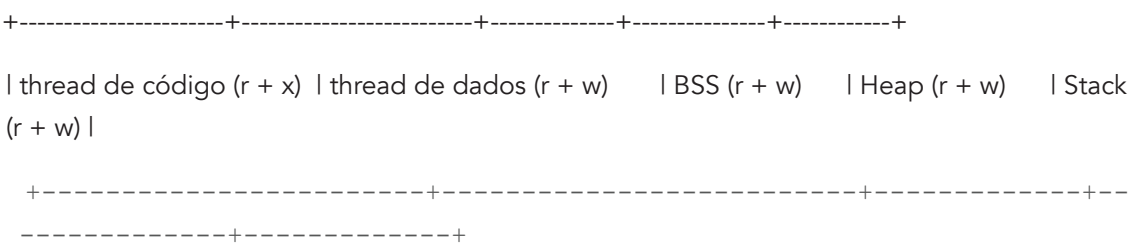
Dessa forma, o programador pode iniciar um programa quando nenhum do programa é na memória principal. Quando a CPU tenta buscar a primeira instrução do programa, ele recebe uma falha de página, porque o memória não contém qualquer parte do programa na memória principal. Este método é chamado (paginação por demanda) demand paging . Se um processo na memória principal tem baixa prioridade ou está dormindo, isso significa que ele não vai executado em breve. Neste caso, o processo pode ser feito backup em disco pelo sistema operacional. Este processo é trocado . O espaço de troca é usada para manter os dados da memória. Processos de usar o Virtual endereços para a transparência. Eles não sabem sobre a memória física. CPU tem uma unidade chamada Unidade de Memória Management (MMU) , que é responsável pela exploração de memória virtual. Quando um processo faz uma referência a uma página que não está na memória principal, a MMU gera uma página falha. O kernel pega e decide se a referência é válida ou não. Se inválido, o kernel envia um sinal de "violação de segmentação" para o processo. Se válido, o kernel recupera a página do processo referenciado a partir do disco.

detalhes Atividade

Layout de memória para um processo

A memória é um conjunto de palavras. Mas não é funcional com esta estrutura simples. Operativo sistemas de dividi-lo em algumas peças cada um tem o seu comportamento personalizado. Por exemplo, o núcleo pode proteger uma parte da memória do processo contra gravação e execução. Na memória do processo, cada secção de memória que tem comportamentos diferentes é chamado de thread .

Quando um programa é carregado na memória, que reside na memória como se mostra pela seguinte layout de memória:



A. Segmento do Código (Texto)

Esta é a área em que as instruções executáveis residem. No mundo Linux / Unix, ele é chamado como " thread de texto ". Ele tem uma permissão de execução. Alguns arquitetura antiga permitida a alteração de código si. Por esse motivo, o thread de código também tinha a permissão de escrita ".

Suponha que temos uma função func chamado () e endereço pontos em algum lugar no thread de código.Porque as funções residem no thread de código, portanto, addr = & func;

B Segmento de dados

Ele contém variáveis globais inicializados declarados pelo programador. As variáveis globais têm um fixo área na memória onde serão definidas na inicialização.

C. BSS

Ele contém variáveis globais não inicializadas.

D. Heap

Ele contém variáveis geradas dinamicamente em tempo de execução. thread de dados contém variáveis que nós criamos em tempo de compilação. Por isso, é de tamanho fixo. Muitas vezes, o programador precisa para criar variáveis em tempo de execução. Isso é chamado de alocação de memória dinâmica . Bibliotecas C modernos fornecem algumas funções para alocar a área de heap como malloc (). A função free () destrói variáveis, que são alocada dinamicamente a partir do espaço de pilha. Tudo é sem nome na pilha. Você não pode chegar a qualquer variável diretamente usando seu nome. Mas você pode fazer referência indiretamente usando um ponteiro. A extremidade da pilha é marcada por um ponteiro chamado de "break". Quando uma referência programa além da fratura, ele vai quebrar. Quando o Gestor de pilha precisa de mais memória, ele chama brk () e chamadas

do sistema sbrk (). O brk () e sbrk () funções são usado para alterar a quantidade de memória alocada no thread de dados do processo. Eles fazem isso, movendo a localização do `break '. A ruptura é o primeiro endereço pós o final do processo de thread de dados não inicializada (também conhecido como o BSS ` ').

NAME brk, sbrk		- change data segment size
LIBRARY Standard C Library (libc, -lc)		
SYNOPSIS #include <sys/types.h>		
#include <unistd.h>		
int brk(const void *addr); void * sbrk(intptr_t incr);		

A função brk () define a pausa para addr. A função sbrk () aumenta o intervalo de bytes Incr, assim, alocação de pelo menos incr bytes de memória nova no thread de dados. Se incr é negativo, a quebra é reduzido por bytes Incr. O valor actual da interrupção pode ser determinado ligando sbrk (0).

5. Stack

Stack é uma estrutura de dados que é acessado no último a entrar primeiro a sair. Existem duas operações para pilhas: Para inserir um novo produto, que deve ser IMPULSO Ed e para recuperar produto, que deve ser POP PED. SP (ponteiro de pilha) é um registo de CPU que aponta para a parte superior da pilha.

Os dados de endereços reside mais elevadas para diminuir endereços na pilha. A chamada de função é o típicoutilização da pilha. O que acontece depois de uma chamada de função? Considere o exemplo programa abaixo:

```
10] i = 4;

11] func (i);

12], k = 2;

13] …
```

Note-se que CPU sempre executa instrução que IP (ponteiro de instrução) mostra. No código acima,IP é 10 antes de chamar func (). Ele será o endereço de func () quando func () é chamado. E, em seguida, IPserá 12 depois func () saiu. Antes func call (), devemos salvar o endereço da próxima linha de código.

Porque nós vamos voltar após a função saiu e continuar a executar o programa a partir da próxima linha.

Para armazenar o endereço da próxima linha, o programa usará pilha. Após esta etapa o programa empurra parâmetros de função (no código acima, 'i' é o parâmetro) empilhar, e variáveis, em seguida, locais da função. Quando a função termina, ele aparece variáveis locais e parâmetros de função aida na stack. Assim, somente o endereço da próxima linha (no código acima, é 12) permanece na pilha. O return é uma CPU instrução específica. Após o retorno, o endereço será exibido da pilha e será atribuído a IP. Daí o programa irá continuar a execução da linha 12.

Vazamento de memória (memory leak)

Um vazamento de memória é o lugar onde a memória alocada não é liberada, embora ela nunca é usada novamente. Há dois tipos comuns de problemas de heap:

I. Libertar ou substituição de dados que ainda está em uso fará com que " a corrupção da memória (memory corruption)".

II. Não libertar os dados que já não está em uso irá causar " vazamento de memória ".

Se o vazamento de memória está em um loop, depois de um tempo o programa vai consumir toda a memória. Vai-se ver que, o seu sistema operativo está ficando mais lento. Um bom programador sempre libera alocados memória explicitamente. Sempre que ele usa malloc (), coloca uma declaração correspondente free (). Lixo coleção (GC), também conhecido como gerenciamento automático de memória, é a reciclagem automática de memória alocada dinamicamente. A coleta de lixo é realizada por um coletor de lixo que libera memória que nunca vai ser usado novamente. Há muitas maneiras para a memória automática gerentes para determinar o que a memória não é mais necessário. Em geral, a coleta de lixo depende na determinação de quais blocos não são apontada por todas as variáveis do programa.

A coleta de lixo foi inventado por John McCarthy em 1958 como parte da implementação de Lisp. Sistemas e linguagens que usam a coleta de lixo pode ser descrito como garbage-colector (coletor de lixo) . Java, Prolog, Smalltalk etc, são lixo coletado línguas. C fornece mais controle sobre o programa de programador. Por essa razão, não se preocupa com a liberação de memória não utilizado. Todos variáveis locais na pilha vai ser libertado e disponível para reutilização depois de sair do seu âmbito. Mas variáveis atribuídas dinâmicas não serão libertados sem um coletor de lixo. Desde C não costuma realizar coleta de lixo, os programadores devem ter cuidado se eles usam malloc ().

