Sistemas de Computadores

Escalonamento

Luis Lino Ferreira

Maio de 2020

V1.6

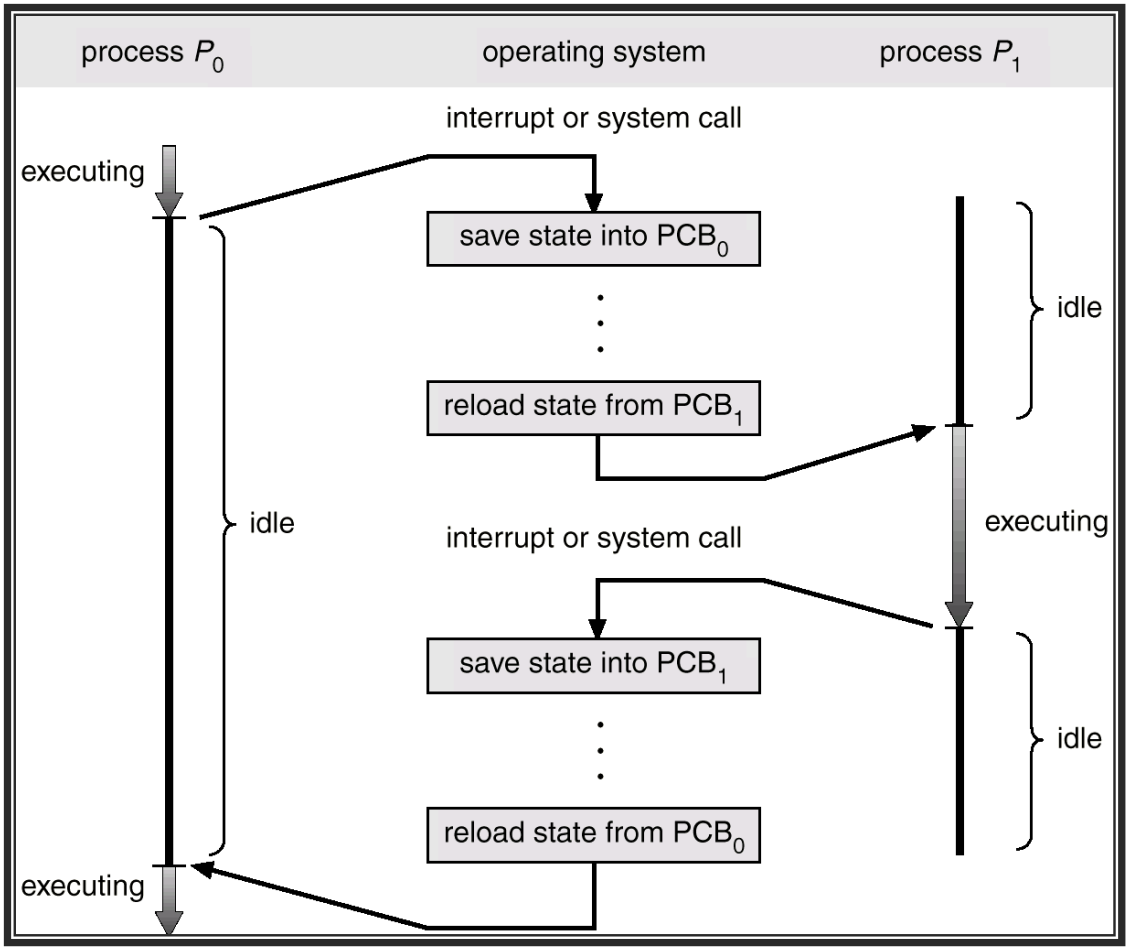
Sumário

- Conceitos básicos
- Gestão das filas de escalonamento
- Critérios de escalonamento
- Algoritmos de escalonamento
- Escalonamento multi-processador
- Escalonamento com recursos partilhados
- Escalonamento em tempo real

Comutação entre Processos

- A partilha do processador requer um mecanismo de comutação de processos, a que se dá o nome de comutação de contexto
- Mecanismo:
  - Salvaguarda do estado do processo que perde a CPU;
  - Restauração do estado do processo que ganha a CPU

Comutação entre Processos



Escalonamento

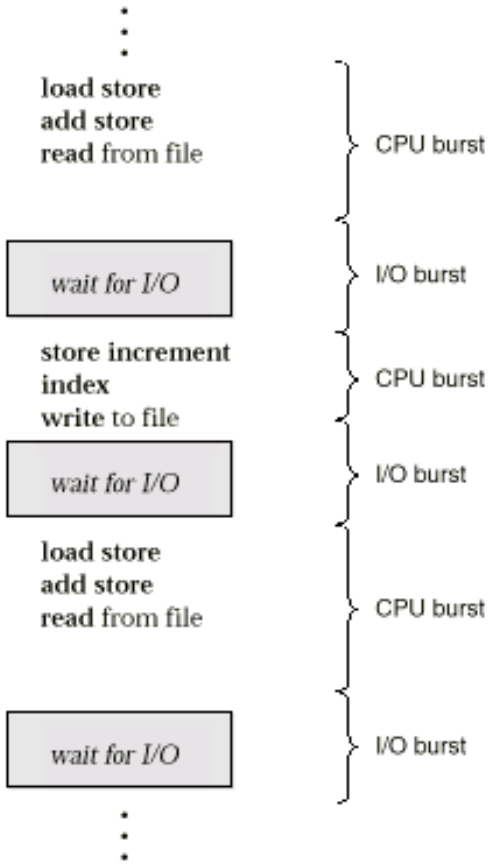
- Um dos objectivos da multi-programação é a maximização da utilização da CPU
- O escalonador tem como objectivo decidir qual o próximo processo a ser executado em função dos seus parâmetros
  - Note-se que em sistemas mono-processador apenas pode ser executado um processo de cada vez

Conceitos Básicos

- Objectivos do escalonamento
  - Optimizar a performance do sistema de acordo com um critério, por exemplo:
    - Dividir a capacidade de processamento da CPU entre vários processos
    - Diminuir o tempo de resposta (sistemas de Tempo-Real)
- Como?
  - Alguns exemplos:
    - Quando um processo está no estado de *Waiting* (por ex: de uma operação de I/O ou de um sinal), outro processo pode entrar em funcionamento
    - Periodicamente é retirado um processo e substituído pelo próximo na fila de *ready*

Caracterização de um programa

- Ciclo de um programa
  - Execução na CPU (*CPU Burst*) seguido de espera até à finalização de um operação de I/O (*I/O Burst*)



Notação Gráfica Utilizada

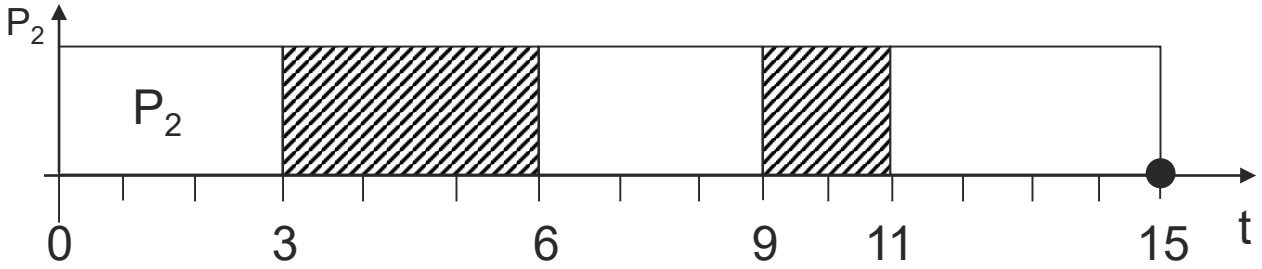
Chegada de um processo ao sistema

Processo em execução (estado de *running*)

Processo à espera (estado de *waiting*)

Finalização de um processo

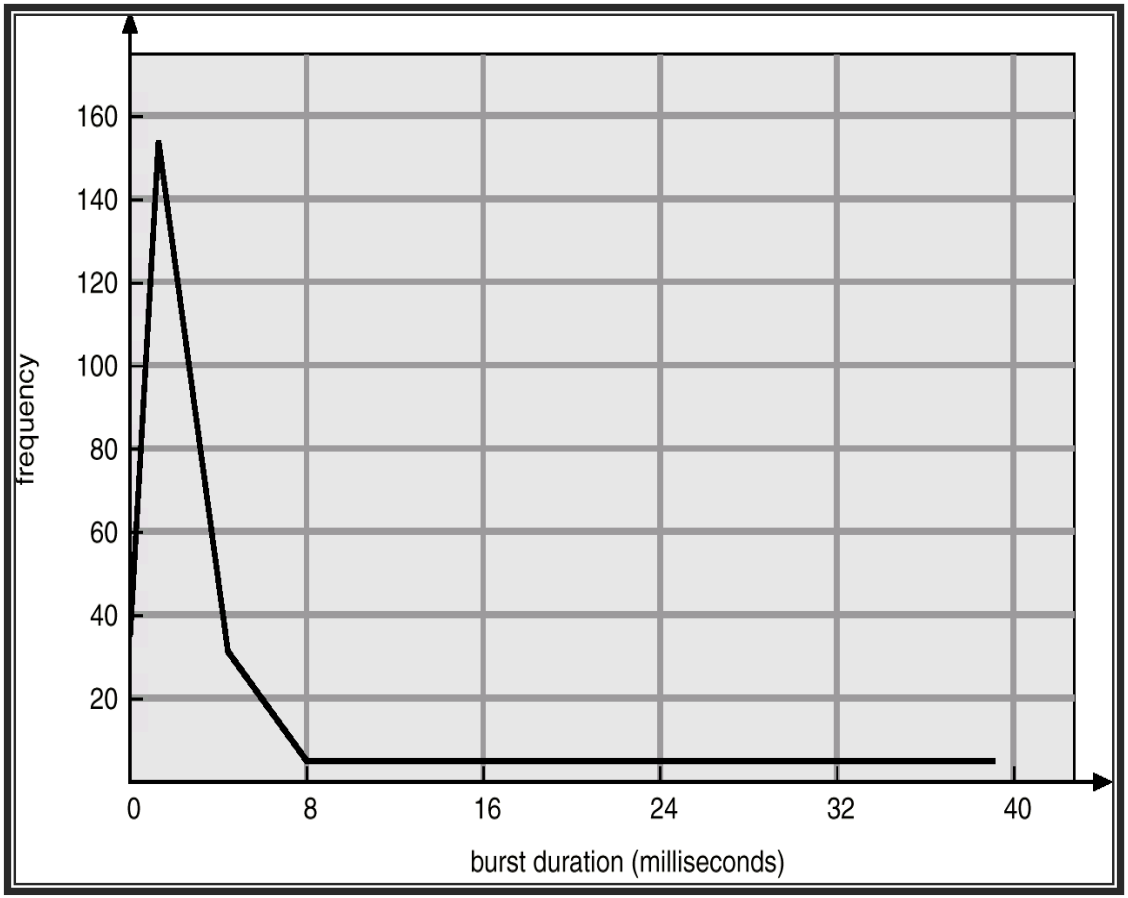
Exemplo:



Histograma dos CPU *Bursts*

A duração dos *CPU Burst* têm tendência a ter uma curva de frequência exponencial, com muitos *CPU Burst* de curta duração e poucos *CPU Burst* de longa duração

- Um programa *I/O-bound* tem muitos *CPU Burst* de curta duração
- Um programa *CPU-bound* tem poucos *CPU Burst,* mas estes são de longa duração

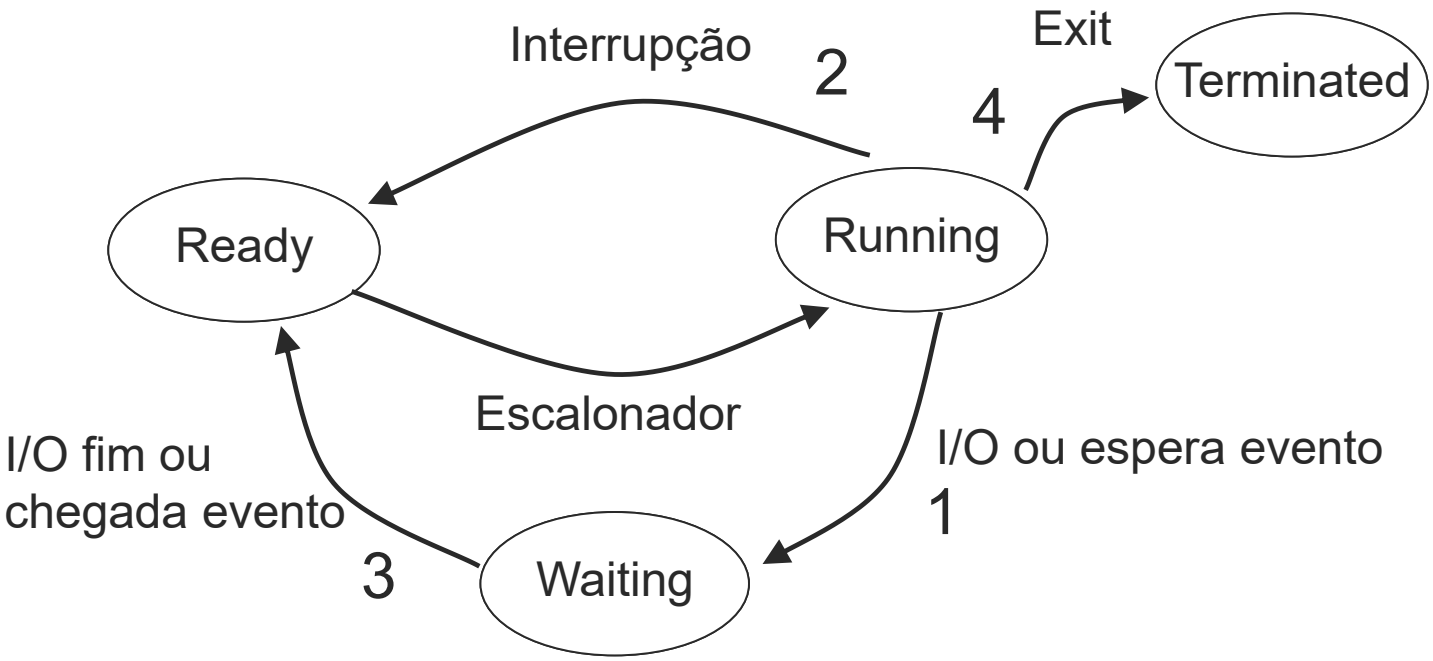


Escalonador da CPU

- O escalonador selecciona entre os processos na fila de *Ready,* aqueles que devem entrar em execução de seguida
- A decisão de escalonamento pode ser feita nas seguintes circunstâncias:

1. O processo comuta do estado *Running* para o estado *Waiting* (pedido de I/O, evocação da função *wait())*
2. O processo comuta do estado *Running* para o estado *Ready* (ocorrência de uma interrupção)
3. O processo comuta do estado *Waiting* para o estado *Ready* (termina um pedido de I/O)
4. O processo termina

Diagrama de Estados

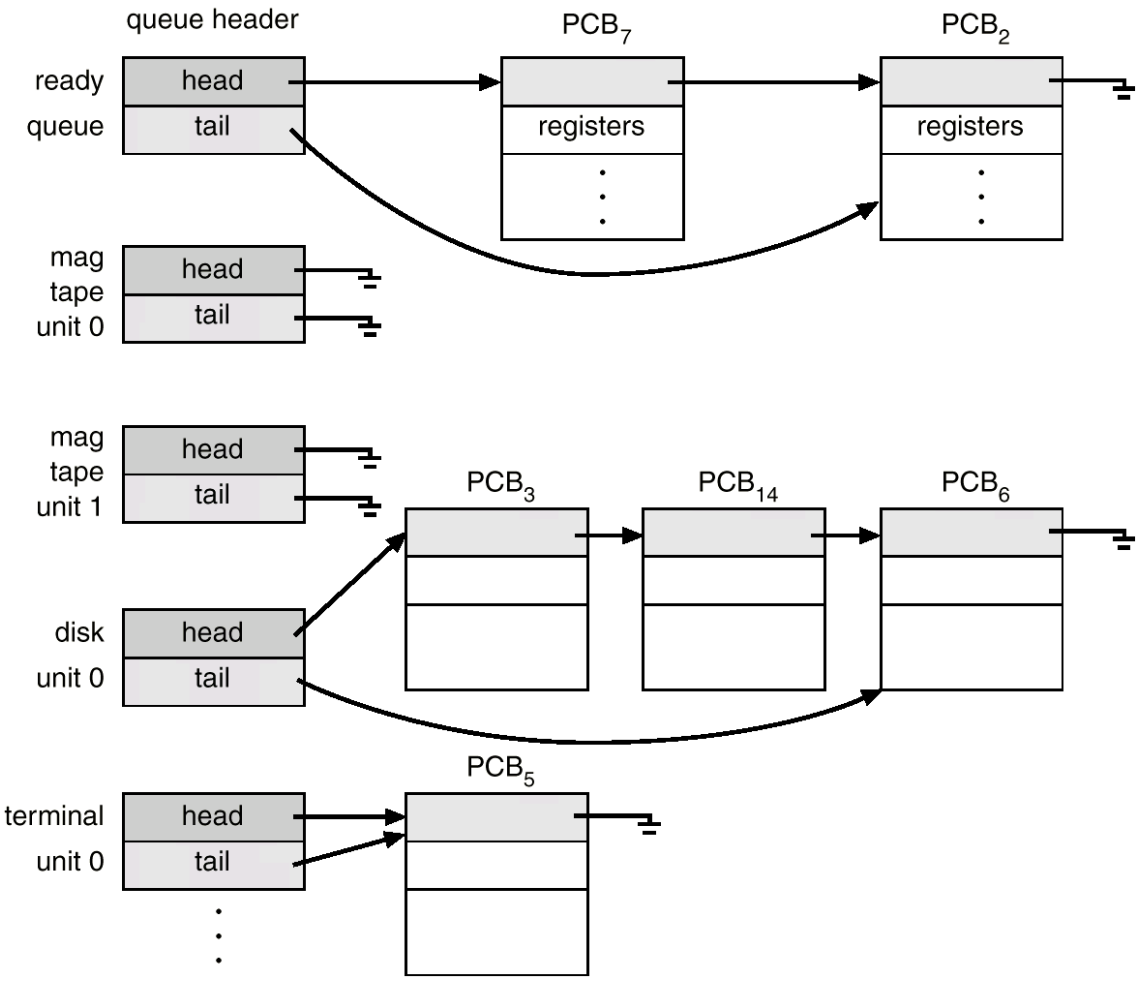


Filas de Escalonamento

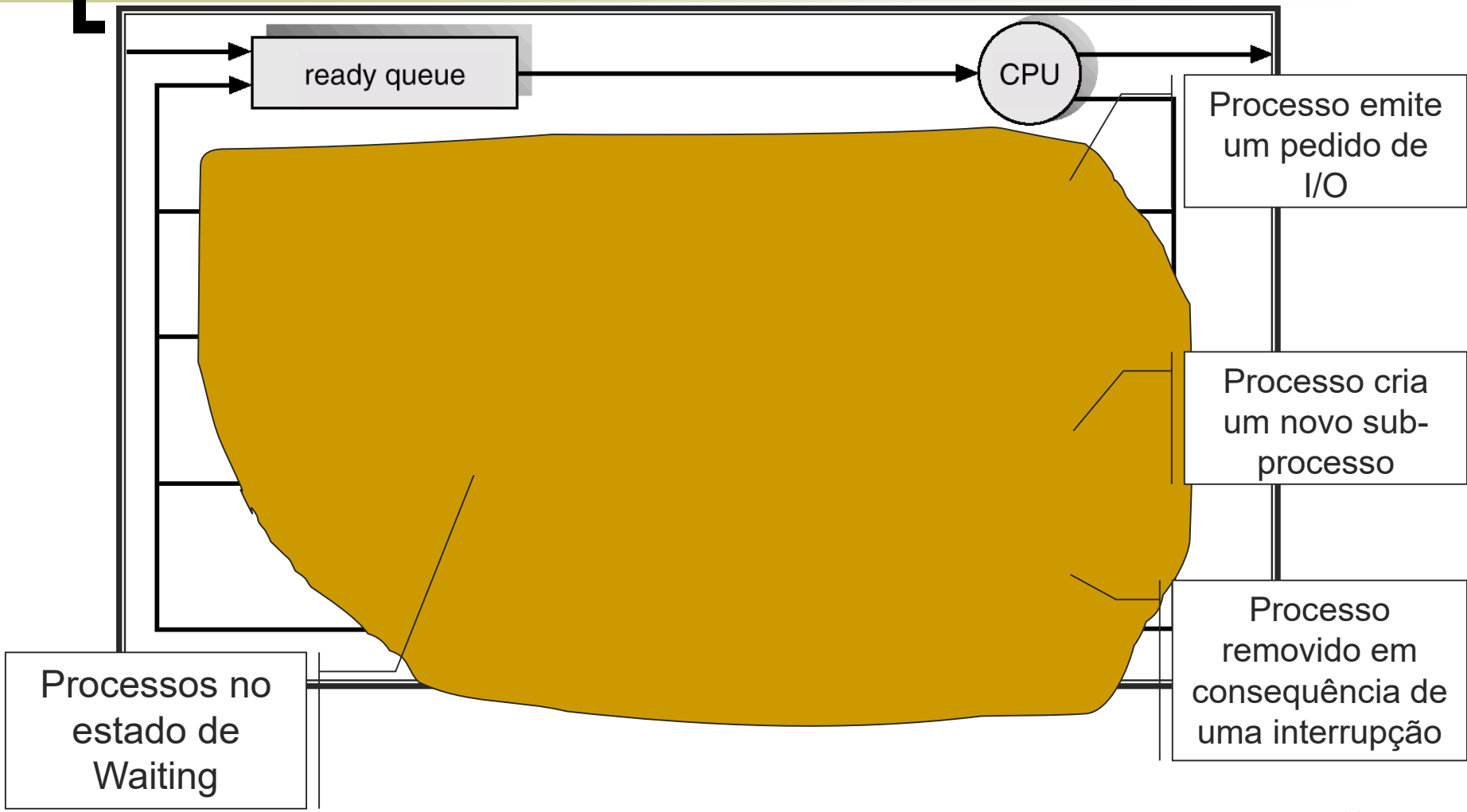
- As filas de escalonamento permitem ao SO saber o estado dos processos
  - A informação sobre os processos é guardada na estrutura *Process Control Block* (PCB)
- Existem filas para cada um dos estados, assim como filas para coordenar o acesso aos dispositivos de I/O

Filas de Escalonamento

- *Ready queue* – esta fila contém os PCBs dos processos residentes em memória que estão no estado *ready*, isto é processos que estão prontos e à espera de serem executados
- *Device queue* – lista dos PCBs dos processos à espera dum dispositivo I/O
  - Exemplo: *Disk unit 0 queue*



Filas de Escalonamento



Filas de Escalonamento

- Portanto, durante a sua execução de um processo várias coisas podem acontecer:
  - o processo pode emitir um pedido I/O, e consequentemente ser colocado numa fila de um I/O *device*
  - O tempo que o escalonador tinha atribuído ao processo (*time slice*) termina e consequentemente ser colocado na fila de *ready*
  - o processo pode criar um novo processo, ficando à espera que ele termine e consequentemente ser colocado na fila de *waiting*
  - o processo pode ser removido da CPU em consequência duma interrupção, transitando para a *Ready queue*

Escalonadores

- Um processo migra entre várias filas de escalonamento durante o seu tempo de vida
- O SO deve seleccionar processos destas filas com base em algum método ou algoritmo
- Há três tipos de escalonadores:
  - curto prazo
  - médio prazo
  - longo prazo
- Os processos podem ser caracterizados como ou:
  - *I/O-bound process* – despende mais tempo a fazer operações I/O do que cálculos na CPU; consequentemente, há bastantes CPU *bursts* de curta duração
  - *CPU-bound process* – despende mais tempo a fazer cálculos na CPU; consequentemente, há poucos CPU *bursts* de longa duração

Escalonadores

- Escalonador de **curto-prazo** (ou escalonador da CPU):
  - Selecciona que processos devem ser executados de seguida e reserva, consequentemente, a CPU
  - Escalonador de curto-prazo é invocado com bastante frequência
    (milli-segundos) $\Rightarrow$ (ser rápido)
  - Exemplos:
    - Linux
    - Windows

Tipos de Escalonamento

- Escalonamento **não preemptivo**:
  - o escalonador apenas pode efectuar a comutação entre processos quando o processo termina ou passa para o estado de *Waiting*
  - Casos 1 e 4 do slide 11
  - Exemplo: Windows 3.1 e Apple Macintosh OS

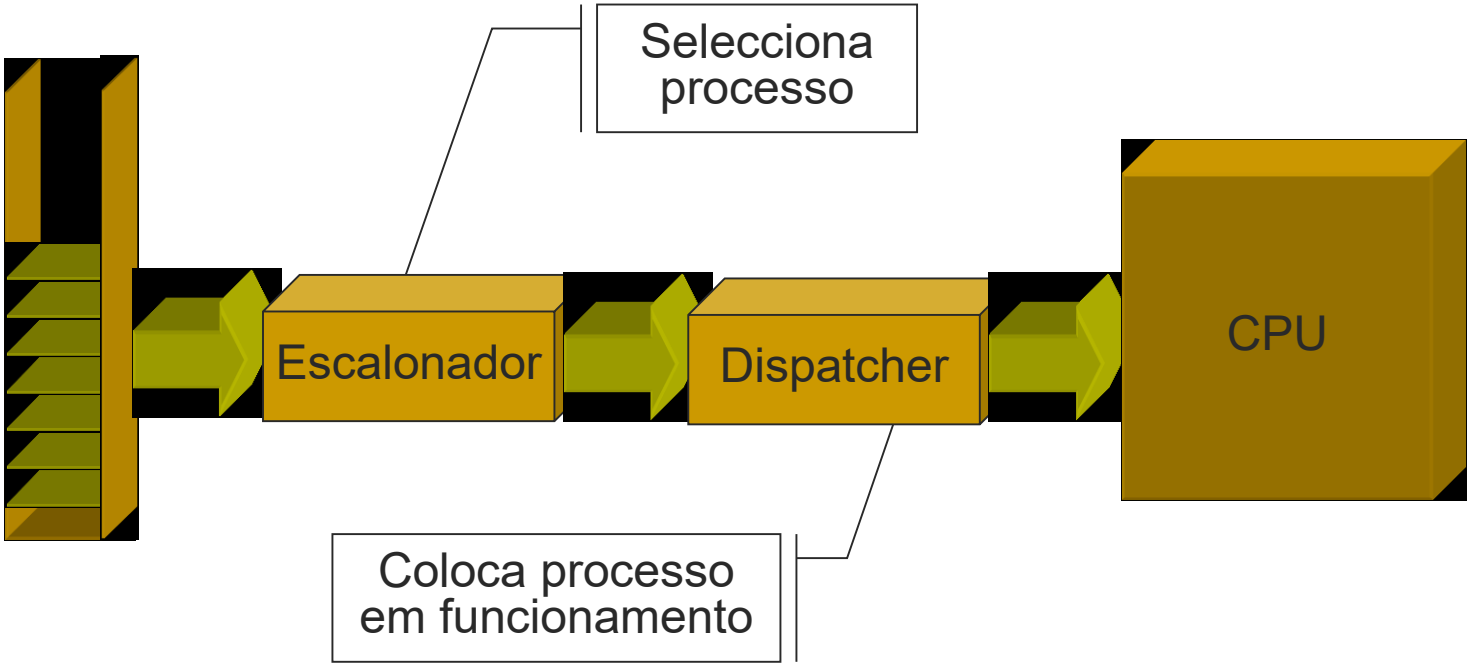
Tipos de Escalonamento

- Escalonamento **preemptivo**:
  - o escalonador pode interromper a execução de um processo antes que este tenha terminado
  - Casos 2 e 3 do slide 11

