Universidade Virtual Africana

INFORMÁTICA APLICADA: CSI 3301

INTRODUÇÃO AOS SISTEMAS OPERATIVOS

Flavio Semedo

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 significativamente 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 Marfim, 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

Flavio Semedo

Par revisor(a)

Elisabeth Andrade

UVA - Coordenação Académica

Dr. Marilena Cabral

Coordenador Geral Programa de Informática Aplicada

Prof Tim Mwololo Waema

Coordenador do módulo

Victor Odumuyiwa

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) a Introdução aos Sistemas Operativos

Este curso apresentará aos alunos os Sistemas Operativos mais populares e modernos. O foco do curso será em Sistemas Operativos baseados em LINUX/UNIX, embora também vai-se aprender sobre Sistemas Operativos alternativos, incluindo o Windows. O curso terá início com uma visão geral da estrutura de Sistemas Operativos. Ao longo das apresentações das unidades, será discutido a história dos computadores modernos, analisando em detalhes cada um dos principais componentes de um Sistema Operativo, tais como, processos, gestão de memórias, threads, ficheiro de sistemas, etc. Um Sistema Operativo é funciona como um sistema intermediário entre o hardware e o software. Estes, tem como propósito fornecer um ambiente em que os utilizadores do sistema possa executar as suas atividade de forma eficaz e eficiente.lors de la conception.

Pré-requisitos

O curso tem como objeto de estudo, familiarizar os alunos com Sistemas Operativos como um gestor de recursos e relacionar o Sistema Operativo com os utilizadores, aplicações e hardware. Para este curso, acredita-se que os alunos já possuem uma boa compreensão de arquitetura e organização de computadores, bem como, conhecimento e habilidades de programação e estruturas de dados (C, C++, Java).

Materiais

Os materiais necessários para completar este curso incluem:

- Um computador pessoal com os sistemas operativos mais populares (Windows, Linux, Unix)
- Um navegador Web
- onexão à internet

Objetivos do Curso

Após concluir este curso, o(a) aluno(a) deve ser capaz de:

- Explicar a função de um Sistema Operativo e como é utilizado.
- Identificar os vários componentes de um computador e como eles interagem com um Sistema Operativo.
- Ilustrar os diferentes tipos de Sistemas Operativos no mercado

- Definir e diferenciar os processos e threads e as suas importâncias.
- Descrever a sincronização e sincronização entre processos.
- Entender a gestão da memória, as sua hierarquias, interações com o Sistema Operativo e alocação dinâmica.
- Saber os conceitos relacionados com ficheiros de sistema e as sua caraterísticas.

Unidades

Unidade 0: Diagnóstico

Nesta unidade, será apresentada as metodologias de avaliação, bem como, a avaliação dos conhecimentos necessários como pré-requisito deste curso.**Unidade 1: introdução aos sistemas operativos**

Nesta unidade, será destacado os princípios de um sistema de operativo. O objetivo essencial desta unidade, é fornecer aos alunos um tour sobre as principais componentes de um sistema operativo e descrever os conceitos básicos na organização de computadores.

Unidade 2: Processo e Threads

Nesta unidade, será discutida dois importantes blocos de construção central de um sistema operativo. Processos que são instâncias de um programa de computador em execução e a thread como uma tarefa específica dentro de um processo? programa.

Unidade 3: Gestão de Memórias

Nesta unidade, será abordada a gestão de memórias, o processo de alocação e a relação do mesmo com os outros componentes de um computador. Por último, será descrito os principais tópicos relacionados com o acesso a memória.

Unidade 4: Sistema de Ficheiros

Nesta unidade, será apresentada uma visão geral dos sistemas de ficheiros, métodos de alocação de ficheiros, bem como os seus algoritmos de alocação no disco.

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

	Avaliação	**Nota**	**Descrição**
1	Avaliação Sumativa	50%	Pontuação de 20/100. Duração: pelo menos 03:00. Esta avaliação pode ser, de preferência um projeto para a avaliação do desenvolvimento das habilidades adquiridos.
2	Unidade 1 (Avaliação Formativa)	10%	Pontuação de 20/100. Avaliação de projetos desenvolvidos durante a apresentação da unidade
3	Unidade 2 (Avaliação Formativa)	10%	Pontuação de 20/100. Avaliação de projetos desenvolvidos durante a apresentação da unidade
4	Unidade 3 (Avaliação Formativa)	15%	Pontuação de 20/100. Avaliação de projetos desenvolvidos durante a apresentação da unidade
5	Unidade 4 (Avaliação Formativa)	15%	Pontuação de 20/100. Avaliação de projetos desenvolvidos durante a apresentação da unidade

Grelha de avaliação

Nota 100	Nota 20	Apreciação dos resultados
90-100	>=18	**Muito bom.** Resultados acima média
80-89	[16-18[	Bom >. Resultados além das expectativas do curso
70-79	[14-16[	Bom. Atingiu os objetivos do curso
60-69	[12-14[	Suficiente >. Resultado aceitável
50-59	[10-12[	Suficiente. Atingiu os objetivos parcialmente. As atividades de correção são necessários para melhor assimilar as atividades e desenvolver as suas habilidades
<=49	<10	Insuficiente. O aluno deve repetir o módulo

Calendarização

Unidade	Temas e Atividades	Estimativa do tempo
Sessão 1	Unidade 0: Diagnóstico Esta sessão é reservada para a apresentação do curso: • Os objetivos e competências do curso • Os pré-requisitos • O desenvolvimento do curso • Conteúdo • Trabalho e prazos para a apresentação e a apresentação do sistema de classificação • Recursos disponíveis sobre o curso Distribuição do tempo de trabalho: • Compreensão dos objetivos e pré-requisitos do curso (2:00) • Atividades da unidade (02:00) • Revisão das atividades do curso (02:00)	16H00

Sessão 2	• Unidade 1: Introdução aos Sistemas Operativos • Leitura dos conteúdos da unidade 1 • Os alunos devem efetuar as atividades de aprendizagem da Unidade 1 • Distribuição do tempo de trabalho: • Leituras dos conteúdos (06:00) • Atividade do curso (8:00) • Avaliação da unidade (02:00) • Pesquisas e consultas os recursos Web adicionais (2:00)	25H00
Sessão 3	Unidade 2: Processos e Threads • Leitura dos conteúdos da unidade 2 • Os alunos devem efetuar as atividades de aprendizagem da Unidade 2 Distribuição do tempo de trabalho: • Leituras dos conteúdos (06:00) • Atividade do curso (8:00) • Avaliação da unidade (02:00) • Pesquisas e consultas os recursos Web adicionais (2:00)	25H00
Sessão 4	Unidade 3: Gestão de Memória • Leitura dos conteúdos da unidade 3 • Os alunos devem efetuar as atividades de aprendizagem da Unidade 3 Distribuição do tempo de trabalho: • Leituras dos conteúdos (08:00) • Atividade do curso (10:00) • Avaliação da unidade (05:00) • Pesquisas e consultas os recursos Web adicionais (2:00)	25H00

Sessão 5	Unidade 4: Sistemas de Ficheiros • Leitura dos conteúdos da unidade 4 • Os alunos devem efetuar as atividades de aprendizagem da Unidade 4 Distribuição do tempo de trabalho: • Leituras dos conteúdos (06:00) • Atividade do curso (8:00) • Avaliação da unidade (02:00) • Pesquisas e consultas os recursos Web adicionais (2:00)	25H00
Sessão 6	Exame fim do semestre	4H00

Leituras e outros Recursos

As leituras e outros recursos deste curso são:

Unidade 0

Leituras e outros recursos obrigatórios:

- Linguagem C (23 ed.). Damas, L., (2014), (FCA Ed.), Portugal.
- Arquiteturas de Computadores, (3 ed.), Delgado, J., Ribeiro, C., (2008), (FCA Ed.), Portugal.
- Computer Organization and Design (3 ed.), Patterson, D., Hennessy, J., (2005), (Elsevier Inc), USA.

Unidade 1

Leituras e outros recursos obrigatórios:

- Linguagem C (23 ed.). Damas, L., (2014), (FCA Ed.), Portugal.
- Arquiteturas de Computadores, (3 ed.), Delgado, J., Ribeiro, C., (2008), (FCA Ed.), Portugal.
- Computer Organization and Design (3 ed.), Patterson, D., Hennessy, J., (2005), (Elsevier Inc), USA.
- Sistemas Operativos, Marques, J., Ferreira, P., Ribeiro, C., Veiga, L., Rodrigues, R., (2009), (FCA Ed.), Portugal. Capitulo 1 e 2.

- Sistema Operacionais (2 ed.). Tanembaum, A., & Woodhull, A. (2002), (Artmed, Ed.) Brasil. Capitulo 1.
- Operating System Concepts (9th ed.), Silberschatz, A., Galvin, P., & Gane, G. (2013), (Willey, Sons, & Inc, Eds.) USA. Capitulo 1 e 2.

Unidade 2

Leituras e outros recursos obrigatórios:

- Linguagem C (23 ed.). Damas, L., (2014), (FCA Ed.), Portugal.
- Arquiteturas de Computadores, (3 ed.), Delgado, J., Ribeiro, C., (2008), (FCA Ed.), Portugal.
- Computer Organization and Design (3 ed.), Patterson, D., Hennessy, J., (2005), (Elsevier Inc), USA.
- Sistemas Operativos, Marques, J., Ferreira, P., Ribeiro, C., Veiga, L., Rodrigues, R., (2009), (FCA Ed.), Portugal. Capitulo 3.
- Sistema Operacionais (2 ed.). Tanembaum, A., & Woodhull, A. (Artmed, Ed.) Brasil. Capitulo 2.
- Operating System Concepts (9th ed.), Silberschatz, A., Galvin, P., & Gane, G. (2013), (Willey, Sons, & Inc, Eds.) USA. Capitulo 3 e 4.

Unidade 3

Leituras e outros recursos obrigatórios:

Linguagem C (23 ed.). Damas, L., (2014), (FCA Ed.), Portugal.

- Arquiteturas de Computadores, (3 ed.), Delgado, J., Ribeiro, C., (2008), (FCA Ed.), Portugal.
- Computer Organization and Design (3 ed.), Patterson, D., Hennessy, J., (2005), (Elsevier Inc), USA.
- Sistemas Operativos, Marques, J., Ferreira, P., Ribeiro, C., Veiga, L., Rodrigues, R., (2009), (FCA Ed.), Portugal. Capitulo 7 e 8.
- Sistema Operacionais (2 ed.). Tanembaum, A., & Woodhull, A. (Artmed, Ed.) Brasil. Capitulo 4.
- Operating System Concepts (9th ed.), Silberschatz, A., Galvin, P., & Gane, G. (2013), (Willey, Sons, & Inc, Eds.) USA. Capitulo 8 e 9.

Unidade 4

Leituras e outros recursos obrigatórios:

- Linguagem C (23 ed.). Damas, L., (2014), (FCA Ed.), Portugal.
- Arquiteturas de Computadores, (3 ed.), Delgado, J., Ribeiro, C., (2008), (FCA Ed.), Portugal.

- Computer Organization and Design (3 ed.), Patterson, D., Hennessy, J., (2005), (Elsevier Inc), USA.
- Sistemas Operativos, Marques, J., Ferreira, P., Ribeiro, C., Veiga, L., Rodrigues, R., (2009), (FCA Ed.), Portugal. Capitulo 9 e 10.
- Sistema Operacionais (2 ed.). Tanembaum, A., & Woodhull, A. (Artmed, Ed.) Brasil. Capitulo 5.
- Operating System Concepts (9th ed.), Silberschatz, A., Galvin, P., & Gane, G. (2013), (Willey, Sons, & Inc, Eds.) USA. Capitulo 11 e 12.

Unidade 0: Diagnóstico

Introdução à Unidade

A multiplicação da força intelectual da humanidade nesta geração, foi conduzido pela origem e penetração dos computadores no quotidiano, com informações tendo seu lugar em diferentes atividades do dia-a-dia. O propósito desta unidade é verificar a compreensão dos conhecimentos que o aluno possui, como pré-requisito deste curso. Antes de explorar os detalhes de como funciona um computador é necessário saber, em geral, a sua estrutura organizacional. Durantes as apresentações das unidades subsequentes, serão apresentadas os algoritmos para a criação de processos e programação de threads, gestão de memória e ficheiros. No entanto, é necessário que os alunos possuem conhecimentos das linguagens de programação.

Avaliação da Unidade

Verifique a sua compreensão!

Perguntas de autoavaliação

Parte 1: Arquitetura de Computadores

1. Qual é a componente de um computador em que todos os programas em execução e dados associados residem?
2. Cite e descreve a componente de um processador que executa as operações aritméticas?
3. Utilizando as categorias na lista abaixo, classifique os exemplos a seguir. Utilize as letras à esquerda das palavras na resposta.

    a) Aplicações

    b) Dispositivo de entrada

    c) Circuitos integrados

    d) Dispositivo de saída

    e) Linguagens de alto nível

    f) Computadores pessoais

    g) Semicondutores

    h) Supercomputadores

    i) Softwares do sistema

1. LCD _____
2. Assembler _____
3. Compilador _____
4. DRAM _____
5. Java _____
6. Microprocessador _____
7. Sistemas Operativos _____
8. Power Point _____
9. Rato do computador _____
10. C++ _____
11. HP PC _____
12. Excel _____
13. Impressora_____

4. Se um computador emitir 30 solicitação de rede por segundo e cada pedido é a média de 64KB, 100Mbit Ethernet é o suficiente?
5. Qual é o programa que converte uma versão simbólica de um instrução em uma versão binária?
6. Qual é o circuito integrado comumente utilizado para construir a memória principal?
7. Qual é o programa que gere os recursos de um computador para o benefício dos programas que são executados neste mesmo computador?
8. Qual é o programa que traduz uma notação de alto nível para a linguagem assembler?

Parte 2: Programação em linguagem C

1. Implemente um programa que adicione um valor (ex. 5000$00) ao salário de um funcionário, caso este seja inferior a 20000$00.
2. Implemente um programa que recebe n números de apostas. Dado o valor do prémio do concurso (ex. prémio é de 3000$00), um valor da aposta e o número de n apostas, determine qual é o índice que mais se aproxima do prémio, sem o ultrapassar, assumindo que não há valores repetidos. No caso de não existir nenhuma aposta válida (com valor maior ou igual ao prémio) escreva "aposta inválida".

3. Escreva um programa que mostre todos os parâmetros que recebeu a partir da linha de comandos.

4. Escreva um programa que lê um número a partir da linha de comando e informa se é maior que a metade do maior inteiro possível.

5. Escreva um programa que solicite dois números 4 números ao utilizador e apresente no ecrã o resultado da sua soma e o dobro de cada um deles

6. Escreva um programa que calcule a média de três módulos do curso Ciências de Computação. Veja o resultado da execução da aplicação:

```
Este programa calcula a média total de 3 módulos do curso Ciências de Computação
Inserir a nota #1: 14
Inserir a nota #2: 12
Inserir a nota #3: 17
A média é 14.333333333333334

Gostarias de continuar a calcular a media dum outro curso?
Inserir S se Sim ou N se não: S
Inserir a nota #1: 15
Inserir a nota #2: 18
Inserir a nota #3: 14
A média é 15.666666666666666

Gostarias de continuar a calcular a media dum outro curso?
Inserir S se Sim ou N se não: N
```

7. Escreva um programa que permite inserir um item em um *ArrayList*? Veja o resultado da execução da aplicação:

```
Index: 0 Nome: Adilson
Index: 1 Nome: Carlitos
Index: 2 Nome: Carla
Carlos foi adicionado no index 1. Aqui se encontra o item.
Index: 0 Nome: Adilson
Index: 1 Nome: Carlos
Index: 2 Nome: Carlitos
Index: 3 Nome: Carla
```

8. Escreva um programa que remove um item em um ArrayList? Veja o resultado da execução da aplicação:

```
Index: 0 Nome: Adilson
Index: 1 Nome: Carlitos
Index: 2 Nome: Carla
Um item foi removido. Veja a lista atualizada.
Index: 0 Nome: Adilson
Index: 1 Nome: Carla
```

9. Escreva um programa que recebe o número de um cliente cujo tamanho consiste em sete (7) carateres de três (3) letras seguido de 4 números. Veja o resultado da execução da aplicação:

```
Insere o número do cliente em formato LLLNNNN
(LLL = letras and NNNN = numeros): FSR1256
Este é um número de cliente válido
```

10. Escreva um programa que abra um ficheiro e indique quantos bytes este contém, obtendo este valor através do acesso sequêncial.

11. Implemente uma função char * segundo(char *s) unicamente através do uso de apontadores.

Unidade 1: Introdução Aos Sistemas Operativos

Introdução à Unidade

Nesta unidade, serão abordados os conceitos fundamentais sobre os sistemas operativos (SO). Os Sistemas Operativos funcionam como uma plataforma de troca de informações entre o hardware do seu computador e as aplicações em execução. Primeiramente, serão introduzidos os tipos de sistemas operativos existentes no mercado e os mais utilizados. Em seguida, será visto a estrutura geral e as funções básica de um SO. Por último, serão apresentadas algumas atividades relativamente ao fundamentos de Sistemas Operativos.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Explicar o conceito de sistemas operativos, os objetivos e como se utiliza.
2. Mostrar os diferentes tipos de Sistemas Operativos e descrever as mais utilizadas.
3. Identificar e descrever os componentes de um Sistema Operativo e a forma como interagem com um Sistema Operativo.
4. Descrever a diferença entre um Sistema Operativo de 32 bits vs 64 bits.

Termos-chave

Sistema Operativo: Controla todos os recursos de um computador e fornece a base como uma aplicação pode ser executada.

CPU: Unidade de Processamento Central

Hardware: termo utilizado para descrever os dispositivo físico de um computador, tal como, hard drive que é fisicamente conectado a um computador

Software: Termo utilizado para descrever uma coleção de programas de computadores

Atividades de Aprendizagem

Atividade 1.1 – Fundamentos de um Sistema Operativo

Introdução

Em geral, todos os utilizadores que utilizam computadores, já tem experiência com um sistema operativo. Sempre que um computador é iniciado, neste processo de inicialização em um determinado período o Sistema Operativo estará a comando. Depois do computador iniciar e estiver disponível para o uso pessoal, os utilizadores podem interagir com o computador através de uma interface gráfica ou de linha de comando. Há diversas tarefas que podem ser efetuadas por parte dos utilizadores, tais como, instalar/desinstalar um programa, executar um programa, entre outras tarefas. Um Sistema Operativo, pode ser considerado, também um gestor de recursos. Desta forma, determina que recursos do computador serão utilizados para resolver um problema e a ordem em que estes serão utilizados. De uma forma geral, um sistema operativo desempenha três principais tipos de funções.

- **Alocação** e atribuição de recursos do sistema, tais como dispositivos de entrada/saída, software, unidade central de processamento, etc.
- **Agendamento:** Esta função coordena os recursos, jobs e segue um determinado tipo de prioridade.
- **Monitoramento:** Esta função monitora e controla as atividades no sistema de computador. Também, mantém os registos das operações, notifica os utilizadores ou operadores de computador de qualquer anormalidade ou erros.

Perspetiva Histórica

Na década de 1940, os primeiros sistemas eletrónicos digitais não tinham um sistemas operativo. Nesta época, os computadores eram tão primitivas comparados com os de hoje. Eram programas que funcionavam a nível de hardware baseadas em uma linguagem máquina. Eventualmente, as linguagens de máquina (que consiste em uma String formado por dígitos binários de 0s e 1s), foram introduzidas para acelerarem o processo de programação. Os sistemas na década de 1950 apenas executavam uma tarefa de cada vez. É permitida apenas uma única pessoa de cada vez para utilizar a máquina. Todos os recursos da máquina, encontravam à disposição de um utilizador.

Originalmente, cada utilizador escreve o código necessário para implementar uma aplicação específica, incluindo as instruções de entrada/saída a nível da máquina altamente detalhadas. A codificação de entrada/saídas necessárias para implementar de funções básicas foi consolidada em um sistema de controlo de entrada/saída (IOCS). A implementação do sistema de controle de entrada / saída pode ter sido o início do conceito de um Sistema Operativo. O utilizador tem o controle total sobre o armazenamento da memória principal e, como resultado, este sistema é conhecido como sistema de alocação contíguo (é o esquema mais simples de alocar e armazenar os ficheiros no disco).

Para cada utilizador do sistema era atribuído a um job. Jobs requer geralmente um tempo de execução considerável toda a vez que o sistema operativo era executado.

Naquele tempos, um único grupo de pessoas projetavam, construíam, programavam e operavam cada máquina. Toda a programação era feita em uma linguagem pura, frequentemente ligando com o fios painéis de conectores para controlar as funções básicas da máquina. Pouco se aplicava as linguagens de programação que na época era desconhecida por muitas pessoas.

No inicio da década de 50, mudou-se a rotina devido a introdução de cartões perfurados. Isto, permitiu a gravação de programas em cartões e lê-los em vez de utilizar cabos e conectores.

