Sistemas Operativos

1.1 – Introdução aos Sistemas Operativos

Componentes de um Sistema de Computação

Hardware – oferece os recursos de computação básicos (CPU, memória, dispositivos de I/O).

Programas do sistema (incluindo o Sistema Operativo) – controlam e coordenam a utilização do hardware pelos vários programas de aplicação de diferentes utilizadores.

Programas de aplicação – definem uma forma de utilização de recursos para resolver os diferentes problemas dos utilizadores (compiladores, bases de dados, jogos, programas de gestão).

Utilizadores (pessoas, outros computadores, ...).

Explicar as funções de um sistema operativo:

Um conjunto de dispositivos eletrônicos (*hardware*) capazes de processar informações de acordo com um programa (*software*).

Fornece as bases para a execução das aplicações, às quais o usuário deseja executar.

Um programa que age como um intermediário entre o utilizador e o hardware criando um ambiente adequado para que um utilizador possa executar os seus programas.

Gere os recursos e a informação entre o software e o hardware.

Explicar a diferença entre o sistema operativo e outros programas do sistema:

Software é uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redireccionamento ou modificação de um dado/informação ou acontecimento. "*Software*" também é o nome dado ao comportamento exibido por essa sequência de instruções quando executada

É composto por diversas funções, bibliotecas e módulos que gera um programa executável ao final do processo de desenvolvimento e este, quando executado, recebe algum tipo de “entrada” de dados (*input*), processa as informações segundo uma série de algoritmos ou sequências de instruções lógicas e libera uma saída (*output*) como resultado deste processamento.

Descrever a organização de um sistema operativo, as suas componentes e os seus serviços:

Um Sistema Operativo gere a sua organização em três categorias:

1.Drives, Discos ou unidade de armazenamento

2.Diretorias ou Pastas

3. Ficheiros

1.) 2.) 3.)

Exemplo: C:\Pasta\Fotos

Este vê os recursos que precisa para uma aplicação correr e também como intermediário entre hardware e software.

Evolução dos Sistemas Operativos:

Sistemas Batch (Primeiros Sistemas Operativos):

O sistema era manuseado apenas pelo operador

Equipado com um leitor de cartões

Utilizador prepara uma tarefa (programa, dados e informação de controlo) para submissão

Operador controla execução das várias tarefas

Resultado da execução entregue posteriormente ao utilizador

Problema: baixo desempenho I/O e processamento não se sobrepõem; leitor de cartões muito lento

Sistemas Multi-programação

Várias tarefas são mantidas em memória ao mesmo tempo, sendo o CPU multiplexado entre elas.

Requisitos:

Gestão de memória - o sistema tem de atribuir a memória às diferentes tarefas.

Escalonamento do CPU - o sistema tem de selecionar a tarefa a executar de entre todas as tarefas prontas a executar.

Proteção entre tarefas.

Suporte a I/O.

Sistema Time-Sharing

Requisitos:

Gestão de memória

Gestão de ficheiros.

Sistemas de proteção.

Algoritmos de escalonamento de tempo de CPU sofisticados e mais.

Sistemas para PC

A evolução de sistemas complexos de janelas introduziu o requisito da multitarefa

A participação em redes de computadores fez evoluir o requisito de proteção de ficheiros (multiutilizador)

Exs: Windows XP/Vista/7 e Linux

Sistemas Operativos Distribuídos

Distribui a computação por vários computadores (Cada computador tem a sua própria memória, mas comunicam com outros computadores através de linhas de comunicação).

Vantagens dos sistemas distribuídos:

Partilha de recursos

Aumento de desempenho e distribuição de carga

Robustez Comunicação

Sistemas Operativos de Rede

Partilha de ficheiros

Facilidades de comunicação

Independente de outros computadores na rede

Sistema Operativo Distribuído

Menos autonomia entre computadores

Dá a impressão de que existe apenas um sistema operativo a controlar um conjunto de computadores em rede

Sistemas de tempo real

São utilizados quando existe um tempo requerido que tem de ser garantido e só ocorre se o resultado for correto no tempo limite.

Sistemas Hard Real-Time

Garantem que a tarefa crítica ocorra num tempo especificado (em vez de um tempo limite).

Incompatível com sistemas time-sharing.

Ex: Sistema de Travagem de um Carro

Sistemas Soft Real-Time

Permite especificar um limite de tempo e gere as prioridades para cumprir com esses limites.

Compatível com sistemas time-sharing

Ex: Transmissão de vídeo.

Organização do Sistema Operativo:

Gestão de processos:

Um processo é um programa em execução.

Um processo requer um determinado número de recursos, incluindo tempo de CPU, memória,

ficheiros, e dispositivos de I/O.

O sistema operativo, através do gestor de processos, é responsável por:

Criar e terminar processos

Suspender e acordar processos

Oferecer mecanismos para:

- Sincronização de processos

- Comunicação entre processos

Identificação de processos em ambientes de Multi-programação (uid - user identification - e gid - group identification)

Gestão de Memória:

A memória principal é um enorme array de bytes cada um com o seu endereço.

É acedida diretamente pelo CPU e dispositivos de I/O

Os programas para serem executados tem de ser carregados, pelo menos parcialmente, para memória principal.

O sistema operativo, através do gestor de memória, é responsável por:

- Manter o registo de quais as partes de memória estão a ser usados num dado instante e por quem.

- Decidir sobre que processos carregar para memória quando existe espaço livre. Atribuir e libertar espaço de memória quando necessário.

- Atribuir e libertar espaço de memória quando necessário.

Gestão de ficheiros

Memória principal é volátil e diminuta para armazenar todos os programas e dados permanentemente.

Os sistemas de computação oferecem a memória secundária.