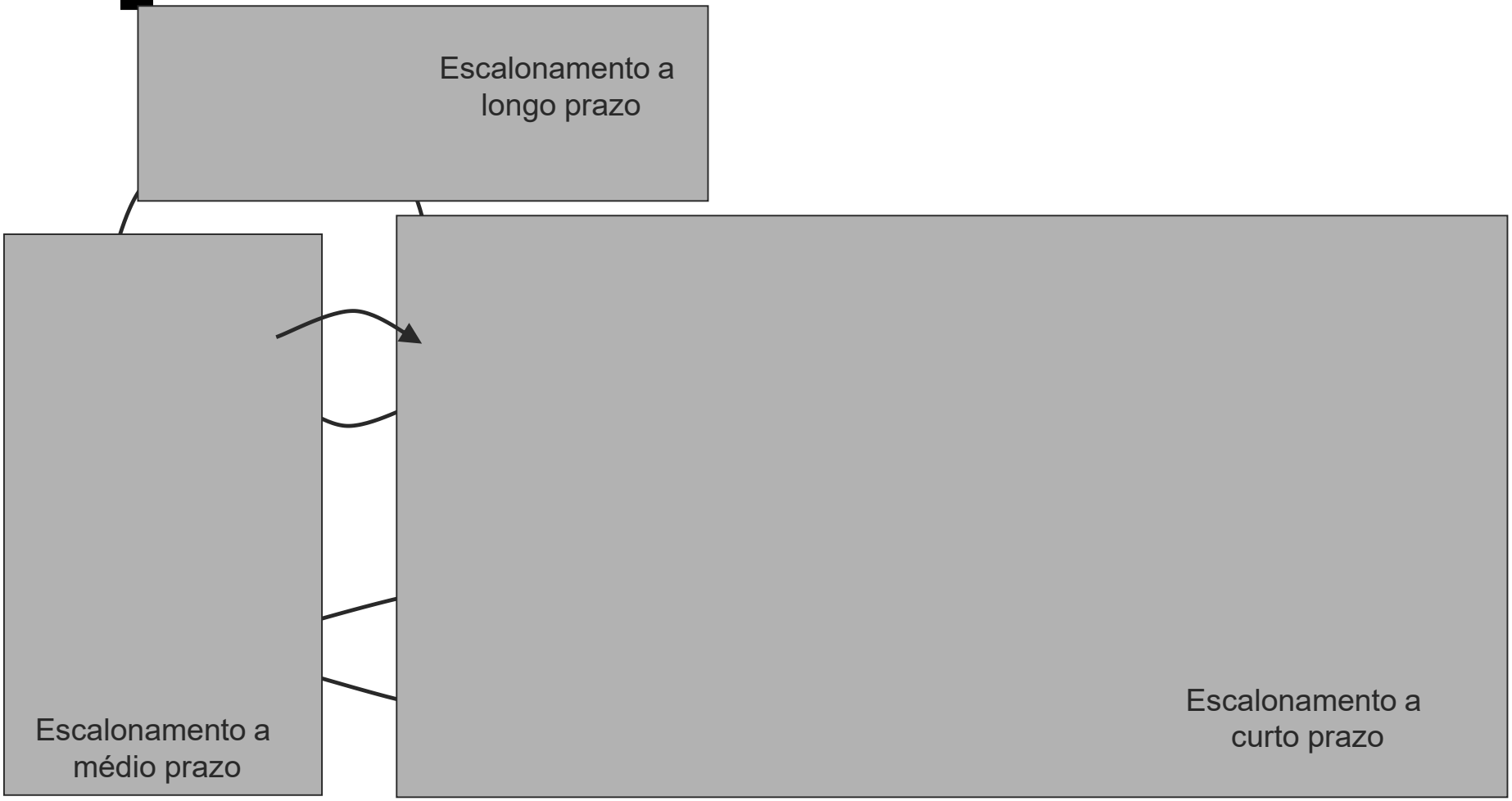
Dispatcher

- Outro componente envolvido na função de escalonamento é o *Dispatcher* :
  - É o módulo que atribui a CPU ao processo seleccionado pelo escalonador de curto-prazo
- *Dispatcher ex*ecuta as seguintes operações:
  - comutação de contexto
  - comutação para o modo de utilizador
  - salto para o endereço adequado de memória do programa por forma a continuar ou iniciar a sua execução
- O tempo de execução do *Dispatcher* deverá ser minimizado

Estrutura de um Escalonador



Tipos de Escalonamento



Algoritmos de Escalonamento

- *First-Came, First-Served* (FCFS)
- *Shortest-Job-First* (SJF)
- Escalonamento por Prioridades
- *Round-Robin* (RR)
- Multi-nível por Filas
- Multi-nível com realimentação por filas

First-Come, First-Served (FCFS)

- O processo que chega à fila de *ready* em primeiro lugar é também o primeiro processo a ser executado
- Simples e fácil de implementar
- Não preemptivo
  - um processo apenas liberta a CPU quando termina ou quando requer uma operação de I/O
- Problemas
  - Efeito de comboio, podendo resultar em grandes tempos de espera
  - Favorece processos *CPU Bound*, dado que os processo *I/O Bound* têm que esperar pelo fim dos processos CPU *Bound*

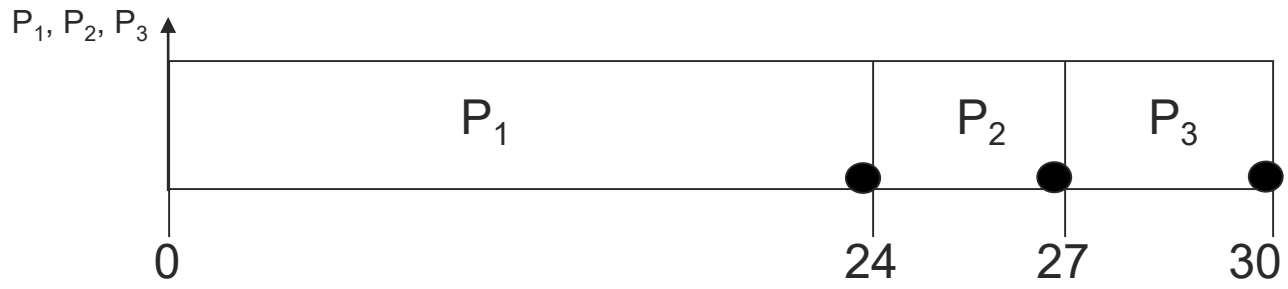
First-Come, First-Served (FCFS)

- Exemplo:

Processo	*Burst Time*
P_1	24
P_2	3
P_3	3

  - Ordem de chegada: {P1, P2, P3}

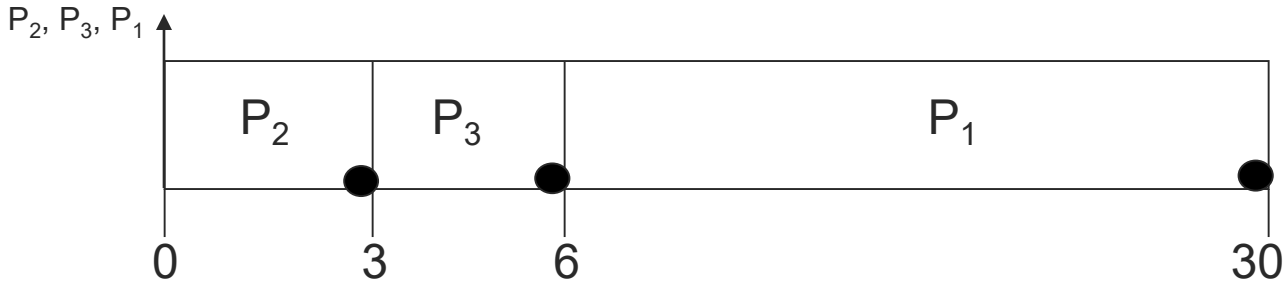
First-Come, First-Served (FCFS)



- Tempo de espera: P1=0, P2=24, P3=27
- Tempo médio de espera: (0+24+27)/3 =17

First-Come, First-Served (FCFS)

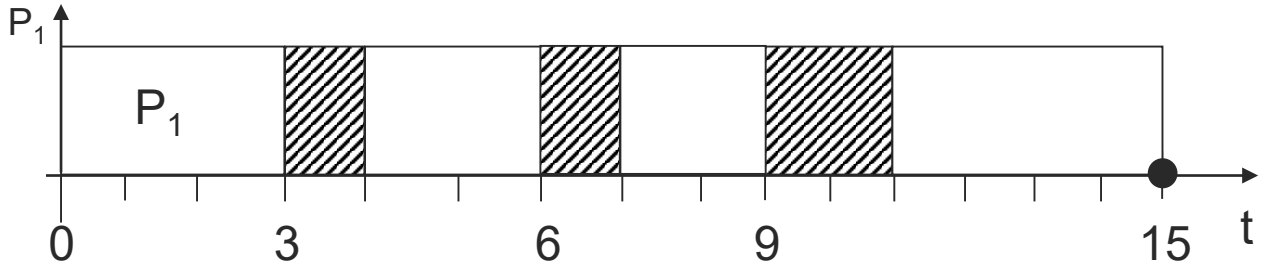
- Ordem de chegada: {P2, P3, P1}



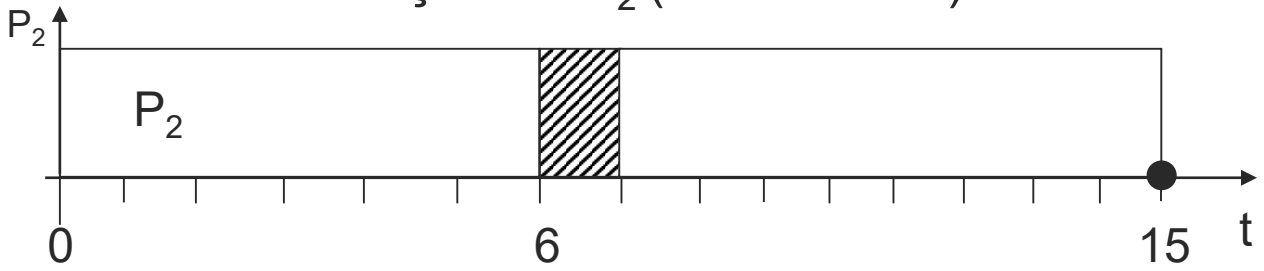
- Tempo de espera: P2=0, P3 = 3, P1=6
- Tempo médio de espera: (0+3+6)/3=3
- O **Efeito de Comboio** deve-se aos processo mais curtos terem que esperar por processos de maior duração

First-Come, First-Served (FCFS)

- Exemplo com processos *I/O Bound* e *CPU Bound*
  - Perfil de execução P_1 (*I/O Bound*)

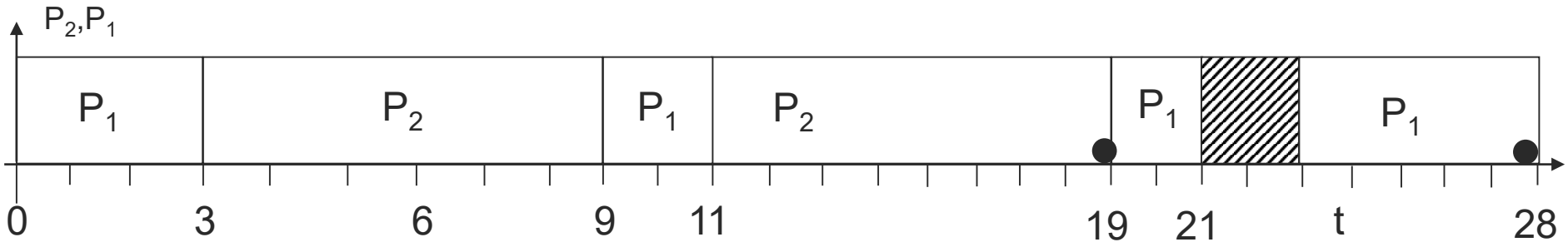


  - Perfil de execução de P_2 (*CPU Bound*)