Em geral, todo este processo requeria um gasto enorme de tempos, porque levava muito tempo para um utilizar processar vários jobs no sistema.

Na década de 70 houve significativamente vários desenvolvimentos no âmbito de sistemas operativos. Os sistemas de comunicações melhoraram, uma vez que, o protocolo TCP/IP (Protocolos de transmissão dos Dados/ Internet Protocolo) foi amplamente utilizado pelas forças militares e as universidades no ambiente de computação.

Atualmente, os Sistemas Operativos não só tem que lidar com a interconexão entre as redes mas, também, tem que lidar com a segurança. Neste sentido, foi necessário codificar os dados proprietários ou privadas que assim mesmo que os dados forem comprometidos, não era para uma pessoa comum com conhecimentos básicos sobre computadores as acederem.

Alguns dos Sistemas Operativos mais utilizados foram desenvolvidos neste período, como o sistema IBM, VM e sistemas UNIX. Sistema Operativos UNIX é particularmente notável porque este é o único sistema que têm-se implementado com sucesso em todo o tipo de computador, desde de microcomputadores a supercomputadores. A figura 1, em uma cenário simples descreve o que é um sistema de computador.

Figura 1: Sistema de computador, (Niu, 2003)

Tipos de Sistemas Operativos

Os computadores têm-se progredido e desenvolvidos, assim como os sistemas operativos. Abaixo encontra-se uma lista básica dos tipos de sistemas operativos e alguns exemplos que se enquadram cada tipo.

GUI - *Graphical User Interface,* um Sistema Operativos GUI contém gráficos e ícones e é comummente navegado utilizando um Mouse (Rato) de computador. Exemplos de Sistemas Operativos GUI são:

- System 7.x
- Windows 98
- Windows CE

Multiprocessamento - Um sistema operativo capaz de suportar e utilizar mais do que um processador de computador. Exemplos de Sistemas Operativos que se enquadram nessa categoria:

- Linux
- Unix
- Windows 8

Multitarefa - Um Sistema Operativo de múltiplas capaz de permitir múltiplos processos de software a serem executados em paralelo. Exemplos de Sistemas Operativos que se enquadram nessa categoria:

- Linux
- Unix
- Windows XP

Na lista seguinte, encontra-se a listagem de diferentes Sistemas Operativos disponíveis atualmente no mercado, e as datas em que estes foram lançadas, as plataformas de desenvolvimento, e as empresas responsáveis.

Sistemas Operativos	Data de Lançamento	Plataforma	Desenvolvedores
AIX e AIXL	História de Unix e Linux	Vários	IBM
AmigaOS	História	Amiga	Commodore
Android	Novembro de 2007	Mobile	Google
BSD	História de Unix e Linux	Vários	BSD
Caldera Linux	História de Unix e Linux	Vários	SCO

Corel Linux	História de Unix e Linux	Vários	Corel
CP/M	História CP/M	IBM	CP/M
Debian Linux	História de Unix e Linux	Vários	GNU
DUnix	História de Unix e Linux	Vários	Digital
DYNIX/ptx	História de Unix e Linux	Vários	IBM
iOS	2010	Mobile	Apple
IRIX	História de Unix e Linux	Vários	SGI
Kondara Linux	História de Unix e Linux	Vários	SGI
Linux	História de Unix e Linux	Vários	Mandrake
MAC OS 8	História dos Sistemas Operativo da Apple	Apple Macintosh	Apple
MAC OS 9	História dos Sistemas Operativo da Apple	Apple Macintosh	Apple
MAC OS 10	História dos Sistemas Operativo da Apple	Apple Macintosh	Apple
MAC OS X	História dos Sistemas Operativo da Apple	Apple Macintosh	Apple
Mandrake Linux	História de Unix e Linux	Vários	Mandrake
MINIX	História de Unix e Linux	Vários	MINIX
MS-DOS 1.x	História MS-DOS	IBM	Microsoft
MS-DOS 2.x	História MS-DOS	IBM	Microsoft
MS-DOS 3.x	História MS-DOS	IBM	Microsoft
MS-DOS 4.x	História MS-DOS	IBM	Microsoft
MS-DOS 5.x	História MS-DOS	IBM	Microsoft
MS-DOS 6.x	História MS-DOS	IBM	Microsoft
Ultrix	História de Unix e Linux	Vários	Ultrix
Unisys	História de Unix e Linux	Vários	Ultrix
Unix	História de Unix e Linux	Vários	Bell Labs
UnixWare	História de Unix e Linux	Vários	UnixWare
VectorLinux	História de Unix e Linux	Vários	VectorLinux
Windows 2000	História da Microsoft Windows	IBM	Microsoft

Windows 2003	História da Microsoft Windows	IBM	Microsoft
Windows 3.x	História da Microsoft Windows	IBM	Microsoft
Windows 7	História da Microsoft Windows	IBM	Microsoft
Windows 8	História da Microsoft Windows	IBM	Microsoft
Windows 95	História da Microsoft Windows	IBM	Microsoft
Windows 98	História da Microsoft Windows	IBM	Microsoft
Windows 10	História da Microsoft Windows	IBM	Microsoft
Windows CE	História da Microsoft Windows	IBM	Microsoft
Windows ME	História da Microsoft Windows	IBM	Microsoft
Windows NT	História da Microsoft Windows	IBM	Microsoft
Windows Vista	História da Microsoft Windows	IBM	Microsoft
Windows XP	História da Microsoft Windows	IBM	Microsoft
Xenix	História de Unix e Linux	Vários	Microsoft

Tabela 1: Lista de Sistemas Operativos (Computer Hope, 2016)

Visão geral do hardware do computador

Sistema de computadores, em geral, possuí diferentes tipos de hardware, placa-mãe, adaptadores, impressoras, etc. Para desenvolver um Sistema Operativo é necessário conhecer o hardware da máquina. Placa-mãe não é algo que fornece uma função específica, em vez disso, é uma coleção de todos os tipos de slots e módulos. Para fazer com que esses componentes funcionam em conjunto, a placa-mãe contém algum tipo de conexão física entre esses componentes, isto é, o que chamamos de barramento do sistema. Esses componentes podem ser divididos em vários grupos: CPU, memória, módulos I/O e bus de sistema.

Para descrever esses componentes e suas conexões, apresenta-se a seguinte exibição de nível superior:

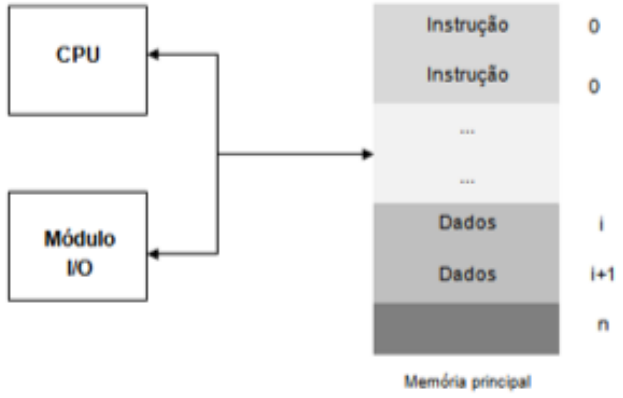


Figura 2: Conexões entre os componentes, ((Niu, 2003))

A figura 2, mostra a conexão entre a memória e o CPU com base em um conjunto de dados transferidos entre os dois componentes. A memória consiste em um conjunto de endereços definidos sequêncialmente. Cada endereço pode ser um byte constituído por 8 bits, ou uma palavra de 16 bits. Para aceder os dados na memória o CPU faz o uso de dois registos internos:

Memory address register - MAR que especifica o endereço na memória para o próximo ler/escrever.

Memory buffer register - MBR contém dados para ser escritos na memória ou dados para ser lidos a partir da memória.

Em relação aos dispositivos de E/S, os dados são transferidos a partir de um dispositivo externo para a memória e o CPU, vice versa. Estes contém buffers para armazenar os dados temporariamente até serem enviados de volta a um dispositivo externo ou secundário.

Diferença entre Sistema Operativo de 32 bits vs 64 bits

À medida que o número de bits aumentam existem duas grandes vantagens.

- Quanto maior for o número de bits significa que os dados podem ser processados em uma grande quantidade e com mais precisão.
- Quanto maior for o número de bits significa que o sistema pode apontar para o endereço uma grande quantidade de números de localizações na memória física.

Os Sistemas Operativos de 32 bits foram amplamente utilizados porque podiam endereçar 4 GB de memória. Algumas aplicações modernas exigem mais do que 4 GB de memória para concluir suas tarefas de modo que, sistemas de 64 bits estão agora se tornando mais potentes, porque podem potencialmente endereçar cerca de até quatro bilhões de vezes.

Desde 1995, quando o Windows 95 foi lançado com suporte para aplicações de 32 bits, a maioria dos softwares e Sistema Operativo tem sido de 32 bits compatíveis.

O problema, é que, enquanto a maioria dos softwares disponíveis hoje é de 32 bits, os processadores que compramos são quase todos de 64 bits.

A questão que se coloca é a seguinte:

Por quanto tempo será está transição de Sistemas Operativos de 32 bits para 64 bits?

A questão principal é que os computadores funcionam a partir do hardware, como o processador (CPU), através do Sistema Operativo ao nível mais alto que são as aplicações. Assim, o hardware do computador foram concebidos em primeiro lugar, em seguida a combinação dos Sistema Operativo com o hardware são desenvolvidos, e por último as aplicações.

Pode-se ver que, algumas décadas atrás, houve a transição de 16 bits para 32 bits do Windows para processadores de 32 bits. Este demorou aproximadamente 10 anos (1985 a 1995) para obter um Sistema Operativo de 32 bits. O mesmo cenário acontece atualmente, que é a transição de 32 bits para 64 bits.

Conclusão

Nesta atividade, introduziu-se os conceitos fundamentais de um Sistema Operativo e a visão geral dos marcos históricos para a progressão e desenvolvimento de um Sistema Operativo. Em seguida, foi descrita os tipos de sistemas e a principal diferença entre um sistema de 32 bits vs 64 bits. Tipos de so, visao geral do hardware

Avaliação

1. Define um Sistema Operativo.
2. Quais são as principais funções de um Sistema Operativo?
3. Cite 3 tipos de Sistemas Operativos mais utilizados atualmente.
4. Qual é a diferença entre um sistema de 32 bits e 64 bits.
5. Descreve as 3 camadas que constituí um Sistema Operativo?

Atividade 1.2 – Estrutura de um Sistema Operativo

Introdução

Pode-se ver a vantagem de um Sistema Operativo em diferentes pontos de vista. Por um lado, os serviços que o sistema fornece e por outro lado, a interface disponível para os utilizadores e programadores. E, o terceiro ponto, são os seus componentes e as interconexões.

Nesta atividade serão explorados os três aspetos importantes de um Sistema Operativo. Também, são considerados quais os serviços são fornecidos pelo SO, como são fornecidos e as diferentes metodologias utilizadas para a conceção deste sistema.

Serviços de um Sistema Operativo

Segundo as definições anteriormente apresentadas, um Sistema Operativo fornece um ambiente de desenvolvimento para a execução de programas. Fornece certos tipos de serviços para as aplicações e para os utilizadores das aplicações. Um serviço difere de um Sistema Operativo para outro, mas pode-se identificar classes comuns. Estes serviços faz com que as tarefas de programação sejam mais fáceis de executar.

Os serviços de um Sistema Operativo, fornece funções que ajudam os utilizadores:

- **Interface do Utilizador** – Geralmente, quase todos os Sistemas Operativos têm um Interface de utilizador (UI).
- **Execução do programa** – O sistema deve ser capaz de carregar um programa na memória e executá-lo.
- **Input/Output** – Um programa em execução talvez requere I/O, que pode envolver ficheiro ou um dispositivo I/O.
- **Manipulação de um sistema de ficheiro** – Obviamente, um programa precisa ler e escrever um ficheiro ou diretório.
- **Comunicação** – Circunstâncias em que um processo precisa trocar informação com outro processo.
- **Deteção de erros** – O Sistema Operativo precisa estar constantemente preparado para possíveis erros.
- **Alocação de recursos** – Quando há múltiplo utilizadores e jobs executados em paralelo, recursos devem ser alocados para cada um deles.
- **Conta do utilizador** – Tem que gerir os recursos utilizados para cada utilizador.
- **Proteção e segurança –** Os proprietários das informações armazenadas em sistema multiutilizador pode querer controlar a utilização dessas informações.

Interface do Utilizador em um Sistema Operativo

Há duas abordagens fundamentais para os utilizadores a interagirem com o Sistema Operativo. A primeira técnica é a linha de comando (command line), que permite os utilizadores digitar os comandos que serão executados pelo o Sistema Operativo. E, a segunda técnica permite que os utilizadores interagem com o sistema via uma Interface Gráfica do utilizador.

Linha de Comando

Alguns Sistemas Operativos incluem o interpretador de comando no kernel. Outros, tal com o Windows e UNIX, trata estes interpretadores de comandos como um programa especial que é executado quando uma job é iniciada. Nos sistemas com múltiplos interpretadores de comandos para escolha, são conhecidos por Shell.

Interface Gráfico de Utilizador

A segunda forma de interagir com o Sistema Operativo é através do GUI, interface amigável. Como introduzido nos parágrafos anteriores, um Sistema Operativos GUI contém gráficos e ícones e é comummente navegado utilizando um Mouse de computador.

Portanto, a escolha dessa abordagem, depende da preferência dos utilizadores.

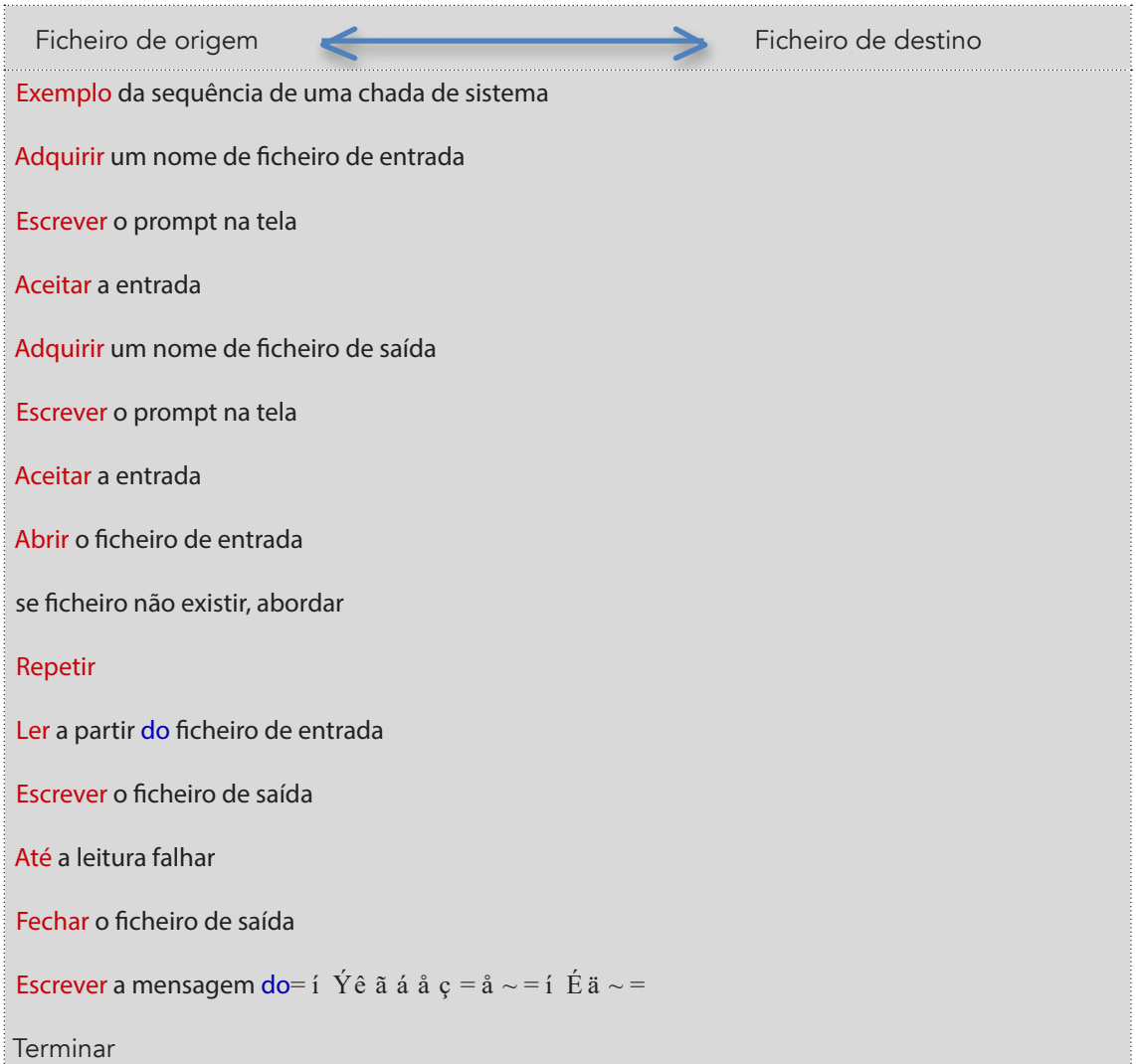
Chamada do Sistema (System Call)

As chamadas do sistema fornece uma interface para os serviços disponibilizados por um Sistema Operativo. Essas chamadas são rotinas geralmente disponíveis escritos em C e C ++, embora algumas tarefas de baixo nível precisam ser escritas utilizando a linguagem máquina (instruções assembler).

Um exemplo de uma chamada de sistema é, escrever um programa que lê os dados em um ficheiro e copia para outro ficheiro. A abordagem aqui, é perguntar ao utilizador o nome dos dois ficheiros. Em sistema interativo esta abordagem requererá uma sequência de chamadas de sistemas:

1. Escrever a mensagem prompt na tela.
2. Ler os carateres que define os dois ficheiros através do teclado.
3. Uma vez que os nome dos ficheiros são obtidos, o programa deve abrir o ficheiro de entrada e criar o ficheiro de saída.

Cada operação requer um tipo de chamada de sistema e, há também possíveis erros durante a sua execução. Em seguida é apresentado como a chamada de sistema é utilizada:



Objetivos de um Sistema Operativo

A fundação de um Sistema Operativo pode ser visto em diferentes perspetivas. Principalmente, se essas perspetivas se concentrarem em utilizadores e níveis de sistema. Como é que o Sistema Operativo é percebida do ponto de vista do utilizador, hardware, e também um software?

Visão do sistema como gestor dos recursos

Para um sistema de computador, um Sistema Operativo é um programa que interage de perto com o hardware. Um sistema de computador tem um conjunto de meios para efetuar trocas, processar e armazenar dados. Assim, do ponto de vista da máquina, um Sistema Operativo pode ser considerado como alocador e gestor de recursos. Um número de solicitações de recursos que também podem, eventualmente, ser conflituantes no sistema e o Sistema Operativo deve decidir como atribuí-los a programas específicos e utilizadores para garantir o desempenho eficiente e justa do sistema. O Sistema Operativo também deve controlar os diferentes dispositivos de I/O e programas aplicacionais encontrados em um sistema de computador para que erros e uso indevido do sistema seja impedida.

Visão do utilizador como uma máquina virtual

A maioria dos utilizadores de computadores sentem na frente de um computador com a unidade do sistema, monitor, teclado e mouse. Em outros casos, os utilizadores podem se sentar em um terminal conectado a um mainframe ou minicomputador ou nos postos de trabalho ligados a outras estações de trabalho em rede. Em todos estes casos, os requisitos dos utilizadores variam. No primeiro caso, o sistema destina-se apenas para um utilizador para monopolizar todos os recursos; o objetivo é maximizar o trabalho que o utilizador está realizando. Assim, o Sistema Operativo é projetado para fornecer facilidade de uso, com algumas considerações para o desempenho, mas nenhuma consideração para a utilização de recursos. No segundo caso, em que os utilizadores estão sentados na frente de um terminal, há também outros utilizadores compartilhando os mesmos recursos em seus respetivos terminais. Para esses casos, o Sistema Operativo deve ser projetado de tal forma que a principal preocupação é a utilização de recursos para garantir todos os recursos disponíveis, tais como tempo de CPU, memória principal e I/O, são utilizados de forma eficiente e todos estão compartilhando os recursos de forma justa. Neste último caso que envolve uma estação de trabalho, os utilizadores têm recursos dedicados à sua disposição, mas também, partilhar recursos como servidores de ficheiros ou servidores de impressão com os outros. O Sistema Operativo para esses casos devem ser projetados para comprometer entre a utilização de recursos e usabilidade individual.

Considerando todas estas e outras circunstâncias de exigências dos utilizadores, um Sistema operativo é um sistema projetado para fornecer um ambiente easy-to-work-in para utilizadores e gerir todas as possíveis necessidades dos seus utilizadores, independentemente da escassez de recursos.