Um ficheiro é uma estrutura de dados que esconde a complexidade dos dispositivos físicos de armazenamento de informação.

O sistema operativo, através do sistema de ficheiros, é responsável por:

Criar e apagar ficheiros

Criar e apagar diretorias

Suportar primitivas para manipular ficheiros e diretorias

Associar ficheiros ao sistema de memória secundária

Gestão do espaço livre em disco Atribuição de espaço em disco

Escalonamento de serviço de I/O~

O sistema operativo é responsável por:

Esconder a complexidade dos dispositivos de I/O ao utilizador, oferecendo-lhe uma interface abstrata de alto nível.

Sistema de proteção

O mecanismo de proteção é responsável por controlar o acesso de processos ou utilizadores tanto a recursos do sistema como do utilizador.

O sistema operativo, através do sistema de proteção, é responsável por:

- Distinguir entre o acesso autorizado e o acesso não autorizado.
- Especificar o controlo a impor.
- Providenciar uma forma de implementar essa proteção.

O interpretador de comandos

Programa que estabelece uma interface entre o utilizador e o sistema operativo para a execução de operações genéricas

Interpreta e executa comandos ao sistema operativo, como por exemplo executar um programa

Implementa um ciclo de leitura, interpretação e execução de comando

No Unix/Linux é conhecida pela Shell

Chamadas ao sistema (system calls)

Oferecem uma interface entre os programas e o sistema operativo.

Genericamente, disponíveis como instruções assembly.

Podem ser agrupadas por serviços

Gestão de ficheiros

Gestão de memória

Gestão de processos

….

1.2 – Arquitetura de Software

Compromissos no desenho de SO

Desempenho

A Sistema Operativo corresponde a uma sobrecarga no desempenho das aplicações.

Funcionalidades adicionais devem ser analisadas em função do custo de desempenho.

Proteção e segurança

Multiprogramação ⇒ partilha de recursos

Desta forma é necessário garantir o isolamento dos recursos, como por exemplo a memória

Correção, Evolutibilidade e Portabilidade

Segurança depende do funcionamento correto do software ⇒ software *trusted* vs software *untrusted*.

Evolutibilidade refere-se à capacidade de o software evoluir (+ funcionalidades).

Portabilidade refere-se adaptação do sistema operativo a diferentes máquinas (hardware) com o mínimo de alterações.

Mecanismos básicos

- Modos de execução

mode bit: modo Supervisor ou Utilizador:

Elemento chave para o conceito de trusted software

modo Supervisor:

Pode executar todas as instruções máquina

Pode referenciar todas as posições de memória

modo Utilizador:

Pode executar apenas um subconjunto de instruções

Pode referenciar apenas um subconjunto de posições de memória

- O conceito de Kernel

Parte do sistema operativo crítica para o funcionamento correto (trusted software - software de confiança)

Executa em modo Supervisor

A instrução máquina trap serve para fazer a mudança de modo utilizador para modo supervisor

Desempenho: modo utilizador vs modo supervisor

Interrupções:

O kernel reage a pedidos que partem de entidades externas

Uma interrupção é um sinal electrónico (interrupt request - IRQ) produzido pela entidade externa (por exemplo um periférico), causando a interrupção do CPU e a execução de uma rotina de atendimento (interrupt service rou0ne - ISR)

Interrupções concorrentes: cli() e sti()

Requisitos de serviços do kernel:

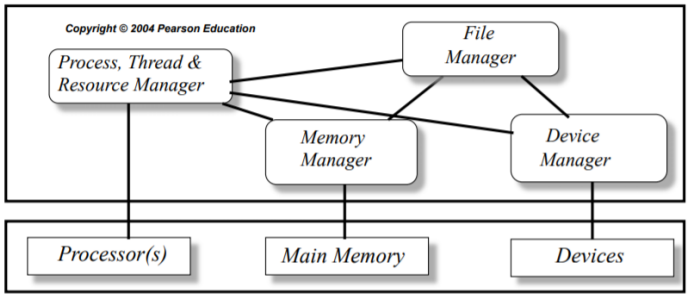
As aplicações veem o kernel como um tipo abstrato de dados (ADT – Abstract Data Type) (ou objeto), o qual mantêm um estado e oferece um conjunto de funções

Implementa a interface POSIX

Se a função é implementada por um módulo ou device driver, o kernel recebe o pedido e redirecciona-o para o módulo ou driver apropriado

Arquitecturas de Software

Organização lógica do *Kernel*



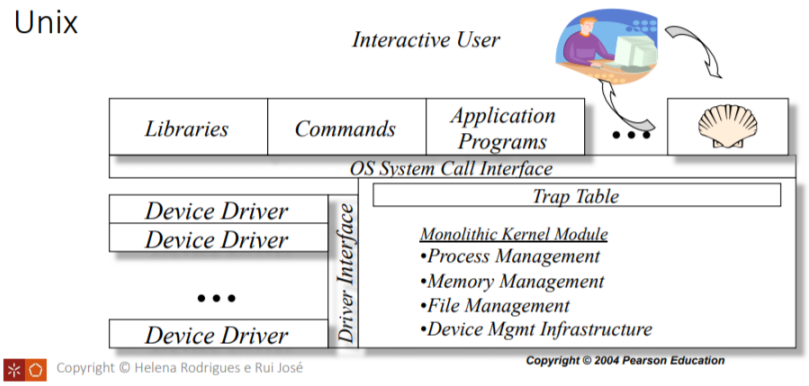
Kernel monolítico

Integra as componentes lógicas do sistema operativo num único módulo

Baseia-se no elevado grau de interação entre os diferentes módulos

Arquitecturas de Software

Unix



Unix

Implementava funcionalidade (processos, memória, ficheiros, E/S) mínima no kernel