First-Come, First-Served (FCFS)

- Resultado
  - Ordem de chegada {P_1, P_2}



  - Os processo *CPU Bound* tem tendência para serem executados primeiro

First-Come, First-Served (FCFS)

- Favorecimento de processos *CPU-bound*
  - Os processos *I/O bound* libertam a CPU sempre que requerem uma operação de I/O
  - Passam para a fila de *waiting*
  - Quando terminam a operação de I/O, voltam a entrar na fila de *ready* na última posição
  - Esperam até que os processo à sua frente terminem ou libertem a CPU

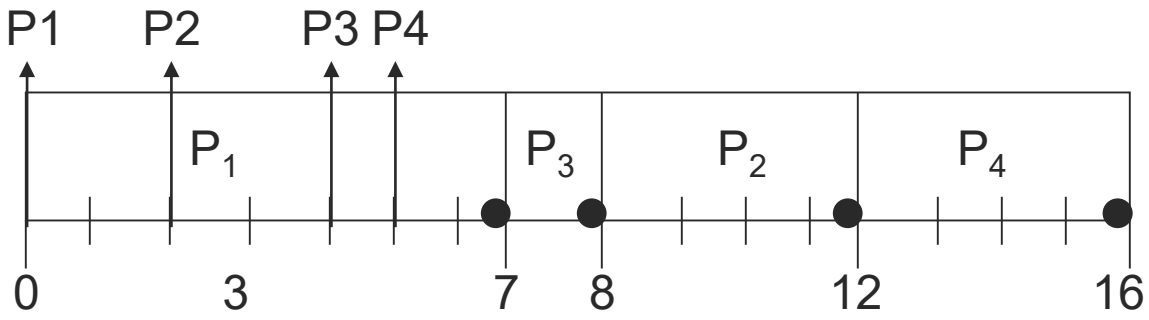
Shortest-Job-First (SJF)

- O escalonador selecciona o processo na fila de *ready* que tiver menor tempo de execução
  - **Não preemptivo**: se chegar à fila de *ready* um processo com um tempo de execução menor que o processo em execução, este não é comutado
  - **Preemptivo**: o processo em execução é comutado pelo processo novo **se o tempo restante de execução** for **maior** que o **tempo de execução do processo novo** (***Shortest-Remaining-Time-First*** (SRTF))

Shortest-Job-First (SJF)

- **SJF - Não Preemptivo**

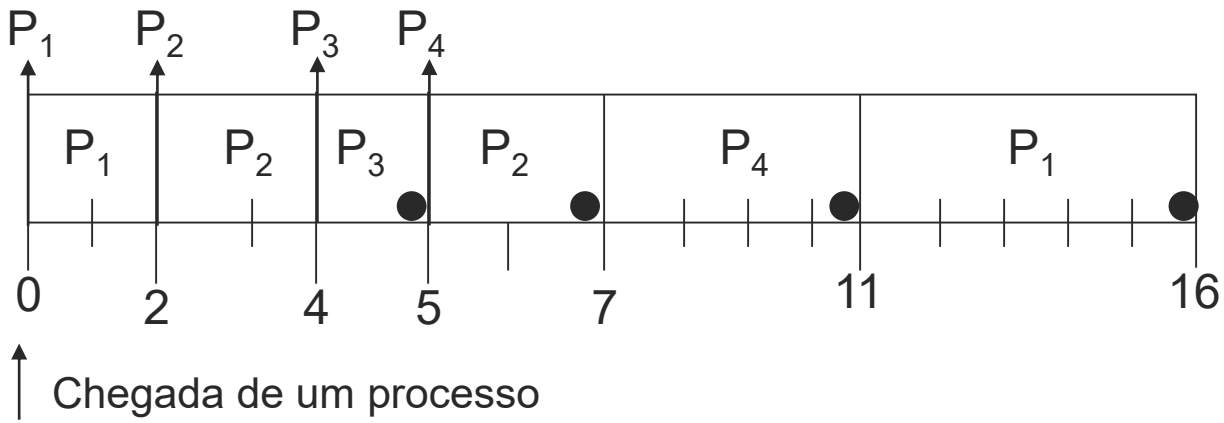
Processo	Chegada	*Burst*
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



Shortest-Job-First (SJF)

- **SJF - Preemptivo**

Processo	Chegada	*Burst*
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



Shortest-Job-First (SJF)

- **Características**
  - É considerado como um algoritmo óptimo* para minimização do tempo médio de espera
  - **Usado essencialmente para escalonamento a longo prazo**
- **Problemas**
  - É necessário conhecer e avaliar o tempo de execução do processo
  - Não é possível saber o tempo de execução com precisão, apenas é possível estimar
    - Pode-se utilizar como tempo de execução o tempo limite especificado pelo utilizador
  - Em casos de carga elevada os processos mais longos são prejudicados (i.e. são os últimos a ser executados)
  - Pouco adequado ao escalonamento de curto prazo devido à dificuldade de prever a duração do próximo *CPU Burst*

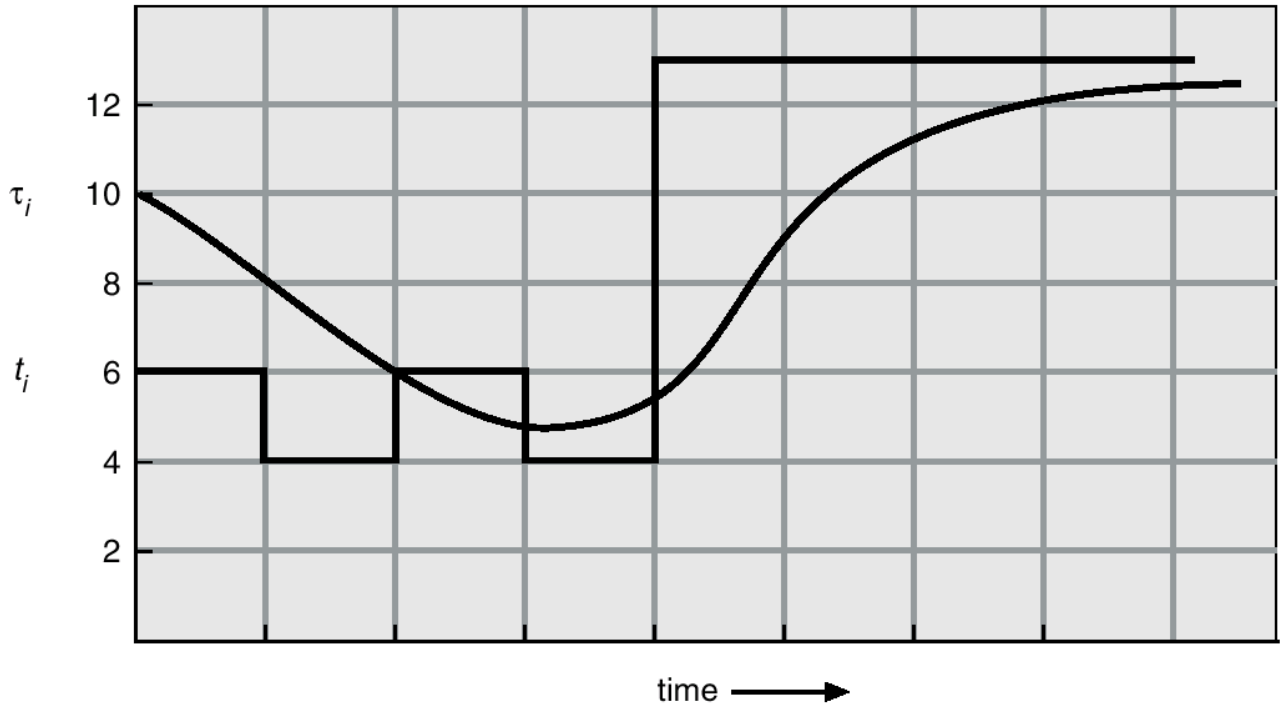
* Não foi ainda provado

Shortest-Job-First (SJF)

- SJF no escalonamento a curto prazo
  - Pode ser utilizada uma estimativa da duração do próximo *CPU Burst* (τ_{n+1}) com base nos anteriores
  - A duração do próximo *CPU Burst* é uma média exponencial da duração dos anteriores *CPU Burst*
  - $\tau_{n+1}=\alpha.t_n + (1- \alpha).t_n$ - média exponencial para um valor de α
    - α: controla o peso relativo da historia passada e recente, $0 \leq \alpha \leq 1$
    - t_n: duração do último *CPU Burst*
    - τ_{n+1}: valor previsto para o próximo *CPU Burst*

Shortest-Job-First (SJF)

$\alpha = 0.5$
$t_0 = 10$



CPU burst (t_i)	6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Escalonamento por Prioridades

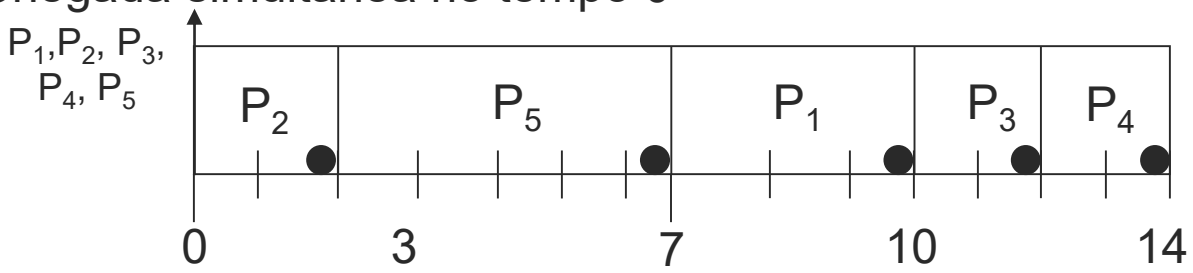
- Cada processo tem uma prioridade associada
- A CPU é alocada ao processo com maior prioridade
- Processos de igual prioridade podem ser escalonados através de FCFS
- Pode ser preemptivo ou não preemptivo
- A prioridade é representada por um número inteiro positivo (nesta disciplina adoptou-se que 0 representa a maior prioridade)

Escalonamento por Prioridades

- Escalonamento não preemptivo

Processo	*Burst*	Prioridade
P1	3	3
P2	2	1
P3	2	4
P4	2	5
P5	5	2

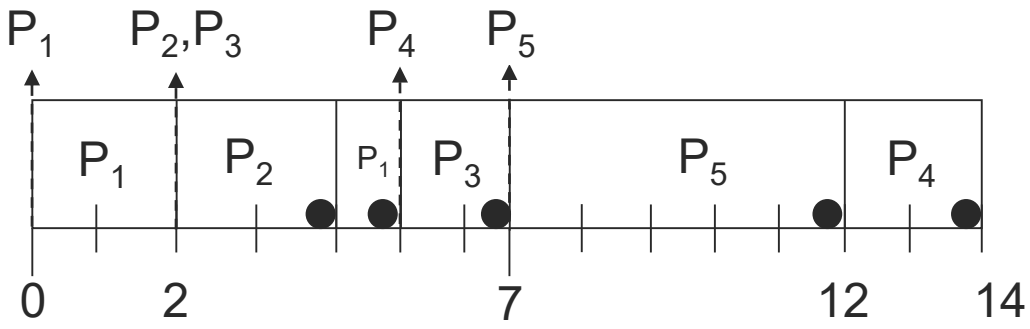
Chegada simultânea no tempo 0



Escalonamento por Prioridades

- Escalonamento preemptivo

Processo	*Burst*	Prioridade	Tempo de chegada
P1	3	3	0
P2	2	1	2
P3	2	4	2
P4	2	5	5
P5	5	2	7



Escalonamento por Prioridades

- Características
  - Em situações de sobrecarga os processos de menor prioridade podem nunca ser executados
    - Uma das soluções para este problema é aumentar a prioridade do processo com o tempo
  - As prioridades podem ser definidas:
    - em função das propriedades do processo
      - Duração dos *CPU Bursts*, *deadlines*, importância do processo, etc
    - politicamente
      - Em função do pagamento do utilizador, o departamento a quem pertence o processo, etc