Desenvolver e Implementar um Sistema Operativo

A primeira questão em desenvolver e implementar um Sistema Operativo são as definições dos objetivos e as especificações. No nível superior, desenvolver um sistema , este será afetado pela escolha do hardware e o tipo de sistema: batch, time shared (tempo partilhado), utilizador único, multiutilizador, distribuição, tempo real, etc. Além do nível superior de desenvolvimento, os requisitos talvez podem ser complicado para especificar. No entanto, os requisitos podem ser divididos em dois grupos básicos: Objetivos do utilizador e objetivos do sistema.

Os utilizadores certamente desejam algumas propriedades em um sistema: O sistema deveria ser conveniente de utilizar, fácil de aprender e de utilizar, confiável, segura e rápida. Claro que, estas especificações não são particularmente úteis no desenvolvimento do sistema, desde de que não há um acordo geral de como adquiri-los.

Há, em suma, não há uma solução única para o problema definir os requisitos para um Sistema Operativo. A ampla gama de sistemas mostra que diferentes requisitos podem resultar em uma ampla variedade de soluções para diferentes ambientes.

Portanto, especificar e desenvolver um Sistema Operativo é uma tarefa altamente criativo.

Estrutura Simples de um Sistema Operativo

Muitos sistemas comercias não tem uma estrutura bem definida. Frequentemente, tais Sistemas Operativos iniciam-se com sistemas pequenos e limitados, e depois desenvolvem além do escopo de origem. OMS-DOS é um exemplo destes sistemas. Foi originalmente desenvolvido e implementado pela um pequeno número de pessoas que não tinham a ideia que tornaria popular. Foi concebida para fornecer a maior funcionalidade em poucos espaços e foi dividida em módulos.

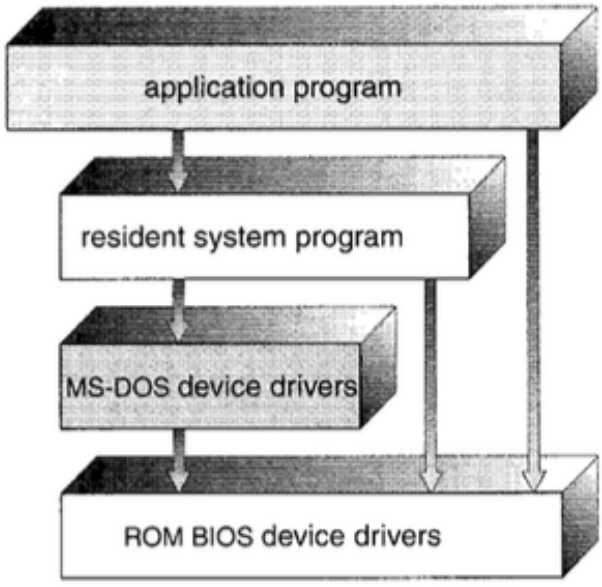


Figure 1: Estrutura do MS_DOS

Em MS-DOS, as interfaces e os níveis de funcionalidade não eram bem separados. Por exemplo, os programas são capazes de aceder as rotinas básicas do I/O para escrever diretamente para as unidades de exibição e de discos. É claro que, o MS-DOS também era limitado por hardware na sua era. A Intel 8088 não tinha capacidade de executar processos no modo dual e sem nenhuma proteção de hardware. Os desenvolvedores do MS-DOS não tinham nenhuma escolha mas sim, deixar a base do hardware acessível.

Um outro exemplo de uma estrutura limitada é o Sistema Operativo original UNIX. UNIX é um SO que inicialmente era limitado pela funcionalidades do hardware. Consistia em duas partes separadas: O kernel e os programas. O kernel que é dividida em uma série de interfaces e drivers de dispositivos, que tem sido adicionado e expandida no últimos anos como a UNIX tem evoluído. Pode-se ver que, o Sistema Operativo UNIX tem sido dividido em camadas. O kernel fornece o ficheiro de sistema, CPU scheduling (agendamento), gestão de memórias, e outras funções do Sistema Operativo através da chamadas de sistema (system calls).

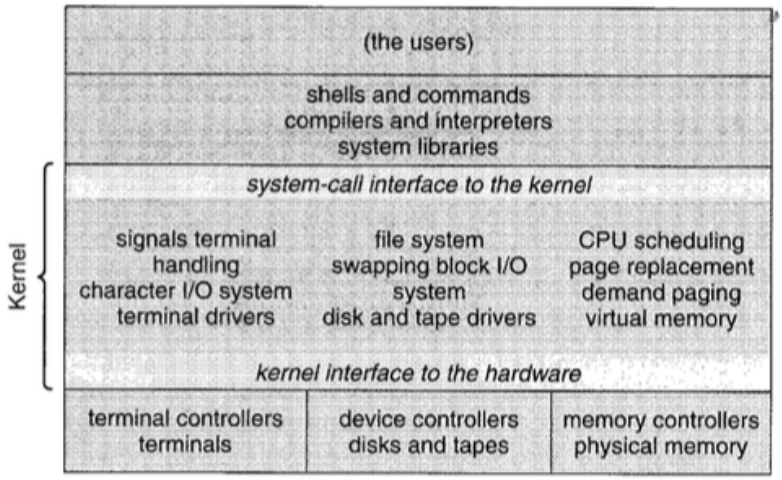


Figure 2: Estrutura do sistema UNIX

Componentes de um Sistema Operativo

A figura 3, ilustra os três componentes de um Sistema Operativo. Em geral são três camadas simples com o processador ou CPU a camada mais baixa e a camada aplicação como a mais alta, como mostrado abaixo:

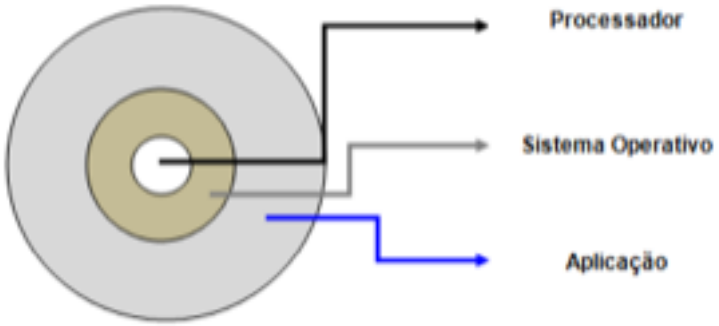


Figure 3: Camadas de um Sistema Operativo

Para executar um Sistema Operativo de 64 bits é necessário o suporte da camada mais baixa, CPU de 64 bits.

Para executar uma aplicação de 64 bits é preciso o suporte das camadas mais baixa, SO de 64 bits, e o CPU de 64 bits.

Microkernel

Nos meados de 1908, os pesquisadores da Universidade Carnegie Mellon, desenvolveu um Sistema Operativo chamado de Math com o kernel modularizado utilizando a abordagem Microkernel. Este método estrutura o Sistema Operativo removendo todos os componentes não essenciais do kernel implementando-os como sistemas e programas de níveis de utilizadores. O resultado foi um kernel pequeno. Há uma preocupação em definir qual é o serviço que deveria manter-se no kernel e qual deveria ser implementado no espaços de utilizador. Os Microkernels fornece mínimos processos e gestão de memória, em adição para a facilidade de comunicação.

Módulos

Talvez, a melhor metodologia para o desenvolvimento de um Sistema Operativo envolve a utilização das técnicas de programação orientada a objetos para criar um kernel modular. Aqui, um kernel tem um conjunto de componentes básicos e dinamicamente ligação em serviços adicionais mesmo durante o tempo de inicialização (booting) ou tempo de execução. Essa estratégia utiliza módulos dinamicamente carregáveis e é comum em implementações modernas de UNIX, Solaris, Linux e MAC OS X.

Máquinas Virtuais

A abordagem em camadas é levado à sua conclusão lógica no conceito de máquina virtual. A ideia fundamental por de trás de uma máquina virtual é para abstração do hardware de simples computador (CPU, memória, disco, etc) em um conjunto diferente de ambientes em execução, criando a ilusão de que cada ambiente em execução está sendo executado em um computador privado.

As razões para a criação de máquinas virtuais estão relacionadas com a capacidade de partilhar o mesmo hardware executado em ambientes diferentes.

Implementação

Embora o conceito de máquinas virtuais é útil, é complicado para implementar. Muito trabalho é requerido para fornecer uma máquina duplicada. A máquina virtual tem dois modos: modo utilizador e kernel como foi introduzido anteriormente. O software máquina virtual pode ser executado no modo kernel, desde de que é um Sistema Operativo. A máquina virtual por si só executa somente o modo utilizador.

Conclusão

Nesta atividade, foram introduzidos a estrutura de um SO e como são criadas e implementadas e os problemas encontrados durante esta processo.

Avaliação

1. Os serviços e funções fornecidos por um SO podem ser divididos em duas categorias principais. Descreve essas duas categorias e mostre como eles se diferenciam?
2. Qual é o propósito de um interpretador de comandos? Porque é geralmente dividido pelo kernel?

Atividade 1.3 – Laboratório

Como podemos interagir com o Sistema Operativo? Como é que o Sistema Operativo expõe os seus serviços? Esta atividade de laboratório incidirá sobre estas questões.

Use qualquer um dos Sistemas Operativo Linux, como o Ubuntu, Red Hat, Fedora, CentOS, Kali, ou qualquer outro para executar as atividades de laboratório neste módulo. O código-fonte são testados no Ubuntu 14.04 neste módulo. Executar sua máquina e faça o login. Utilizar o terminal para ter acesso ao Shell.

O Shell é um interpretador de linha de comando que funciona como uma interface primária entre um utilizador e o Sistema Operativo. É um processo (programa em execução) que lê comandos. Não faz parte do Sistema Operativo, mas faz o uso intensivo de muitos recursos do Sistema Operativo.

Como funciona a Shell?

1. Quando um utilizador faz o login no sistema, um Shell é iniciado.
2. O Shell exibe um aviso que informa ao utilizador que ele está pronto para aceitar um comando.
3. Quando o utilizador insere um comando, o Shell cria um processo filho que executa o programa para o comando correspondente
4. Quando o processo filho estiver em execução, o Shell espera que o processo filho termine.
5. Quando o processo filho terminar, o Shell exibe o prompt e tenta ler o próximo comando.

Durante a execução do Shell, será solicitado a seguinte interface:

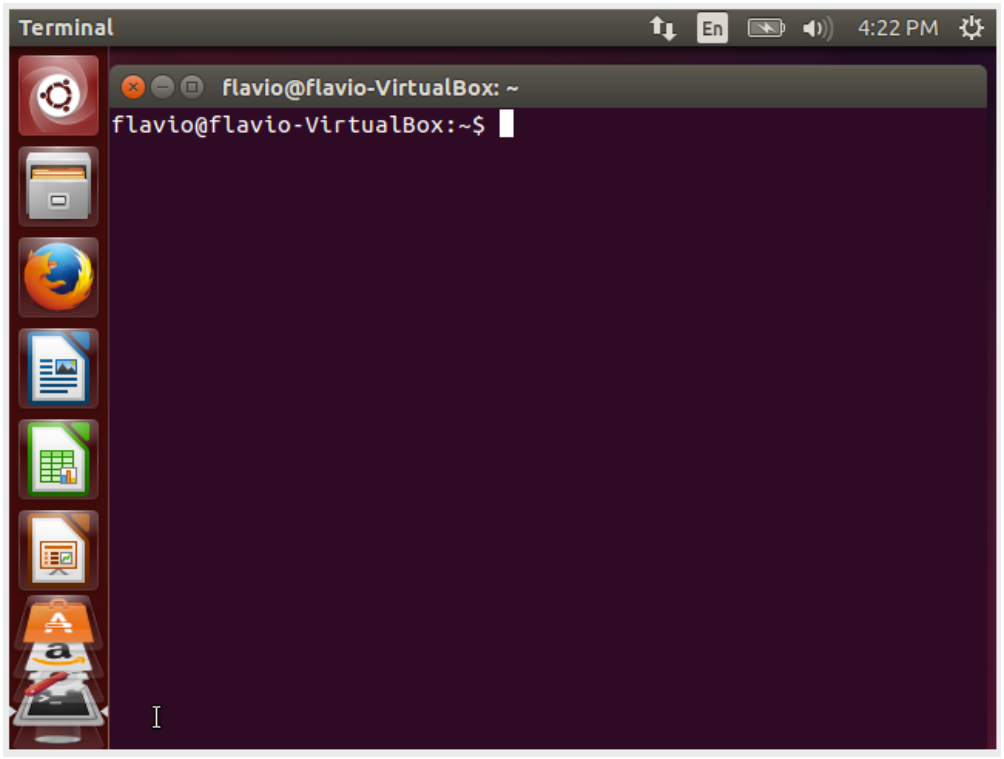


Figure 1: Interface terminal do Sistema Operativo Ubuntu

Veja alguns exemplos das chamadas de sistema. O Linux tem vários editores de texto, tais como, vim, emacs, ou gedit para escrever programas em C.

1. A chamada de sistema fork() retorna verdadeiro no processo pai e falso no processo filho. Digite o seguinte comando:

```
gedit fork.c (pressione Enter).
```

2. Digite o código apresentado no quadro seguinte:

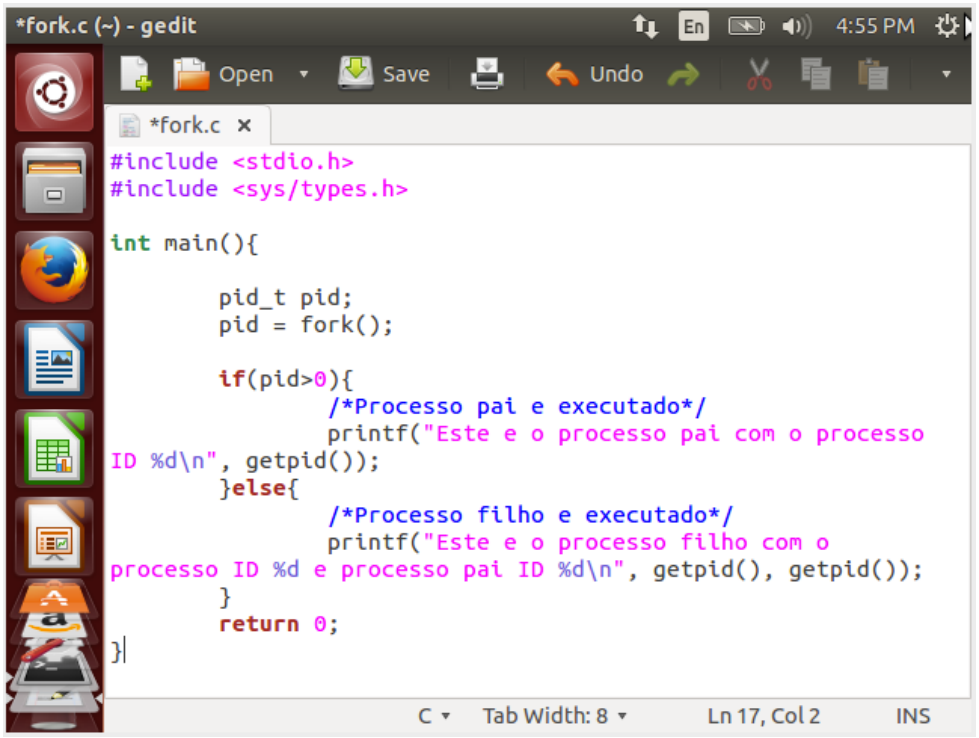


Figure 2: Editor gedit chamada de sistema fork()

Salve o ficheiro e feche o editor. Na linha de comando escreve o seguinte comando para executar o programa no prompt Shell:

```
$ gcc –o fork fork.c
```

Em seguida digite o seguinte comando:

```
$ ./fork
```

Veja o resultado da execução:

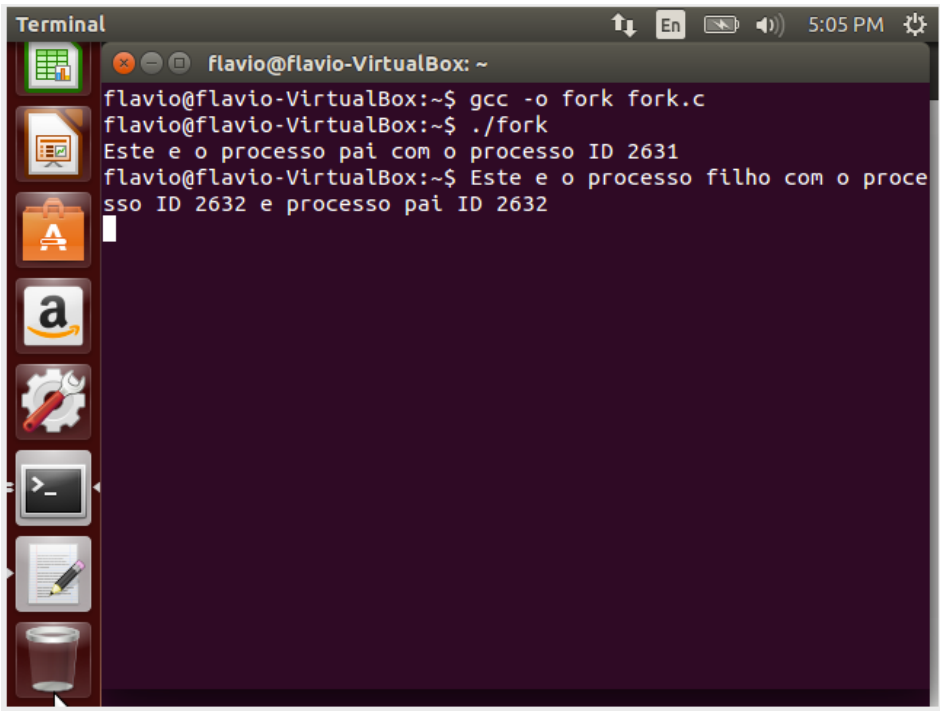


Figure 3: Resultado chamada do sistema fork()

O código a seguir, abre um ficheiro existente chamado meuficheiro duas vezes, escreve uma vez e lê uma vez através de dois descritores de ficheiros diferentes.

```
#include < string.h >
#include < unistd.h >
#include < fcntl.h >
int main (void) {
int fd[2];
char buf1[12] = "Este é um teste";
char buf2[12];
fd[0] = open("meuficheiro",O_RDWR);
fd[1] = open("meuficheiro",O_RDWR);
write(fd[0],buf1,strlen(buf1));
write(1, buf2, read(fd[1],buf2,12));
close(fd[0]);
close(fd[1]);
return 0;
}
```

O seguinte exemplo do código é para mostrar como utilizar um creat().

```
/*  criar.c */

#include <stdio.h>

#include <sys/types.h>       /* define types utilizado por sys/stat.h */     #include <sys/stat.h>
/* define S_IREAD & S_IWRITE      */

int main(){

            int fd;

            fd = creat("datafile.dat", S_IREAD | S_IWRITE);

            if (fd == -1)

               printf("Error em abrir datafile.dat\n");

            else{

               printf("datafile.dat aberto para read/write access\n");          printf("datafile.dat está
               vazia\n");

             }

            close(fd);

            exit (0);

}
```

Conclusão

A Shell é uma interface que não é parte do kernel. É utilizado para executar chamadas de sistema do Sistema Operativo Linux. Nesta atividade, foi apresentada os passos necessários para escrever programas em C e aceder ás chamadas de sistema do Sistema Operativo Unix.

Exercício prático

1. Escreve um programa em C que imprime os seus dados pessoais, execute-o e mostre os resultados.

Resumo da Unidade

Nesta unidade, foram introduzidos os conceitos fundamentais de um Sistema Operativos, descrevendo a suas função, objetivos, a sua estrutura e a diferença entre um SO de 32 bits vs 64 bits. Também foi apresentado a interface Shell que não é a parte do Kernel.

Avaliação da Unidade

Verifique a sua compreensão!

Perguntas de Autoavaliação

Leia o primeiro capitulo do livro Sistemas Operacionais dos autores, Andrew S. Tanembaum e Albert S. Woodhull.

1. Leia sobre as diferentes extensões da estrutura da máquina virtual.
2. Como pode um modelo cliente-servidor ser utilizado em um único computador?
3. Descreve as vantagens da estrutura da máquina virtual sobre a estrutura em camadas de um Sistema Operativo.
4. Que vantagem é que o serviços de conta de um Sistema Operativo tem para os utilizadores?
5. Quais são os objetivos de um Sistema Operativo? Como descreves um Sistema Operativo como um gestor de recursos?

