Slides 1

Sistema operativo é o programa base que estabelece a **interface entre os programas e o *hardware*** e que é simultaneamente **fácil de usar e eficiente**.

Sistema computacional

Componentes:

- *Hardware*;
- Sistema operativo;
- Programas de aplicação;
- Utilizadores.

Fornece

- **Serviços**
  - Serviços *standard* que são implementados pelo hardware.
- **Coordenação**
  - Coordena várias aplicações e utilizadores de modo a garantir segurança, eficiência e justiça na utilização dos recursos.
- **Controlo**
  - Controla a execução dos programas prevenindo erros e uso impróprio do computador.

Papéis

- **Árbitro**
  - Gere recursos partilhados, como *CPU*, memória, discos, impressoras, etc.
- **Ilusionista**
  - Fornece às aplicações/programador abstrações de recursos com capacidades superiores às existentes, como memória infinita, uso exclusivo de *CPU*, etc.
- **Adaptador**
  - Serviços comuns: sistema de ficheiros da *UI*.
  - Separa aplicações dos dispositivos *I/O*.

Organização do computador

- *CPU's* e controladores de dispositivos *I/O* executam em paralelo;
- Cada controlador de dispositivo trata um tipo particular;
- Controladores de dispositivo têm *buffer* locai;
- *CPU* move dados de/para memória e de/para *buffers* locais;
- Transferências de *I/O* são do dispositivo para o *buffer* local do respetivo controlador e depois para a memória;
- Controlador do dispositivo informa *CPU* que terminou a operação através do envio de uma interrupção.

Slides 2

Gestão de processos

Processo é um **programa em execução.**

- **Processo** -> Entidade ativa;
- **Programa** -> Entidade passiva;
- Necessita de recursos (**CPU, memória, I/O, ficheiros**), quando acaba liberta-os.

Processos com uma ***thread*** têm um ***Program Counter (PC)*** indicando a próxima instrução a executar. Processos ***multi-threaded*** têm um ***PC*** por ***thread***.

É responsabilidade do SO fazer o **planeamento de processos na CPU**, criar, suspender, retomar e excluir processos e fornecer **mecanismos de sincronização e comunicação de processos.**

Gestão de memória

Memória física é **escassa** e **cara.**

Vantagens da memória virtual

- Determinar o que deve estar em memória em cada altura, **maximizando a utilização da *CPU***;
- Utilização **transparente** da memória;
- **Segurança** na utilização da memória;
- **Partilha** da memória pelos vários processos.

Atividades

- Conhecer quais as **zonas de memória** livres e ocupadas;
- Decidir que processos e dados **mover para dentro ou para fora** da memória;
- **Alocar e desalocar espaço** de memória.

Gestão de armazenamento

O **sistema operativo** disponibiliza uma vista lógica uniforme do espaço de armazenamento. **Ficheiro é um conjunto de dados.**

Sistema de ficheiros

- Ficheiros organizados em **diretorias/pastas**;
- Ficheiros com **permissões** que garantem **segurança** do acesso aos dados.

Subsistema de I/O

O **sistema operativo esconde** pormenores dos dispositivos *hardware* do utilizador.

Responsabilidades

- Gestão de memória *I/O*;
- Interface geral dos *device drivers*;
- *Drivers* para dispositivos específicos de *hardware*.

Proteção e segurança

Proteção é um mecanismo do **sistema operativo** para **controlar o acesso aos recursos** de processos e utilizadores.

Segurança é a **defesa do sistema operativo** de ataques internos e externos.

De modo a garantir a **segurança do sistema**, a maioria dos **sistemas operativos** podem ser executados em dois modos:

- **Utilizador**
  - Tem **restrições** de segurança;
  - **Acesso interdito a certas zonas** de memória e dispositivos.

- ***Kernel***
  - **Sem restrições** de segurança;
  - Pode **executar todas as instruções** e acessos;
  - Instruções privilegiadas.