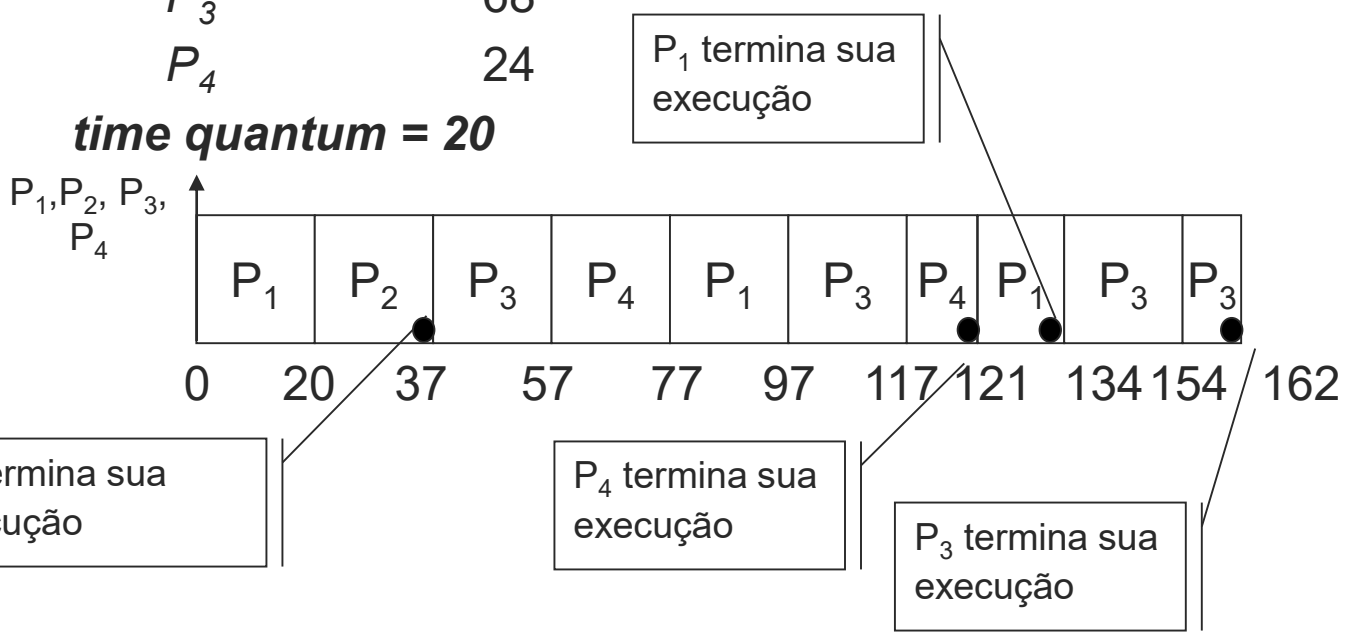
Round-Robin (RR)

- Algoritmo:
  - Cada processo obtém uma pequena unidade de tempo da CPU, ***time quantum* ou *time slice***, vulgarmente 10 -100ms
  - No fim de cada ***time quantum*** (*q*) o processo é comutado e adicionado à cauda da fila *ready*
  - Caso o processo termine a sua execução ou passe para o estado de *waiting* durante o *time quantum* atribuído, o escalonador selecciona o processo seguinte para execução
  - O escalonador necessita de um ***timer*** de modo a que seja periodicamente interrompido após cada ***time quantum***
- Especialmente adaptado para **Sistemas Partilhados Multi-utilizador**

Round-Robin (RR)

Processo	*Burst*
P_1	53
P_2	17
P_3	68
P_4	24

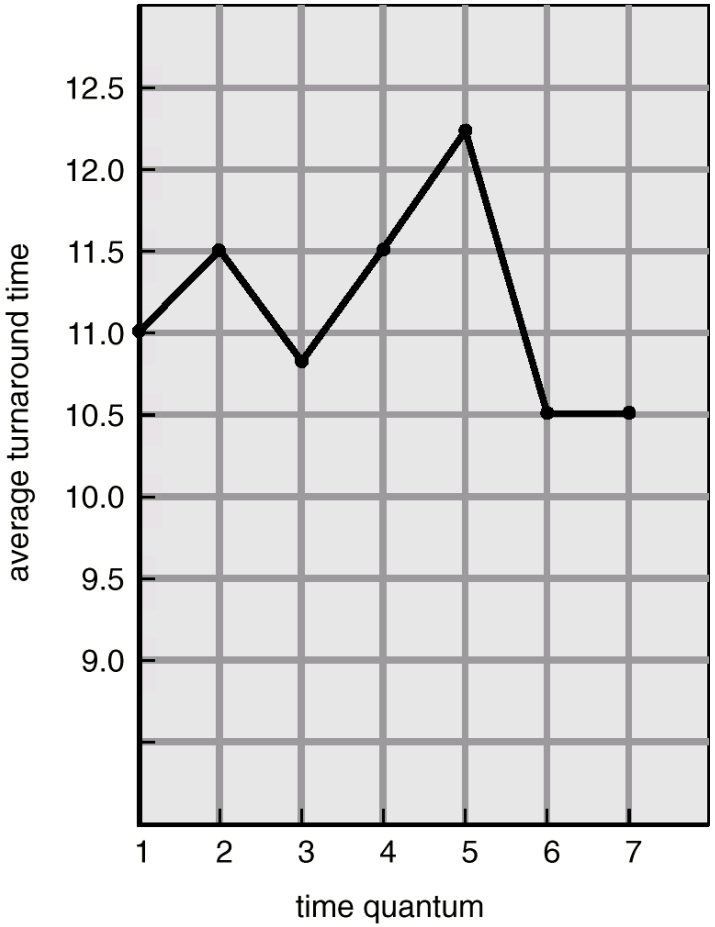
time quantum = 20



Round-Robin (RR)

- **Características**
  - Melhor tempo de resposta do que SJF
  - O escalonamento em RR pode ser visto como dividir a capacidade de processamento da CPU pelo vários processos
- **Problemas**
  - Se o *time quantum* for pequeno o *overhead* do algoritmo pode ser grande (FCFS)
  - Favorece processo *CPU bound*
    - Os processos *I/O bound* usam a CPU durante um tempo inferior ao *time quantum* e após o que passam para a fila de *waiting*
    - Os processo *CPU bound* usam o seu *time quantum* na totalidade e voltam para a fila de *ready*

Round-Robin (RR)



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Turnaround time

tempo que decorre entre o instante em que um processo é submetido e o instante em que é concluído

Turnaround time também está dependente do valor do *time quantum*

Round-Robin (RR)

- Algumas questões:
  - E se quisermos dar maior importância a um processo?
    - Que alterações devem ser feitas ao algoritmo RR?
  - Se reduzirmos o *time quantum* para um valor muito pequeno.
    - Que problemas ocorrem?

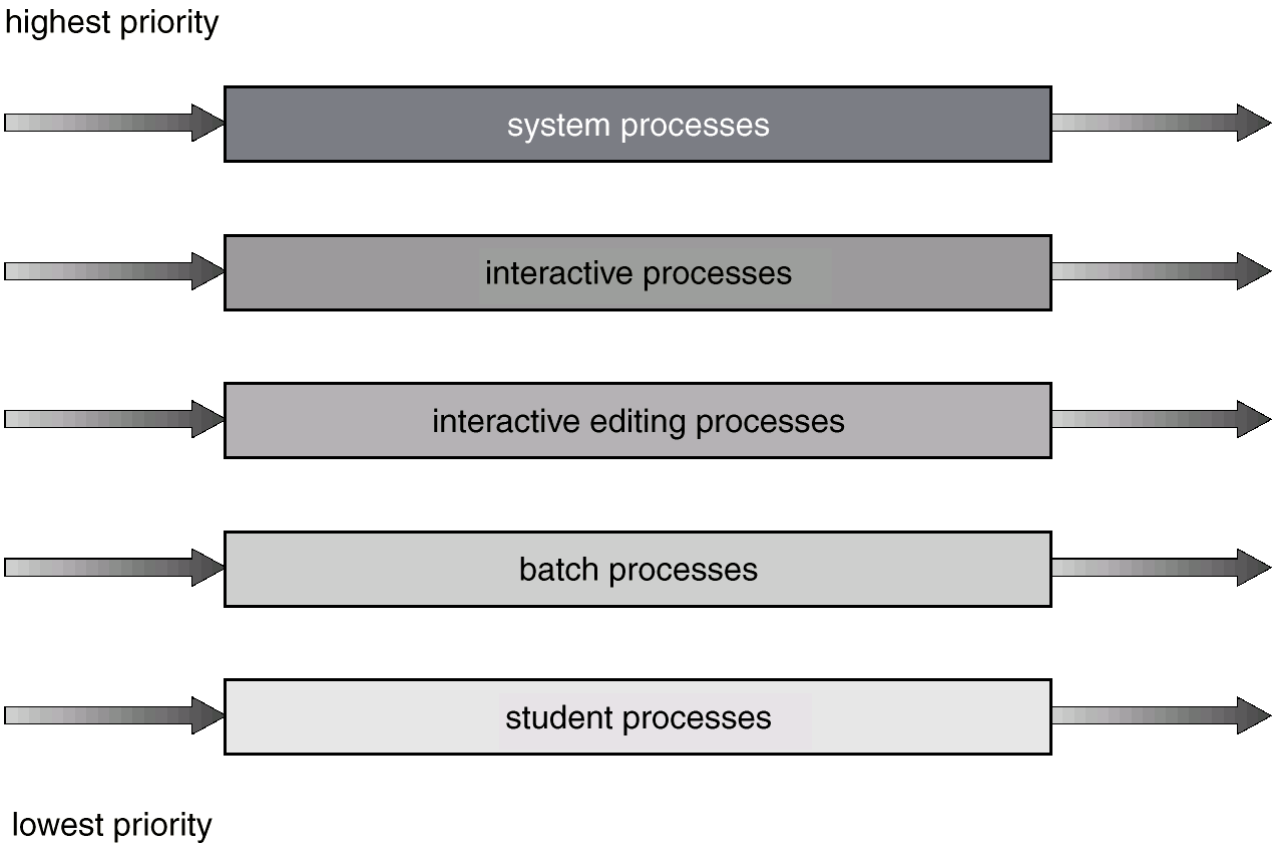
Multi-nível por Filas

- Este tipo de escalonamento é usado quando é fácil classificar os processos em classes distintas, por ex.:
  - Processos interactivos
  - Processos *batch*
  - Aplicações multimédia
- A fila *ready* é dividida em várias filas, uma por cada classe de processos

Multi-nível por Filas

- Cada fila pode ter o seu próprio algoritmo de escalonamento, por ex.:
  - Processos interactivos: RR
  - Processos *batch*: FCFS
  - Processo do sistema: prioridades
- Cada fila, poderá, tem prioridade absoluta sobre a outra
  - i.e. um processo da fila de *batch* apenas corre quando não existirem processos na fila dos processos interactivos

Multi-nível por Filas

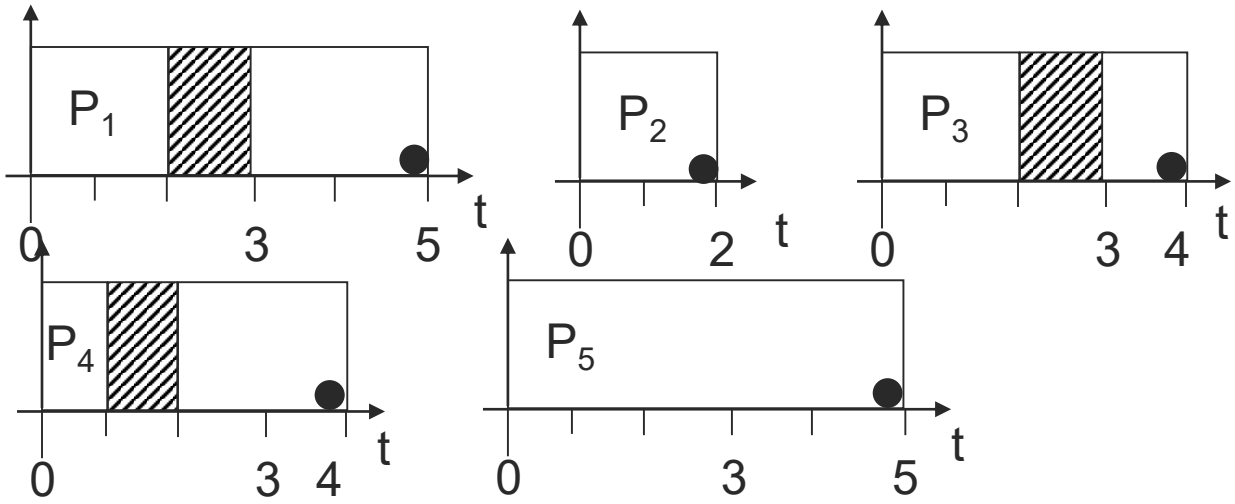


Multi-nível por Filas

- Também é necessário efectuar o escalonamento entre as filas:
  - **Prioridades fixas**
    - Em casos de carga elevada os processos atribuídos às filas de menor prioridade são preteridos
  - **Round-Robin**
    - cada fila obtém uma certa quantidade de tempo da CPU que pode ser escalonado pelos seus processos, por exemplo:
      - 80% para processos *foreground* em RR
      - 20% para processos *background* em FCFS

Multi-nível por Filas

- Exemplo: 2 processos do sistema: P_1 e P_2
  2 processos interactivos: P_3 e P_4
  1 processo batch: P_5
- Perfis de execução