Por que precisamos de alocação dinâmica de memória? Todas as variáveis declaradas estaticamente no tempo de compilação na pilha, será destruída enquanto as funções estão saindo (quando a função principal do programa será sair). Mas às vezes o programador não pode saber quanto espaço o programa precisa. Por exemplo, o programa lê palavras de spam a partir de um arquivo e colocá-los para a memória.

Pode haver linhas 10 ou 1000 linhas. Se assumirmos uma linha pode ser no máximo de 32 bytes, em seguida, no primeiro caso, precisamos de 320 bytes de memória e, no segundo caso, precisamos de 32.000 bytes de memória. Como resultado, o programador não sabe sobre os requisitos de memória de antemão.

Como solução, ele pode alocar 1000 linhas de texto estaticamente. Mas se tivermos 100 palavras, isso vai desperdiçar nossa memória. Ou se o banco de dados de spam é tão grande (por exemplo, 10.000 linhas) o programa não irá funcionar corretamente.

```
char wordtable[1000][32];
```

A melhor solução é usar memória dinâmica. O programa aloca 32 bytes para cada linha em um loop.

NAME

malloc, calloc, realloc, livre, reallocf - Funções de Alocação de memória de uso geral

BIBLIOTECA

Padrão C Biblioteca (libc, lc)

SINOPSE

```
#include <stdlib.h>

void * malloc (size size_t);

void free (* ptr void);
```

A função malloc () aloca tamanho bytes de memória. A função free () faz com que o alocado memória referenciado por ptr a ser disponibilizado para alocações futuras.

O código a seguir ilustra um erro DE programação básico de vazamento d memória:

```
01]

02] int main ()

03] {

04] char * str;

05] char * tmp;

06]

07] str = malloc (sizeof (char) * 32);

08] tmp = malloc (sizeof (char) * 32);

09] fscanf (stdin, "% s", str);

10] tmp = str;

11]
```

```
12] do_something_with_tmp ();

13] do_something_with_str ();

14] free(str);

15] free (tmp);

16]

17] return 0;

18]}

19]
```

Na linha 10, o programador está usando tmp ponteiro para preservar endereço do str ponteiro. Então realizar alguns processos com tmp . E em seguida, usando str . Na linha 14 e 15, ele está liberando os ponteiros como um bom programador. Mas ele se esquece de sua atribuição (tmp = str). Linha 14 terá sucesso, mas a linha 15 falhará se tmp ainda aponta na str . Porque str não pode ser desalocados novamente. Outro ponto é que, o endereço programador perdida de tmp alocados na linha 8. Assim, ele nunca libera tmp .

Ponteiros

Um ponteiro é um grupo de células (geralmente dois ou quatro) que podem conter um endereço. Se ch é um char e p é um ponteiro para esse caractere:

p & ch

+ ----- + ------ + ------ + ----- + ----- + ------- + ----- +

| ... | ... | &ch | ..... | ..... | char | ..... |.....

+ ------ + ----- + ------ + ----- + ----- + ------- + ----- +

O operador unário & dá o endereço de um objeto. O indirection ou dereference operador * dá o "conteúdo de um objeto apontado por um ponteiro ''. Para declarar um ponteiro usamos dereferencoperador.

Considere o "ptr.c" programa abaixo que demonstra o uso de ponteiros:

// Ptr.c Programa - O uso de ponteiros

```
1 #include <stdio.h>

2 #include <stdlib.h>

3

4 int main ()

5 {

6 char ch1 = 'g';
```

```
7 char CH2 = 's';
8 char * ptr;
9
10 printf ( "Char 1 é% c \ n", CH1);
11 printf ( "Char 2 é% c \ n", CH2);
12
13 printf ( "Endereço de Char 1 é% p \ n", e CH1);
14 printf ( "Endereço de Char 2 é% p \ n", e CH2);
15
16 ptr = & CH1;
17 CH2 = * ptr;
18
19 printf ( "Char 1 é% c \ n", CH1);
20 printf ( "Char 2 é% c \ n", CH2);
21
22 printf ( "Endereço do ptr é% p \ n", ptr);
23
24 return 0;
25}
```

Nós declaramos duas variáveis caractere em linha 6 e 7. Em seguida, declarou um ponteiro de char em linha 8. ptr é um variável que contém o endereço da ch1 após a linha 16. CH2 é atribuído ao valor "reside no endereço ptr pontos "na linha 17. O endereço PTR pontos para tratar de ch1 por causa da linha 16. Então, estes dois

linhas são iguais para a linha seguinte:

```
CH2 = CH1;
```

Nós podemos fazer operações aritméticas em ponteiros.

ptr ++; / * Pontos próximo endereço. * /

(* Ptr) ++; / * Incrementa o que PTR aponta. * /

ptr ++ incrementos ptr para apontar próximo objeto. Ele não incrementa ptr por 1 byte. Valor adicionado depende do tamanho do objecto. Gerenciamento de variáveis criados dinamicamente requer o uso de ponteiros. Quando o programador precisa de memória em tempo de execução para armazenar uma estrutura de dados, ele exige usando a função malloc ().

void * malloc (size size_t);

A função malloc () aloca size bytes de memória e retorna um ponteiro para o espaço alocado.

typedef struct rulelist rulelist;

```
struct rulelist {
int attr;
char ruleline[256];
rulelist *next;
};
rulelist *ll;
if((ll = (rulelist *) malloc(sizeof(rulelist))) == NULL) return -1;
```

Desde C passa argumentos para funções por valor, não há nenhuma maneira direta para a função chamado para alterar uma variável na função de chamada, se não é uma variável global. Todas as variáveis locais são acessíveis localmente. Se você passá-lo para funcionar e, em seguida, alterar o seu valor, não efeito sobre a variável de chamar a função.

```
  void swap(int x, int y) /* WRONG */
{
int temp;
temp = x;
x = y;
y = temp;
}
swap(a, b);

Mas endereços são acessíveis de qualquer lugar.

void swap (int * px, int * py) / * * intercâmbio px e py * * /
{
int temp;
temp = * px;
* Px = * py;
```

```
* Py = temperatura;

}

swap (&a, &b);
```

Argumentos de ponteiro ativam uma função para acessar e alterar objetos na função que o chamou, subtração de ponteiro também é válido: se p e o ponto q para elementos da mesma matriz, e p <q, então q-p + 1 é o número de elementos de P a Q, inclusive. Este fatos podem ser usado para escrever mais um versão do strlen :

```
int strlen (char * s)

    {

char * p = s;

while (* p! = '\ 0')

p ++;

return p - s;

  }
```

Na sua declaração, p é inicializado para s , ou seja, para apontar para o primeiro caractere da string. No enquanto loop, cada personagem por sua vez, é examinada até que o '\ 0' no final é visto. Porque p aponta para caracteres, p ++ avança P para o próximo carácter de cada vez, e PS indica o número de caracteres avançou mais, isto é, o comprimento da corda.

IMPORTANTE!!!

Quando um ponteiro é declarado que não aponta qualquer lugar. Utilizado este programa forma irá falhar. Isto é um dos famosos bug de segurança. Você deve definir a variável ponteiro para apontar algum lugar antes de usar isto. Isto pode ser de duas maneiras:

I. Alocação de memória para este ponteiro

II.Atribuí-la a resolver de uma variável existente.

int * Num;

* Num = 11;

O código acima irá falhar. código correto deve ser algo como abaixo:

```
int i = 5;

int * Num;

Num = & i;

* Num = 11;
```

Outra forma é a alocação de memória para o ponteiro de pilha:

int * Num;

Num = (int *) malloc (sizeof (int));

* Num = 11;

Conclusão

Nesta atividade, reveja o esquema de memória para processos e apresentou a memória

técnicas de gestão em programas C. Em particular, a criação de memória usando o malloc função, a condição de fuga de memória e uso de ponteiros em C foram destacados.

Avaliação

1. Escreva o seguinte código que ilustra threads.

```
//mem.c
#include <stdio.h>
#include <stdlib.h>
int uig;
int ig = 5;
int func()
{
return 0;
}
int main()
{
int local;
int *ptr;
ptr = (int *) malloc(sizeof(int));
printf("An address from BSS: %p\n", &uig);
printf("An address from Data segment: %p\n", &ig);
printf("An address from Code segment: %p\n", &func);
printf("An address from Stack segment: %p\n", &local);
printf("An address from Heap: %p\n", ptr);
printf("Another address from Stack: %p\n", &ptr);
```

```
free(ptr);

return 0;

}
```

Executar o código e examinar última linha de saída, o que mostra um outro endereço de pilha:

a. Porque &ptr está no pilha do segmento?

b. Porque a distância é de 4 bytes entre o local e o &ptr?

c. Porque &ptr é inferior a & local?