Sistema de ficheiros

Sistema *FAT32* (*File Allocation Table*) é uma **tabela** que indica onde estão os **dados de cada arquivo** na memória, que está **dividida em blocos**, **podendo cada ficheiro ocupar mais que um bloco**. No entanto, **um bloco não pode ser ocupado por mais que um ficheiro**.

Organização do disco
- *Boot sector*;
- 2 cópias da *FAT*;
- Zona de dados.

Como encontrar conteúdo de um ficheiro
- **Cluster inicial** indicado na **entrada da diretoria;**
- **Clusters seguintes** numa **lista ligada** através da *FAT***.**

FAT
- ***Array* de números** de *clusters* (cada um com 32 *bits*);
- Para cada *cluster* indica se o **seguinte está livre ou ocupado**.

Diretorias
- Constituídas por **entradas de diretorias** de tamanho fixo;
- Entradas **definem nome e metadados** de ficheiros e diretorias;
- Entradas **definem *cluster* inicial**.

Por exemplo, se cada *cluster* suporta 1024 *bits*, quando tiver cheio avança para o próximo.

Desvantagem -> Não é muito eficiente para ficheiros muito grandes, porque para chegarmos ao último *cluster*, temos de percorrer todos os anteriores.

Slides 3

Modos de operação

Chamadas ao sistema providenciam uma **forma segura** de alternar entre **modo de utilizador** e o **modo *kernel***.

Programas de sistema

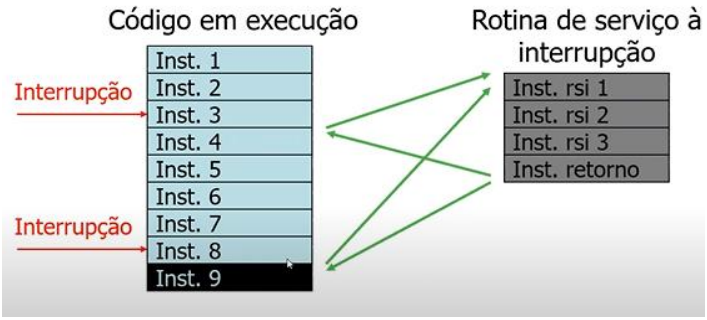
Oferecem um **ambiente confortável** para o **desenvolvimento e execução** de programas.

Podem ser divididos em:

- Manipulação de ficheiros;
- Informação de estado;
- Modificação de ficheiros;
- Suporte às linguagens de programação;
- Carregamento e execução de programas;
- Comunicações.

Interrupções/Exceções

Quando o ***CPU* é interrompido**, o processo atual para de imediato (**salvaguardando o endereço do código interrompido para posteriormente voltar a ele**) e a sua exceção é transferida para a **rotina de atendimento de interrupções**. **Quando a execução estiver concluída volta ao processo interrompido**.



As **interrupções** são originadas pelo **hardware** e as **exceções** são originadas pelo **software**.

Tipos de interrupções

- **Interrupções de *I/O***
  - O **dispositivo de *I/O* pede atenção** e a **rotina de atendimento** deve aceder ao dispositivo para determinar a ação necessária.

- **Interrupções de *timers***
  - Dizem ao processador que um **certo intervalo de tempo** já decorreu;
  - *Timer* **local ou externo**.

- **Interrupções entre processadores**
  - Um processador emite uma interrupção para outro processador num **sistema multiprocessador**.

Slides 4

Organização de *I/O*

Métodos assíncronos

Quando as **chamadas de sistema** suspendem a execução de um programa até que um dispositivo de *I/O* responda. O programa é movido para uma **fila de espera** até que a **chamada ao sistema termine**.

Métodos síncronos

Mesmo que um **dispositivo de *I/O* deixe de responder**, a **execução do programa não é bloqueada**, enquanto fazem pedidos ao dispositivo *I/O*.

Multiprogramação

Execução de **vários programas em simultâneo**, ou seja, suporta **vários processos em execução ao mesmo tempo**. No entanto, alguns programas **pausam a sua execução** à espera de uma resposta de um dispositivo *I/O*, e a solução da multiprogramação é **executar outro programa**, utilizando os recursos da *CPU.*

A **utilização da *CPU*** é alternada rapidamente entre vários processos através de um **escalonamento de processos**.