Avaliação

Rúbrica	Pontuação
Qualidade da execução do problema (Clareza, organização, rigorosidade)	5pts
Nota da realização do Trabalho	12pts
Esforço suplementares (pesquisas, resolução de problemas..)	3pts

Nota	Comentário
]14,20]	Sucesso
]10,14]	Os resultados abaixo das expectativas, mas aceitável
<10	Insuficiente

Leituras e outros Recursos

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

1. Marques, J., Ferreira, P., Ribeiro, C., Veiga, L., Rodrigues, R., Sistemas Operativos, (2009), (FCA Ed.), Portugal. Capitulo 1 e 2.
2. Andrew S. Tanenbaum e Albert S. Woodhull, Sistemas Operacionais, 2ª Edição, Editor Porto Alegre, 2000, Capitulo. 1.
3. Abraham Silberschartz, Operating System Concepts, 7ª Edição, Editora John Wiley / Sons, inc, 2005, Capitulo. 1.
4. Saylor. Org, acedido em 22, de Fevereiro de 2016, https://learn.saylor.org/mod/page/view.php?id=922
5. Computer Hope, acedido em 22, de Fevereiro de 2016, http://www.computerhope.com/os.htm#03
6. Buchnell University, acedido em 22, de Fevereiro de 2016, http://www.eg.bucknell.edu/~cs315/wordpress/lab/lab-1/
7. PS Exam, acedido em 22, de Fevereiro de 2016, http://www.psexam.com/Notes-for-Computer-Science/operating-systems-history-of-operating-system-article/Print.html
8. Gizmos freeware, acedido em 22, de Fevereiro de 2016, https://web.archive.org/web/20151117104721/http://www.techsupportalert.com/content/32-bit-and-64-bit-explained.htm

Unidade 2: Processos E Threads

Introdução à Unidade

Nesta unidade, será discutida dois conceitos importantes de construção centrais de sistemas operativos modernos que são. os processos que são instâncias de um programa de computador em execução e a Thread como uma tarefa específica dentro de um programa. Esses conceitos são essenciais para a compreensão de como um Sistema Operativo quandoexecuta um programa e a comunicação entre cada uma das camadas da arquitetura de computador. Será apresentado, primeiramente uma visão geral de cada um dos conceitos, incluindo definições, finalidades de utilização e tipos. Em seguida, será discutido as semelhanças e diferenças entre os processos e as Threads. Por último, conclui-se a unidade com a introdução do conceitos relacionados com a mudança de contexto (Context Switch) e do papel importante que estes desempenham na programação de CPU, que será apresentado de forma mais aprofundada nas unidades seguintes.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Identificar processos e threds no sistema computacional?
2. Discutir a importância e a utilização de processos e Threads em um Sistema Operativo.
3. Explicar a diferença entre um processo e uma Thread.
4. Discutir mudança de contexto e como é utilizado em um Sistema Operativo.

Termos-chave

Processo: Um programa em execução

Thread: Uma tarefa específica dentro de um programa

Mudança de contexto: Mudança de um processo para o outro processo.

Atividades de Aprendizagem

Atividade 2.1 – Introdução aos Processos

Introdução

Todos os computadores modernos podem executar um conjunto de tarefas ao mesmo tempo. Enquanto executam um programa do utilizador, um computador também pode estar lendo a partir de um disco e dando saída a texto para uma tela ou impressora. Em um sistema multi programação, a CPU também alterna de um programa para o outro, executando cada por dezenas ou centenas de milissegundos. Acredita-se que estritamente falando, a CPU em qualquer instante do tempo executa só um programa de cada vez, funcionando para vários programas, dando aos utilizadores a ilusão de paralelismo. Monitorar múltiplas atividades paralelas é um problema difícil. Com as evoluções constantes nessa área desenvolveram um modelo que torna o paralelismo mais fácil de tratar (Tanenbaum & Woodhull, 2000).

Como mencionado anteriormente, um processo é um programa em execução. Um processo é mais de que o código de um programa, conhecido como seção de texto (text section). Também inclui a atividade corrente, como representado pelo valor do contador do programa (program counter) e os conteúdos do registo do processador. Um processo também inclui o pilhas de processo (Stack), que contém dados temporários (como parâmetros da função, variáveis locais), e a seção de dados (data section), que contém variáveis globais. Um processo também pode incluir a heap, memória alocado dinamicamente durante a execução do processo.

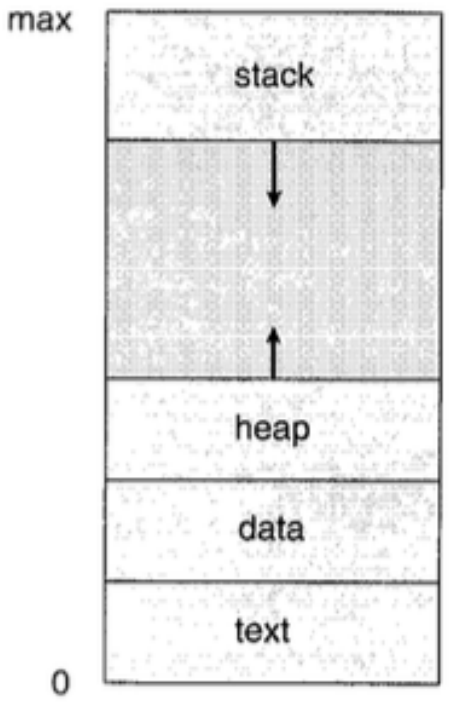


Figure 1: Processo na memória, (Silberschartz, 2005)

Para enfatizar que um programa por si só não é um processo. Um programa é uma entidade passivo, tal como um ficheiro que contendo uma lista de instrução armazenado no disco (geralmente chamados de ficheiros executáveis) , contudo, um processo é uma entidade ativa, com um contador de programa especificando a próxima instrução para executar e um conjunto de recursos associados. Um programa torna um processo quando um ficheiro executria na mema um processo quando um ficheiro execut fuer instante do tempo executa sb.m ambientes de exeuçável é carregado na memória, (Silberschartz, 2005).

Processo é uma instância de um programa que está sendo executado. É um programa que foi carregado a partir do disco para a memória e está em execução. Enquanto um programa é apenas uma coleção de instruções passiva, o processo é a real execução dessas instruções e um programa pode ser de vários processos. Este é a unidade básica de execução num Sistema Operativo. Um processo consome certos recursos como, memória, dispositivos de I/O tempo de CPU para realizar suas operações. É o dever do Sistema Operativo gerir todos esses requisitos de processos e levá-los à extinção. Um processo é a unidade básica de execução em um Sistema Operativo com o número único atribuído como um identificador (PID). Embora a CPU está executa apenas um programa ao mesmo tempo, também pode trabalhar em vários programas, trocando e executando por dezenas ou centenas de milissegundos. Este, por sua vez, exige mais controle e particionamento de programas que resultaram em um processo. Assim, o Sistema Operativo terá muitos processos em execução ao mesmo tempo e com os recursos necessários.

Colocar um exemplo claro (ver analogia entre programa e processo, página 62, Tanenbaum).

Estado de um Processo

Enquanto o processo é executado, muda de estado. O estado de um processo é definido em parte por sua atividade. Cada processo pode estar em um dos seguintes estados. Veja a figura seguinte:

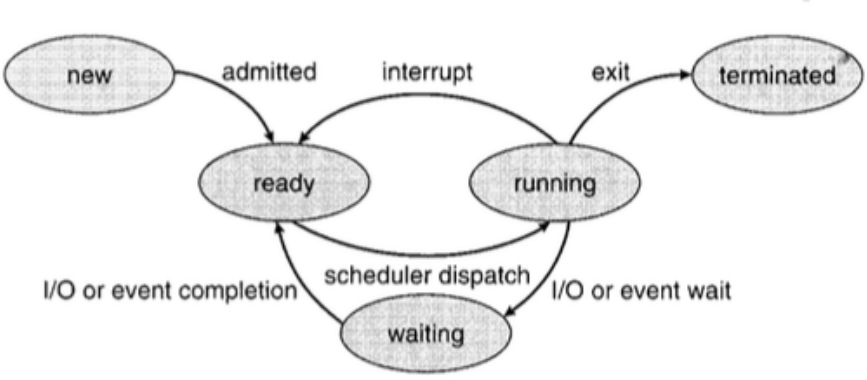


Figure 2: Diagrama de estado do processo, (Silberschartz, 2005)

- **Novo (New)** – O processo está sendo criado.
- **Execução (Running)** – As instruções estão sendo executados.

- **Espera (Waiting)** – O processo está a espera da ocorrência de algum evento.
- **Pronto (Ready)** – O processo está a espera para ser atribuído a um processador.
- **Terminado (Terminated)** – Termina a execução do processo.

Implementação de um Processo

Para implementar o modelo de processo, o Sistema Operativo mantém uma tabela chamada tabela de processos, com uma entrada por processo. Esta entrada contém informações sobre o estado do processo, sobre seu contador de programa, sobre o ponteiro da pilha, alocação de memória, etc... e tudo mais sobre o processo que deve ser salvo quando o processo alterna de um estado em execução para o estado pronto a fim de que possa ser reiniciado mais tarde como se nunca tivesse sido interrompido.

Agendamento do Processo

O objetivo da multiprogramação é o de ter processos em execução constantemente., para maximizar a utilização do CPU. O objetivo da partilha de tempo (time sharing) é mudar a CPU entre os processos com tanta frequência que os utilizadores podem interagir com cada programa enquanto estiver em execução. Para alcançar este objetivo, o agendador de processos seleciona o processo disponível para a execução do programa no CPU. Para um simples processador, nunca terá mais do que 1 (um) processo em execução. Neste caso, se houver mais processos, terão que esperar até que o CPU liberte e pode ser reagendado.

Filas de Escalonamento

Enquanto os processos entram no sistema, são colocados na filas de trabalho (job queue), que consiste em todos os processos no sistema. Os processos que se encontram na memória e prontos a espera para serem executados são guardados em uma lista chamada de filas prontas (ready queue). Este é geralmente armazenada como um lista ligada (linked list). Um cabeçalho da fila pronta, contém ponteiros para as primeiras e as finais PCB (Bloco de Controle de Processos) na lista.

Agendamento

Os processos migram-se entre as várias filas de escalonamento durante o seu tempo de vida. O Sistema Operativo deve selecionar para o agendamento, processos a partir dessas filas de alguma forma. O processo de seleção é realizada pelo um agendador apropriado.

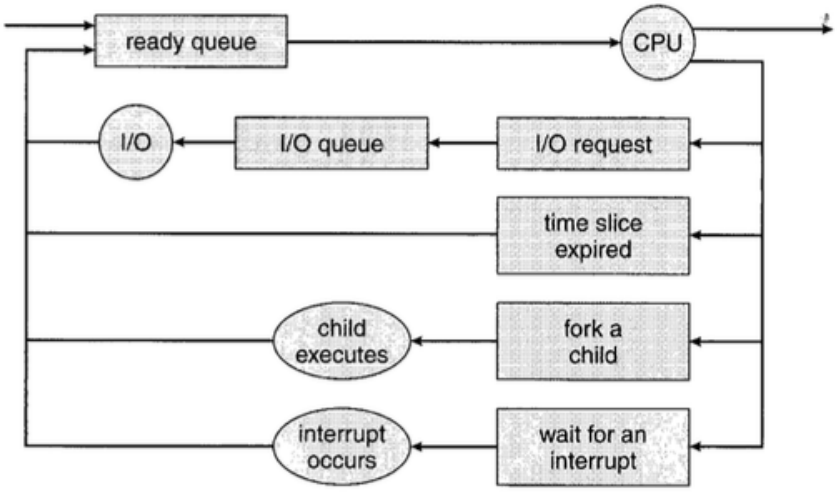


Figure 3: Diagrama de representação de agendamento de processos (Silberschartz, 2005)

Utilizando o comando ps

Abre a linha de comando no terminal Ubuntu e digite o comando ps.

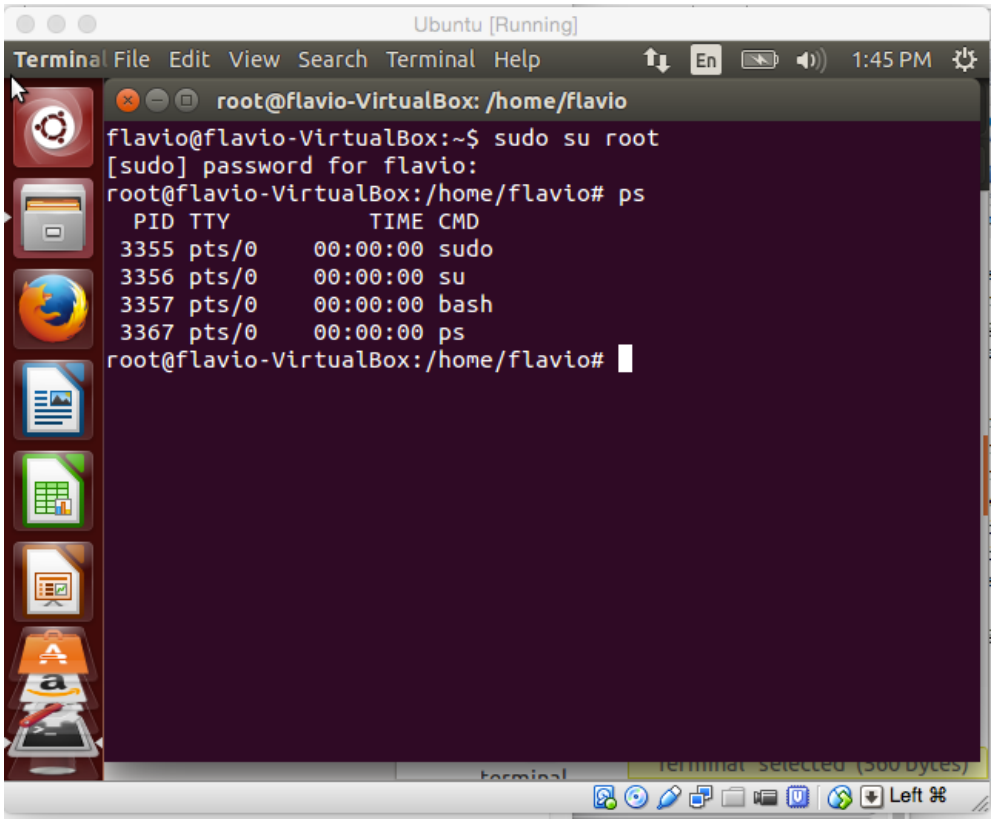


Figure 4: Resultado da execução do comando ps

Qual é o significado de cada uma das colunas?

- **PID** é o ID do processo, um número único da a cada processo em execução. O sistema Operativo utiliza um contador de 32 bits last_pid para rastrear o último PID atribuído a um processo. Quando é criado um processo, o contador é incrementado e o seu valor se torna o PID do novo processo.
- **TTY,** mostra que processo terminal é ligado. O pts significa que o Shell. /0 mostra que este é o primeiro Shell executado pelo utilizador. Se iniciar um outro Shell teria como resultado pts /1 para o campo TTY.
- Time indica o tempo total requerido pelo processo para terminar.
- **CMD** (comando), é o nome do comando em execução. Aqui, apenas são apresentados os processos que o utilizador root está atualmente em execução (neste terminal ou Shell). Estes são os bash (programa de Shell de root), e do próprio comando ps. Como pode-se notar, o bash está sendo executado simultaneamente com o comando ps.

Agora abra um novo terminal e execute o seguinte comando

```
# ps -e
```

Como resultado do comando em execução:

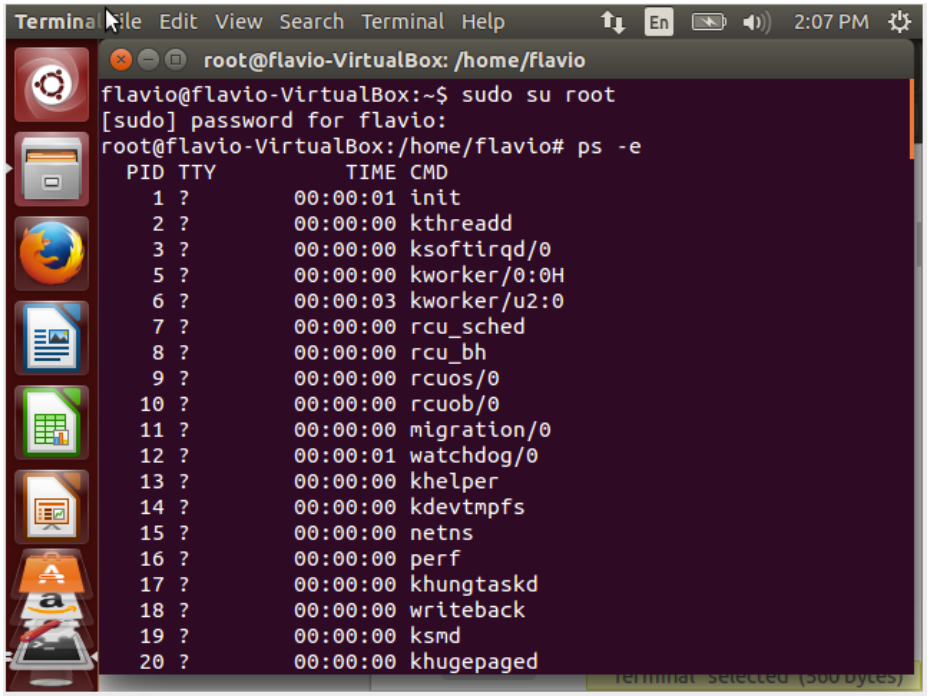


Figure 5: Resultado da execução do comando ps -e

1. Com base no resultado explique as alterações efetuadas?
2. Qual seria o resultado para ps –l? O que representa cada coluna?
3. Agora combina ps com E e L, tal como ps -e -l / ps -el e observa as alterações.
4. Faça o mesmo com o Gestor de Tarefas no Sistema Operativo Windows.

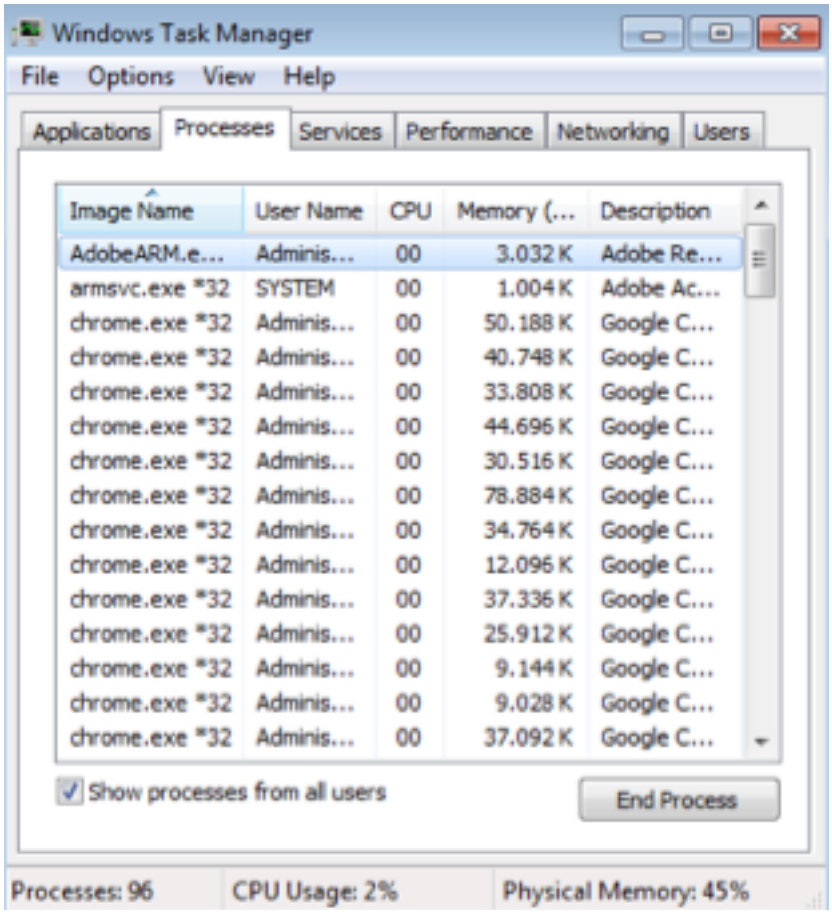


Figura 1: Gestor de Tarefas Windows

Mudança de Contexto (Context Switch)

Sempre que uma interrupção acontece, a CPU deve entrar em um estado salvar (save) do processo atualmente em execução, em seguida, muda para o modo kernel para lidar com a interrupção, e depois fazer uma restauração do estado do processo interrompido. Do mesmo modo, uma mudança de contexto ocorre quando o intervalo de tempo para um processo expira e um novo processo é carregada a partir da fila pronta. Este será instigado por uma interrupção do timer, o que irá fazer com que o estado do processo atual ser salvo e o novo estado do processo ser restaurado.

Salvar e restaurar estados envolve salvar e restaurar todos os registos e contador de programa (s), bem como os blocos de controle de processos descritos acima. Mudança de contexto acontece muito frequentemente, e a sobrecarga de fazer a comutação apenas faz com que CPU perde tempo, então mudanças de contexto precisa ser tão rápido quanto possível. Alguns hardware dispõe de características especiais para acelerar isso, assim como, uma única máquina de instrução para salvar ou restaurar todos os registos de uma só vez.

Conclusão

Nesta atividade, foram ilustrada os conceitos importantes sobre os estados e como operam estes processos.

Perguntas de autoavaliação

1. O que é um processo no âmbito de Sistemas Operativos?
2. Descreve os estados de um processo?
3. O que significa mudança de contexto?

Atividade 2.2 – Operações com Processos

Introdução

Nesta atividade serão apresentadas o ciclo para a criação de um processo.