2. Considere o programa "ptr.c" dada na secção 1.3.2.3. Executar este programa e examinar asaída. Observe que, após a linha 7, char1 e char2 ambos são iguais a 'g' (valor da char1).

Note-se também que, o endereço de PTR é igual ao endereço de CH1.

Resumo da unidade

Nesta unidade, apresentaram o processo, conceitos de thread e de gerenciamento de memória e destacou as técnicas e funções para a criação e gestão. Em particular, a criação de processos funções foram apresentados e a gestão e manipulação de processos usando vários sinais do sistema foi destacada. A função de criação de linha e vários threads associados técnicas de gestão foram descritos. Finalmente, a função de criação de memória, a fuga de memória condição e uso de ponteiros em C foram explicados.

Avaliação da unidade

Verifique a sua compreensão!

Exercícios diversos

1. Faça o seguinte programa C que Forks um processo separado, salvá-lo,

executá-lo e anotar a sua saída.

int main ()

```
{

Pid_t pid;

/ * Garfo outro processo * /

pid = fork ();

if (pid <0) {ocorreu / * Erro * /

fprintf (stderr, "Fork Falha");

exit(-1);

}
```

```
else if (== pid 0) {/ * processo filho * /

execlp ( "/ bin / ls", "LS", null);

}

else {/ * processo pai * /

/ * Pai irá esperar para que a processo filhocompletar * /

wait (NULL);

printf ( "processo filhoComplete");

exit (0);

}

}
```

2. Considere o esqueleto código fornecido abaixo para o problema delimitada-Buffer com a solução Shared-Memory, que modela o paradigma para

processos cooperando, produtor processo produz informação que é consumida por um consumidor processo. O buffer compartilhada é implementado como um

matriz circular com dois ponteiros lógicos: in e out. A variável in pontos

para a próxima posição livre na memória buffer; pontos out para a primeira posição em pleno o buffer . O buffer está vazio, quando in= ut; o buffer está cheio quando em in + 1 mod n = out. Escreva um programa completo que usa threads para implementar, inserir e remover funções. Guarda-o, executa-o e desacartar asaída. Suponha que os itens são gerados aleatoriamente números inteiros.

// Dados compartilhados

```
        #define BUFFER_SIZE 10

Typedef struct {

        . . .

} item;

item buffer[BUFFER_SIZE];

int in = 0;

int out = 0;

// code for Insert function

while (true) {

  /* Produce an item */

   while (((in + 1) % BUFFER SIZE)  == out) ;  /*do nothing -- no
```

```
free buffers */
            buffer[in] = item;
            in = (in + 1) % BUFFER SIZE;
{
//code for remove function
while (true) {
        while (in == out)   ;  // do nothing -- nothing to consume
             // remove an item from the buffer
             item = buffer[out];
             out = (out + 1) % BUFFER SIZE;
        return item;
    {
```

3. Considere o programa abaixo. Descrever o estado da memória após a linha 10, 12 e 14.

O que é a saída do programa?

1	`#include <stdio.h>`
2	`int main() {`
3	`int i = 0;`
4	`int j = 2;`
5	`int** p1;`
6	`int* p2;`
7	`int* p3;`
8	`p2 = &i;`
9	`p3 = &j;`
10	`p1 = &p2;`
11	`j = *p3 + 1;`
12	`*p2 = **p1 - 1;`
13	`p1 -= 2;`
14	`*(p1[2]) = 3 * *p2;`

15	printf("i = %d, j= %d\n", i, j);
16	return 0;
17	}

Sistema de classificação

Como guiado pelos Regulamentos Instituição de classificação da Oferta

Leituras de unidade e outros recursos

1. 1. Mark Mitchell, Jeffrey Oldham, e Alex Samuel; Programação Avançada Linux;

   Copyright © 2001 pela New Riders Publishing; Primeira edição: junho de 2001

2. Programação. Sistema Linux: Falando diretamente para o kernel e C Library, Robert Love

   "O'Reilly Media, Inc.", 14 de maio de 2013

3. Linux Kernel Desenvolvimento, Robert Love, Pearson Education, 22 de junho de 2010

Unidade 4. Comunicação Interprocessos

Introdução à Unidade

A unidade anterior sobre "processos", discutiu a criação de processos e mostrou como um processo pode obter o status de saída de um processo filho. Essa é a forma mais simples de comunicação entre dois processos, mas é de nenhuma maneira o mais poderoso. Uma vez que os mecanismos não fornecer qualquer maneira para o pai comunique com o filho a não ser através de argumentos e variáveis em ambiente de linha de comando, nem qualquer maneira para que o filho se comunque com o pai exceto através de status de saída do filho. Nenhum destes mecanismos proporciona qualquer meio para comunicar-se com o processo filho enquanto ele não está realmente funcionando, nem estes mecanismos permitem a comunicação com um processo fora da relação pai-filho.

Nesta unidade, descreve os meios de comunicação interprocessos que estão rodeados de limitações. Apresentamos várias maneiras para comunicação entre pais e filhos, entre processos "independentes", e até mesmo entre os processos em diferentes máquinas. Comunicação Interprocessos (IPC) é a transferência de dados entre processos. Por exemplo, um navegador da Web pode solicitar uma página da Web de um servidor Web, que envia dados HTML. Esta transferência de dados geralmente usa soquetes em uma ligação telefónica. Em outro exemplo, você pode querer imprimir os nomes de arquivos em um diretório usando um comando como ls | lpr. O shell cria um processo ls e um processo lpr separada, ligando os dois com um pipe, representada pela "|" símbolo. Um cano permite a comunicação unidirecional entre dois processos relacionados. O processo ls escreve dados em o pipe, e o processo lpr lê dados a partir do pipe.

Nesta unidade, vamos discutir três tipos de comunicação entre processos:

I. Pipes permitir a comunicação sequencial de um processo para um processo relacionado.

II. FIFOs são semelhantes a pipes, exceto que os processos não relacionados podem se comunicar porque o pipe é dado um nome no sistema de arquivos.

III. Sockets apoiar a comunicação entre processos não relacionados, mesmo em diferentes computadores.

Estes tipos de IPC diferem pelos seguintes critérios:

I. Se restringir a comunicação para processos relacionados (processos com um comum ancestral), a processos não relacionados que compartilham o mesmo sistema de arquivos, ou para qualquer computador ligado a uma rede.

II. Se um processo de comunicação é limitado a apenas gravar dados ou somente leitura de dados.

III. O número de processos autorizados a comunicar.

IV. Se os processos que comunicam são sincronizadas pelo IPC-por exemplo, um processo de leitura é suspensa até que os dados estão disponíveis para ler

Objectivos da Unidade

Após a conclusão desta unidade, você deve ser capaz de:

- Explicar os conceitos de comunicação Interprocessos
- Comparar e contrastar os três tipos de IPC
- Criar cada um dos três tipos de IPC
- Utilizar mecanismos IPC para gerir a colaboração entre os processos

TERMOS-CHAVE

[Inter-process

comunicação (IPC)]: A transferência de dados entre processos

[Pipe]: Um pipe permite a comunicação unidirecional entre dois processos relacionados

[FIFOs]: Semelhante a pipes, exceto que os processos não relacionados pode comunicar, porque o pipe é dado um nome no sistema de arquivo.

[Sockets]: Apoiar a comunicação entre processos não relacionados, mesmo em diferentes computadores.

Actividades de Aprendizagem

Actividade 1.1 - Pipes

Introdução

Um pipe é um dispositivo de comunicação que permite a comunicação unidirecional. Dados gravados no "write end" do pipe é lida de volta dos "ler fim." Pipes são dispositivos seriais; os dados leia sempre a partir do pipe na mesma ordem em que foi escrito. Tipicamente, um pipe é usado para comunicar entre dois segmentos em um único processo ou entre processos pai e filho.