Pretendia no início ter um grau elevado de portabilidade

Ambiente de computação específicos eram criados como extensões a esta funcionalidade mínima. Ex: Interpretador de comandos

Kernel monolítico:

Melhor desempenho

Menor Evolutibilidade

Menor robustez/fiabilidade

Microkernel

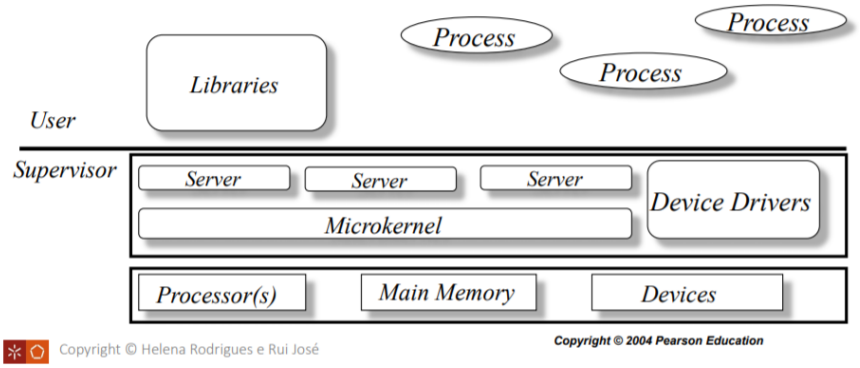
As componentes não essenciais são removidas do kernel e implementadas ao nível dos programas do sistema e utilizador

Normalmente, um microkernel oferece uma funcionalidade mínima de gestão de processos, memória, I/O e comunicações

As funções do sistema operativo executadas em modo utilizador implicam normalmente variadas chamadas ao (micro)kernel (trusted so4ware)

Arquitecturas de Software

Organização do *microkernel*



Microkernel

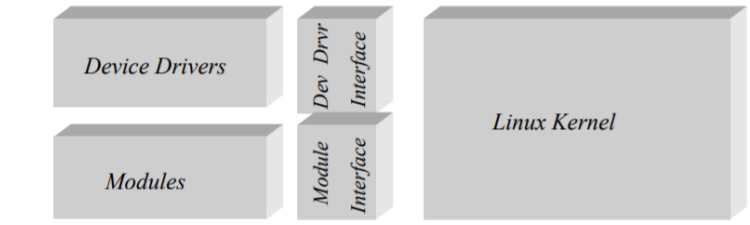
Maior modularidade, maior flexibilidade, maior evolutibilidade

Maior imunidade a bugs, maior robustez

Melhor adaptabilidade a sistemas distribuídos

O Sistema operativo Linux - Arquitectura

Organização do *Kernel Linux*



Módulos Linux

O Linux é baseado numa arquitetura tipo kernel monolítico

Um módulo Linux é um conjunto de funções e estruturas de dados compilados como um programa independente

Os módulos podem ser instalados estaticamente, quando o kernel inicia, ou dinamicamente, em tempo de execução

Os módulos Linux oferecem uma interface para ser utilizada pelas aplicações

Os módulos Linux executam em modo supervisor

Os módulos Linux são instalados e removidos com os comandos insmod/modprobe e rmmod respetivamente

Novas funcionalidades podem ser adicionadas sem necessidade de modificar o kernel

Maior facilidade de debug

Menos memória utilizada

Exemplos:

Ex1: Interface para um sistema de ficheiros distribuído (nfs) (analisar o conteúdo do ficheiro /proc/modules)

Ex2: outros sistemas de ficheiros (Atenção, pelo menos a interface para o sistema de ficheiros que contém a / terá de fazer parte do kernel base)

Ex3: device drivers

Ex4: Chamadas ao sistema

2.1 – Gestão de Processos

Num sistema de computação decorrem simultaneamente várias tarefas

Como suportar essas tarefas ao nível do sistema operativo (multitarefa)?

Gestão de recursos

Gestão das tarefas

Tipo de tarefas

Processo

Um programa em execução

Uma instância de um programa em execução num sistema de computação

Uma entidade que pode ser atribuída e executada pelo processador

Caracterizada pela execução de uma sequência de instruções, um estado corrente e um conjunto de instruções e informação do sistema

Elementos de um processo

identificador

estado de execução

program counter

registos do CPU

instruções e dados informação de accoun.ng: criador do processo, utilizador, grupo, tempos de CPU, ...

estado de I/O: lista de dispositivos em utilização

prioridade

Process Control Block (PCB)

Contém os elementos do processo

Criado e mantido pelo kernel

Possibilita o suporte a múltiplos processos

Os processos competem pelos recursos do sistema: memória, disco, processador, periféricos e se possuir todos estes recursos está pronto a ser executado pelo processador, se um dos recursos que necessita está indisponível o processo bloqueia até o recurso estar disponível.

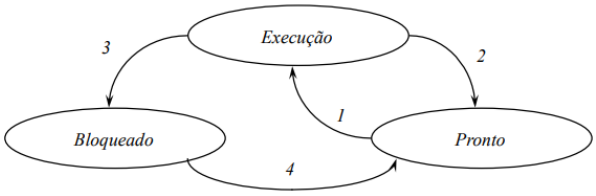
O sistema operativo mantém informação sobre que processos estão bloqueados e que processos esperam por que recursos.

O gestor de recursos é responsável por atribuir os recursos aos processos bloqueados segundo uma determinada política.