Criar um processo

Processos podem criar outros processos através de chamadas de sistemas (system calls) adequados, tais como fork ou Spawn. O processo que faz a criação é denominado o pai (parent) do outro processo, o que é denominado o seu filho (child).

Cada processo recebe um identificador inteiro, denominado seu identificador de processo ou PID como apresentado anteriormente. O PID pai (PPID) também é armazenado para cada processo.

Em sistemas típicos UNIX o agendador de processo é denominado sched, e é dado PID 0. A primeira coisa que faz em tempo de inicialização do sistema é executar o init, o que dá esse processo PID 1. Init em seguida, inicia todos os daemons do sistema e logins de utilizadores, e se torna o processo pai de todos os outros processos. Figura seguinte, mostra uma árvore típico de processo para um sistema Linux, e outros sistemas têm uma estrutura semelhante, embora as árvores não são idênticas:

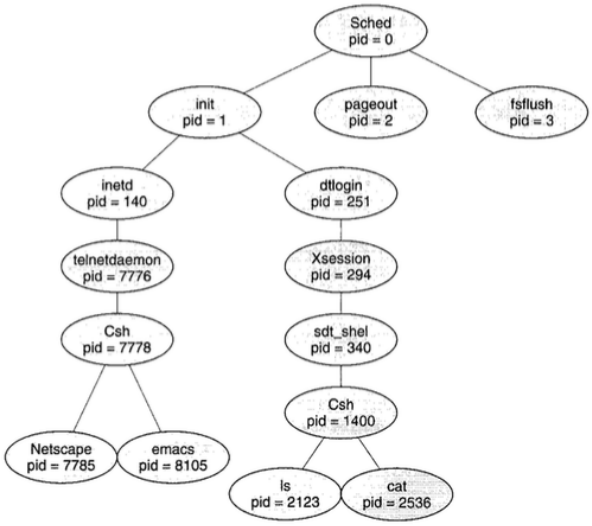


Figure 6: Árvore de processos para sistemas Linux, (Silberschartz, 2005)

Dependendo da implementação do sistema, um processo filho pode receber uma certa quantidade de recursos compartilhados com seu processo pai. Processo filho pode ou não ser limitado a um subconjunto dos recursos inicialmente atribuída ao pai, impedindo fuga de processo filho de consumir todo um determinado recurso do sistema.

Há duas opções para o processo pai depois de criar a criança:

- Aguardar o processo filho terminar antes de prosseguir. O processo pai faz uma chamada do sistema wait(), tanto para um processo filho específico ou para qualquer processo filho, o que faz com que o processo pai bloquear até que a wait() retorna. Shells UNIX normalmente espera para os seus processos filhos completar antes de emitir uma nova alerta.
- Executados simultaneamente com o processo filho, continuaram a processar sem esperar. Esta é a operação visto quando um Shell UNIX executa um processo como uma tarefa de fundo. Também é possível para o processo pai ser executado por um tempo, e depois esperar para o processo filho, que pode ocorrer em uma espécie de uma operação de processamento paralelo.

Utilizando o pstree e os comandos relacionados

1. Utilize os comando pstree para ver todos os processos organizados como uma árvore.
2. Utilize a versão apropriada dos comandos ps e pstree e liste 5 processos que se encontram no estado sleeping.
3. Exibe um processo que tem como avô (grandparent), juntamente com ID de seu pai e avô.

No entanto, o Sistema Operativo Windows não tem essa disposição hierárquica de processos e todos os processo são iguais. Quando um processo, o pai, cria outro processo, o filho, o pai será dado um sinal especial conhecido como manipulador (handler) para controlar o processo filho. Mas esse manipulador pode ser passado para outro processo livremente o que invalida a hierarquia.

Utilizando fork() e createProcess() para chamadas de sistema

O trecho de código seguinte mostra um processo fork e exec para criar um processo no sistema UNIX:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
   pid_t pid;
   // Fork processo filho
   pid = fork();
   if(pid < 0){
       //Um erro ocorre
       fprintf(stderr, "Fork falhou!");
      exit(-1);
      else if(pid == 0){
      execlp("/bin/ls", "ls", NULL);
    } else{
      // Processo pai espera para o
      //processo filho completar
     wait(NULL);
     printf("Processo filho completo");
     exit(0);
    }
}
```

Compile e execute o programa acima apresentado. Quantas linhas são impressas pelo programa?

O trecho do código seguinte, ilustra o processo mais complicado para Windows, que deve fornecer todas as informações de parâmetro para o novo processo. O createProcess () faz a tarefa semelhante do fork() em um Sistema Operativo Windows. A diferença é que o fork() tem o processo filho que herda o espaço de endereço (address space) do processo pai, enquanto createProcess() requer o carregamento de um programa especificado no espaço de endereço do processo filho na criação do processo. Além disso, o fork() nenhum parâmetro é requerido, enquanto que createProcess() aguarda nada menos do que dez parâmetros, incluindo o

nome da aplicação a ser executada, os parâmetros de linha de comando e outros atributos relacionados à segurança.

```
#include <stdio.h>

#include <windows.h>

int main(VOID){

    STARTUPINFO si;

    PROCESS_INFORMATION pi;

    // Alocar memorias;

    ZeroMemory(&si, sizeof(si));

    si.cb = sizeof(si);

    zeroMemory(&pi, sizeof(pi));

    //Cria o processo filho


    if(!CreateProcess(NULL, "C:\\WINDOWS\\system32\\mspainy.exe",
NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){

          fprintf(stderr, "Falha na criação do processo!");

          return -1;

          waitForSingleObject(pi.hProcess, INFINITE);

          printf("Filho completo");

          Closehandle(pi.hProcess);

          CLoseHandle(pi.hThread);

     }

}
```

Depois de um processo ser criado e executar as suas funções, será levado a terminar. Este pode surgir a partir da saída normal, onde um processo termina a execução da sua declaração final e pede ao Sistema Operativo para excluí-lo utilizando o exit() e chamadas de sistema exit do processo em UNIX e Sistema Operativo Windows, respetivamente. Nesse ponto, o processo pode retornar um valor de status (geralmente um número inteiro) ao seu processo pai (através da chamada de sistema wait()). Todos os recursos do processo, incluindo memória física e virtual, ficheiros abertos e I/0 são alocados pelo Sistema Operativo. Em alguns casos, um processo pode encerrar devido a erros de saída, erros de saída fatal ou ser morto (killed) por outro processo. Se o término for solicitada de outro processo, kill() para UNIX e terminar processo() no Windows será utilizado.

Terminar um Processo

Processos podem solicitar o seu próprio término, fazendo a chamada de sistema exit(), geralmente retornando um int. Este int é repassada para o processo pai se está em espera, wait(), e é tipicamente zero (0) na conclusão bem sucedida, e alguns não 0s códigos em caso de problemas.

Código do processo filho

```
int exitCodigo;

     exit( exitCodigo );

    // return exitCodigo; tem o mesmo efeito quando executado a partir
do main( )
```

Código do processo pai

```
pid_t pid;

    int estado;

    pid = wait( &estado );

    // pid indica qual proc. filho é terminado.
```

Para saber mais sobre o estado de saída (exit status) de um programa, pode-se utilizar macros da biblioteca sys/wait.h que verifica o status de terminação e retorna o status de saída.

WIFEXITED (status) retorna verdadeiro (true) se o processo filho terminar normalmente, isto é, chamando o exit(3) ou _exit (2), ou por retornar a partir da função main().

WEXITSTATUS (status) retorna o status de saída do processo filho. Este é composto por menos de 8 bits do argumento que o processo filho especificado em uma chamada para exit(3) ou _exit(2) ou como o argumento para uma instrução de retorno na função main(). Este só deve ser utilizado se WIFEXITED retornar verdadeiro.

1. Adiciona a forma correta de wait() e exit() no seguinte programa, para que o processo filho termine com sucesso e o processo pai espera o filho para terminar a execução exibindo "Eu sou ...".
2. Compile e execute o programa para ver os efeitos dessas chamadas.

```
#include<unistd.h>

#include<sys/wait.h>    /* Definir o wait  */

#include<stdio.h>

#include<stdlib.h>  /*Definir o exit */

void main()

      {

        pid_t pid, child;

        pid=fork();


        if(pid==0){          /*Codigo executado no processo filho */

             printf ( " Eu sou o PID filho %d\n", getpid());

      }

      else {

          printf ( " Eu sou o PID pai %d\n", getpid());

      }

}
```

Qual o processo, filho ou o pai, imprime a declaração "Eu sou ..." na primeira linha? Porquê?

Intercomunicação entre Processos

Processos independentes que operam simultaneamente em um sistemas são aqueles que não podem afetar outros processos ou ser afetado por outros processos.

Processos dependentes são aqueles que podem afetar ou ser afetado por outros processos. Há várias razões por que são permitidos processos dependentes:

- **Partilha de de informações** - Pode haver vários processos que precisam de acesso ao mesmo ficheiro, por exemplo(pipelines).
- **Computação acelerada** - Muitas vezes, uma solução para um problema pode ser resolvido mais rápido se o problema pode ser dividido em sub-tarefas a serem resolvidos simultaneamente (em particular quando estão envolvidos múltiplos processadores.)
- **Modularidade** - A arquitetura mais eficiente pode ser a fragmentar um sistema em módulos dependentes. (por exemplo, base de dados com uma arquitetura cliente-servidor).

Conveniência - Mesmo um único utilizador pode ser multitarefas, como edição, compilação, impressão e executar o mesmo código em janelas diferentes.

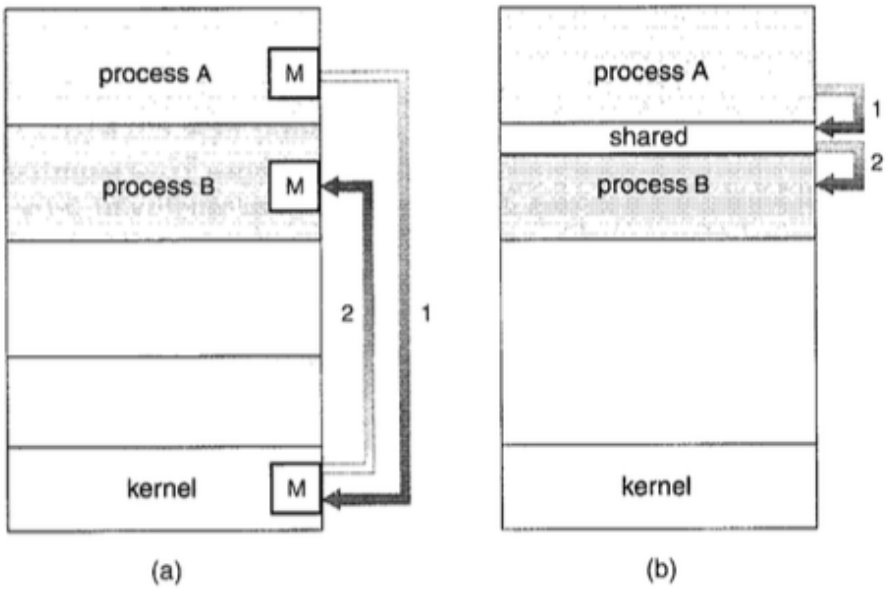


Figure 7: Modelo de comunicação

A figura 8, apresenta uma arquitetura multi-processo, utilizado o Google Chrome como referência. Os websites contém conteúdos ativos com o JavaScript, HTML5, Flash, para fornecer um dinâmica e rica da experiencia da Web browser. Infelizmente estas aplicações muitas vezes cotém erros, o que resulta em crash do Web browser, Neste exemplo, pode-se notar que cada website é separada pela uma tab e para navegar entre eles o utilizador precisa de um clique entre eles.

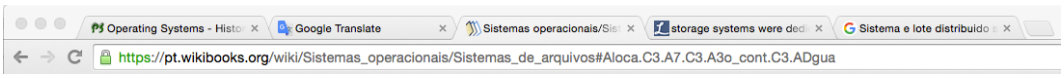


Figure 8: Exemplo de uma arquitetura multi-processo com a Google Chrome, (fonte: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html, 2016)

Processos automáticos

Processos automáticos ou em lotes (batch) são processos que não são conectados a um terminal. Em vez disso, estas são tarefas que podem ser enfileirados em uma área de spooler, onde esperam para ser executado com base no algoritmo FIFO (first-in, first-out). Tais tarefas podem ser executados utilizando um dos dois critérios:

- Em uma determinada data e hora: feito com o comando at.
- Às vezes, quando a carga total do sistema é baixo o suficiente para aceitar trabalhos extras: Por padrão, as tarefas são colocadas em uma fila onde esperam para serem executados até que a carga do sistema seja inferior a 0,8. Em grandes ambientes, o administrador do sistema pode preferir o processamento em lote, quando grandes quantidades de dados têm de ser processados ou quando as tarefas exigem muitos recursos do sistema para serem executados. O processamento em lote também é utilizado para otimizar o desempenho do sistema.

Daemons

Daemons são processos de servidor que são executados de forma contínua. Na maioria das vezes, eles são inicializados na inicialização do sistema e aguarda em segundo plano (background) até que seja necessário o seu serviço. Um exemplo típico é o daemon da rede, xinetd, que é iniciado em quase todos os procedimento de inicialização. Depois que o sistema é inicializado, o daemon da rede apenas aguarda até que um programa cliente, como um cliente de FTP, precisa conectar-se.

Comunicação Sistemas Cliente-Servidor

Sockets

Um Socket é um ponto final para comunicação. Dois processos se comunicam através de uma rede e frequentemente utilizam um par de Sockets conectados como um canal de comunicação. Software que é projetado para operação cliente-servidor também pode utilizar Socket para a comunicação entre dois processos em execução no mesmo computador, por exemplo, a interface do utilizador para um programa de base de dados pode se comunicar com o gestor de base de dados backend utilizando Socket. Se o programa for desenvolvido desta forma desde o início, torna-se muito fácil de porta-lo a partir de um sistema de computador único para uma aplicação de rede.

Uma Socket é identificado por um endereço IP (Protocolo Internet) concatenado com um número de porta, por exemplo 200.120.44.5:80.

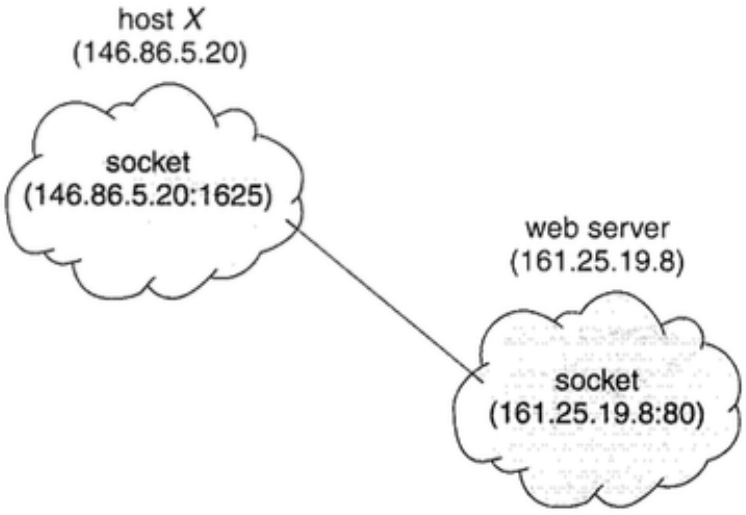


Figure 9: Comunicação utilizando Sockets, (Silberschartz, 2005)

Nota: Para aprofundar o seu conhecimento sobre processos, recomenda-se a leitura do capítulo 2, do livro Sistemas Operativos (Tanenbaum & Woodhull, 2000).

Conclusão

Nesta atividade, foram introduzidos os conceitos importantes sobre as operações com processos. Um processo é um programa em execução que requer alguns recursos concedidos pelo Sistema Operativo para a sua execução. Cada processo pode ser descrito pelo seus recursos de espaço de endereço, estado de processador e Sistema Operativo. Um processo é criado em diferentes eventos, incluindo solicitação do utilizador e inicialização do sistema. Um processo também pode criar outro processo quando ele é simples para passar alguns jobs para novos processos. Há diferentes tipos de processos baseados nos requisitos de cada aplicação. Pode ser interativo, automático ou daemon.

Perguntas de autoavaliação

1. Descreve as ações que um kernel realiza para a mudança de contexto entre os processos?2.
2. Qual é a diferença entre um programa e um processo?
3. Indique as três razões básicas para a criação de um processo?
4. Como um sistema diferencia de um processo?
5. Quantos processos são criados quando o código abaixo é executado?