Em uma concha, o símbolo | cria um pipe. Por exemplo, este comando shell faz com que o shell para produzir dois processos filhos, um para ls e um para less:

```
$ls | less
```

O Shell também cria um pipe de ligação na saída padrão da sub-processo ls com o entrada padrão dos menos processo. Os nomes dos ficheiros listados por LS são enviados para menos da mesma ordem como se eles foram enviados diretamente para o terminal. Capacidade de dados de um pipe é limitada. Se o processor escritor, escreve processo mais rápido do que o processo leitor consome os dados, e se o pipe não pode armazenar mais dados, os blocos de processo escritor até mais capacidade se torna disponível. Se o leitor tenta ler

mas não há dados disponíveis, ele bloqueia até que os dados estejam disponíveis. Assim, o pipe automaticamente sincroniza os dois processos

Detalhes da actividade

Ciração de Pipes

Para criar um pipe, invocar o comando pipe. Fornecer uma matriz de inteiros de tamanho 2. A chamada para pipe armazena o descritor de arquivo de leitura em posição de matriz 0 e o descritor de arquivo escrito na posição 1.

Por exemplo, considere o seguinte código:

```
pipe_fds int [2];

int read_fd;

int write_fd;

pipes (pipe_fds);

read_fd = pipe_fds [0];

write_fd = pipe_fds [1];
```

Dados gravados no descritor de arquivo read_fd pode ser lido de volta de write_fd

1 comunicação entre processos pai e filho

Uma chamada para tubulação cria descritores de arquivos, que são válidas apenas dentro desse processo e seus filhos. Um descritore de arquivo do processo não pode ser passado para processos não relacionados; No entanto, quando o processo de chama garfo, descritores de arquivos são copiados para o novo processo filho. Assim, pipes só pode se conectar processos relacionados.

No "pipe.c" programa fornecido abaixo, um garfo gera um processo filho. A criança herda o descritores de arquivos tubulação. O pai escreve uma string para o pipe, e que a criança lê-lo. A amostra programa converte esses descritores de arquivo para * fluxos de arquivo usando fdopen. Porque nós usamos fluxos em vez de descritores de arquivo, podemos usar a biblioteca C padrão de nível superior as funções de I / O tais como printf e fgets.

```
// Programa "pipe.c" - através de um pipe para se comunicar com um
processo filho

#include <stdlib.h>
```

```
#include <stdio.h>
#include <unistd.h>
/* Write COUNT copies of MESSAGE to STREAM, pausing for a second
between each. */
void writer (const char* message, int count, FILE* stream)
{
for (; count > 0; --count) {
/* Write the message to the stream, and send it off immediately. */
fprintf (stream, "%s\n", message);
fflush (stream);
/* Snooze a while. */
sleep (1);
}
}
/* Read random strings from the stream as long as possible. */
void reader (FILE* stream)
{
char buffer[1024];
/* Read until we hit the end of the stream. fgets reads until
either a newline or the end-of-file. */
while (!feof (stream) && !ferror (stream) && fgets (buffer, sizeof
(buffer),
stream) != NULL) fputs (buffer, stdout);
}
int main ()
{
int fds[2];
pid_t pid;
/* Create a pipe. File descriptors for the two ends of the pipe are
placed in fds. */
```

```
pipe (fds);
/* Fork a child process. */
pid = fork ();
if (pid == (pid_t) 0) {
FILE* stream;
/* This is the child process. Close our copy of the write end of
the file descriptor. */
close (fds[1]);
/* Convert the read file descriptor to a FILE object, and read
from it. */
stream = fdopen (fds[0], "r");
reader (stream);
close (fds[0]);
}
else {
/* This is the parent process. */
FILE* stream;
/* Close our copy of the read end of the file descriptor. */
close (fds[0]);
/* Convert the write file descriptor to a FILE object, and write
to it. */
stream = fdopen (fds[1], "w");
writer ("Hello, world.", 5, stream);
close (fds[1]);
}
return 0;
}
```

No início do main, fds é declarado para ser uma matriz de inteiros com o tamanho da chamada pipe 2.O cria um pipe e coloca as de leitura e gravação de arquivos descritores nessa matriz. O programa então bifurca uma criança processo. Depois de fechar a leitura final do pipe, o processo pai começa a escrever strings para o pipe. Depois de fechar a extremidade de escrita

do pipe, a criança lê cordas a partir do pipe. Note-se que depois escrita na função de escritor, o pai libera o pipe chamando fflush. Caso contrário, a cadeia não pode ser enviada através do pipe imediatamente.

Quando você invocar o comando ls l menos, dois garfos ocorrer: uma para o processo filho ls e uma para a menos processo filho. Ambos os processos herdam os descritores de arquivos tubulação para que eles possam comunicam através de um pipe. Para se ter processos não relacionados comunicar, usar um FIFO em vez disso, como serão discutidos em seções posteriores em "FIFO".

Redirecionar a entrada padrão, fluxos de saída e de erro

Frequentemente, você vai querer criar um processo filho e criar uma extremidade de um pipe como a sua entrada padrão ou saída padrão. Usando a chamada dup2, você pode equiparar a um descritor de arquivo com outro. Por exemplo, para redirecionar a entrada padrão de um processo para a fd descritor de arquivo, use esta linha:

dup2 (fd, STDIN_FILENO);

O STDIN_FILENO constante simbólica representa o descritor de arquivo para a entrada padrão, que tem a chamada valor 0.The fecha entrada padrão e, em seguida, reabre-lo como uma duplicata de fd assim que os dois podem ser utilizados indiferentemente. Descritores de arquivos equiparadas compartilham a mesma posição do arquivo

e o mesmo conjunto de sinalizadores de status de arquivo. Assim, caracteres lidos do fd não são reler a partir padrão de entrada.

O "dup2.c" programa dado abaixo usa dup2 para enviar a saída de um pipe para o tipo de comando (tipo lê linhas de texto da entrada padrão, classifica-as em ordem alfabética, e os imprime na saída padrão). Depois de criar um pipe, os garfos de programa. O processo pai imprime algumas strings para o pipe. O processo do filho atribui o descritor de arquivo de leitura do pipe para sua entrada padrão usando dup2. Em seguida, ele executa o programa de classificação

```
//Programa "dup2.c" - Redirect Output from a Pipe with dup2

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main ()

{

int fds[2];
```

```
pid_t pid;
/* Create a pipe. File descriptors for the two ends of the pipe are
placed in fds. */
pipe (fds);
/* Fork a child process. */
pid = fork ();
 if (pid == (pid_t) 0) {
/* This is the child process. Close our copy of the write end of
the file descriptor. */
close (fds[1]);
/* Connect the read end of the pipe to standard input. */
dup2 (fds[0], STDIN_FILENO);
/* Replace the child process with the "sort" program. */
execlp ("sort", "sort", 0);
  }
  else {
/* This is the parent process. */
FILE* stream;
/* Close our copy of the read end of the file descriptor. */
close (fds[0]);
/* Convert the write file descriptor to a FILE object, and write
to it. */
stream = fdopen (fds[1], "w");
fprintf (stream, "This is a test.\n");
fprintf (stream, "Hello, world.\n");
fprintf (stream, "My dog has fleas.\n");
fprintf (stream, "This program is great.\n");
fprintf (stream, "One fish, two fish.\n");
fflush (stream);
close (fds[1]);
/* Wait for the child process to finish. */
```

```
    waitpid (pid, NULL, 0);
      }
    return 0;
    }
```

popen e pclose

Um uso comum de pipe é para enviar dados para ou receber dados de um programa que está sendo executado em um sub-processo. As funções popen e pclose facilitar esse paradigma, eliminando a necessidade de invocar tubulação, garfo, dup2, exec, e fdopen. Compare o "popen.c" programa fornecido abaixo, que usa popen e pclose, ao exemplo anterior programa "dup2.c".

```
    //Programa "popen.c" - Exemplo usando o  popen
    #include <stdio.h>
    #include <unistd.h>
    int main ()
        {
    FILE* stream = popen ("sort", "w");
    fprintf (stream, "This is a test.\n");
    fprintf (stream, "Hello, world.\n");
    fprintf (stream, "My dog has fleas.\n");
    fprintf (stream, "This program is great.\n");
    fprintf (stream, "One fish, two fish.\n");
    return pclose (stream);
     }
```

A chamada para popen cria um processo filho de executar o comando sort, substituindo chamadas para tubulação, fork, dup2, e execlp. O segundo argumento, "w", indica que este processo quer escrever para o processo filho. O valor de retorno de popen é uma extremidade de um pipe; a outra extremidade está ligada a a entrada padrão do processo filho. Após a escrita acabamentos, pclose fecha o processo filho de córrego, aguarda o processo terminar, e retorna seu valor de status.