Gestão de Processos

Após a sua criação, um processo vai alternando entre 3 estados:

- execução
- bloqueado
- pronto



Pseudo-paralelismo

Em ambientes de multi-programação múltiplos processos progridem em paralelo no sistema de modo a maximizar a utilização do CPU (multitarefa)

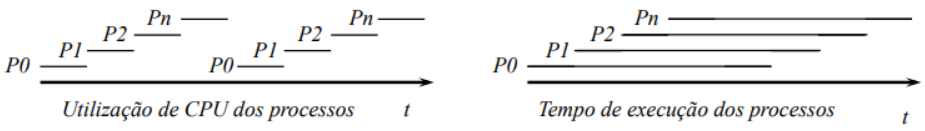
Em cada instante, o processador só executa as instruções de um determinado processo

Os processos são suspensos e reiniciados várias vezes durante a sua execução

Gestão de Processos

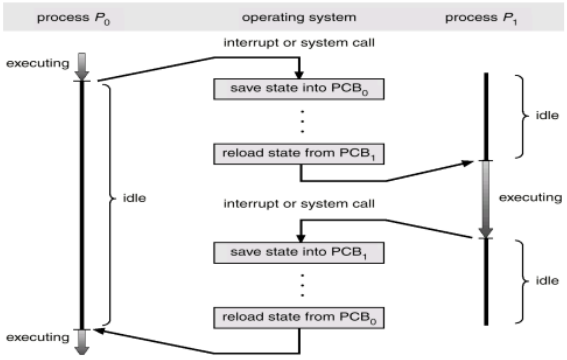
Pseudo-paralelismo

O tempo de execução de um processo varia de execução para execução (a menos que o processo tenha requisitos de tempo real...)



Gestão de Processos

Trocas de contexto entre processos



Copyright © 2000 Pearson Education

Escalonamento de processos (Scheduling)

uma das principais funções do SO: faz a gestão do CPU que é um recurso crítico do sistema

tenta otimizar a sua utilização, mantendo-o tanto quanto possível ocupado

a sua relação com os restantes sistemas é importante. Exemplo: gestão de memória

responsável pela transição execução → pronto do diagrama de estado e pronto → execução
decide que processo vai executar, quando e por quanto tempo

Gestão de Processos

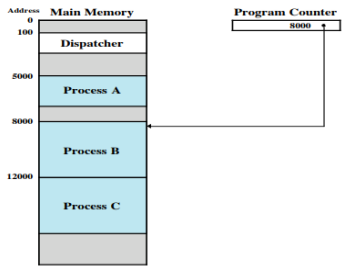


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

Gestão de Processos

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.3 Traces of Processes of Figure 3.2

Gestão de Processos

1	5000		27	12004	
2	5001		28	12005	
3	5002			Time out	
4	5003		29	100	
5	5004		30	101	
6	5005		31	102	
	Time out		32	103	
7	100		33	104	
8	101		34	105	
9	102		35	5006	
10	103		36	5007	
11	104		37	5008	
12	105		38	5009	
13	8000		39	5010	
14	8001		40	5011	
15	8002			Time out	
16	8003		41	100	
	I/O request		42	101	
17	100		43	102	
18	101		44	103	
19	102		45	104	
20	103		46	105	
21	104		47	12006	
22	105		48	12007	
23	12000		49	12008	
24	12001		50	12009	
25	12002		51	12010	
26	12003		52	12011	
				Time out	

100 = Starting address of dispatcher program
shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

Principais operações suportadas pelo sistema operativo

Criação de um processo

Execução de um programa

Terminação de um processo

Obter informação sobre um processo

Criação de processos

Conceptualmente existe um processo que carrega o kernel para memória.

O método convencional para criação de processos é a execução de uma chamada ao sistema (system call)

A chamada ao sistema é invocada pelo processo conhecido como processo pai e dá origem a um novo processo conhecido como processo filho

O modelo para criação de processos define a partilha de recursos entre os processos pai e filhos

Os processos pai e filho executam concorrentemente ou, em alternativa, o processo pai espera que o processo filho termine

Terminação de processos

Um processo executa a última instrução e de seguida requisita ao kernel que o elimine:

- passagem de informação entre processo pai e filho.
- os recursos do processo são apagados pelo sistema operativo.

Processo pai pode terminar a execução de um processo filho quando:

- o processo filho excede os recursos.
- a tarefa atribuída ao processo filho não é necessária.
- o processo pai termina.

Modelo Posix (Unix e Linux)

Chamada ao sistema fork():

- cria um novo processo
- novo process identifier (PID)
- o espaço de endereçamento é duplicado
- partilha de recursos, mas execução independente

execução concorrente de processos

hierarquia de processos

Chamada ao sistema wait():

- permite ao processo pai esperar pelo fim da execução de um dos filhos
- se há mais do que um filho que ainda não terminou bloqueia o processo pai até que pelo menos um dos filhos termine
- retorna o pid do filho que terminou
- permite ao pai saber como é que o processo filho terminou a sua execução

Chamada ao sistema exit():

- permite a um processo terminar-se a si próprio
- não retorna ao processo que a invocou
- o kernel através da chamada ao sistema wait() faz a interface entre pai e filho

Gestão de Processos

```
main()
{
        int fpid;
        if ( (fpid = fork()) == -1){
                perror("Erro ao fazer fork");
                exit(1);}
        else if (fpid == 0)   {
                /* Processo filho */
                printf("** Filho **\n");
                printf("Pid do Filho: %d\n",getpid());
                printf("Pid do Pai: %d\n",getppid());
                exit(0);}
        else {
                /* Processo pai */

                printf("** Pai **\n");
                printf("Pid do Filho: %d\n",fpid);
                printf("Pid do Pai: %d\n",getpid());
                exit(0);}
```


Chamada ao sistema execve() (variantes execl, execlp, execle, execv, execvp, execve):