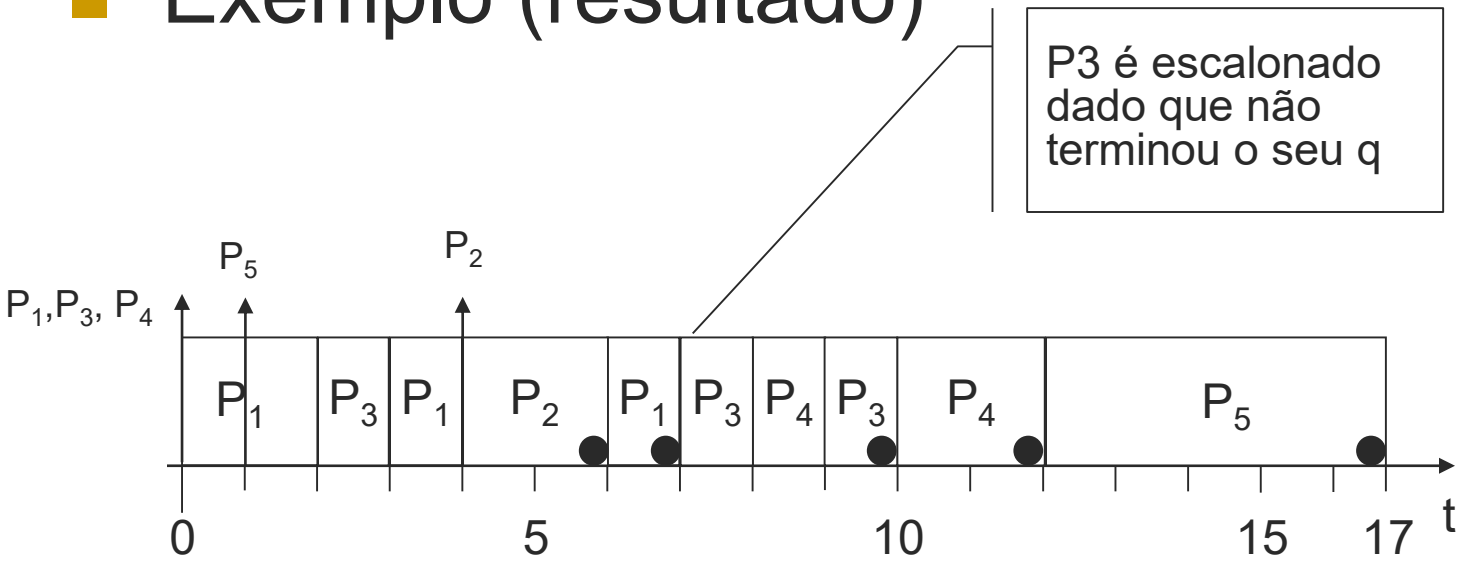


Multi-nível por Filas

- Exemplo (cont.)
  - Algoritmos de escalonamento
    - Processos interactivos: RR com *time quantum* igual a 2
    - Processos *batch*: FCFS
    - Processos do sistema: prioridades ($P_1 = 2$, $P_2 = 0$)
  - Chegadas dos processos
    - $P_1 – 0$
    - $P_2 – 4$
    - $P_3 – 0$
    - $P_4 – 0$
    - $P_5 – 1$

Multi-nível por Filas

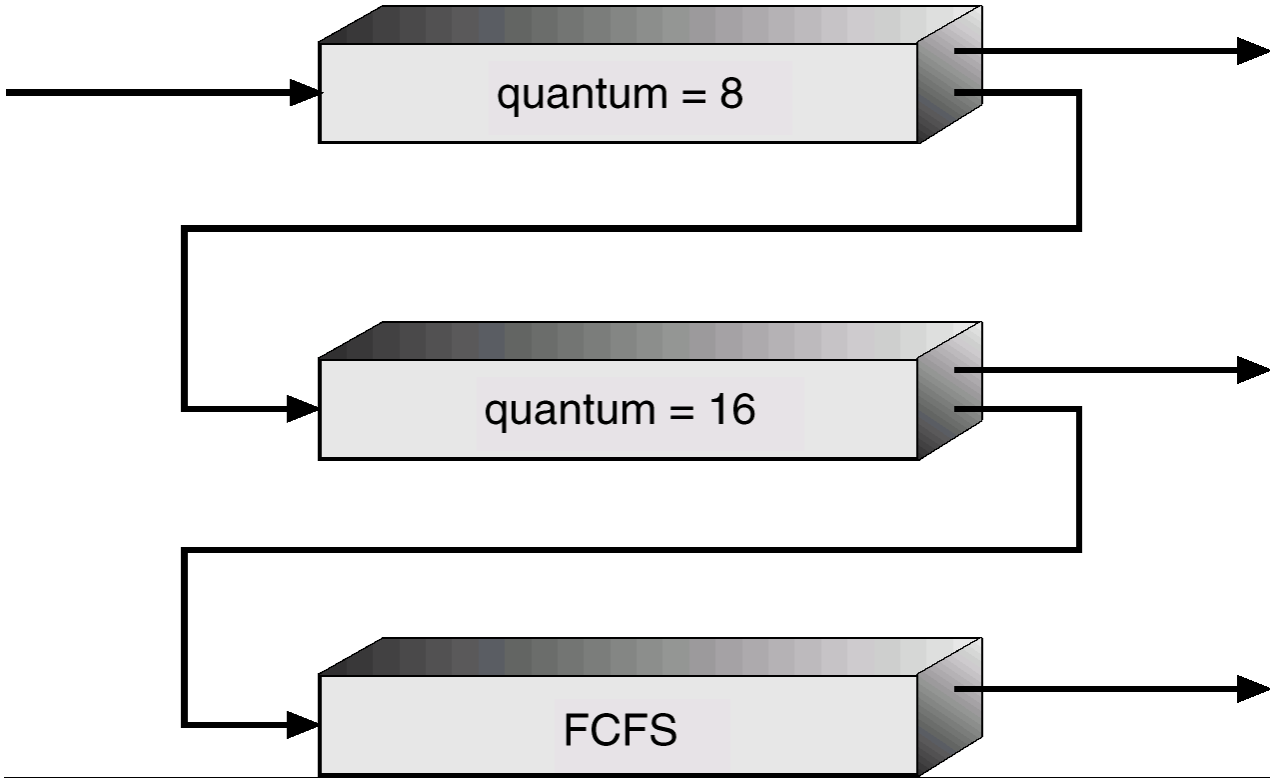
- Exemplo (resultado)



Multi-nível com Realimentação por Filas

- Permite que um processo se mova entre filas, i.e. **acrescenta adaptabilidade aos escalonamento multi-nível**
- A passagem para um fila de prioridade inferior é feita quando, por exemplo:
  - o processo utiliza mais tempo da CPU (*time quantum*) do que aquele que lhe estava destinado
  - um processo que espere demasiado tempo numa fila de prioridade inferior pode ser passado para um fila de prioridade superior

Multi-nível com Realimentação por Filas

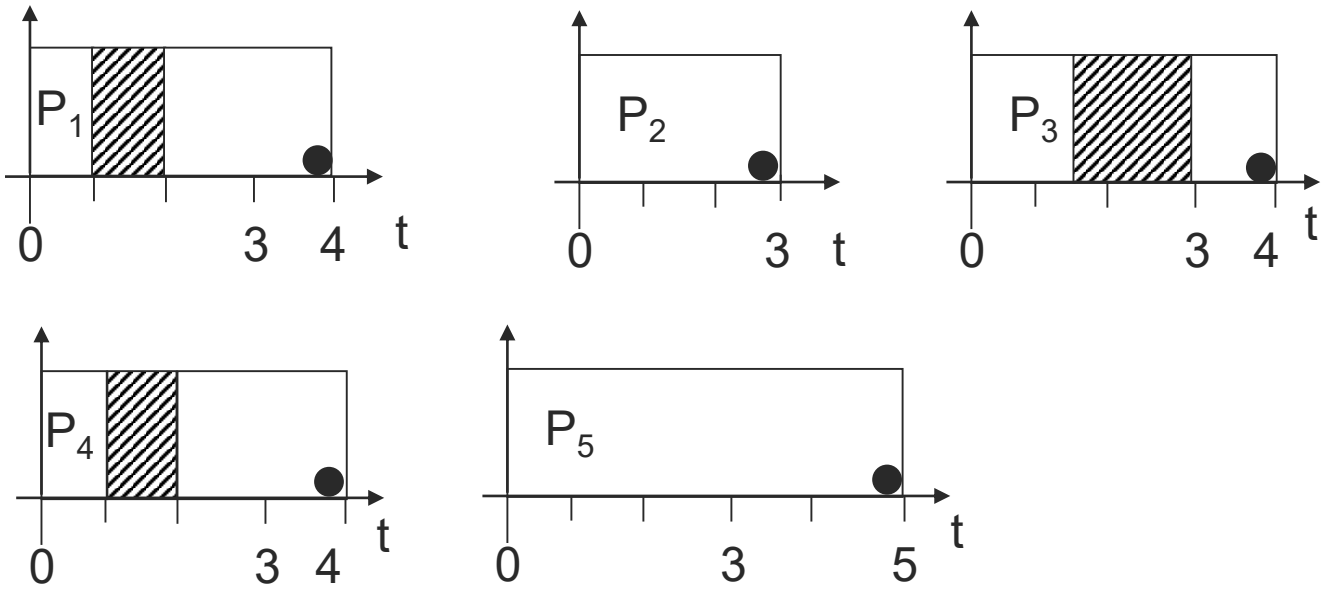


Multi-nível com Realimentação por Filas

- Três filas:
  - Q0 – RR com *time quantum* de 8 ms
  - Q1 – RR com *time quantum* de 16 ms
  - Q2 – FCFS
- Escalonamento
  - Um novo processo entra na fila Q0, a qual segue uma política RR. Quando ganha a CPU, o processo recebe 8 ms; se não terminar em 8 ms, o processo é passado para a fila Q1
  - Em Q1, o processo é servido novamente por uma política de escalonamento RR e recebe 16 ms adicionais;
    se mesmo assim não termina, o processo é passado para a fila Q2 com uma política FCFS

Multi-nível com Realimentação por Filas

- Exemplo:
  - Perfis de execução

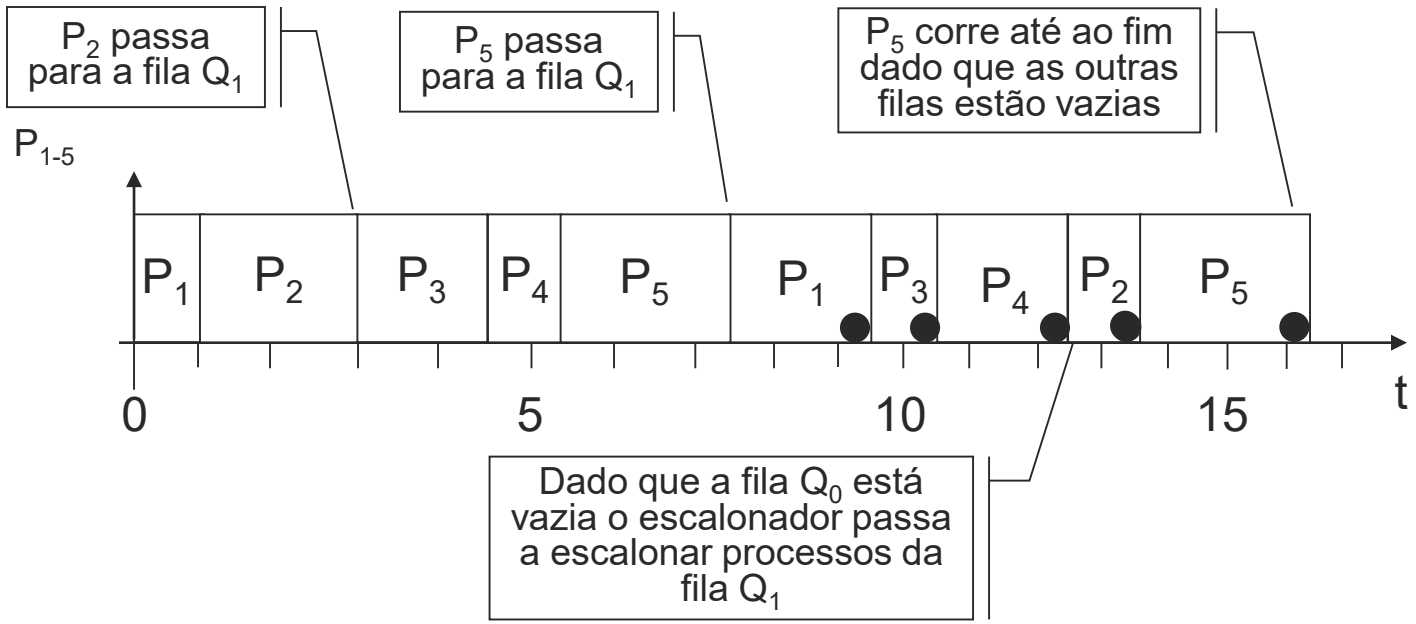


Multi-nível com Realimentação por Filas

- Exemplo (cont.)
  - Algoritmos de escalonamento
    - Q_0 RR com time quantum igual a 2
    - Q_1 RR com time quantum igual a 3
    - Q_2 FCFS
  - Chegadas dos processos
    - P_1 – 0
    - P_2 – 0
    - P_3 – 0
    - P_4 – 0
    - P_5 – 0

Multi-nível com Realimentação por Filas

- Exemplo (resultado)



Multi-nível com Realimentação por Filas

- Caracterizado por:
  - Número de filas
  - Algoritmos de escalonamento de cada fila
  - Método utilizado para passar um processo para uma fila de mais alta ou de mais baixa prioridade
  - Método utilizado para decidir em que fila o processo é colocado, após a sua entrada no sistema

Escalonamento Multiprocessador

- Multi-processamento assimétrico
  - Só um processador acede às estruturas de dados do sistema, aliviando a necessidade de partilha de dados
  - É usada uma única fila *Ready*, e não uma fila por processador, para evitar que haja algum processador inactivo enquanto outros têm processos nas suas filas de *Ready* à espera

Escalonamento Multiprocessador

- Multi-processamento simétrico
  - Todos os processadores são indênticos
  - Cada processador á responsável pelas suas decisões de escalonamento
    - As filas de processos podem ser locais ou globais ao sistema
  - Os processos podem migrar entre processadores
  - Suportada pelos SOs principais!!!