Timesharing

O ***CPU*** altera o processo em execução mesmo que este **não necessite de esperar**, pois a cada processo é atribuído um **tempo máximo de ocupação consecutiva do processador**.

Este fenómeno, diminui muito o tempo de resposta de aplicações interativas e permite que vários utilizadores usem o mesmo sistema computacional como se **dispusessem do sistema exclusivo**.

Estrutura do sistema operativo

- **Monolítico**
  - Contém **todas as funcionalidades** de forma estática;
  - Permite código otimizado, pouco flexível e ocupa mais memória.

- **Modular**
  - Permite **adição/configuração de funcionalidades** através de integração de módulos;
  - Custo/*Overhead* da *API*, mais flexível e menos memória.

- ***Microkernel***
  - *Kernel* apenas com **serviços básicos***: thread, adress space* e *ipc*;
  - Várias funcionalidades associadas ao **sistema operativo** correm em modo utilizador;
  - Pouca memória, verificável, mudanças de *kernel mode* para *user mode* são frequentes.

Slides 5

Máquinas virtuais

- Levam a organização em camadas ao limite;
- Os recursos do computador parecem ser exclusivos apesar de serem partilhados;
- O sistema operativo cria a ilusão de que cada processo ocorre no seu processador e memória;
- Cada sistema operativo tem o seu conjunto de processos em execução.

Processo

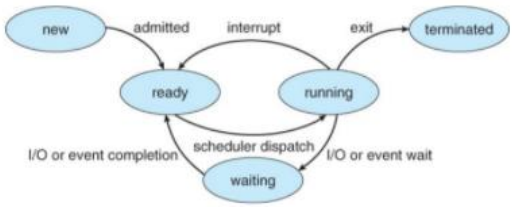
Métodos de ocorrência

- ***Foreground*:** Interage com o utilizador;
- ***Background*:** Executa sem interação.

Um **processo** consiste numa **cópia da imagem executável do programa** na memória física, que tem associada uma *stack*, *heap*, etc.

Estados de um processo

- ***New*** -> Está a ser criado;
- ***Running*** -> Instruções em execução;
- ***Waiting*** -> À espera de um evento;
- ***Ready*** -> À espera de ir para o processador;
- ***Terminated*** -> Terminou a execução.



Process Control Block (PCB)

É a estrutura que, no **sistema operativo**, armazena a informação sobre um **processo.**

Inclui os seguintes dados:

- Estado do processo;
- *Program Counter*;
- Registos do *CPU*;
- Tipo de escalonamento;
- Informação sobre a memória do processo;
- Informação sobre *I/O*;
- *Accounting*.

Os **processos** estão armazenados em listas ligadas, ficando no estado de espera até serem executados.

Árvore de processos

Um **processo** pode dar origem a outro processo, sendo o **principal o processo pai** e o **originado o processo filho**, criando assim uma **árvore de processos**. Pode assim, ser criada uma hierarquia de processos.

O processo filho pode saber o ***process id (pid)*** do processo pai, quando o **processo filho morre**, é enviado o **sinal *SIGCHLD*** ao pai, o processo pai recolhe o ***exit code* dos filhos** e quando morre, o filho é **herdado** pelo **processo 1 (*init*).**

Tipos de processos

- ***I/O* intensivos**
  - Fazem muitas chamadas ao sistema relacionadas com *I/O*;
  - Pequenos períodos de utilização do *CPU*.

- ***CPU* intensivos**
  - Fazem poucas chamadas *I/O***;**
  - Longos períodos de utilização do *CPU***.**

Num **sistema com *timesharing*** e de modo a otimizar a utilização do CPU é positivo que a lista de processos em execução seja **equilibrada entre os dois tipos**.