- substitui o espaço de endereçamento do processo que a invoca pelo novo programa
- não é criado um novo processo
- tipicamente é invocada após um fork() pelo processo filho (ex: interpretador de comandos)

Gestão de Processos

```
main(){
int fpid, status;
if ( (fpid = fork()) == -1){
        perror("Erro ao fazer fork"; exit(-1);}
else if (fpid == 0)       {
        /* Processo filho */
        execl("/usr/ucb/ls","ls","-la",NULL);
        perror("Erro ao fazer execl"); exit(-1);}
else {
        /* Processo pai */
        wait(&status);
        if (WIFEXITED(status)){
                printf("Filho terminou normalmente>");
                printf("%d\n",WEXITSTATUS(status));}
        if (WIFSIGNALED(status)){
                printf("Filho terminou com sinal ->");
                printf("%d\n",WTERMSIG(status));
        exit(0);}
}
```

2.2 – Gestão de Processos

Interação entre tarefas

Uma tarefa é independente se a sua execução não afeta ou não é afetada pela execução de outras tarefas no sistema

Uma tarefa é cooperante se a sua execução afeta ou é afetada pela execução de outras tarefas no sistema

Partilha de informação: diferentes tarefas num sistema podem necessitar dos mesmos dados.

Desempenho: diferentes tarefas podem estar a cooperar para o mesmo objetivo, contribuindo para um aumento de desempenho

Modularidade: uma tarefa pode ser dividida em subtarefas que comunicam entre si.

Conveniência: facilitar a execução concorrente de várias tarefas dentro de uma mesma aplicação

Exemplo de scheme:

```
(define x 10)

(parallel-execute (lambda () (set! x (*x x))) (lambda () (set! X (*x x x))))
```

Para que duas tarefas sejam cooperantes, o sistema operativo terá de permitir a comunicação entre processos e para comunicarem é necessário ocorrer: Memória partilhada ou Troca de Mensagens.

Threads

Um processo é um programa em execução e pode ser visto como tendo duas componentes fundamentais:

- A gestão dos recursos necessários para a execução do programa (variáveis, ficheiros abertos, processos filhos, etc...)
- Uma linha de execução que vai percorrendo o código correspondente ao programa até atingir o final (program counter, uma stack e registos que vão guardando os valores das variáveis que estão a ser processadas num determinado momento)

O conceito de thread vem separar estas duas componentes e permite assim que a um mesmo conjunto de recursos gerido por um processo possam corresponder diversas linhas de execução.

Permite suportar tarefas concorrentes no âmbito de um único processo

Multithreading

Possibilidade de suportar múltiplas threads de execução no mesmo processo

Situação análoga à de ter múltiplos processos na mesma máquina

Como as threads têm muitas semelhanças com o conceito de processo são por vezes designadas por processos ligeiros (lightweight process)

ATENÇÃO: Threads não são processos!

Uma thread tem em exclusivo:

Um thread ID Um program counter

Um conjunto de registos

Uma stack

Uma thread partilha com as restantes threads do mesmo processo

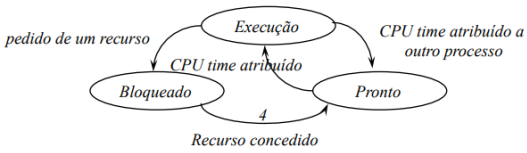
Um process ID

Código Memória de dados (variáveis globais)

Recursos (ficheiros abertos, sinais, etc..)

Gestão de processos

Tal como nos processos a execução de uma *thread* também pode passar por vários estados



Problema?

As threads partilham as variáveis globais

Ao contrário dos processos não existe protecção entre threads.

Uma thread pode apagar completamente os dados globais

Dados partilhados não são normalmente um problema

As várias threads fazem parte do mesmo programa, escrito pela mesma pessoa

São feitas de forma articulada para cooperarem e não para serem intencionalmente malcomportadas

Quais são as vantagens de ter multithreading?

Agilidade na resposta (responsiveness):

- Programas interativos podem ser mais solícitos a responder aos pedidos dos utilizadores
- Nota: não confundir com desempenho

Partilha de recursos

- Forma de partilhar recursos mais simples do que a memória partilhada entre múltiplos processos

Eficiência

- Criação mais simples do que nos processos
- Trocas de contexto mais simples do que nos processos

Melhor uso de arquiteturas multiprocessador

Um programa pode criar linhas de execução adicionais recorrendo à classe Thread (ou interface runnable).

2.3 – Gestão de Processos

Sincronização de Processos e threads

Dois ou mais processos/threads interagem/cooperam para executar uma aplicação comum.

Ex: um processo/thread apenas poderá prosseguir depois de outro ter executado uma determinada operação.

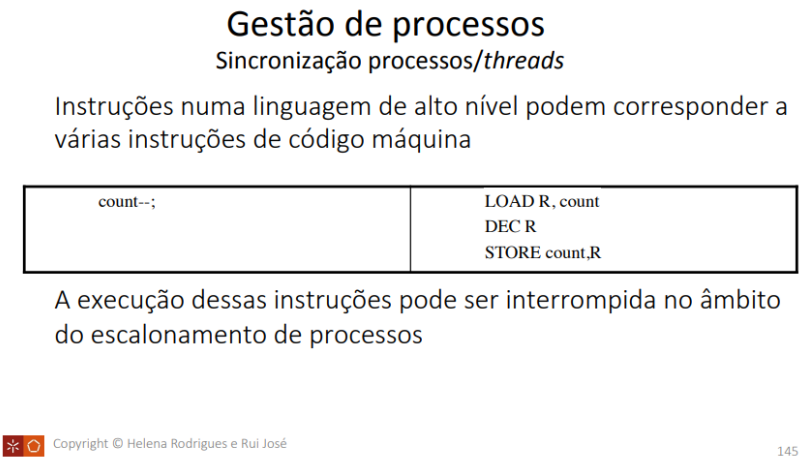
dois processos/threads acedem em simultâneo a uma variável

A competição pela obtenção de um recurso único ou limitado impõe a necessidade de mecanismos de sincronização que garantam que o recurso é utilizado de forma consistente pelos processos/threads envolvidos.