Escalonamento Multiprocessador

- Multi-processamento simétrico
  - As filas de processos são guardadas numa estrutura partilhada
    - Nota: é necessário fazer o lock a esta lista sempre que se tentar efectuar alterações
  - Vai existir contenção no acesso às filas
    - Mais problemático quando existem muitos CPUs
  - Em relação a um sistema uniprocessador pode existir maior *overhead* quando um processo tem que mudar de processador. Porquê?

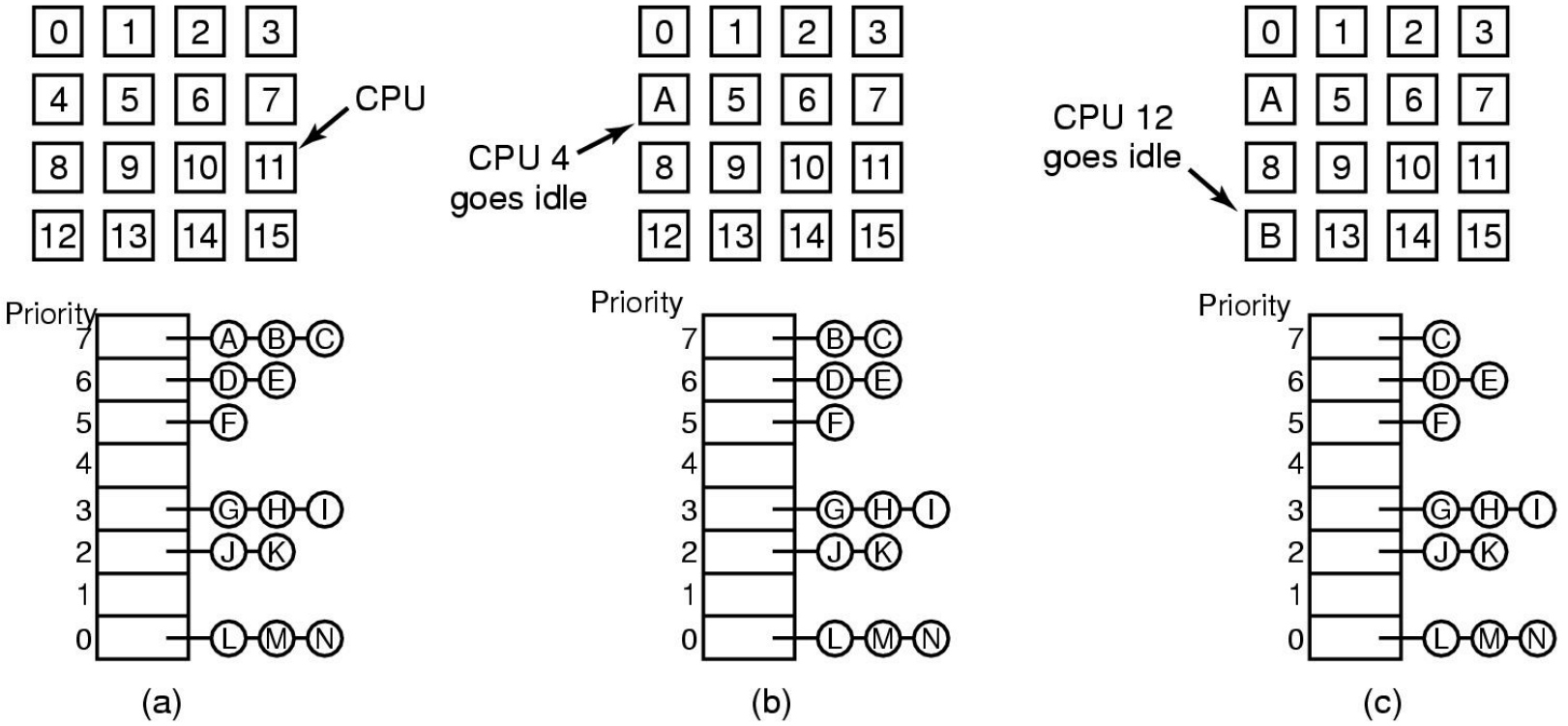
Escalonamento Multiprocessador

- Afinidade do processador
  - Será que é vantajoso, em termos de performance, migrar um processo para um CPU diferente sempre que uma CPU se encontrar livre?
    - Não:
      - o código e os dados do processo já se podem encontrar na cache da CPU, logo podemos não ter vantagens em migrar o processo
      - O processo poderá ter que cooperar com outros processo que se encontrem na mesma CPU e é mais rápido faze-lo utilizando a cache da CPU em que se encontra
  - Consequentemente, os SO actuais permitem definir a **Afinidade**, de um processo a uma CPU
  - Esta propriedade é definida pelos programadores

Escalonamento Multiprocessador

- Balanceamento de carga
  - Objectivo: manter a carga de todos os processadores do sistema uniforme
  - Técnicas:
    - Push
      - Uma tarefa verifica periodicamente a carga de um determinado processador e envia os processos extra para outros processadores
    - Pull
      - Um processador que está com pouca carga vai executar processos que se encontrem nas filas de *Ready* de outro processador

Escalonamento Multiprocessador



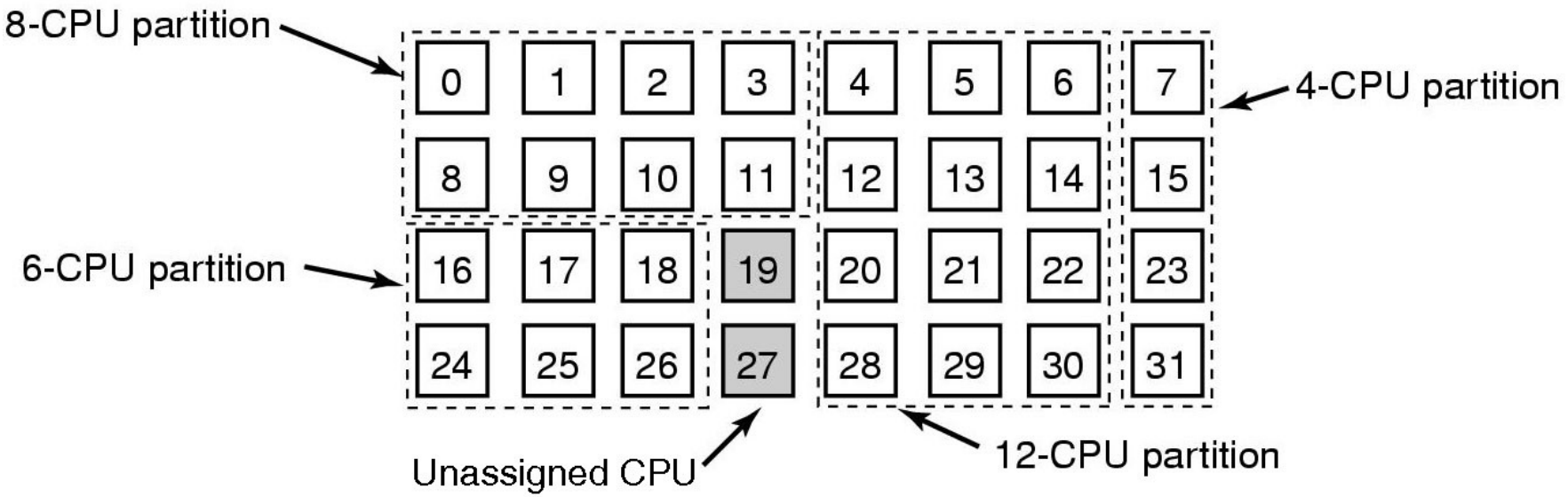
Escalonamento Multiprocessador

- Algorítmos específicos

Escalonamento Multiprocessador

- Space Sharing
  - Os processos são escalonados em conjunto, sendo-lhes atribuído um conjunto de processadores onde vão correr isoladamente
  - Os processadores podem ficar no estado de *idle* se um processo for realizar operações de I/O
  - Especialmente indicada para sistemas não interactivos, p.e. sistemas de previsão meteorológica, simulações, etc.
  - Pode também ter vantagens quando os processo partilham dados entre si

Escalonamento Multiprocessador



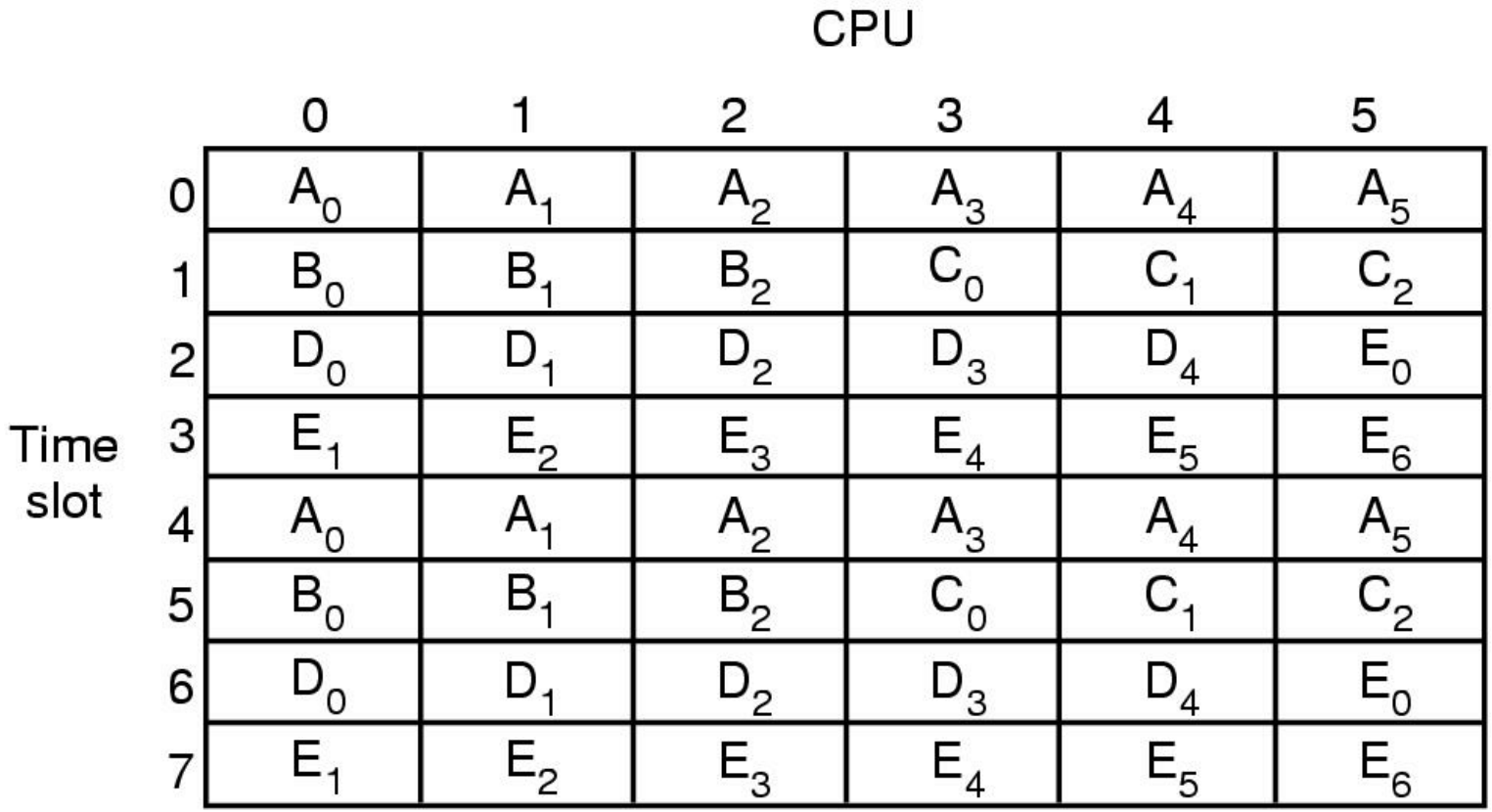
Escalonamento Multiprocessador

- Gang Scheduling
  - Grupos de processo relacionados são escalonados como uma unidade (gang)
  - Todos os membros de um gang correm simultaneamente em diferentes CPUs
  - Todos os membros de um gang iniciam e terminam o seu timeslice ao mesmo tempo

Escalonamento Multiprocessador

- Vantagens
  - A comunicação entre processo é mais rápida e pode ser feita durante o timeslice