Criação de processos

Valor de retorno do *fork()* (ao usar *fork()* estamos basicamente a criar uma cópia do pai)

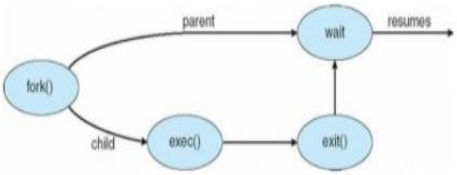
- **Processo pai ->** *pid* do filho;
- **Processo filho ->** 0.

Partilha de recursos

- Pai e filhos partilham recursos;
- Filhos partilham um subconjunto dos recursos do pai;
- Pai e filhos não partilham recursos.

Execução

- Pai e filhos executam em paralelo;
- Pai espera que filho(s) terminem.



Funções

- **Exec() ->** Substitui a execução de um programa pela execução de outro programa e não retorna nada a não ser que dê erro;
- **Wait() ->** O processo pai espera pelo processo filho terminar, ou seja, **até receber o *exit code* do processo filho**.

Slides 7

Threads

É um **caminho de execução de um processo** e este pode ser **dividido em várias *threads*** portanto, *threads* são partes de um processo que decorrem em paralelo, **partilhando uma memória única**.

Cada *thread* tem o seu **Program Counter**, ***set* de registos**, **estado** e ***stack***.

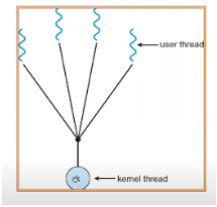
Vantagens

- **Modular ->** Cada *thread* trata de uma **atividade distinta**;
- **Responsivo ->** Respondem imediatamente quando surge algum evento da qual elas estão à espera;
- **Melhor desempenho**;
- Partilha de recursos e arquitetura multiprocessador.

As ***threads*** podem ser implementadas ao **nível do utilizador** e ao **nível do kernel**.

Modelos *multithreading*

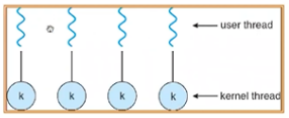
Many-to-one



Várias *threads* do utilizador são mapeadas uma *thread kernel* ou núcleo.

- Se uma delas bloqueia todas bloqueiam;
- Não tira partido de vários processadores.

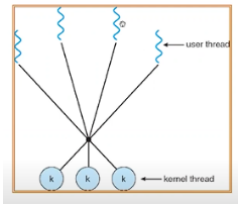
One-to-one



Cada ***thread* do utilizador** mapeada numa *thread* núcleo.

- Existe **número máximo de *threads*** para evitar perdas de *performance*;
- Caso uma delas bloquear, as outras continuam ativas.

Many-to-many



Várias threads do utilizador mapeadas em várias *threads* núcleo.

- Número de ***threads* do utilizador** maior que o número de ***threads* núcleo**;
- Se uma thread bloquear, o processo continua ativo.

Bibliotecas -> *Pthreads*, *Java Threads*, etc.

Thread Pools

Criação de um **número pré-definido de *threads*** quando o processo é iniciado, atribuindo trabalho à medida que for necessário.

Vantagens

- Mais rápido do que criar *threads* à medida da necessidade;
- Limita o número de *threads* no sistema.

Slides 8

Fork() e exec()

- **fork() ->** Duplica apenas a *thread* que invocou a função;
- **forkall() ->** Duplica todas as *threads*;
- **exec() ->** Substitui o processo incluindo todas as *threads*.

Cancelamento de *threads*

Cancelar uma thread antes desta terminar por si;

- **Assíncrono ->** *Thread* terminada imediatamente;
- **Síncrono ->** *Thread* verifica periodicamente se deve terminar.

Atendimento de sinais

Usados para **notificar** processos de certos eventos.

Tipos de sinais

- **Síncronos ->** Gerados pelo próprio processo;
- **Assíncronos ->** Gerados por um evento externo ao processo.

Opções

- Enviado **apenas para uma** *thread* a que o sinal se aplica;
- Enviado **para todas** as *threads*;
- Enviado para **subconjunto** de *threads*;
- ***Thread* específica** recebe todos os sinais.