O primeiro argumento para popen é executado como um comando shell em um sub-processo em execução / bin / sh.

O shell procura a variável de ambiente PATH da maneira usual para encontrar programas para executar.

Se o segundo argumento é "r", a função retorna fluxo de saída padrão do processo filho para que o pai pode ler a saída. Se o segundo argumento é "w", a função retorna o fluxo filho de entrada padrão do processo para que o pai pode enviar dados. Se ocorrer um erro, popen retornos um ponteiro nulo. Chamada pclose para fechar um fluxo retornado por popen. Depois de fechar o especificado

córrego, pclose aguarda o processo filho para terminar.

Conclusão

Essa atividade apresentou um pipe como um dispositivo IPC que permite a comunicação unidirecional. Nós descrevemos as instruções para a criação de um pipe, e como o pipe pode ser usado para a comunicação entre processos pai e filho. Além disso, mostramos como redirecionar padrão de um processo de Entrada, Saída e Erro de streams, bem como a utilização das funções popen e pclose para eliminar o necessário chamar pipe, fork, dup2, exec, e funções fdopen.

Avaliação

1. Pratique o código de exemplo fornecido nesta atividade.

Actividade 1.2 - FIFO

1.2.1 Introdução

Um arquivo primeiro a entrar, primeiro a sair (FIFO) é um pipe que tem um nome no sistema de arquivos. Qualquer processo pode abrir ou fechar o FIFO; os processos em cada extremidade do pipe não precisam de ser ligados uns aos outros.

FIFOs são também chamados de named pipes.

Você pode fazer uma FIFO usando o comando mkfifo. Especifique o caminho para o FIFO no linha de comando. Por exemplo, criar um FIFO em / tmp / fifo invocando o seguinte:

```
$ mkfifo / tmp / fifo

$ ls -l / tmp / fifo

PRW-rw-rw- 1 usuários samuel 0 16 de janeiro 14:04 / tmp / fifo
```

O primeiro caractere da saída do ls é p, indicando que este arquivo é na verdade um FIFO (named pipe). Em uma janela, ler a partir do FIFO invocando o seguinte:

$ Cat </ tmp / fifo

Em uma segunda janela, escreva para o FIFO, invocando o seguinte:

```
$ Cat> / tmp / fifo
```

Em seguida, digite algumas linhas de texto. Cada vez que você pressionar Enter, a linha de texto é enviada através do FIFO e aparece na primeira janela. Feche o FIFO pressionando Ctrl + D na segunda janela.

Remova o FIFO com esta linha:

```
$ Rm / tmp / fifo
```

Detalhes da atividade

Criando um FIFO

Criar uma programação FIFO usando a função mkfifo. O primeiro argumento é o caminho a que para criar o FIFO; O segundo parâmetro especifica proprietário, grupo e mundo da tubulação permissões (permissões de arquivo do sistema). Porque um pipe deve ter um leitor e um escritor, o permissões devem incluir tanto ler e escrever permissões. Se o pipe não pode ser criada (por exemplo, se um arquivo com esse nome já existe), mkfifo retorna -1. Include <sys / types.h> e <Sys / stat.h> se você chamar mkfifo.

Acessando um FIFO

Acessar um FIFO apenas como um arquivo comum. Para comunicar-se através de um FIFO, um programa deve abrí-lo para escrever, e outro programa deve abrí-lo para a leitura. Ou as funções de baixo nível de E / S (Aberto, escrever, ler, perto, e assim por diante) ou C biblioteca de funções de I / O (fopen, fprintf, fscanf, fclose, e assim por diante) pode ser usado.

Por exemplo, para escrever um buffer de dados para um FIFO usando rotinas de baixo nível de E / S, você poderia usar este código:

int fd = open (fifo_path, O_WRONLY);

write (fd, dados, data_length);

close (fd);

Para ler uma string a partir do FIFO usando a biblioteca C funções de I / O, você pode usar este código:

```
FILE * FIFO = fopen (fifo_path, "r");

fscanf (FIFO, "% s", buffer);

fclose (FIFO);
```

A FIFO pode ter vários leitores ou vários escritores. Bytes de cada escritor são escritos automicamente até um tamanho máximo de PIPE_BUF (4KB no Linux). Pedaços de simultânea os escritores podem ser intercaladas. Regras semelhantes se aplicam a simultânea lê

Diferenças de Pipes nomeados do Windows

Tubos em sistemas operacionais Win32 são muito semelhantes a pipes Linux (consulte a biblioteca Win32 documentação para obter detalhes técnicos sobre estes). A principal diferença para o Win32 se que o nome pipes funções são mais como soquetes. Win32 pipes nomeados pode conectar processos em separado

computadores ligados através de uma rede. No Linux, tomadas são utilizadas para este fim. Além disso, Win32 permite várias conexões leitor-escritor em um pipe nomeado sem intercalação de dados e Win32 pipes podem ser usados para a comunicação de duas vias (Note que apenas o Windows NT pode criar uma chamada pipe; Programas do Windows 9x pode formar apenas conexões de clientes).

Conclusão

Essa atividade apresentou um FIFO (também referida como "pipes") como um dispositivo que permite IPC processo de comunicação independente. Nós descrevemos as construções para criar e acessar uma FIFO e destacou as diferenças de Linux para Windows named pipe.

Avaliação

Pratique fazendo os programas completos desenvolvidos nesta unidade.

Actividade 1.3 - Sockets

Introdução

Um socket é um dispositivo de comunicação bidireccional que pode ser utilizado para comunicar com outro processo na mesma máquina ou com um processo em execução em outras máquinas. Sockets autorização comunicação entre processos em computadores diferentes. Programas de Internet, como Telnet, rlogin, FTP, conversa, eo uso Wide Web sockets mundo. Por exemplo, você pode obter o WWW página de um servidor Web usando o programa Telnet porque ambos usar soquetes para a rede comunicações (Normalmente, você usaria telnet para conectar um servidor Telnet para logins remotos. Mas você também é possível usar o telnet para conectar a um servidor de um tipo diferente e digite comentários diretamente na isto). Para abrir uma conexão com um servidor WWW em www.codesourcery.com, use telnet www.codesourcery.com 80. A magia constante 80 especifica uma conexão com o servidor Web programa em execução www.codesourcery.com em vez de algum outro processo. Tente digitar GET / depois a conexão é estabelecida. Isso envia uma mensagem através do socket para o servidor Web, que responde enviando HTML da casa da página e, em seguida, fechar a conexão para exemplo:

```
$ telnet www.codesourcery.com 80
```

Trying 206.168.99.1...

Connected to merlin.codesourcery.com (206.168.99.1).

Escape character is '^]'.

```
GET /

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">

...
```

Detalhes da atividade

Conceito de Socket

Conceitos de soquete

Quando você cria um socket, você deve especificar três parâmetros: o estilo de comunicação, namespace,e protocolo. Um estilo de comunicação controla como os dados de socket trata transmitidos e especifica o número de parceiros de comunicação. Quando os dados são enviados através de uma tomada, é

empacotado em pedaços chamados pacotes. O estilo de comunicação determina como esses pacotes são tratadas e como eles são abordados a partir do emissor para o receptor.

- estilos de conexão garantir a entrega de todos os pacotes na ordem em que foram enviados. Se os pacotes são perdidas ou reordenados por problemas na rede, o receptor solicita automaticamente sua retransmissão do remetente.

Um soquete-estilo de conexão é como uma chamada de telefone: Os endereços do remetente e destinatário são fixada no início da comunicação, quando a ligação é estabelecida.