Um processo/thread executa uma tarefa seguindo para isso uma sequência de passos (algoritmo)

Execução sequencial desses passos pode ser correta num cenário em que apenas exista um processo/thread a executar a tarefa

Execução pode falhar se vários processos/threads estiverem a executar concorrentemente essa mesma tarefa ou a manipular os mesmos dados



Race conditions

Situação em que um ou mais processos/threads manipulam concorrentemente os mesmos dados e em que o resultado final vai depender da ordem em que as diferentes instruções são executadas

Situação em que não há garantia de que os valores obtidos são equivalentes aos valores que seriam obtidos nos casos em que as duas threads/processos seriam executados sequencialmente em alguma ordem

Correção de programas concorrentes (I)

Na programação concorrente temos de garantir que os resultados obtidos são equivalentes aos resultados que seriam obtidos nos casos em que os processos/threads são executados sequencialmente em alguma ordem

A execução dos processos/threads não necessita de ser sequencial - grau de concorrência

Esta parte está nos resumos do Zé

Correcção de programas concorrentes (II)

Definir as secções críticas dos programas e garantir:

1. Exclusão mútua no acesso à secção crítica. Se o processo/thread Pi está a executar uma secção crítica nenhum outro processo/thread pode estar a executar a mesma secção crítica.

2. Progresso. Nenhum processo/thread a executar fora de uma secção crítica pode bloquear outro processo/thread que deseja entrar na mesma secção crítica.

3. Tempo de espera limitado. Nenhum processo/thread, após requisitar a execução da sua secção crítica, poderá ficar indefinidamente à espera.

Variável de exclusão mútua (mutex)

objeto do Sistema Operativo para a implementação de secções críticas em programas concorrentes

oferece duas operações genéricas:

- fechar (mutex): chamada pelo processo/thread quando quer entrar na secção crítica
  - se a secção crítica estiver aberta, o processo/thread entra na secção crítica e fecha-a
  - se a secção crítica estiver fechada, o processo/thread bloqueia até esta ser aberta; nessa altura, entra na secção crítica e fecha-a
- abrir (mutex): chamada pelo programa quando quer sair da secção crítica
  - abre a secção crítica
  - se houver processos/threads bloqueados na secção crítica, acorda um

Semáforo S – variável inteira positiva partilhada

Tem associada uma fila de processos/threads bloqueados apenas suporta duas operações atómicas:

```
wait (S) {
        if (value(S) == 0)
            block(P); // insere P na fila do semáforo;
        value(S) --;
}
post(S) {
        value(S)++;
        if (value(S) == 1)
            wakeup(Q); // remove Q da fila do semáforo
}
```

Os programas produtor/consumidor são programas concorrentes em que existem processos/threads produtor que geram algum tipo de recurso (dados ou não) que será consumido pelos processos/threads consumidor. Ex. Serviços de impressão

Semáforos genéricos

Os semáforos são utilizados para sincronizar processos/threads noutras situações que não de exclusão mútua.

São utilizados para sincronizar programas do tipo produtor/consumidor.

Os problemas de sincronização são: o consumidor só pode consumir após o produtor ter produzido; em alguns casos, se houver limite de produção, o produtor só pode produzir sempre que o limite não for atingido.

Metodologia para serialização de secções críticas

1. Identificar as secções críticas

Uma secção crítica é um conjunto de instruções que leem e escrevem uma ou mais variáveis partilhadas por vários processos/threads

2. Associar um objeto de exclusão mútua a cada uma delas

A mesma variável partilhada tem que estar protegida sempre pelo mesmo objeto de exclusão mútua.

Usam-se objetos de exclusão mútua diferentes para proteger variáveis completamente independentes, que podem ser acedidas concorrentemente

3. Invocar fechar(mutex) no início e abrir(mutex) no fim de cada secção crítica; ou

4. Inicializar S=1 e invocar wait(S) no início e post(S) no fim de cada secção crítica

Monitores Java – exclusão mútua

Todos os objetos em java têm associado um único semáforo/lock

Normalmente, quando os métodos de um objeto são invocados, o lock é ignorado

Se o método é declarado synchronized, a invocação do método requer que a thread obtenha o lock, isto é, faça fechar no lock.

Se o lock está fechado, a thread é bloqueada na lista de threads bloqueadas no objeto

Se o lock está aberto, a thread fecha o lock e executa o método. Quando a execução do método termina a thread liberta o lock

Se a lista de threads bloqueadas no lock não está vazia quando o semáforo é libertado, uma das threads é desbloqueada e poderá executar o método depois de fechar o lock

2.4 – Gestão de Processos

Escalonamento

Num sistema de computação ...

de um modo geral, um processo/thread representa qualquer actividade em execução, tenha sido criada pelo utilizador ou o próprio SO.

estas actividades competem pelo CPU e deste modo surge a necessidade de fazer o seu escalonamento

o escalonamento de processos/threads refere-se à tarefa de fazer a gestão da partilha do CPU entre um conjunto de processos/threads prontos a executar

Em sistemas com um único processador …

apenas um processo/thread pode estar em execução num dado momento ...