Sincronização de processos

Condição de corrida -> Quando **vários processos/*threads*** acedem a dados partilhados ao mesmo tempo e o **resultado depende da ordem de execução**.

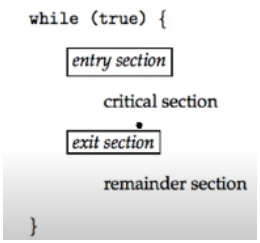
Região crítica -> Zona de código que **manipula dados partilhados** e que não pode ser executada concorrentemente por mais do que um processo/*thread*.

Região de entrada -> Código que realiza o pedido de acesso à região crítica.

Região de saída -> Código executado após a saída da região crítica.

Condições para a região crítica

- Se há um processo na região crítica, nenhum outro processo pode entrar (**exclusão mútua**);
- Se não há ninguém na região crítica, e alguém quer entrar, deverá entrar o mais rapidamente possível (**progresso**);
- Se um processo quer e está na "fila" para entrar, não pode ser ultrapassado por outros, ou seja, não pode esperar infinitamente (**espera limitada**).



Slides 9

Semáforos

Tipo de dados abstrato que **permite a sincronização** de *threads*/processos sem *busy waiting*.

- **Estado interno** que é um valor inteiro;
- **Operações atómicas** de incremento e decremento da variável interna;
- Bloqueia se a **operação torna o valor do semáforo negativo**.

Deadlock

Processos que estão **bloqueados** à espera de um **evento** que é resultado da **execução de um dos processos bloqueados**.

Adiamento indefinido (*starvation*)

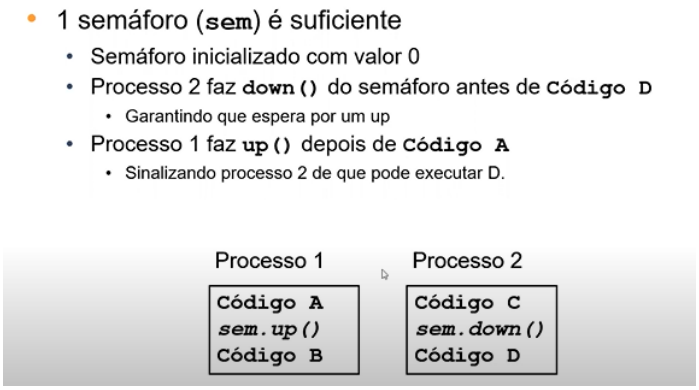
Um processo pode nunca ser removido da fila de espera de um semáforo, ou seja, **espera infinitamente**. Não cumpre o requisito da espera limitada.

Signaling

Mecanismo de sincronização onde um processo avisa outro que algo aconteceu.

- **Sincronizar código** em diferentes *threads*/processos, impondo uma ordem na sua execução.

Problema: o código D só poderá executar caso o código A já tenha terminada a sua execução



Slides 10

Rendevouz

Mecanismo de sincronização através do qual dois processos se encontram antes de continuar. Permite **sincronizar determinadas operações em processos distintos**.

Problema: código B e código D só serão executados caso código A e código C estiverem concluídos

- Usando 2 semáforos (**`arrived1`** e **`arrived2`**)
  - Semáforos inicializados com valor 0
  - Processo 1 faz **`arrived1.up()`** e **`arrived2.down()`** após **`Código A`**
    - Garantindo que espera por um **`arrived2.up()`**
  - Processo 2 faz **`arrived1.up()`** e **`arrived2.down()`** após **`Código C`**
    - Garantindo que espera por um **`arrived1.up()`**

Processo 1	Processo 2

`Código A`
`arrived1.up()`
`arrived2.down()`
`Código B`
	`Código C`
	`arrived2.up()`
	`arrived1.down()`
	`Código D`

Slides 11

Monitores

Tipo de dados abstrato que armazena dados privados manipuláveis através de métodos públicos, que são acedidos em **exclusão múltipla**. **Apenas um processo pode estar ativo dentro um monitor de cada vez.**

Variáveis de condição

Permitem bloquear um processo até que determinada condição se verifique.