- estilos Datagram não garantem a entrega ou chegada ordem. Os pacotes podem ser perdidos ou reordenadas em trânsito devido a erros de rede ou outras condições. Cada pacote deve ser rotulado com seu destino e não é garantido para ser entregue. Os sistema garante apenas "melhor esforço", então os pacotes podem desaparecer ou chegar em uma ordem diferente do que o transporte.

Um soquete de estilo datagrama se comporta mais como correio postal. O remetente especifica o receptor de endereço para cada mensagem individual.

Um namespace tomada especifica como endereços de socket são escritos. Um endereço de socket identifica uma final de uma conexão de soquete. Por exemplo, endereços de socket no "espaço local" são comuns nomes de arquivos. Em "namespace Internet," um endereço de socket é composto pelo endereço de Internet (também conhecido como um endereço de Protocolo de Internet ou endereço IP) de um host conectado à rede e um número da porta. O número da porta distingue entre múltiplos soquetes no mesmo host.

Um protocolo especifica a forma como os dados são transmitidos. Alguns protocolos são TCP / IP, o primário protocolos usados pela Internet rede; o protocolo de rede AppleTalk; eo UNIX locais protocolo de comunicação. Nem todas as combinações de estilos, namespaces e protocolos são suportada

Chamada a Sistema

Sockets são mais flexíveis do que as técnicas de comunicação previamente discutidas. Estes são o sistema chama envolvendo sockets:

socket-Cria um soquete

close-Destroys um soquete

connect -Cria uma conexão entre dois soquetes

bind-etiquetas um soquete de servidor com um endereço

listen-configura um soquete para aceitar condições

accept-Accepts uma conexão e cria um novo socket para a conexão

Sockets são representados por descritores de arquivos.

A. Criando e Destruindo um Sockets

O soquete e funções estreitas criar e destruir bases, respectivamente. Quando você cria um socket, especificar as três opções de socket: namespace, estilo de comunicação, e de protocolo. Para o parâmetro de namespace, usar constantes começando com PF_ (abreviando "protocolo famílias"). Para exemplo, PF_LOCAL ou PF_UNIX especifica o namespace local, e especifica PF_INET Namespaces Internet. Para o parâmetro de estilo de comunicação, usar constantes começando com SOCK_. Use SOCK_STREAM para um socket de estilo conexão ou usar SOCK_DGRAM para um Tomada de estilo datagrama.

O terceiro parâmetro, o protocolo, especifica o mecanismo de baixo nível para transmitir e receber dados. Cada protocolo é válido para uma determinada combinação de estilo namespace. Porque há geralmente um melhor protocolo para cada tal par, especificando 0 geralmente é o protocolo correto. Se a tomada

bem-sucedido, ele retorna um descritor de arquivo para o socket. Você pode ler ou escrever para o soquete usando ler, escrever, e assim por diante, como com outros descritores de arquivos. Quando tiver terminado com um soquete, chame

close para removê-lo.

B. Chamando connect

Para criar uma conexão entre dois soquetes, as chamadas de clientes conectar, especificando o endereço de um soquete do servidor para se conectar. Um cliente é o processo que inicia a conexão, e um servidor é o processo de espera para aceitar conexões. As chamadas de clientes ligar para iniciar uma ligação a partir de um socket local à tomada de servidor especificado pelo segundo argumento. O terceiro argumento é o comprimento, em bytes, da estrutura de endereço apontado pelo segundo argumento. endereço de socket formatos diferentes de acordo com o espaço de nomes socket.

C. Informações de envio

Qualquer técnica para escrever para um descritor de arquivo pode ser usado para escrever a uma tomada. Para mais detalhes, revisitar a unidade que apresenta funções O baixo nível de Linux I / e algumas das questões em torno sua utilização. A função de envio, que é específico para os descritores de arquivos socket, fornece uma alternativa para escrever com algumas opções adicionais; veja a página do manual para obter informações.

Servidores

O Ciclo de vida do servidor consiste na criação de um socket-estilo de conexão, ligando um endereço para seu soquete, colocando uma chamada para ouvir que permite conexões para o socket, fazer chamadas para aceitaronexões de entrada e, em seguida, fechar o soquete. Os dados não são lidos e escritos diretamente através do soquete do servidor; Em vez disso, cada vez que um programa aceita uma nova ligação, o Linux cria um separada socket para usar na transferência de dados através dessa ligação. Nesta seção, apresentamos bind, ouvir, e aceitar.

Um endereço deve ser ligado à tomada do servidor usando bind se um cliente é encontrá-lo. seu primeiro argumento é o descritor de arquivo de socket. O segundo argumento é um ponteiro para um endereço de socket estrutura; O formato deste depende de família de endereço do socket. O terceiro argumento é o comprimento da estrutura de endereço, em bytes. Quando um endereço é ligado a uma tomada de estilo de conexão, deve invocar o ouvir para indicar que é um servidor. Seu primeiro argumento é o descritor de arquivo de socket.

O segundo argumento especifica quantas conexões pendentes estão na fila. Se a fila estiver cheia, conexões adicionais serão rejeitadas. Isto não limita o número total de ligações que um servidor pode manipular; limita apenas o número de clientes que tentam se conectar que ainda não tenham sido aceitaram.

Um servidor aceita pedidos de ligação de um cliente, invocando aceitar. O primeiro argumento é o descritor de arquivo de socket. O segundo argumento aponta para uma estrutura endereço de soquete, que é preenchido com o endereço de soquete do cliente. O terceiro argumento é o comprimento, em bytes, do endereço de socket estrutura. O servidor pode usar o endereço do cliente para determinar se ele realmente quer comunicar com o cliente. A chamada para aceitar cria um novo socket para comunicação com o cliente e retorna o descritor de arquivo correspondente. A tomada servidor original continua a aceitar novas ligações de clientes. Para ler dados de um soquete sem removê-lo a partir da entrada filas, utilize recv. Leva os mesmos argumentos de leitura, além de um argumento flags adicionais. Uma bandeira de MSG_PEEK faz com que os dados sejam lidos, mas não removida da fila de entrada.

Sockets locais

Sockets conectando processos no mesmo computador pode usar o namespace local representado por os sinônimos PF_LOCAL e PF_UNIX. Estes são chamados de bases locais ou UNIX-domain soquetes. Seus endereços de socket, especificados por nomes de arquivos, são utilizadas apenas durante a criação de conexões.

O nome do soquete é especificado no sockaddr_un struct. Você deve definir o campo sun_ family para AF_LOCAL, indicando que este é um espaço de nomes local. O campo sun_path especifica o nome do arquivo de usar e pode ser, no máximo, 108 bytes de comprimento.

O comprimento real do sockaddr_un struct deve ser calculado usando a macro SUN_LEN. Qualquer filename pode ser usado, mas o processo deve ter permissões de gravação do diretório, que permitem adicionar arquivos ao diretório. Para se conectar a um soquete, um processo deve ter permissão de leitura para o

Arquivo. Apesar de computadores diferentes podem compartilhar o mesmo sistema de arquivos, apenas os processos em execução no o mesmo computador pode se comunicar com os soquetes de namespace locais.

O único protocolo admissível para o namespace local é 0. Porque ele reside em um sistema de arquivos, um socket local é listado como um arquivo. Por exemplo, observe os 's' iniciais:

```
$ ls -l /tmp/socket

srwxrwx--x 1 user group 0 Nov 13 19:18 /tmp/socket
```

Chamada unlink para remover um socket local quando tem sido terminado.

Um exemplo usando Sockets namespace local

Nós ilustramos soquetes com dois programas. O programa do servidor ", socket-server.c", cria um namespace socket local e escutas de ligações sobre ele. Quando ele recebe uma conexão, ele lê o texto mensagens da conexão e imprime-los até que a conexão é fechada. Se uma destas mensagens é "parar" o programa do servidor remove o soquete e termina. O programa de soquete-servidor toma o caminho para o socket como seu argumento de linha de comando.