.... mas aparentemente executam ao mesmo tempo, por isso dizem-se concorrentes

os restantes processos têm de esperar a sua vez mesmo que estejam prontos para serem executados

Necessário otimizar utilização do CPU

se um processo se encontra em execução no processador e chega a um ponto do programa onde necessita de executar uma instrução de I/O de dados, isso significa que o CPU não estaria a fazer nada de útil até essa instrução estar terminada

como operações I/O são tipicamente muito mais lentas do que a execução de instruções no CPU, qualquer paragem à espera de dados significa uma quebra substancial no desempenho do sistema

CPU deve estar sempre ocupado a executar algum processo e não pode ficar parado à espera que um processo termine I/O

Objetivos (eficiência)

Maximizar a utilização do CPU e outros recursos (100% do tempo ocupado!)

Maximizar o número de processos executados por unidade de tempo (throughput)

Minimizar o tempo de total de execução (turnaround)

Objetivos (conveniência)

Minimizar o tempo de resposta aos utilizadores interativos

Características dos processos

Um processo/thread pode ser:

- Tipo I/O-bound vs CPU-bound
- interativo ou batch
- Tempo real ….

Os processos/threads ….

- podem ter diferentes prioridades num sistema
- São também caracterizados pelo seu comportamento no sistema (tempo de CPU acumulado, etc)

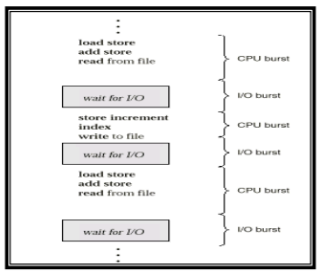
Gestão de processos

Escalonamento

A execução de um processo alterna entre:

CPU-burst- Períodos de avanço na execução do programa

I/O-burst – Períodos (curtos) de bloqueio à espera de sinais, operações de entrada-saída de dados (I/O), etc…



Duração e números de CPU bursts varia muito de programa para programa

CPU bound

- Processo caracterizado por CPU-bursts mais longos poucas interrupções causadas por I/O

I/O bound

- Processo caracterizado por muitos CPU-bursts de curta duração Interrupções causadas por i/O são muitas

Gestão de processos

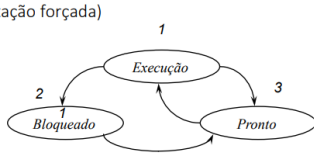
Escalonamento

Escalonamento de processos/threads diz respeito ao conjunto de decisões relacionadas:

- com a transição 3 -> 1
- e, eventualmente com a transição 1-> 3 (desafectação forçada)

Escalonamento de processos é responsável por decidir…

- qual o próximo processo/thread a executar
- quando começa esse processo/thread a executar
- durante quanto tempo esse processo executa



Gestão de processos

Escalonamento

Estratégias de escalonamento:

- ***Nonpreemptive scheduling*** ou "*run to completion*";
  - permite que um processo, uma vez escolhido, execute até que este liberte voluntariamente o CPU (transição 1 → 2) ou termine a execução.
  - vantagens: Fácil implementação.
  - problemas: não se adequa a sistemas multitarefa (por exemplo sistemas multiutilizador com *time-sharing)*

Gestão de processos

Escalonamento

Estratégias de escalonamento:

- ***Preemptive scheduling*** (desafectação forçada)
  - suspensão de processos, mesmo que tenham todas as condições para executar
  - implica mecanismos de interrupção dos processos.
  - problemas: as *race conditions* e consequentemente a necessidade de implementar mecanismos complexos como os semáforos e as mensagens...
  - Garantem normalmente tempos de resposta rápidos a processos/ *threads* prioritários ou garantem a partilha equilibrada do CPU pelos diferentes *processos/threads.*

Gestão de processos

Escalonamento

CPU scheduler (Dispatcher)

- escalonamento de curto prazo
- objectivo genérico é promover uma utilização produtiva do CPU e do sistema
- objectivos específicos determinam políticas de escalonamento diversas
- seleciona um processo de entre os processos em memória prontos a executar e atribui-lhe o processador.

Gestão de processos

Escalonamento

Níveis de escalonamento:

- *Long-term scheduler* – seleciona quais os processos a carregar em memória
  - invocado pouco frequentemente (ordem dos segundos ou minutos) => moderado na rapidez de execução.
  - controla o grau de multiprogramação
  - normalmente não existe em sistemas de time-sharing
- *Short-term scheduler* (*CPU scheduler*) – seleciona qual o processo a executar de seguida.
  - invocado frequentemente (na ordem dos milissegundos) ⇒ rapidez na execução.

Existem diversos algoritmos de escalonamento adequados a cenários diferentes e que otimizam diferentes variáveis de funcionamento do sistema.

First-Come, First-Served

Round-robin

Shortest job first

Prioridades

Multilevel queue

Multilevel feedback queue

"First-Come, First-Served" (FCFS)

O processador é atribuído pela ordem de pedidos dos processos.

É implementado por uma fila FIFO

O tempo médio de espera de cada processo é, no entanto, muitas vezes longo

Por definição é nonpreemptive

É particularmente inadequado a sistemas de partilha de tempo

Gestão de processos

Escalonamento

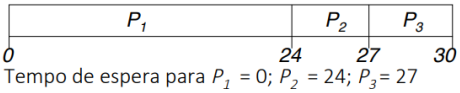
"First-Come, First-Served" (FCFS)

Exemplo:

Process	Burst Time
P_1	24
P_2	3
P_3	3

Submetidos pela seguinte ordem: P_1 , P_2 , P_3

O diagrama de Gantt para o escalonamento é:



Tempo de espera para P_1 = 0; P_2 = 24; P_3 = 27

Média: (0 + 24 + 27)/3 = 17; E se a ordem de chegada fosse: P_2 , P_3, P_1

"Shortest Job First" (SJF)

Associa a cada processo o tempo de execução do próximo CPU burst.