```c
#include <stdio.h>
#include <unistd.h>
int  main()
{
      /*  fork  um processo filho  */
      fork();
      /*  fork  outro processo filho */
      fork();
      /*  e outro fork */
      fork() ;
      return ;
}
```

Laboratório – Processos, processo filho e fork()

Objetivo: Neste laboratório, o objetivo é entender como criar processos filho utilizando a função fork. Essa função (fork()) cria um novo processo por duplicar o processo de chamada. O novo processo, referido como processo filho, é uma cópia exata do processo de chamada, conhecido como o processo pai. O PID do processo filho é devolvida ao processo pai, e o 0 é retornado ao processo filho. Em caso de falha, -1 é retornado ao processo pai.

Exercício 1:

Escreve um programa em C, que recebe um parâmetro na linha de comando n, e n exibe as mensagens, uma após a outra com intervalos de 1 segundo entre 1 e o seguinte. Utilize a função sleep(). Em seguida, compile e execute o programa.

Uma vez que o programa está sendo executado faça o seguinte:

1. Pesquise o PID do processo utilizando o comando ps:
    a. A partir do mesmo Shell utilizado para executar o programa.
    b. A partir de um Shell diferente.
2. Execute 10 instâncias do programa em paralelo (utilizando &) :
    a. Utilize o comando ps para exibir todos os PIDs dos processos;
    b. Escolhe e termina cinco (5) desses processos a partir do mesmo Shell;

Atividade 2.3 - Threads

Introdução

Uma Thread é uma unidade básica de utilização da CPU, que consiste de um contador de programa, uma pilha, e um conjunto de registos, processos tradicionais têm uma única Thread de controle (e um ID de discussão.) - Há um contador de programa, e uma sequência de instruções que podem ser realizadas a qualquer momento. Nesta atividade, será introduzido a noção do Thread e descrever os APIs para Phtreads, Win32.

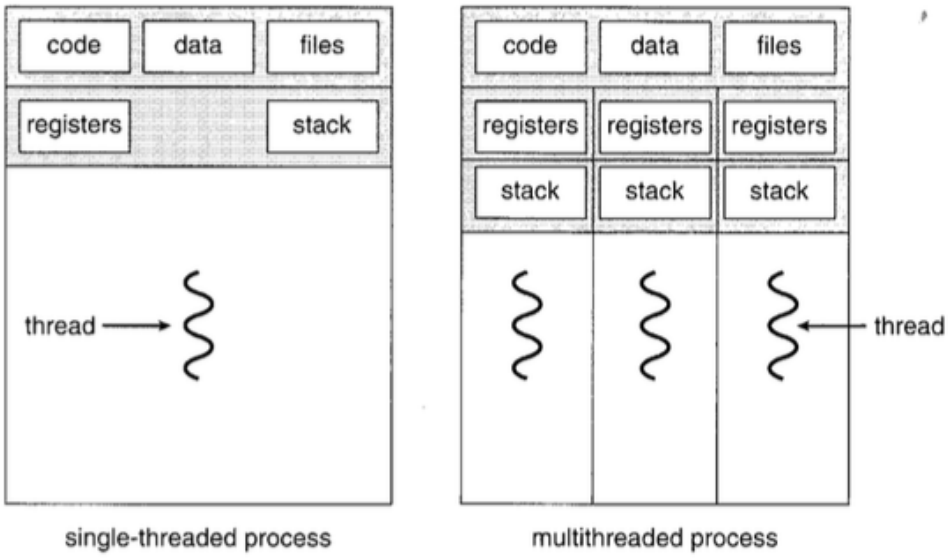


Figure 10:Simples Thread e multithread

Em certas situações, uma única aplicação pode ser necessário para executar algumas tarefas similares. Por exemplo um servidor aceita solicitações de uma página Web, imagens, música, vídeo e mais pelo cliente. Um servidor Web com muitas transações ou solicitações pode ter talvez milhares de cliente em paralelo conectados. Se o servidor Web executar um simples processo thread, poderia ser capaz de servir somente um cliente de cada vez. O tempo que o cliente pode ter a espera para uma resposta do pedido pode ser elevado.

Uma solução é ter o servidor em execução como uma simples processo que aceita o pedido. Quando o servidor receber o pedido, cria-se um processo separado para servir o pedido. De facto o método de criação de um processo foi utilizado antes do thread tornar-se popular. A criação do processo era moroso e utilização intensivo de recursos.

Finalmente, muitos Sistemas Operativos Kernel são agora multithread. Vários Threads operaram no kernel, e cada Thread executa uma tarefa específica, tais como a gestão de dispositivos ou manipulação da interrupção.

Threads são muito úteis para a programação moderna sempre que um processo tem várias tarefas a serem executadas de forma independente dos outros. Isto é particularmente verdadeiro quando uma das tarefas podem bloquear, e é desejável para permitir que as outras tarefas prosseguirem sem bloqueio.

Por exemplo, em um processador de texto, uma discussão de fundo pode verificar a ortografia e gramática enquanto um segmento de primeiro plano processa a entrada do utilizador (combinações de teclas), enquanto ainda um terceiro carrega imagens do disco rígido, e uma quarta faz backups automáticos periódicos do ficheiro que está sendo editado .

Outro exemplo é um servidor Web - Múltiplos Threads permite múltiplos pedidos para serem respondidas em simultâneo, sem ter as solicitações de serviço sequencialmente ou separar processos para cada solicitação. O último é a forma como este tipo de coisa foi feita antes do conceito de Threads foi desenvolvido.

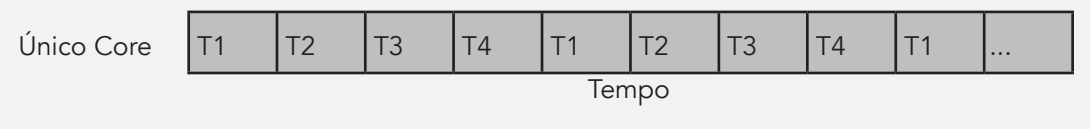
Benefícios

- **Receptividade** – Uma Thread pode dar uma resposta rápida, enquanto outros são bloqueados ou demorados fazendo cálculos intensivos.
- **Partilha de recursos** - Por padrão, Thread partilha códigos comuns, dados e outros recursos, que permite que várias tarefas serem executadas simultaneamente em um único espaço.
- **Económico** – A criação e gestão de Threads (e da mudança de contexto entre eles) é muito mais rápido do que realizar as mesmas tarefas para processos.
- **Escalabilidade**, ou seja, utilização de arquiteturas de multiprocessadores - Um único Thread só pode ser executado em uma CPU, não importa quantas podem estar disponíveis, ao passo que a execução de uma aplicação multithread pode ser dividida entre os processadores disponíveis.

1. Programação Multicore

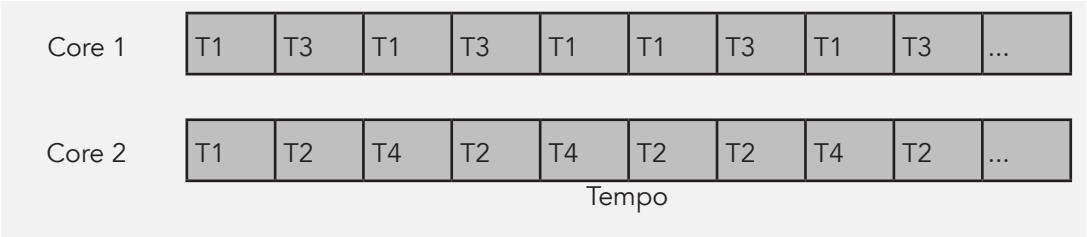
Uma tendência recente na arquitetura de computador é produzir chips com multicore, ou CPUs em um único chip.

Uma aplicação multithread em execução no chip de um único core teria que intercalam as Threads, conforme mostrado na figura seguinte.



Execução concorrente em um único core

Em um chip de multicore, contudo, os Threads podem ser distribuídos entre os core (núcleo) disponíveis, permitindo o processamento paralelo, como mostrado na figura.



Execução em paralelo

Para os Sistemas Operativos, multicore chips exigem novos algoritmos de escalonamento para fazer melhor utilização dos vários cores disponíveis.

Como multithread se torna mais penetrante e mais importante (milhares em vez de dezenas de Threads), CPUs foram desenvolvidos para suportar Threads simultâneas por core no hardware.

1.1 . Desafios na Programação

Para programadores de aplicações, existem cinco áreas onde chips multi-core apresentam novos desafios:

Identificação de tarefas – Analisar aplicações para encontrar atividades que podem ser realizadas simultaneamente.

Balance - Encontrar tarefas para executar simultaneamente que fornecem valor igual. Isto é, não desperdiçar Threads em tarefas triviais.

Data splitting - Para evitar que os Threads interferem uma com a outra.

Dependência de dados - Se uma tarefa é dependente dos resultados de um outro, em seguida, as funções precisam de ser sincronizados para assegurar o acesso na ordem adequada.

Teste e depuração - inerentemente é mais difícil em situações de processamento paralelo, como as condições de nas execuções tornam-se muito mais complexo e difícil de identificar.

0.2. Tipos de Paralelismo

Na teoria, há duas formas diferentes de paralelizar a carga de trabalho:

O **paralelismo de dados** (data parallelism) divide os dados em cima entre multicores (Threads) e executa a mesma tarefa em cada subconjunto de dados. Por exemplo dividindo uma imagem em partes e realizando o mesmo processamento de imagem digital em cada partes em diferentes do core.

Tarefa em paralelo (task parallelism) divide as diferentes tarefas a serem realizadas entre os diferentes núcleos e executa-os simultaneamente.

2. Modelos Multithread (Muiltarefas)

Threads do utilizador são suportados acima do kernel, sem o apoio do kernel. Estes são os Threads que os programadores de aplicações i colocariam em seus programas.

Threads do kernel são suportados dentro do kernel do Sistema Operativo em si. Todos os Sistema Operativo modernos suportado nível do Thread do kernel, permitindo que o kernel execute várias tarefas simultâneas e/ou a servir várias chamadas do kernel simultaneamente.

Em uma implementação específica, os Threads do utilizador devem ser mapeados para kernel Threads, utilizando uma das seguintes estratégias.

Modelo muitos-para-um

No modelo de muitos-para-um, muitos Threads de utilizadores são mapeados em um único segmento do kernel. A gestão dos Threads é feita pela biblioteca de Threads no espaço do utilizador, que é muito eficiente.

No entanto, se um bloqueio acontecer na chamada de sistema, em seguida, todo os processos bloqueiam, ainda que os outros Threads do utilizador, de outro modo serem capazes de continuar.

Porque um único Thread no kernel pode operar apenas em um único CPU, o modelo de muitos-para-um não permite que processos individuais serem dividido entre várias CPUs.

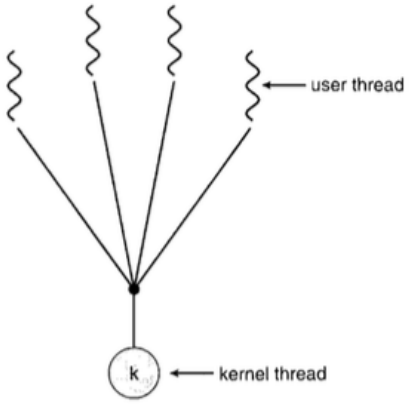


Figure 11: Modelo Muitos por um

Modelo um-para-um

O modelo um-para-um cria uma Thread do kernel separado para lidar com cada Thread do utilizador.

O modelo um-para-um supera os problemas listados acima envolvendo o bloqueio de chamadas do sistema e a separação dos processos em várias CPUs. No entanto, a sobrecarga na gestão do modelo de um-para-um é mais significativo, envolvendo mais sobrecarga e abrandar o sistema.

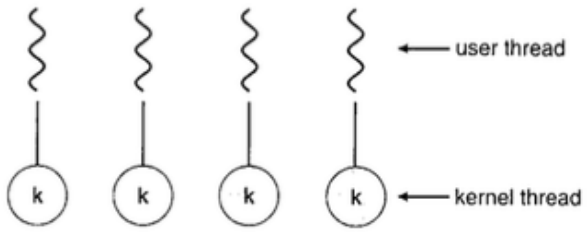


Figure 12: Modelo um-para-um

Modelo muitos-para-muitos

Os modelo multiplexes muitos-para-muitos, qualquer número de Threads do utilizador para um número igual ou menor de Threads do kernel, combinam as melhores características dos modelos de um-para-um e muitos-para-um.

Os utilizadores não têm restrições sobre o número de Threads criados. Bloqueio nas chamadas de sistema do kernel não bloqueiam todo o processo. Os processos podem ser divididos em vários processadores.

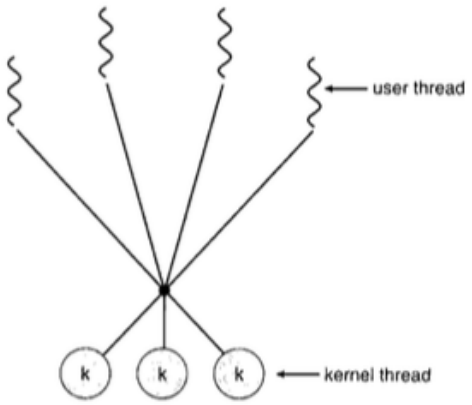


Figure 13: Modelo muitos-para-muitos

Comunicação entre os processos

O processos precisam se comunicar uns com os outros para poderem assim, partilhar os recursos. Como o Sistema Operativo lida com esta comunicação entre os processos?

A tarefa fundamental dos Sistemas Operativos modernos é a gestão de múltiplos processos dentro de um único processador, multiprocessador e sistemas informáticos distribuídos. O problema de design fundamental na gestão dos vários processos é o paralelismo: a execução simultânea de múltiplos processos.

Paralelismo ou execução em simultâneo surge em três contextos diferentes:

- Múltiplas aplicações: aplicações em execução em simultânea.
- Aplicações estruturadas: uma aplicação estruturada como um conjunto de processos concorrentes (Threads).
- Estrutura do Sistema Operativo: é implementado como um conjunto de processos.

Paralelismo oferece grandes benefícios em termos de eficiência de processamento e na estruturação do programa.

Comunicação entre processos

Há uma interação frequente entre os processos em execução simultânea. Existem três formas em que os processos em execução simultâneo interagem uns com os outros:

1. A competição por recursos

- Ocorre quando os processos independentes, que não são destinadas a trabalharem em conjunto competem para a utilização da mesma ou de recursos partilhados, e.g. da impressora, memória, ou ficheiro.
- Não há troca de informações entre estes processos concorrentes.
- Processos não têm conhecimento da existência um do outro.

2. A cooperação de partilha recursos

- Ocorre quando os processos não estão conscientes da utilidade de cada um, e utiliza atualiza dados partilhados sem referência a outros processos, mas sabe que outros processos podem ter acesso aos mesmos dados.
- Os processos devem cooperar para assegurar uma boa gestão dos dados que partilham.
- Processos estão cientes da existência de um ao outro da forma indireta.

3. Cooperação com base na comunicação

- Ocorre quando vários processos se comunicam uns com os outros, por exemplo, com a passagem de mensagens, a fim de fornecer uma maneira de sincronizar ou coordenar as suas diversas atividades.
- Não há nada partilhada entre processos.
- Processos estão conscientes da existência de um ao outro diretamente.

1. O que entendes por Multiprogramação?
2. Define um espaço de endereçamento?

Descreve o modelo Multitarefa?

Gestão de Thread

Pthreads são a implementações Linux de mecanismo de gestão de Threads. Pthreads são definidas como um conjunto de tipos de linguagem de programação C e chamadas de procedimentos. Os fornecedores geralmente oferecem uma implementação Pthreads sob a forma de um ficheiro header/include e uma biblioteca, que se liga com o seu programa. Os dois nomes utilizados na gestão de Threads frequentemente são: pthread_t para objetos da Thread e pthread_t para os atributos do objeto da Thread.

A função pthread_create é utilizado para criar um nova Thread, e a função pthread_exit para terminar uma Thread.

Normalmente, quando um programa inicia-se e torna-se um processo, que inicia com uma Thread padrão que leva à conclusão de que cada processo tem, pelo menos, uma Thread de controlo. Um processo pode criar Threads adicionais utilizando a seguinte função:

```
#include <pthread.h>

int pthread_create(pthread_t *restrict tidp, const pthread_
attr_t *restrict attr, void *(*start_rtn)(void), void *restrict
arg)
```

- O primeiro argumento é um endereço tipo pthread_t. Uma vez que a função é chamada com êxito, a variável cujo endereço é passado como primeiro argumento conterá o ID Thread de um Thread recém-criado.
- O segundo argumento pode conter certas qualidades que queremos para a nova Thread. Pode ser uma prioridade etc.
- O terceiro argumento é um ponteiro da função. Isso é algo a ter em mente que cada Thread inicia com uma função e que o endereço da função é passado aqui como o terceiro argumento para que o kernel saiba a partir de qual função deve iniciar o Thread.
- Como a função pode aceitar alguns argumentos também pode-se passar esses argumentos na forma de um ponteiro para um tipo void, que é o quarto argumento.

```
/* Criar Thread */

#include <stdio.h>

#include <pthread.h>

/* Todas as funções da Thread e tipo de dados são definidos
em pthread.h */

void *funFilho(void *p){

      printf ("ID filho é ---> %d\n", getpid( ));

}

      void main(){

        /*Declarar o tipo de variavel pthread_t :*/

        pthread_t filho;

        pthread_create (&filho, NULL, funFilho, NULL);

        printf ("ID pai é ---> %d\n", getpid());

        pthread_join (filho, NULL);

        printf ("Sem mais filhos!\n") ;

}
```

O compilador precisa conectar este programa com ficheiro da biblioteca Pthreads. Isto pode ser feito através da adição de -lpthread:

```
$ gcc lab1.c -o threads -lpthread
```

Depois, execute o programa:

```
$ ./threads
```

Resultado da execução:

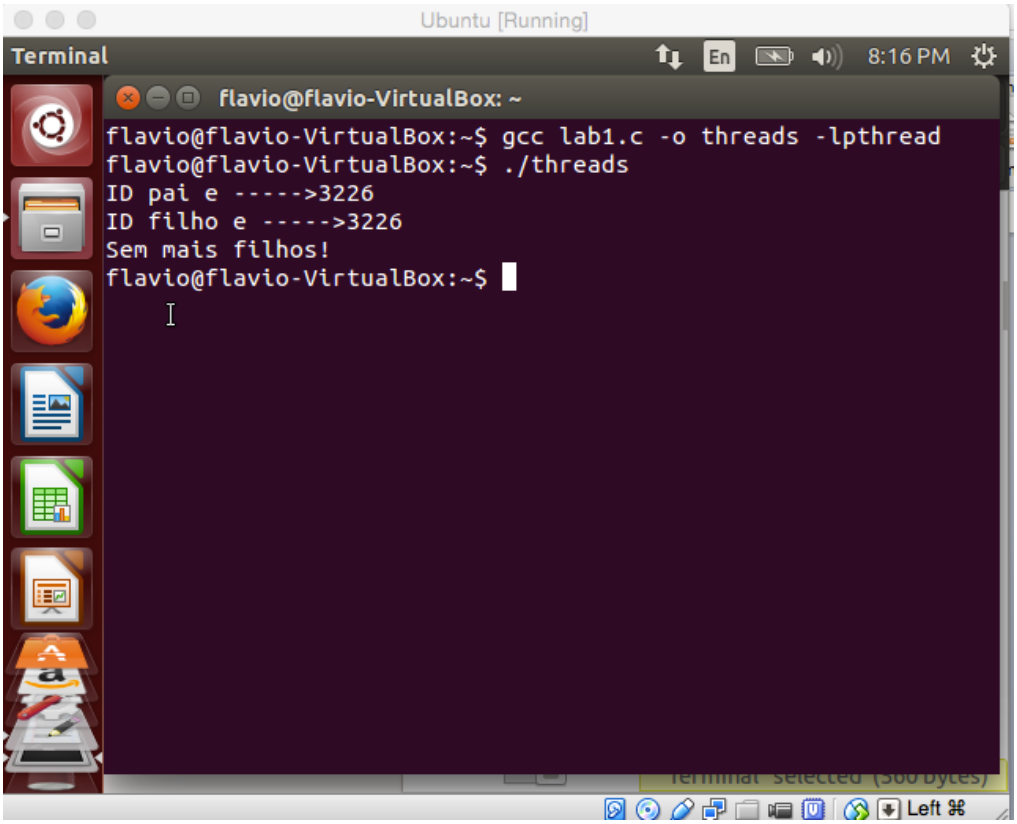


Figura 2: Criar uma Thread

1. Qual é o resultado do programa?
2. Quantos Threads foram criados? Quais as instruções foram executados por cada Thread?
3. Os números de processos são iguais ou diferentes? Porquê?
4. Qual é o Thread que termina primeiro a sua execução? Porquê?

Terminar uma Thread

```
/*Terminar uma Thread */

#include<stdio.h>

#include<string.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

pthread_t tid[2];

int ret1,ret2;

void* fazAlgo(void *arg)

{

    unsigned long i = 0;

    pthread_t id = pthread_self();


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


    if(pthread_equal(id,tid[0]))

    {

        printf("\n Processamento de primeiro Thread terminado\n");

        ret1  = 100;

        pthread_exit(&ret1);

    }

    else

    {

        printf("\n Processamento de segundo Thread terminado\n");

        ret2  = 200;

        pthread_exit(&ret2);

    }

    return NULL;

}
```

```
int main(void)
{
    int i = 0;
    int err;
    int *ptr[2];

    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &fazAldo, NULL);
        if (err != 0)
            printf("\nThread não criado");
        else
            printf("\n Thread criado com sucesso!\n");

        i++;
    }
    pthread_join(tid[0], (void**)&(ptr[0]));
    pthread_join(tid[1], (void**)&(ptr[1]));

    printf("\n Valor do primeiro Thread e [%d]\n", *ptr[0]);
    printf("\n Valor do segundo Thread e [%d]\n", *ptr[1]);