// Programa "socket-server.c" - local Namespace Socket Server

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <unistd.h>

/* Read text from the socket and print it out. Continue until the

socket closes. Return nonzero if the client sent a "quit"

message, zero otherwise. */

int server (int client_socket)

{
```

```
  while (1) {
int length;
char* text;
/* First, read the length of the text message from the socket. If
read returns zero, the client closed the connection. */
if (read (client_socket, &length, sizeof (length)) == 0)
return 0;
/* Allocate a buffer to hold the text. */
text = (char*) malloc (length);
/* Read the text itself, and print it. */
read (client_socket, text, length);
printf ("%s\n", text);
/* Free the buffer. */
free (text);
/* If the client sent the message "quit," we're all done. */
if (!strcmp (text, "quit"))
return 1;
  }
}
int main (int argc, char* const argv[])
{
 const char* const socket_name = argv[1];
  int socket_fd;
  struct sockaddr_un name;
  int client_sent_quit_message;
/* Create the socket. */
  socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0);
/* Indicate that this is a server. */
  name.sun_family = AF_LOCAL;
  strcpy (name.sun_path, socket_name);
```

```
  bind (socket_fd, &name, SUN_LEN (&name));
/* Listen for connections. */
  listen (socket_fd, 5);
/* Repeatedly accept connections, spinning off one server() to deal
with each client. Continue until a client sends a "quit" message.
*/
  do {
struct sockaddr_un client_name;
socklen_t client_name_len;
int client_socket_fd;
/* Accept a connection. */
client_socket_fd = accept (socket_fd, &client_name,
&client_name_len);
/* Handle the connection. */
client_sent_quit_message = server (client_socket_fd);
/* Close our end of the connection. */
close (client_socket_fd);
  }
while (!client_sent_quit_message);
/* Remove the socket file. */
close (socket_fd);
unlink (socket_name);
return 0;
}
```

O programa cliente ", socket-client.c", se conecta a uma tomada de namespace local e envia uma mensagem.O caminho do nome para o socket e a mensagem são especificados na linha de comando.

// Programa "socket-client.c" - local de cliente Namespace socket

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
```

```
#include <sys/un.h>
#include <unistd.h>
/* Write TEXT to the socket given by file descriptor SOCKET_FD. */
void write_text (int socket_fd, const char* text)
  {
/* Write the number of bytes in the string, including
NUL-termination. */
int length = strlen (text) + 1;
write (socket_fd, &length, sizeof (length));
/* Write the string. */
write (socket_fd, text, length);
   }
int main (int argc, char* const argv[])
  {
const char* const socket_name = argv[1];
const char* const message = argv[2];
int socket_fd;
struct sockaddr_un name;
/* Create the socket. */
socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0);
/* Store the server’s name in the socket address. */
name.sun_family = AF_LOCAL;
strcpy (name.sun_path, socket_name);
/* Connect the socket. */
connect (socket_fd, &name, SUN_LEN (&name));
/* Write the text on the command line to the socket. */
write_text (socket_fd, message);
close (socket_fd);
return 0;
   }
```

Antes de o cliente enviar a mensagem de texto, ele envia o comprimento de que o texto enviando os bytes de o comprimento variável inteiro. Do mesmo modo, o servidor lê o comprimento do texto por leitura a partir da Tomada em uma variável inteira. Isto permite que o servidor para atribuir uma memória intermédia de tamanho adequado para segure o texto da mensagem antes de lê-lo da tomada.

Para experimentar este exemplo, iniciar o programa de servidor em uma janela. Especificar um caminho para um socket -

Por exemplo, / tmp / socket.

$. / socket-server / tmp / socket

Em outra janela, execute o cliente algumas vezes, especificando o caminho mesmas tomadas além de mensagens de enviar para o cliente:

```
$. / socket-client / tmp / socket "Olá, mundo".

$ ./socket-client / Tmp / socket "Este é um teste."
```

O programa servidor recebe e imprime essas mensagens. Para fechar o servidor, enviar a mensagem "Sair" de um cliente:

$ ./socket-client / Tmp / socket "quit"

O programa servidor termina.

Internet de Domínio Socket

Soquetes do domínio Unix pode ser usado apenas para comunicação entre dois processos no mesmo computador. Soquetes Internet no domio, por outro lado, pode ser utilizado para ligar em processos diferentes máquinas ligadas por uma rede.

Sockets conectando processos através da Internet utilizando o espaço nominal Internet representado por PF_INET. Os protocolos mais comuns são TCP / IP. O Protocolo de Internet (IP), um protocolo de baixo nível, move pacotes através da Internet, dividir e reunir os pacotes, se necessário. Isso garante só entrega "melhor esforço", por isso, os pacotes podem desaparecer ou ser reordenadas durante o transporte.

Cada computador participante é especificado usando um número de IP único. O Transmission Control Protocol (TCP), em camadas em cima de IP, fornece transporte ordenou-conexão confiável. ele permite ligações telefónicas-like a ser estabelecida entre computadores e garante que os dados é entregues de forma confiável e em ordem.

Endereços de socket de Internet conter duas partes: uma máquina e um número de porta. Esta informação é armazenado numa estrutura sockaddr_in variável. Defina o sin_family campo para AF_INET para indicar que este é um endereço namespace Internet. O sin_addr campo armazena o endereço de Internet do desejado máquina como um número IP inteiro de 32 bits. Um número de porta distingue uma determinada máquina é diferente soquetes. Porque diferentes máquinas armazenar valores de vários bytes em diferentes ordens de byte, use htons para converter o número da porta para rede byte ordem . Veja a página man para ip para mais informações.

Para converter nomes de host legíveis, tanto números em notação de ponto padrão (como 10.0.0.1) ou nomes DNS (tais como www.codesourcery.com) em números IP de 32 bits, você pode usar gethostbyname . Isso retorna um ponteiro para a struct hostent estrutura; o h_addr campo contém número IP do hospedeiro.

Note-se que os nomes de DNS são úteis porque é mais fácil de lembrar nomes do que números. o Domain Name Service (DNS) associa nomes como www.codesourcery.com com computadores 'números de IP exclusivo. DNS é implementado por uma hierarquia mundial de servidores de nomes, mas você não precisa entender protocolos DNS para usar nomes de host Internet em seus programas.

O programa de exemplo " socket-inet.c " fornecido abaixo ilustra o uso da internet de domínio soquetes. O programa obtém a home page do servidor Web cujo nome de host é especificado na linha de comando.

// Programa socket-inet.c - Lê de um servidor WWW

```
#include <stdlib.h>

#include <stdio.h>

#include <netinet/in.h>

#include <netdb.h>

#include <sys/socket.h>

#include <unistd.h>

#include <string.h>

/* Print the contents of the home page for the server's socket.

Return an indication of success. */

void get_home_page (int socket_fd)

{

 char buffer[10000];

  ssize_t number_characters_read;

/* Send the HTTP GET command for the home page. */

  sprintf (buffer, "GET /\n");

  write (socket_fd, buffer, strlen (buffer));

/* Read from the socket. The call to read may not

   return all the data at one time, so keep

   trying until we run out. */

   while (1) {
```

```
number_characters_read = read (socket_fd, buffer, 10000);
if (number_characters_read == 0)
return;
/* Write the data to standard output. */
fwrite (buffer, sizeof (char), number_characters_read, stdout);
   }
}
int main (int argc, char* const argv[])
{
  int socket_fd;
  struct sockaddr_in name;
  struct hostent* hostinfo;
/* Create the socket. */
  socket_fd = socket (PF_INET, SOCK_STREAM, 0);
/* Store the server's name in the socket address. */
  name.sin_family = AF_INET;
/* Convert from strings to numbers. */
  hostinfo = gethostbyname (argv[1]);
  if (hostinfo == NULL)
return 1;
  else
name.sin_addr = *((struct in_addr *) hostinfo->h_addr);
/* Web servers use port 80. */
name.sin_port = htons (80);
/* Connect to the Web server */
  if (connect (socket_fd, &name, sizeof (struct sockaddr_in)) ==
-1) {
        perror ("connect");
return 1;
  }
```

```
    /* Retrieve the server's home page. */

      get_home_page (socket_fd);

      return 0;

    }
```

Este programa leva o nome do servidor Web na linha de comando (não uma URL- isto é,

sem o "http: //"). Ele chama gethostbyname para traduzir o nome do host em um IP numérico abordar e, em seguida, se conecta a um soquete de fluxo (TCP) para a porta 80 em que o host. Servidores de web falam atraves do Protocolo de transporte de hipertexto (HTTP), para que o programa emite o HTTP GET comando e o servidor responde enviando o texto da home page.