Operações

- **wait() ->** Bloqueia o processo/*thread* e liberta o monitor, permitindo que **outro processo execute primitivas do monitor**;
- **signal() ->** Acorda um e apenas um processo dos processos bloqueados numa variável de condição

Diferença entre monitores e semáforos

Embora um semáforo possa ser utilizado para implementar um monitor, este é de baixo nível.

Vantagens dos monitores sobre os semáforos

- **Suporto ao nível do sistema de operação ->** Implementação é feita pelo kernel;
- **Universalidade ->** Construções de baixo nível.

Desvantagens dos monitores sobre os semáforos

- **Conhecimento especializado ->** Exige ao programador um domínio completo dos monitores.

Slides 12

Escalonamento do *CPU*

Permite a multiprogramação decidindo que processos são executados em cada instante de tempo, ou seja, de todos os processos que estão no estado *ready* qual será executado no(s) CPU('s).

Execução do escalonador

- ***Preemptive***
  - Processo muda do estado ***running* -> *ready***;
  - Processo muda do estado ***waiting* -> *ready***;

- ***Non preemptive***
  - Processo muda do estado ***running* -> *terminated***;
  - Processo muda do estado ***running* -> *waiting***;

Escalonador *preemptive*

É capaz de **retirar um processo que estava em execução**, portanto é ativado quando o processo transita do estado ***running* -> *ready*** e do estado ***waiting* -> *ready***.

Escalonador non preemptive

Só muda um processo que está em execução caso este se dispôs a isso, portanto só será ativado ou **quando o programa termina** (processo transita do estado *running -> terminated*) ou quando transita do estado ***running* -> *waiting***.

Dispatcher encarrega-se de colocar o processo selecionado pelo escalonador em **execução** no *CPU*.

↳ ***Dispatcher latency* ->** Tempo que o *Dispatcher* demora entre parar um processo e reiniciar o processo selecionado pelo escalonador.

Avaliação do escalonamento

- **Utilização do CPU ->** Manter CPU ocupado;
- **Débito ->** Número de processos que terminam por unidade de tempo;
- **Tempo do processo (*turnaround time*) ->** Tempo entre submissão do processo até este terminar;
- **Tempo de espera ->** Tempo que o processo está à espera no estado *ready*;
- **Tempo de resposta ->** Tempo entre o pedido e primeira resposta (eventualmente parcial) a esse pedido.

Algoritmos de escalonamento

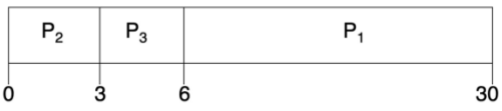
FCFS (First-Come, First-Served)

Process	*Burst Time*
P_1	24
P_2	3
P_3	3

Se os processos chegarem pela ordem 1, 2, 3, então:

Tempo de espera: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
Tempo médio de espera: (0 + 24 + 27)/3 = 17

Mas se os processos chegarem pela ordem 2, 3, 1, então:



Tempo de espera: $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
Tempo médio de espera: (6 + 0 + 3)/3 = 3 !!!

SFJ (Shortest Job First)

Ordena os processos considerando a duração do próximo CPU burst. Executa primeiro os processos com CPU burst mais curtos.

Opções

- **Non preemptive ->** Uma vez atribuído o *CPU*, o processo fica em *running* até terminar o *CPU burst*;
- **Preemptive ->** Se um processo entra na fila de *ready* com um *CPU burst* menor do que o tempo restante do *CPU burst* do processo em execução, atribuir o *CPU* ao processo que entrou em *ready*. **Também conhecido como *Shortest-Reamining-Time-First (SRTF)*.**

É ótimo do ponto de vista de tempo médio de espera de um conjunto de processos.