    return 0;
}
```

Resultado da execução do programa:

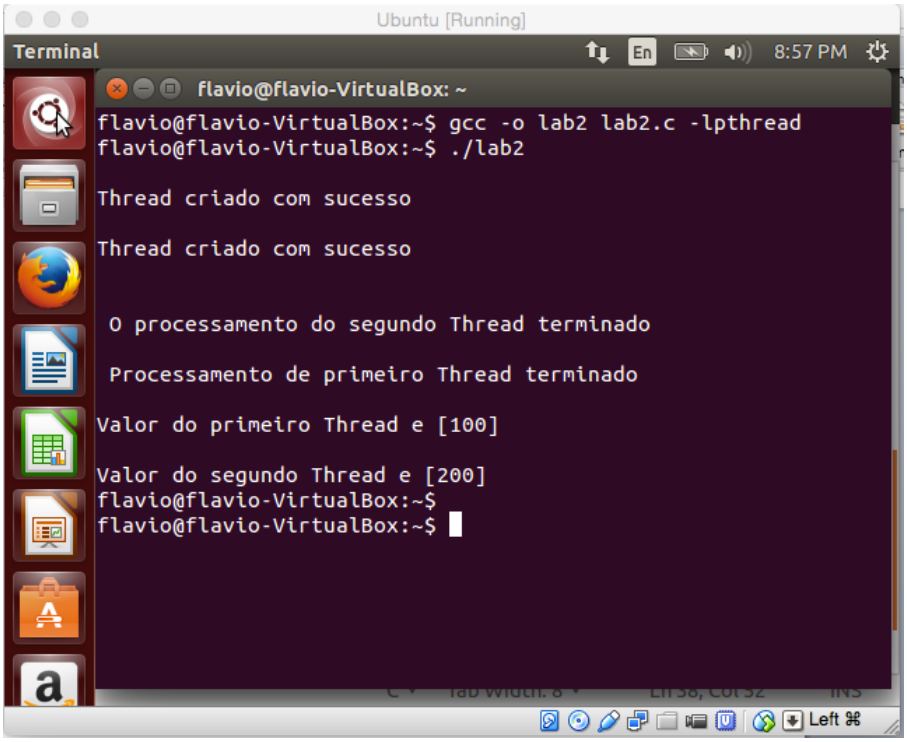


Figura 3: Terminar uma Thread

1. Com base nos resultados descreve as atividades deste programa?

Para consolidar o seu conhecimento leia o capítulos apresentadas para esta unidade.

Recomenda-se a leitura dos seguintes tópicos: Biblioteca das Threads e problemas com as Threads, em que será discutido os problemas considerados com os programas multithread. Também, nesta atividade, serão discutidos alguns pontos importantes, como, fork(), exe() e chamada de sistema, sinais, cancelação de Thread, entre outras, de modo que, o aluno possa entender e aplicar as operações com Threads.

Outras leituras recomendadas:

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html.

Conclusão

Nesta atividade, foi discutida os conceitos relacionados com as Threads, programação multicore, os modelos multithread e as suas características.

Uma Thread é uma unidade básica da utilização da CPU, que consiste em um contador de programa, uma pilha, um conjunto de registos, e um Thread ID. Um processo pode ter um ou mais Threads que partilham os mesmos recursos como o espaço de endereços e ficheiros abertos de um processo. Os Threads diferem dos processos porque existe uma dependência entre Threads e também trabalham em colaboração para realizar uma tarefa, como também, partilham os mesmos recursos. Os Threads tem também, uma similaridade com processos como caso de possuir um ID único da Thread semelhante com PID dos processos.

Avaliação

1. Qual é a importância da utilização de Threads?
2. Quais são as desvantagens da implementação de Threads a nível do kernel?
3. Quais são os recursos utilizados quando uma Thread é criada?
4. Como eles diferem daqueles utilizados quando um processo é criado?

Resumo da Unidade

Um processo é um programa em execução e uma Thread é um fluxo de controle dentro de um processo. Um processo multithread contém vários fluxos diferentes de controle dentro do mesmo endereço Nesta unidade, foi apresentado dois blocos de construção centrais de Sistemas Operativos modernos: Processos e Threads. Esses, são essenciais para a compreensão de como um Sistema Operativo executa um programa e a comunicação entre cada uma das camadas de arquitetura do computador. Introduziu-se os conceitos gerais sobre as operações entre os processos e Threads. Também, é notável, os benefícios de dos sistemas multithread que incluem o aumento da capacidade de resposta para a partilha de recursos do utilizador dentro do processo, económico, e a capaz de tirar proveito das arquiteturas multiprocessadores.

Avaliação da Unidade

Verifique a sua compreensão!

Sequência de Fibonacci

1. A sequência de Fibonacci é uma série de números 0,1,1,2,3,5,8,13,... que pode ser expresso da seguinte forma:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

   a. Escreve um programa em C utilizando a chamada de sistema fork() que gera sequência de Fibonacci no processo filho. O número de sequência será inserido na linha de comando. Por exemplo, se 5 é inserido, os primeiros cinco números da sequência de Fibonacci serão exibidas pelo processo filho.

   b. Repita o exercício anterior, utilizando o CriarProcesso() na API Win32. Neste caso, será necessário especificar um programa separado para ser chamado de CriarProcesso().

Sistema de avaliação

Cada resposta ou ação executada com sucesso relataram 5%. O sistema de pontuação é descrito pelo seguinte quadro:

Nota para respostas corretas cujo total será 20 pontos	Descrição
>= 17	Superou os objetivos
> 14 < 17	Atingiu os objetivos
> 10 < 14	Atingiu os objetivos parcialmente

Leituras e outros Recursos

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

Leituras primários

Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.)

Silberschartz, A. (2005). Operating System Concepts (Vol. 7). (i. Editora John Wiley / Sons, Ed.)

Leituras secundários

Sistemas Operativos. (n.d.). Retrieved 2016, from

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html

https://sites.google.com/site/uopops/pm

https://ruby-hacking-guide.github.io/thread.html

Unidade 3: Gestão De Memória

Introdução à Unidade

A memória está presente em todas as operações efetuadas pelo computador. Nas unidades anteriores falamos de como um Sistema Operativo funciona e como os processos e Threads operam, mostrando as suas relações e diferenças. Discutimos o modelo multiprocessos e multitarefas e como estes podem aumentar o desempenho de um computador partilhando o CPU. Também, alguns processos precisam ser guardados na memória, isto é, a memória deve ser partilhada. É absolutamente importante ter um entendimento sólido das funções da memória, de modo que sejas capaz de utilizar a memória no seu programa de forma eficiente. Será discutido como é que as memórias são alocadas para diferentes finalidades. Por último, será descrito os principais tópicos relacionados com o acesso a memória.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Definir a memória e descrever a sua hierarquia.
2. Mostrar como os Sistemas Operativos interagem com a memória
3. Descrever os algoritmos para alocação dinâmica da memória e explicar os métodos de acesso.
4. Definir uma memória virtual e como funciona.

Termos-chave

Memória: Ponto central para as operações de um computador moderno

Processos: Permite efetuar o controlo das partes constituintes da memória de um computador

Sincronização: Permite sincronizar processos

Paginação: É a gestão da memória que permite o espaço do endereço físico de um processo serem não contíguos.

Segmentação: Prove os computadores com espaços de endereços completamente independentes

Troca : Permite efetuar trocas entre os processos na gestão de memória

Atividades de Aprendizagem

Atividade 3.1: Gestão da Memória

Gestão da memória é um campo complexo da ciência da computação e existem muitas técnicas que estão sendo desenvolvidas para torná-la mais eficiente. A parte do Sistemas Operativos que faz a gestão hierárquica da memória é o gestor da memória. O gestor de memória faz o controlo das partes da memória que estão em utilização e as que estão livres, alocar memória para processos quando necessitarem, e desalocar terminarem e, não só, gerir a troca entre a memória principal e o disco, quando a memória principal é pequeno para armazenar todos os processos.

Esta atividade destina-se a apresentá-lo algumas das questões básicas da gestão de memória que os programadores enfrentam no dia-a-dia.

A gestão da memória é normalmente dividido em três áreas, embora as distinções são aparecem um confusas:

- Gestão da memória a nível de hardware – a gestão da memória a nível de hardware está relacionada com os dispositivos electrónicos que armazenam os dados. Como exemplo, tem-se a RAM (Random Accesss Memory).
- Gestão da memória a nível dos Sistemas Operativos – neste nível, a memória deve ser alocada para programas de utilizadores, e reutilizado por outros programas quando não é mais preciso. O Sistema Operativo pode "fingir" que o computador têm mais memória do que realmente é a nível físico da máquina.
- Gestão da memória a nível das aplicações – este, envolve o fornecimento da memória necessária para objetos de um programa e estruturas de dados a partir dos limitados recursos disponíveis, e a reciclagem que a memória reutiliza quando já não é necessário. Porque as aplicações não pode, em geral, prever com antecedência a quantidade de memória necessária, precisam de código adicional para lidar com a mudança dos requisitos da memória.

Espaço de endereçamento

A memória é dividida em blocos com o mesmo tamanho, denominados de páginas. As páginas têm uma dimensão que é obviamente uma potência de 2 para que toda a aritmética que se relaciona com o seu tamanho seja simples (Marques, Ferreira, Ribeiro, Veiga, & Rodrigues, 2009). Exemplo do endereço:

(página, deslocamento)

A página indica o número da página e o deslocamento indica o deslocamento dentro da página.

Endereço físico vs lógico

O conceito de um espaço de endereçamento lógico que é vinculado a um espaço de endereço físico separada é fundamental para a gestão da memória.

- Endereço lógico – é gerado pelo CPU, e também conhecido como endereço virtual.
- Endereço físico – é o endereço da unidade de memória.

Alocação de memória contíguo

A memória principal deve alocar tanto o Sistema Operativo como também os processos variados de utilizadores. É preciso, portanto, atribuir as partes da memória principal da maneira mais eficiente possível.

A memória é usualmente dividido em duas partes: uma parte para o Sistema Operativo e a outra para os processos de utilizadores. O Sistema Operativo pode ser alocado tanto a nível mais baixo como, a nível mais alto da memória.

Paginação

Paginação é a gestão da memória que permite o espaço do endereçamento físico de um processo serem não contíguos. Em qualquer computador, existe um conjunto de endereços de memória que os programas podem produzir. Quando utiliza uma instrução, tal como:

MOVE REG, 1000

Isto significa que o conteúdo é copiado para o endereço da memória 1000 ou vice versa dependo do computador. A ideia é fazer com que o programador não precise se preocupar com a gestão de memória quando escreve um programa.

Troca (Swapping)

Com um sistema de lotes, organizar a memória em partições fixas é simples e efetivo. Cada job é carregado em uma partição quando aproxima do inicia da fila e permanece na memória até que termine. Um processo deve estar na memória para ser executado. No entanto um processo pode ser colocado temporariamente fora da memória e depois trazido de volta para a memória para execução continuada. Uma troca consiste em encaminhar cada processo no seu todo, executá-lo temporariamente e, então, devolvê-lo novamente ao disco. A operação de sistema de troca é ilustrada na figura 1:

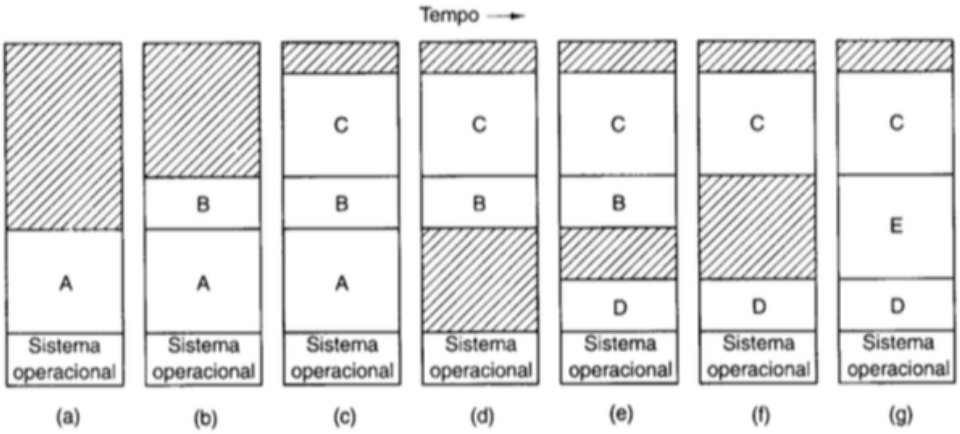


Figure 1: As alterações de alocação de memória enquanto os processos entram e saem da memória (nota-se que as operações sombreadas correspondem a memória não utilizada)

Segmentação

Um aspeto importante na gestão de memória e que tornou inevitável com a paginação é a separação da vista de utilizadores de memória e a atual memória física. Como introduzido no capítulo do livro (Silberschatz, 2005) a vista de utilizador de memória não é o mesmo com o atual memória física. A vista de utilizador é mapiado em memória física. Este mapeamento permite diferenciar entre memória lógica e a memória física.

Embora muitos sistemas são a demanda paginada (discutido no capítulo 10 do livro (Silberschatz, 2005)), existem ainda muitos que não são, em muitos casos, as estratégias de gestão de memória mais simples pode ser melhor, especialmente para sistemas pequenos dedicados. Pretende-se que o aluno aprende os conceitos relacionados com: troca, divisórias, paginação e segmentação.

Leitura: Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.) Capitulo 4 (Gestão de Memória).

Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc. Capitulo 9 (Gestão de Memória).

(Respeite os direitos autorias)

Outras Leituras:

< http://pages.cs.wisc.edu/~solomon/cs537-old/s07/memory.html >, acedido em 23 de Fevereiro de 2016.

Conclusão

É imprescindível que os alunos consciencializam sobre a importância da gerência da memória para um bom desempenho de um computador. Nesta atividade, foi introduzida os aspetos relevantes para entender o funcionamento da memória como parte integrante de um computador, nomeadamente…..

Perguntas de autoavaliação

Responda as seguintes questões com base nas referências acima apresentadas.

1. Identifique duas diferenças entre endereço físico e lógico?
2. Explica a diferença entre fragmentação interna e externa?
3. Quando um processo é colocado para fora da memória, este perde sua capacidade de utilizar o CPU (pelo menos por um tempo). Descreva outra situação em que um processo perde a sua capacidade de utilizar o CPU.
4. Considere um espaço de endereçamento lógico de oito páginas de uma pesquisa 1024 palavras, mapeados em uma memória física de 32 frames.
    a. Quantos bits existem no endereço lógico?
    b. Quantos bits existem no endereço físico?
5. Um sistema de computador tem espaço suficiente para armazenar quatro programas em sua memória principal. Esses programas ficam inativos esperando E/S metade do tempo. Que fração de tempo da CPU é desperdiçada?
6. Considere uma troca em disco no qual a memória consiste nos seguintes tamanhos de lacuna pela ordem de memória: 10k, 4k, 20, 18k, 7k, 12k e 15k. Que lacuna é tomada para sucessivas solicitações de segmento de

    (a) 12k

    (b) 10k

    (c) 9k

no caso do algoritmo de primeiro ajuste? Agora repita a pergunta para melhor ajuste, pior ajuste e próximo ajuste.

Atividade 3.2: Estrutura da tabela de Páginas

Introdução

Nesta atividade, será explorada as técnicas mais comuns para a estrutura de tabelas. Em seguida, será apresentado um laboratório para a sincronização dos processos.

Paginação Hierárquica

Os computadores modernos suportam um grande espaço de endereço lógico. Em tal ambiente, a própria tabela de página torna-se excessivamente grande. Por Exemplo, considere um sistema com um espaço de endereçamento lógico de 32 bits. Se o tamanho da página de tal sistema com 4 KB (), em seguida, uma tabela de página pode consistir em até 1 milhão de entradas (). Assumindo que cada entrada consistem em 4 bytes, cada processo pode precisar de até 4 MB de espaço de endereço físico para a tabela de página sozinho.

Assumindo que cada entrada consiste em 4 bytes, cada processo pode precisar mais do que 4 MB de espaço da memória física para a tabela sozinha. Claramente, pode-se não querer de atribuir a tabela de página de forma contígua na memória principal. Uma solução é dividir a tabela em pequenas partes.

Uma forma é dois níveis de algoritmo de paginação. Veja a figura 2:

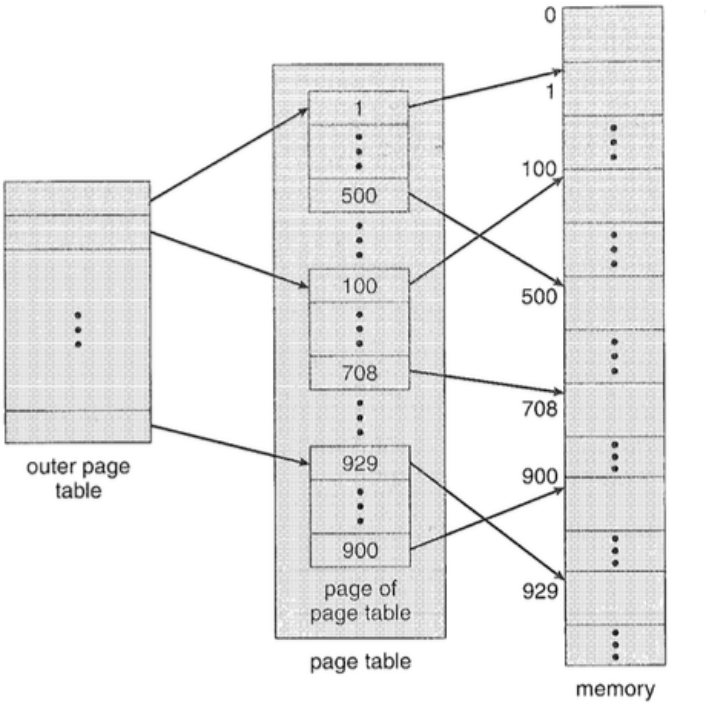


Figure 2: Esquema de dois níveis da tabela de páginas, (Silberschatz, 2005)

Lembre-se do exemplo das máquinas de 32 bits com o tamanho de página de 4 KB. O endereço lógico é dividido em um número de páginas consistindo em 20 bits e um deslocamento de página (page offset) consistindo em 12 bits. Porque é paginado, a tabela de página, o número da página é dividida em um número de página de 10-bit e 10-bit página offset. Então o endereço físico é a seguinte:

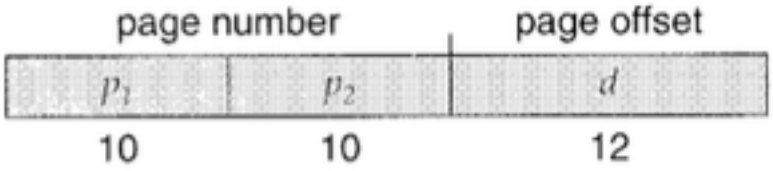


Figure 3: Endereço físico, (Silberschatz, 2005)

Onde, o P1 é um índice para a tabela de páginas externa e P2 é o deslocamento (offset) dentro da página da tabela de páginas externa.

Para os sistemas de 64 bits do espaço do endereço físico, esquema da paginação nível dois não é apropriado. Para ilustrar este ponto, suponha que o tamanho da página em um sistema é 4 KB (). Neste caso, a tabela de página consistem mais de () entradas. Se utilizar o esquema de nível dois então a junção de tabelas pode convenientemente ser uma página longa, ou conter 4 byte () de entrada. Veja o exemplo do endereço:

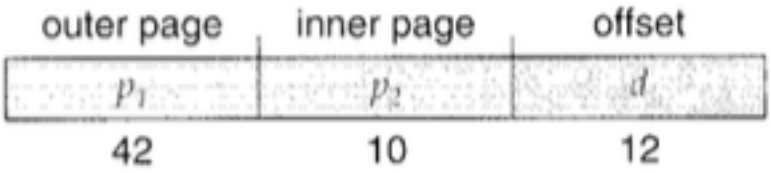


Figure 4: Endereço lógico, (Silberschatz, 2005)

O próximo passo seria o esquema de paginação nível quatro, onde o nível dois a própria tabela de página externa também é paginada. Para arquiteturas de 64 bit, tabelas de páginas hierárquicos são geralmente considerados inadequados. Por exemplo, o 64 bit UltraSPARC pode requerer sete níveis de paginação.

Leituras recomendadas para esta atividade

Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.) Capitulo 4 (Gestão de Memória).

Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc. Capitulo 9 (Gestão de Memória).

(Respeite os direitos autorias)

Ler: < http://www2.cs.uidaho.edu/~krings/CS240/ >, acedido em 24 de Fevereiro de 2016.

Outras leituras:

< http://www.cs.ucsb.edu/~rich/class/cs170/notes/MemoryManagement/ >, acedido em 26 de Dezembro de 2016

Conclusão

Tendo em vista os aspetos observados, nota-se que o fator determinante do método utilizado em um sistema particular é um hardware. Para a melhor compreensão das atividades proposta para esta unidade, recomenda-se a leitura de cada referências antes citadas, com vista a responder as perguntas de auto avaliação.

Perguntas de autoavaliação

Responda as seguintes questões com base nas referências acima apresentadas.

1. A maioria dos sistemas permitem os programas atribuem mais memórias para seu espaço de endereço durante a sua execução. Dados alocados nos segmentos heap do programa é um exemplo de uma memória atribuída. O que é necessário para suportar alocação dinâmica de memória nos seguintes esquemas?
    a. Alocação contíguo da memória
    b. Segmento pura
    c. Paginação pura
2. Porquê as segmentações e paginação são as vezes combinados em um esquema?
3. Explique por que a partilha de um módulo de reentrada é mais fácil quando a segmentação é utilizado, do que quando paginação pura é utilizado.
4. Um computador com endereços de 32 bits utiliza uma tabela de página de dois níveis. Endereços virtuais são divididos em um campo de 9 bits da tabela de páginas de primeiro nível e um deslocamento. Qual é o tamanho das páginas e quantas existem no espaço de endereços?

5. A seguir apresenta-se a listagem de um programa curto na linguagem assembly para um computador com páginas de 512 bytes. O programa está localizado no endereço 1020, e seu ponteiro de pilha está em 8192 (a pilha cresce em direção a 0). Forneça a cadeia de referências de páginas gerada por esse programa. Cada instrução ocupa 4 bytes (1 palavra) e referências tanto de instruções como de dados contam na cadeia de referências.

   - Carregue a palavra 6144 no registador 0
   - Coloque o registador 0 na pilha
   - Chame um procedimento em 5120, colocando o endereço de retorno na pilha
   - Subtraia a constante imediata 16 do ponteiro da pilha
   - Compare o parâmetro real com a constante imediata 4
   - Salte se igual a 5152

Conclusão

Conclui-se que, até o final desta atividade de laboratório o aluno estará familiarizado com a função fork e também terá uma compreensão clara da função wait como um meio para sincronizar o execução de um processo pai e filho.,

Atividade 3.3: Algoritmos de gestão de memória

Os algoritmos de gestão de memória, que são da exclusiva responsabilidade do sistema operativo, determinam que decisões devem ser tomadas e quando devem ser tomadas, utilizando os mecanismos básicos de memória virtual para as executar.

Nesta unidade, são apresentados os algoritmos de gestão de memória usados no Sistemas Operativos.

- Quais são os objetivos dos algoritmos da gestão de memória?
- Onde se deve colocar um bloco (segmento ou página) de programa dada a memória primária livre?
- Quando transferir um bloco de memória secundária para primária e vice versa?
- Que bloco retirar de memória quando não existe mais memória primária livre ou quando a que existe disponível é inferior a um determinado valor considerado mínimo para o bom funcionamento de sistema?

Alocação da Memória

Os algoritmos de alocação de memória decidem, dada a memória livre, onde colocar um bloco de memória de uma determinada dimensão. Os Sistemas Operativos tem a necessidade de utilizar estes algoritmos em diversas situações:

- Criação e terminação de processos - quando um processo se inicia em uma memória virtual, o Sistema Operativo aloca blocos de memória pelo menos para o código, os dados e pilhas e a pilha que são libertados quando o processo termina.
- Expansão do espaço de endereçamento - quado um programa precisa alocar a memória dinâmica, chama uma rotina de biblioteca que pode resultar num pedido a Sistema Operativo pra expandir o espaço de endereçamento ou quando a pilha cresce automaticamente com a execução do programa.

Alocação de segmentos

A alocação de segmentos é mais complicada do que a alocação de páginas, porque introduz o problema da dimensão variável do segmento e da necessidade de encontrar um bloco com a dimensão suficiente para satisfazer o pedido. Este aspeto é idêntico ao que se passa nos sistemas com partições variáveis.