Por exemplo, para recuperar a home page do site www.codesourcery.com , Invocar esta:

```
$ ./socket-inet Www.codesourcery.com

<html>

<Meta http-equiv = "Content-Type" content = "text / html; charset =
iso-8859-1 ">

...
```

Nota sobre números de porta padrão: Por convenção, os servidores Web para ouvir conexões na porta 80. A maioria dos serviços de rede de Internet está associada com um número de porta padrão. Por exemplo, servidores seguros da Web que usam SSL escutar as conexões na porta 443, e servidores de correio (que falar SMTP) usar a porta 25.

Em sistemas GNU / Linux, as associações entre o protocolo / serviço nomes e números de porta padrão são listados no arquivo / etc / services. A primeira coluna é a protocolo ou nome do serviço. A segunda coluna lista o número da porta e o tipo de conexão: TCP para conexão-orientado, ou udp para datagrama. Se você implementar serviços de rede personalizadas usando soquetes internet de domínio, utilize números de porta acima de 1024.

Sockets Pairs

Como vimos anteriormente, a função de pipe cria dois descritores de arquivos para o início e o fim de um pipe. Pipes são limitados porque os descritores de arquivo deve ser utilizado por processos relacionados porque a comunicação é unidirecional. O socketpair função cria dois descritores de arquivos para dois sockets conectados no mesmo computador.

Estes descritores de arquivos permitir a comunicação de duas vias entre os processos relacionados. três seu primeiro parâmetros são os mesmos que os da chamada tomada: Eles especificar o tipo de domínio, ligação, e protocolo. O último parâmetro é uma matriz de duas inteiro, que é preenchido com as descrições de arquivo dos dois soquetes, semelhantes a tubulação. Quando você chamar socketpair , você deve especificar

PF_LOCAL como o domínio.

Conclusão

Esta actividade apresentada tomada como um dispositivo de comunicação bidireccional que pode ser usado para comunicar com um outro processo na mesma máquina, ou com um processo em execução no outro máquinas. Vários conceitos relacionados a tomadas foram descritos e demonstrados, incluindo As chamadas do sistema envolvendo sockets, servidores e ciclo de vida do servidor, soquetes Internet de domínio e Pairs soquete.

Avaliação

- Pratique o código de exemplo fornecido nesta atividade.

Resumo da Unidade

Esta unidade apresentou três tipos de dispositivos de comunicação IPC, os pipes, FIFOs e Sockets. Um pipe permite uma forma de comunicação entre dois processos relacionados. FIFOs são semelhantes aos tubos, exceto que os processos não relacionados podem comunicar, porque o pipe é dado um nome no sistema de arquivos. Sockets apoiar a comunicação entre alheios

processos, mesmo em computadores diferentes. Os conceitos por trás de cada dispositivo de comunicação foram descritos, e os construtos para a criação e a manipulação do dispositivo.

Avaliação da Unidade

Verifique a sua compreensão!

Exercícios diversos

Instruções:

Considerar o esqueleto código dado abaixo demonstra que o bounded-

problema de buffer, que modela o paradigma para processos cooperantes. Um produtor processo produz informação que é consumida por um consumidor processo. Faça um conjunto de programas de trabalho, que utiliza Pipes, FIFOs, e Sockets mecanismos IPC, respectivamente, como método de solução para o problema de buffer delimitada.

```
//Shared data

#define BUFFER_SIZE 10

Typedef struct {

        . . .

} item;

item buffer[BUFFER_SIZE];

int in = 0;
```

```
int out = 0;

// code for Insert function

while (true) {

  /* Produce an item */

  while (((in + 1) % BUFFER SIZE)  == out) ;  /*do nothing -- no
free buffers */

              buffer[in] = item;

              in = (in + 1) % BUFFER SIZE;

    {

//code for remove function

while (true) {

        while (in == out)  ;  // do nothing -- nothing to consume

              // remove an item from the buffer

              item = buffer[out];

              out = (out + 1) % BUFFER SIZE;

        return item;

  }
```

Avaliação do curso

1. Implementa usando o linux GCC, crie dois processos threads cujo maximo de subprocesos filhos na fila do buffer são 10?

2. Desenvolva um algoritmo em C Linux GCC que reserva 1MB espaco em memoria?

3. Implemente usando funções socket GCC um servidor web que recebe requisições de paginas web?

LAB

- TÍTULO LAB : Aplicação Socket TCP/IP Remoto entre Um Servidor E Vários Clientes Conversando Ao Mesmo Tempo.
- OBJETIVOS DO LABORATÓRIO : Testar as habilidades dos alunos ate esta unidade, permitindo-lhe implementar na prática uma aplicação SOCKET TCP/IP usando os conceitos de programação C LINUx GCC implementando um chat entre vários cliente e um único servidor simultaneamente.

- OS DETALHES DOS EXERCÍCIOS DE LABORATÓRIO / ATIVIDADES PARA O TÍTULO LABORATÓRIO - PARA CADA EXERCÍCIO:
- OBJECTIVOS DO EXERCÍCIO : Testar e consolidar os conhecimentos teóricos obtidos ao longo do módulo em particular a unidade quatro no que tange a Socket TCP/IP
- RECURSOS NECESSÁRIOS: Um computador rodando qualquer distribuição Linux com o copilador GCC e também com acesso a internet.
- TEMPO REQUERIDO: Duas semanas
- DESCRIÇÃO DO EXERCÍCIO DE LABORATÓRIO / ATIVIDADE: A aplicação Chat é uma aplicação socket TCP/IP que roda um servidor capaz de aceitar pelo menos 10 requisições cliente na fila do buffer, podendo os clientes conversarem entre si e simultaneamente, mas todas as comunicações precisam passar pelo servidor. Sempre que o servidor ficar offline os cliente não conseguem comunicar. O sistema deve gerar varios threads filhos.
- RESULTADOS E REQUISITOS DE APRESENTAÇÃO: Conhecimento em programação C estruturada, processos concorrente, e Socket C, Sistemas operacional Linux e resultados será apresento em que vários alunos usando seus computadores possam através da aplicação Cliente.c conversarem entre si em tempo real, como acontece com por exemplo o chat de facebook, MSN, etc
- CRITÉRIOS DE APRECIAÇÃO: O aluno deve implementar os dois algoritmos, comentar os códigos, e defender presencialmente sendo os pesos são:
- lidade do algoritmo e comentário do código: 20%
- Estrutura de algoritmo -lógica adequada (fork, threads): 30%
- Defesa oral e entendimento do que foi feito: 50%

AS REFERÊNCIAS OU LINKS IMPORTANTES

1. UNIX Systems Programming: Communication, Concurrency, and Threads, By Kay A. Robbins, Steven Robbins, Prentice Hall Professional, 2003
2. UNIX Network Programming: Interprocess communications, Volume 2 , W. Richard Stevens Prentice Hall PTR, 1999

Leituras e outros Recursos

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

1. Mark Mitchell, Jeffrey Oldham, and Alex Samuel; Advanced Linux Programming; Copyright © 2001 by New Riders Publishing; FIRST EDITION: June, 2001
2. UNIX Systems Programming: Communication, Concurrency, and Threads, By Kay A. Robbins, Steven Robbins, Prentice Hall Professional, 2003
3. Interprocess Communications in Linux, By John Shapley Gray, Prentice Hall Professional, 2003

4. UNIX Network Programming: Interprocess communications, Volume 2 , W. Richard Stevens Prentice Hall PTR, 1999

5. Sistemas Operacionais - Conceitos e Aplicações inclui Java, Abraham Silberschatz et al Editora Campus, Rio de Janeiro,2000;

6. Distributed Operating Systems, Andrew Tanenbaum, Ed Prentice Hall - Perason - USA, 2000.

7. Operating Systems - Concurrent And Distributed Software Design, Jean Bacon, Tim Harris, Addison Wesley - USA, 2003.

8. C - Completo e Total, Herbert Schildt, 3edição, Editora -Makron Books, S.Paulo, 1996.

9. The Complete Reference C 4Ed, Herbert Schildt, Mc Graw Hill, USA 1999;

Sede da Universidade Virtual africana

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Escritório Regional da Universidade Virtual Africana em Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org