Utiliza estes tempos para escalonar o processo com o tempo mais pequeno.

Dois esquemas:

nonpreemptive – assim que o CPU é atribuído a um processo, o processo tem garantia que termina o seu CPU burst.

preemptive – se um processo novo é submetido com um tempo de CPU burst menor que o tempo que falta executar ao processo corrente, preempt. Este esquema é conhecido como o Shortest-Remaining-Time-First (SRTF).

Gestão de processos

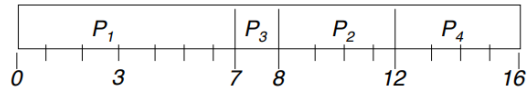
Escalonamento

"Shortest Job First" (SJF)

Exemplo:

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF *(nonpreemptive)*



Tempo de espera para P_1 = 0; P_2 = 6; P_3 = 3; P_4 = 7

Média = (0 + 6 + 3 + 7)/4 = 4

Dificuldades do "Shortest Job First" (SJF)

como determinar a duração do próximo CPU burst?

nos sistemas batch pode ser utilizado o limite de tempo de execução especificado quando um job é submetido.

nos sistemas de partilha de tempo podemos aproximar o SJF se utilizarmos uma estimativa do tempo de execução do próximo CPU burst calculada com base na duração dos CPU bursts anteriores.

Gestão de processos

Escalonamento

Scheduling com prioridades

- a cada processo é associada uma prioridade (número inteiro); o processador é atribuído ao processo com maior prioridade (normalmente inteiro menor = maior prioridade);
  - *preemptive* - se a prioridade de um processo é mais alta que a prioridade do processo corrente, *preempt.*
  - *Nonpreemptive* - ... o processo de maior prioridade é simplesmente colocado à cabeça da lista de processos prontos.
- problema = *Starvation* - processos de baixa prioridade podem não executar nunca
- solução ≡ *Aging* – à medida que o tempo progride, a prioridade dos processos que estão à espera de executar incrementa.

Gestão de processos

Escalonamento

Scheduling com prioridades

atribuição de prioridades:

- externa - definidas segundo critérios externos ao SO, normalmente associadas a factores económicos e políticos
- interna - o SO utiliza factores quantificáveis para determinar a prioridade de um processo, como por exemplo, os limites de tempo, os requisitos de memória, o número de ficheiros abertos, a relação entre tempo médio de I/O *burst* e tempo médio de CPU *burst.*
  - por exemplo dar maior prioridade a processos que utilizam I/O mais frequentemente (executam pouco e esperam muito);

Round Robin (RR)

é atribuído a cada processo um pequeno intervalo de tempo de CPU (4me quantum), normalmente 10-100ms. Após este intervalo, o processo é suspenso e inserido no final da fila de processos prontos.

se existem n processos na fila de espera e o quantum é q, então cada processo obtém 1/n do tempo do CPU, dividido em quantidades de tempo q de cada vez. Nenhum processo espera mais de (n-1)q unidades de tempo até nova execução.

desempenho :

q grande ⇒ FCFS

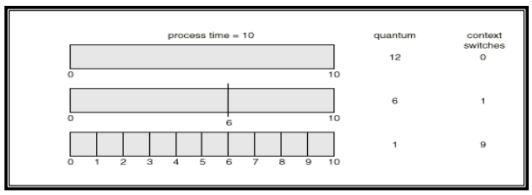
q pequeno ⇒ q tem que ser grande relativamente ao tempo de mudança de contexto (overhead)

Gestão de processos

Escalonamento

Round Robin (RR) (cont.)

um *quantum* pequeno incrementa o número de mudanças de contexto.
tempo de execução dos processos varia com o *quantum*:



[sil99] - cap. 6

Multilevel Queue Scheduling

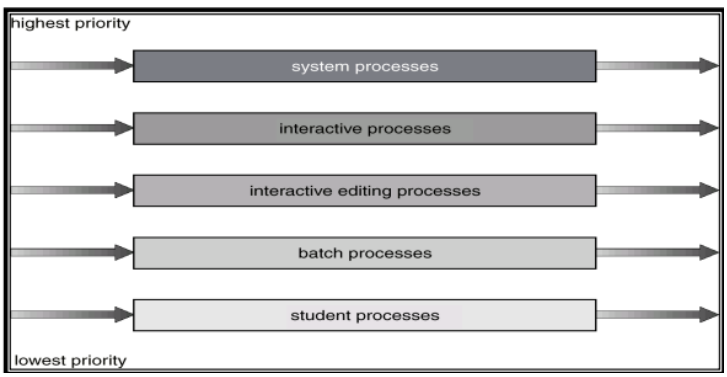
a fila de processos prontos é dividida em filas separadas. Exemplo: fila dos processos fg e fila do processo bg.

cada fila tem o seu algoritmo de scheduling próprio. Exemplo: fg - RR; bg FCFS.

é necessário fazer scheduling entre filas:

prioridade fixa; i.e., apenas executa processos em bg se a fila dos processos fg está vazia. Situação de starva4on é possível.

fatia de tempo – é atribuída a cada fila uma fatia de tempo de CPU a qual será dividia pelos processos da fila; i.e., por ex: 80% to fg com RR e 20% to bg com FCFS



Multilevel Feedback Queue

a fila de processos prontos é dividida em filas separadas.

processos podem passar de umas filas para as outras

se um processo usa muito CPU passa para uma fila com menor prioridade

processos interativos e com muito I/O mantêm-se na fila prioritária.

processos nas filas com menor prioridade vão sendo subidos através de um processo de aging