A estrutura de dados que descreve a memória dividida em segmentos é tipicamente uma lista de blocos de memórias livres. Com base nesta estrutura existem diversos algoritmos que diferem na política de escolha do bloco a devolver e que iremos analisar.

1) Best Fit

Este algoritmo pesquisa a lista à procura do bloco cuja dimensão seja a mais próxima da dimensão pedida. Este método segue o raciocínio intuitivo de atribuir ao pedido o bloco que mais se lhe aproxima.

2) Worst Fit

Este algoritmo pesquisa a lista à procura do maior bloco livre. A ideia pode parecer paradoxal, mas tem por objetivo permitir que o fragmento resultante seja ainda suficientemente grande para poder satisfazer outros pedidos.

3) First Fit

O algoritmo First Fit pesquisa a lista por ordem crescente ou decrescente de endereços e escolhe o primeiro bloco livre com dimensão suficiente para satisfazer o pedido.

4) Next Fit

O Next Fit é um First Fit modificado, em que a pesquisa do bloco livre começa no ponto onde terminou a pesquisa anterior, em vez de ser sempre no início da lista.

5) Buddy

Este algoritmo organiza a memória em blocos de dimensão .Normalmente, b*2, designando-se o método por Buddy binário.

Substituição de páginas

Os algoritmos de substituição de páginas são importantes no caso da memória paginada. Os programas executados em um determinado momento geralmente tem a dimensão superior à memória física do computador.

1) LRU (Menos Usado Recentemente)

Uma boa aproximação do algoritmo óptimo é escolher a página que não seja acedida há mais tempo, pois, pela localidade de referência, é aceitável pensar que há uma reduzida probabilidade de ser acedida nos instantes seguintes.

NRU (Não Usado Recentemente)

Um outro método bastante utilizado e que resulta de uma simplificação do LRU é escolher uma página que não tenha sido acedida recentemente.

2) FIFO

O algoritmo FIFO escolhe a página que esteja carregada há mais tempo em memória primária. Para o realizar mantem-se uma fila com as várias páginas em memória. Quando uma página é colocada na memória primária é inserida no fim da lista, quando se necessita de uma página retira-se a primeira lista.

3) Clock

O algoritmo second chance FIFO tem a desvantagem de implicar a movimentação de páginas entre as lista utilizadas. Uma solução que permite evitar esta movimentação consiste em manter as páginas numa dos apresentados nesta unidade foram retiradas do liimplicar a movimentaçm a dimensminou a pesquisa anterior, em vez de ser sempúnica lista circular.

4) Espaço de trabalho

Em relação a todos os métodos anteriores pode –se colocar uma questão: o conjunto de páginas a analisar deve ser o conjunto global das páginas físicas ou só páginas do processo onde ocorreu a falta de página? Define-se o espaço do trabalho de um processo num determinado intervalo de tempo como sendo o conjunto de páginas acedidas pelo processo nesse intervalo de tempo.

Conclusão

Os algoritmos de gestão de memória determinam as decisões tomadas pelo Sistema Operativo no que toca a alocação, transferência e substituição de blocos na memória. Nesta atividade, introduziu-se os conceitos relacionados com a arquitetura da memória e com este influencia no desempenho de cada algoritmo supra apresentados.

Resumo da Unidade

Em virtude do que foi apresentado nesta unidade, é desejável que sejas capaz de executar um processo cujo espaço de endereço lógico é maior do que o espaço de endereço físico disponível. A memória virt l permite-nos executar grandes processos e aumentar a memória física que realmente dispomos. A maioria de Sistemas Operativos oferecem caraterística mapear ficheiros na memória. O processo Kernel tipicamente requere que a memória seja atribuído utilizando paginas que são fisicamente contíguos.

Esta unidade, permitirá aos alunos consolidarem o seu entendimento no que concerne a gestão memória e como os processos operam na gestão de memória.

Avaliação da Unidade

Verifique a sua compreensão!

Laboratório Linux/Unix

Objetivos de aprendizagem: nesta atividade de laboratório, o aluno aprenderá como utilizar o sistema e exec para executar comandos. Também vai aprender como registar um manipulador sinal para um processo e como utilizar os sinais para sincronizar o processo de execução.

Dado um ficheiro (comando.txt) com uma lista de sinais:

Ls –laR

touch main.c

mkdir src

cp main.c src/

Escreva um programa em C que recebe o nome de ficheiro como parâmetro duma linha de comandos e executa os comandos um depois do outro.

a) Primeiro utilizando a função do sistema

b) Depois utilizar o exec.

Sistema de avaliação

Cada resposta ou ação executada com sucesso correto relataram 5%. O sistema de pontuação é descrito pelo seguinte quadro:

Nota para respostas corretas cujo total será 20 pontos	Descrição
>= 17	Superou os objetivos
> 14 < 17	Atingiu os objetivos
> 10 < 14	Atingiu os objetivos parcialmente

Leituras e outros Recursos

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

Leituras primárias

Andrew S., T., & Albert S., W. (2000). Sistemas Operacionais (2nd Edition ed.). Porto Alegre.

Marques, J. A., Ferreira, P., Ribeiro, C., Veiga, L., & Rodrigues, R. (2009). Sistemas Operativos. FCA.

Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc.

Leituras secundárias:

1. The Linux Kernel, http://www.win.tue.nl/~aeb/linux/lk/lk.html#toc4 , acedido em 24 de Fevereiro de 2016
2. Bucknell University, http://www.eg.bucknell.edu/~cs315/wordpress/lab/, acedido em 24 de Fevereiro de 2016
3. Computer Hope, http://www.computerhope.com/win98.htm, acedido em 27 de Fevereiro de 2016
4. WayBackMachine, https://web.archive.org/web/20151117104721/http://www.techsupportalert.com/content/32-bit-and-64-bit-explained.htm, acedido em 23 de Fevereiro de 2016
5. Ruby Hacking Guide, https://ruby-hacking-guide.github.io/thread.html, acedido em 22 de Fevereiro de 2016
6. DEI-ISEP, http://www.dei.isep.ipp.pt/~luis/sop2/index.html, acedido em 27 de Fevereiro de 2016

Unidade 4: Sistemas De Ficheiros

Introdução à Unidade

Os sistemas de ficheiros desempenham um papel importante no sistema operativo. Da perspectiva do utilizador, o sistema de ficheiros é um armazém de ficheiros. No entanto, nos bastidores há muita complexidade. Vamos discutir uma visão geral dos sistemas de ficheiros, uma visão sobre alocação de ficheiros e métodos, bem como algoritmos de alocação no disco.

Objetivos da Unidade

Após a conclusão desta unidade, deverá ser capaz de:

1. Descrever os sistema de ficheiros e a sua função.
2. Discutir os diferentes métodos de alocação de ficheiros.
3. Explicar e descrever alocação no disco e algoritmos associados

Termos-chave

Ficheiro: O ficheiro é nomeado como uma coleção de informações relacionadas que é gravado num dispositivo armazenamento secundário.

Alocação contiguo: Armazena os ficheiros em blocos sequencial

Diretório: Catálogo de nome dos ficheiro que estabelece associações entre nos e os seus descritores

Actividades de Aprendizagem

Actividade 4.1 - Ficheiros

Introdução

Os computadores podem armazenar informações sobre diferente tipos de mídia para armazenamento, como discos magnéticos, fitas magnéticas e discos ópticos. De modo que o sistema de computador será conveniente de utilizar, o sistema operativo fornece uma visão lógica uniforme de armazenamento de informações. O sistema operativo resume-se a partir das propriedades físicas dos seus dispositivos de armazenamentos para definir uma unidade lógica, o ficheiro. Os ficheiros são mapeados pelo sistema operativo para dispositivos físicos. Estes dispositivos de armazenamento são geralmente não volátil, para que o conteúdo sejam persistentes através de falhas de energia e reinicialização do sistema.

O ficheiro é nomeado como uma coleção de informações relacionadas que é gravado num dispositivo armazenamento secundário. Do ponto de vista de um utilizador, um ficheiro é uma cota menor de armazenamento lógica secundário, ou seja, os dados não podem ser gravados para o armazenamento secundário a menos que eles estão dentro de um ficheiro.

Atributos de um Ficheiro

Um ficheiro é chamado, para a conveniência dos seus utilizadores e é referido pelo seu nome. Um nome é normalmente uma sequência de caracteres, como main.c. Quando um ficheiro é nomeado, torna-se independente do processo, do utilizador, e mesmo do sistema que a criou. Em geral, os atributos de um ficheiro variam de um Sistema Operativo para outro.

Um ficheiro consiste em:

- **Nome** – o nome simbólico do ficheiro é a única informações guardadas em um formato legível.
- **Identificador** – este é o identificador único de um ficheiro em um ficheiro de sistema.
- **Tipo** – informações necessárias para sistemas que suportam diferentes tipos de ficheiros.
- **Tamanho** – o tamanho de um ficheiro (em bytes).
- **Proteção** – Controlo de acesso as informações determinam quem pode ler, escrever, executar, apagar.
- **Tempo, data e identificação do utilizador** – esta informação pode ser guardada para a última criação, modificação e utilização.

Operações com Ficheiros

Um ficheiro é um tipo abstrato de dados. Para uma boa definição de um ficheiro é preciso considerar as operações que podem ser efetuadas sobre o ficheiro. O Sistema Operativo pode fornecer chamadas de sistemas para criar, escrever, repor, apagar ficheiros, etc.

- **Criar um ficheiro** – dois passos são necessários para criar um ficheiro. Primeiro, espaço no sistema de ficheiro tem que ser encontrado para o ficheiro e, segundo, uma entrada para o novo ficheiro deve ser feito no diretório.
- **Escrever um ficheiro** – Para escrever um ficheiro, deve fazer uma chamado de sistema especificando o nome do ficheiro e a informação que deve ser escrita.
- **Ler um ficheiro** – para ler a partir de um ficheiro, utiliza-se uma chamada de sistema que especifica o nome do ficheiro e onde o próximo bloco do ficheiro deve ser colocado.
- **Reposição de um ficheiro** – o diretório é procurado para a entrada apropriada, e a posição atual do ficheiro apontador é reposicionado para um dado valor.
- **Eliminar um ficheiro** - para eliminar um ficheiro, procura-se o diretório para o ficheiro nomeado.
- **Truncating um ficheiro** – talvez o utilizar queira eliminar um conteúdo do ficheiro mas guardar os seus atributos.

Tipos de Ficheiros

Quando implementamos um ficheiro de sistema, certamente todo o Sistema Operativo, considera-se que, ou o SO pode reconhecer e suportar o tipo de ficheiro. Veja alguns exemplos de tipos de ficheiros na tabela seguinte:

Tipos de ficheiros	Extensão	Função
Executável	Exe, com, bin ou none	Pronto a executar o programa
objecto	Obj, o	compilado
batch	Bat, sh	comandos
Texto	Txt, doc	Processadores ou editores

Leituras recomendadas para esta atividade

Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.) Capitulo 5 (Gestão de Memória).

Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc. Capitulo 10 (Gestão de Memória).

(Respeite os direitos autorias)

Ler: < http://www2.cs.uidaho.edu/~krings/CS240/ >, acedido em 24 de Fevereiro de 2016.

Outras leituras:

<http://www.cs.ucsb.edu/~rich/class/cs170/notes/FileSystem/index.html>, acedido em 26 de Dezembro de 2016

Conclusão

Todas as aplicações de um computador armazenam informações, isto é, enquanto o processo é executado, este poderá armazenador grandes quantidades de informação dentro do seu espaço e endereço, o que as vezes nem sempre é possível. Nesta atividade, foi abordado os conceitos sobre os ficheiros, tipos e a operações efetuadas. Recomenda-se, para esta atividade as leituras dos seguintes capítulos apresentado na lista de leitura recomendado.

Pergunta de autoavaliação

1. Fornece cinco nomes diferentes de caminho para o ficheiro /etc/passwd/. (sugestão: pense na entradas de diretório "." E "..").
2. Os sistemas que suportam ficheiros sequenciais sempre têm uma operação para retroceder ficheiros. Será que, os sistemas que suportam acesso aleatório a ficheiros precisam disso?
3. A alocação contíguo de ficheiros leva à fragmentação de disco, como mencionado no capítulo. Essa fragmentação é interna ou externa? Faça uma analogia com algo discutido no capítulo anterior do livro.
4. Alguns sistemas eliminam automaticamente todos os ficheiros de utilizadores quando o utilizador faz logs off ou terminar o job, a menos que, o utilizador explicita o pedido para guardar. Outros sistemas guarda todos os ficheiros a menos que o utilizador os elimine. Porquê?

Actividade 4.2 – Métodos de Acesso a um Ficheiro

Introdução

Os ficheiros guardam informações. Quando é utilizado, esta informação deve ser acedida e lida em uma memória de computador. A informação no ficheiro pode ser acedida em várias formas. Alguns sistemas oferece um único método de acesso para ficheiros. Outros, tal com a IBM, suporta muitos métodos de acesso, e escolhe o mais correto para uma aplicação em particular.

Acesso sequencial

O método de acesso mais simples é acesso sequencial. A informação no ficheiro é processado em ordem, um de cada vez. Este modo de acesso é a mais comum, como o exemplo de editores e compiladores.

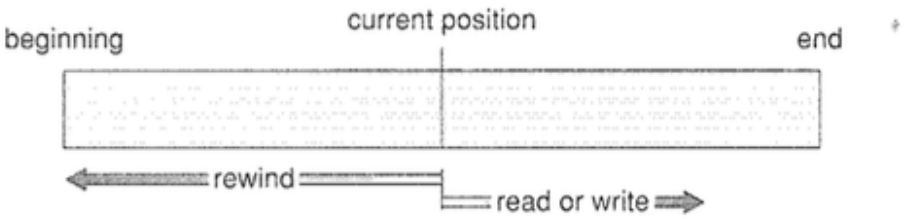


Figure 1: Ficheiro de acesso sequencial

Acesso Direto

Um outro método é o acesso direto. Um ficheiro é criado com um tamanho fixo, com relatórios lógicos que permite os programa ler e escrever rapidamente em uma ordem particular. O método de acesso direto é baseado em um modelo de disco de ficheiro, desde de que os discos permitem o acesso randomizado a qualquer bloco de ficheiro.

Estrutura do Diretório

Até este ponto foram discutido um ficheiro de sistema, Na realidade um sistema pode ter 0 ou mais ficheiros de sistema, e o ficheiro de sistema pode variar para tipos diferentes. Por exemplo, o sistema típico Solaris talvez tenha um pouco UFS ficheiro de sistema, um VFS, e alguns NFS.

O ficheiro de sistema de computadores, pode ser extensiva. Alguns sistemas armazenam milhões de ficheiros em terabytes do disco. Para gerir todos estes dados, precisa-se ser organizado. Esta organização envolve a utilização dos diretórios.

- **Estrutura de Armazenamento** – Um disco pode ser utilizado em o ficheiro de sistema.
- **Resumo do Diretório** – O diretório pode ser visto como uma tabela de símbolo que traduze os nomes dos ficheiros nos seus diretórios de entrada.
- **Nível mais simples do Diretório** – A estrutura mais simples do diretória é nível mais simples do diretório. Todos os ficheiros encontram-se no mesmo diretório, que é fácil de suportar e entender.
- **Nivel dois do diretório** – Neste nível cada utilizador tem o seu próprio diretório de ficheiros do utilizador.
- **Nível três do diretório** – uma vez visto nível dois do diretório, a generalização é estender a estrutura do diretório para uma árvore arbitrário.
- Grafo do diretório geral
- Grafo acíclico do diretório

Montar um Ficheiro de Sistema

Assim como um ficheiro deve ser aberto antes é utilizado, um ficheiro de sistema também deve ser montado antes de ser disponibilizado para os processos do sistema. Mais especificamente o diretório do sistema pode construir uma estrutura de múltiplos volumes, que deve ser montado para fazer que sejam disponíveis entre os ficheiros de sistema nome do espaço.

Leituras recomendadas para esta atividade

Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.) Capitulo 5 (Gestão de Memória).

Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc. Capitulo 10 (Gestão de Memória).

(Respeite os direitos autorias)

Ler: < http://www2.cs.uidaho.edu/~krings/CS240/ >, acedido em 24 de Fevereiro de 2016.

Outras leituras:

<http://www.cs.ucsb.edu/~rich/class/cs170/notes/FileSystem/index.html>, acedido em 26 de Dezembro de 2016

Conclusão

Quando visto de fora, um sistema de ficheiro é uma coleção de ficheiros e de diretórios, mais as operações sobre os mesmo. Por isso é crucial entender os métodos de acesso aos sistemas de ficheiros e a sua estrutura para assim poder operar sobre o mesmo.

Pergunta de autoavaliação

1. Similarmente, alguns sistemas suportam muitos tipos de estruturas para um ficheiro com dados, enquanto outros simplesmente suportam um stream de bytes. Quais são as vantagens e desvantagens?
2. Explique o propósito das operações open e close.
3. Dado um exemplo de uma aplicação em qual os dados em um ficheiro deve ser acedidos na seguinte ordem:
   a) Sequêncial
   b) Randomizado
4. Em alguns sistemas os subdiretórios podem ser lidos e escritos por um utilizador autorizado, apenas um ficheiro comum pode ser.
   a. Descreve os problemas de proteção que pode acontecer.
   b. Sugere um esquema para lidar com ca um dos problemas de proteção nomeado na alínea a.
5. O espaço livre em disco pode ser monitorizado utilizando uma lista de blocos livres ou um mapa de bits. Os endereços de disco requerem D bits. Para um disco com B blocos, F dos quais estão livres, declare a condição sob qual a lista de livres utiliza menos espaço que o mapa de bits. Para D tendo o valor de 16 bits, expresse a sua resposta como percentagem do espaço em disco.

Actividade 4.3 – Laboratório Ficheiro de Sistema Unix

Introdução

Objetivos de aprendizagem: nesta atividade de laboratório, o aluno aprenderá a diferença entre ficheiros texto e ficheiros binários.

1. Escreva um programa C que copia um ficheiro de texto utilizando funções de Entrada/Saida ANSI das funções C (fgetc / fputc, scanf, fprintf, fgets / fputs). Os nomes de ficheiros de origem e de destino são especificados como parâmetros da linha de comando.
2. Escreva uma função criarFicheiro que contém apenas um destino de compilação para gerar o ficheiro executável a partir do ficheiro de origem.
3. Modifica a função criarFicheiro adicionando um novo alvo com o nome install. O alvo deve criar um diretório nomeado bin em diretório pai e copiar o ficheiro executável.
4. Modifica o programa criarFicheiro que, será capaz de gerar todos os executáveis e cada um com nomes diferentes.

Conclusão

Nesta atividade, o aluno aplicou os conhecimento aprendidos para operar com o texto e ficheiros binários.

Resumo da Unidade

Os ficheiros são um dos objetos mais valiosos manipulados pelo Sistema Operativo. Esta unidade, permitirá os alunos a desenvolverem as suas habilidades de manipular de ficheiros que é o core de um Sistema Operativo. Nesta unidade com base nos capítulos dos livros recomendados para a leitura, foi discutido vários métodos para armazenar informações no armazenamento secundário.

Avaliação da Unidade

Verifique a sua compreensão!

Bloco de controlo de ficheiros

Considere um ficheiro que consiste em 100 blocos. Suponha que o bloco de controle do ficheiro já está na memória. Calcule quantas operações de disco I/O são necessárias para estratégias de alocação contíguos, ligados, e indexado (nível individual), se, por um bloco, as seguintes condições é estabelecida. No caso alocação contígua, suponha que não há espaço para aumentar no início, mas há espaço para aumentar no final. Assume que o bloco de informação a ser adicionada é armazenado na memória.

a. O bloco é adicionado no início
b. O bloco é adiciona no meio
c. O bloco é adicionado no final
d. O bloco é removido a partir do início
e. O bloco é removido a partir do meio
f. O bloco é removido a partir do final

Sistema de avaliação

Cada resposta ou ação executada com sucesso correto relataram 5%. O sistema de pontuação é descrito pelo seguinte quadro:

Nota para respostas corretas cujo total será 20 pontos	Descrição
>= 17	Superou os objetivos
> 14 < 17	Atingiu os objetivos
> 10 < 14	Atingiu os objetivos parcialmente

Leituras e outros Recursos

1. As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.
2. Tanenbaum, A., & Woodhull, A. (2000). Sistemas Operacionais (Vol. 2). (P. Alegre, Ed.) Capitulo 5 (Gestão de Memória).
3. Silberschatz, A. (2005). Operating System Concepts (7ª ed.). (B. Zobrist, Ed.) John Wiley & Sons, inc. Capitulo 10 (Gestão de Memória).
4. (Respeite os direitos autorias)
5. Ler: < http://www2.cs.uidaho.edu/~krings/CS240/ >, acedido em 24 de Fevereiro de 2016.
6. Outras leituras:
7. <http://www.cs.ucsb.edu/~rich/class/cs170/notes/FileSystem/index.html>, acedido em 26 de Dezembro de 2016

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