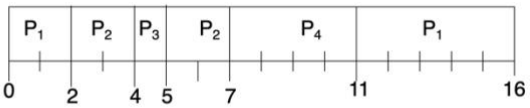
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

SJF (non-preemptive)

Tempo médio de espera= (0 + 6 + 3 + 7)/4 = 4

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 (preemptive)



Tempo médio de espera= (9 + 1 + 0 + 2)/4 = 3

Escalonamento por prioridades

Neste algoritmo é atribuída uma **prioridade a cada processo**, sendo primeiro executado os mais prioritários.

Pode ser *preemptive* ou *non preemptive* e o **algoritmo *SJF* é um caso particular deste algoritmo.**

Para **evitar *starvation*** dos processos menos prioritários, a prioridade pode ser aumentada proporcionalmente ao tempo de espera.

- **Prioridade estática ->** A prioridade de um processo a partir do momento que é definida não altera (FCFS);
- **Prioridade dinâmica ->** A prioridade de um processo pode alterar à medida que chegam novos processos com prioridades maiores (SRTF).

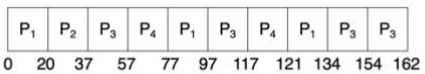
RR (Round Robin)

Cada processo **pode usar o CPU no máximo**, por determinado tempo (*time quantum*). Se o processo não bloquear antes do tempo definido é retirado de execução e passa para o fim da fila de *ready*
Se existem *n* processos na fila de *ready*, e o *time quantum* é *q*, então cada processo usa cerca 1/*n* do processador e **um processo nunca espera mais que (*n*-1)*q* unidade de tempo.**

q = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

O escalonamento será:



Tipicamente RR tem maior tempo médio de espera do que SJF, mas melhor tempo de resposta

Slides 13

FIFO multi-nível

Fila de *ready* é dividida

- Foreground (interativa);
- Background (batch).

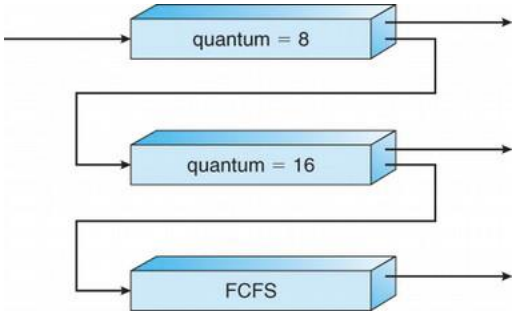
Cada fila pode ter a sua política de escalonamento

- Foreground -> RR;
- Background -> FCFS.

O escalonamento entre as 2 filas pode ser **baseado em prioridades**, por exemplo, os processos em background só irão executar se a fila *ready* de *foreground* estiver vazia ou para evitar *starvation*, ou **baseado em divisão do tempo**, cada fila tem um certo tempo de CPU disponível.

FIFO multi-nível com realimentação

Um exemplo é a criação de três filas, uma com um *time quantum* de 8ms, outra com 16ms e a última com um escalonamento *FCFS*. Estas foram descritas por ordem decrescente de prioridade. A segunda só pode ser executada quando a primeira está vazia e a terceira quando as duas anteriores o estiverem também.
Um novo processo começa na fila dos 8ms, se esgotar este intervalo de tempo antes de terminar a sua execução, passa para a fila doa 16ms e se o mesmo ocorrer para a última, a do *FCFS*.



Earliest Deadlien First

Escolhe para execução sempre o processo que tem a deadline mais próxima.

Memória virtual

Objetivos

- Eficiência da utilização da memória
  - Partilhada pelos processos;
  - Manter em memória apenas o necessário;
  - Endereços usados pelos processos não são endereços de memória física.
- Segurança
  - Mecanismos de segurança que impeçam que um processo altere as zonas de memória dos outros processos.
- Transparência
  - Processo tem acesso a muita memória (eventualmente mais que a física);
  - Processo corre como toda a memória lhe pertencesse.
- Partilha de memória
  - Vários processos acedem à mesma zona de memória.

Cada processo corre num espaço de endereçamento virtual (igual para todos).

Os endereços virtuais usados pelos processos e os endereços físicos que lhes correspondem podem ser distintos. Os endereços de memória virtuais têm de ser convertidos em endereços físico.