Escalonamento Multiprocessador



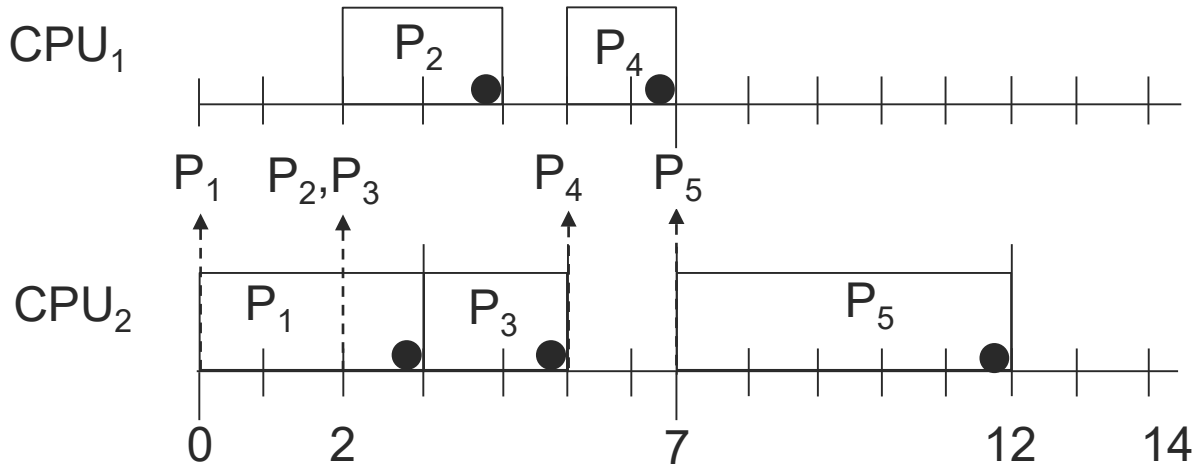
Escalonamento Multiprocessador

- Aplicação de algorítmos clássicos

Escalonamento Multiprocessador

- Escalonamento por prioridades

Processo	*Burst*	Prioridade	Tempo de chegada
P1	3	3	0
P2	2	1(+ alta)	2
P3	2	4	2
P4	2	5	5
P5	5	2	7



Escalonamento Multiprocessador

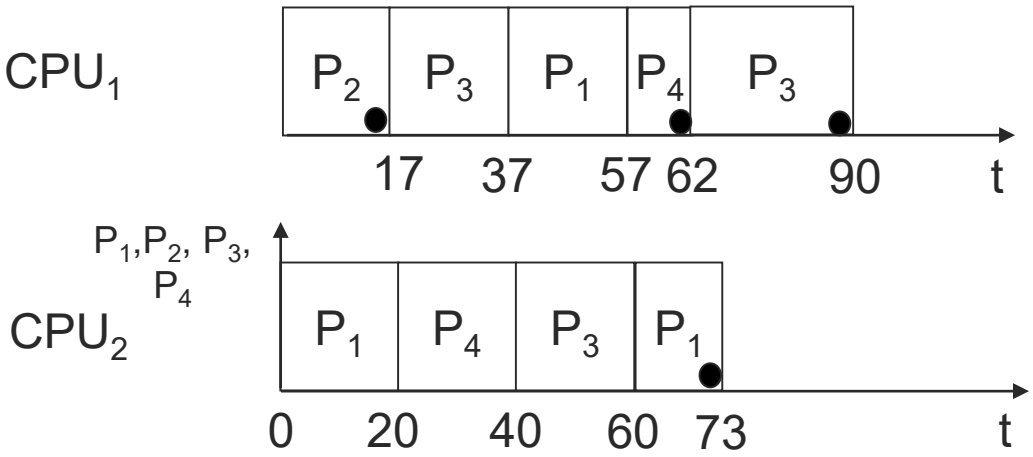
- Notas:
  - Assume-se que custo de mudar de processador é muito baixo e que se pode desprezar.
  - Existe uma fila global de processos acessível por todos os processadores
    - Identifique os problemas de concorrência que podem ocorrer no acesso a essa lista e como se podem resolver.

Escalonamento Multiprocessador

Processo	*Burst*
P_1	53
P_2	17
P_3	68
P_4	24

time quantum = 20

Escalonamento Round-Robin



Acesso a recursos partilhados

- Recursos:
  - Ficheiro (escrita)
  - Acesso a uma zona crítica de um programa

Acesso a recursos

Chegada de um processo ao sistema

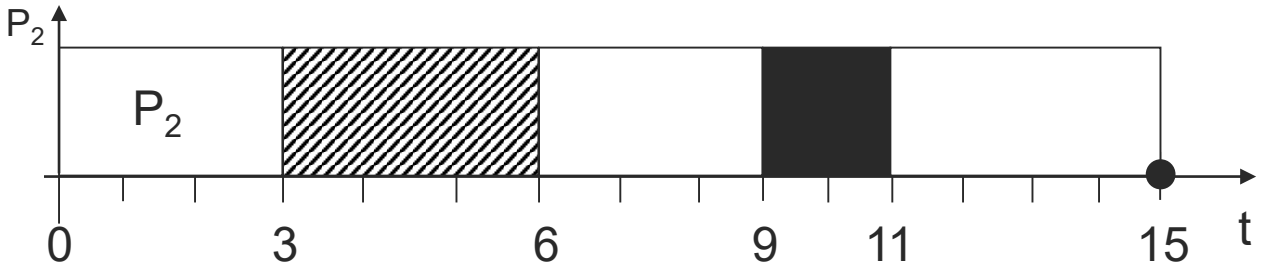
Processo em execução (estado de *running*)

Processo à espera (estado de *waiting*)

Finalização de um processo

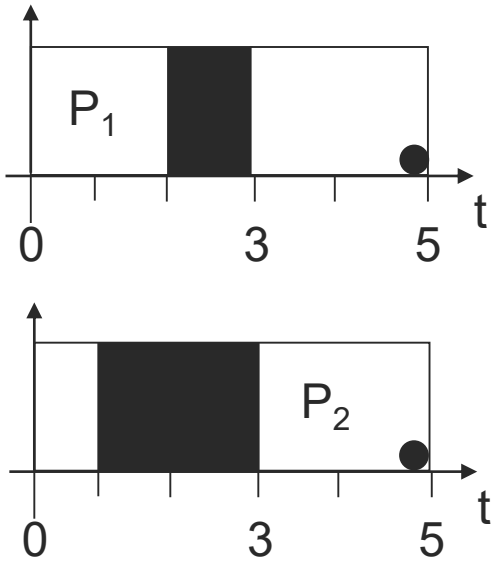
Processo a aceder a um recurso partilhado

Exemplo:

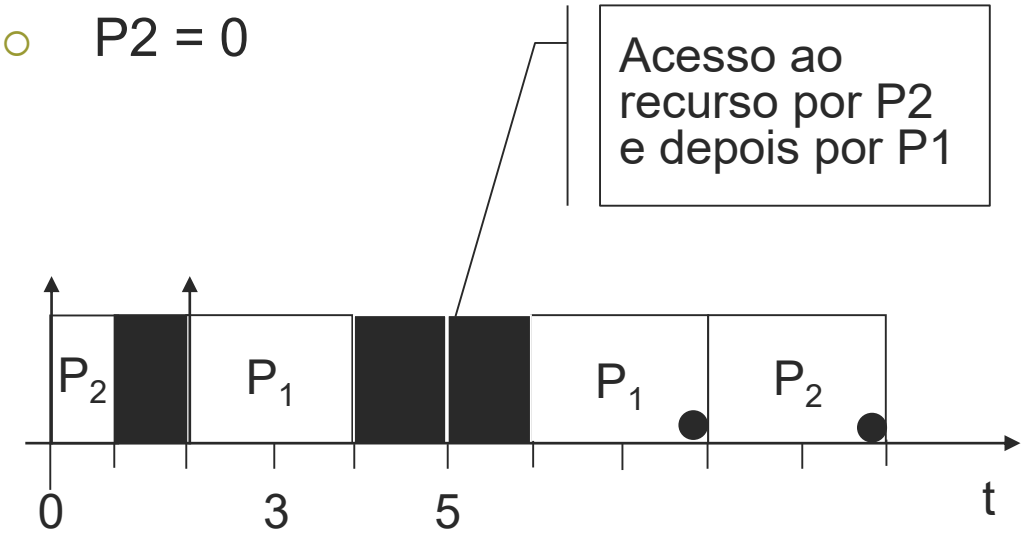


Acesso a recursos partilhados

- Prioridade:
  - P1 = 0
  - P2 = 1



- Tempo de chegada:
  - P1 = 2
  - P2 = 0



- Assuma escalonamento por prioridades

Acesso a recursos partilhados

P1

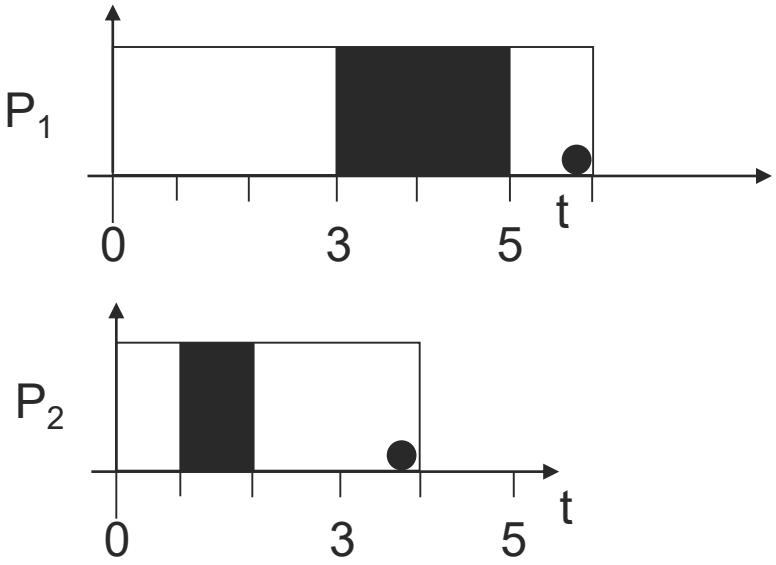
```
…
delay(3)
down(S1)
…
delay(2)
up(S1)
…
delay(1)
exit(0)
```

P2

```
…
delay(1)
down(S1)
…
delay(1)
up(S1)
…
delay(2)
exit(0)
```

Acesso a recursos partilhados

- Representação gráfica



Acesso a recursos partilhados

- Exercícios
  1. Considere que prioridade de P1 é 0 e de P2 é 1, o tempo de lançamento de P1 é 1,5 e P2 é 0
  2. Assuma escalonamento em *round-robin* com *time quantum* igual a 2 e que os dois processo são lançados no tempo zero

Escalonamento Tempo-Real

- Tipos de aplicações
  - Industriais
  - Avionics
  - Automóveis
  - Multimédia
- Tipos de sistemas tempo real
  - Sistemas críticos (*Hard Real-Time*)
  - Sistemas não críticos (*Soft Real-Time*)

Escalonamento Tempo-Real

- Sistemas críticos (*Hard Real-Time*)
  - É necessário garantir que a(s) tarefa(s) consideradas críticas terminem antes de um determinado tempo (*deadline*), caso contrário o seu não cumprimento pode resultar em graves danos para o sistema
  - Exemplos:
    - Aplicações aeroespaciais
    - ABS de um carro
    - Sistema de automação

Escalonamento Tempo-Real

- Sistemas não críticos (*Soft Real-Time*)
  - O funcionamento do sistema é apenas ligeiramente afectado caso não seja possível cumprir um determinada *deadline*.
  - Exemplos:
    - Aplicações multimédia
    - Jogos de computador

Escalonamento Tempo-Real

- Os métodos de escalonamento devem garantir *à priori* que o sistema cumpre as suas metas temporais
- É normalmente necessário conhecer o tempo de execução das tarefas
  - Periódicas (por ex. para aquisição de dados)
  - Esporádicas (por ex. para o tratamento de alarmes)
- O sistema é muito pouco dinâmico
  - Escalonamento por prioridades
  - Escalonamento utilizando o algoritmo *Earliest Deadline First (EDF)*

Exemplos de Escalonadores

Exemplo de Escalonadore

- Linux
- Windows (livro)
- Solaris (livro)

Linux

Linux

- Em Linux, o escalonamento também inclui a execução das tarefas do *kernel*
- Estas tarefas do *kernel* incluem as tarefas requisitadas por processos em execução e as tarefas internas ligadas aos *device drivers*
- A execução em modo *kernel* pode ocorrer de 3 formas:
  - Um programa em execução requisita explicitamente um serviço do SO através de uma chamada ao sistema
  - Implicitamente quando a gestão de memória virtual gera uma "falha de página"
  - Um *device driver* gera uma interrupção que leva a CPU a iniciar uma rotina do *kernel* para atendimento da interrupção

Linux

- O Linux usa 3 métodos de escalonamento de processos:
  - Um algoritmo de tempo partilhado para o escalonamento de processos “convencionais”
  - Dois algoritmos de tempo-real para processos em que a previsibilidade dos parâmetros temporais (por ex. período e *deadline*) do processo são importantes para a sua execução correcta
    - Por ex. uma tarefa responsável por descodificar um filme deve mostrar um fotograma 30 vezes por segundo

Linux

- É a classe de escalonamento de cada processo que determina qual o algoritmo a usar:
  - SCHED_FIFO: First-in-first-out tempo real
  - SCHED_RR: Round-Robin tempo real
  - SCHED_OTHER: Escalonamento hierarquico com realimentação por filas

Linux

- O escalonador escolhe para entrar em execução o processo com maior prioridade (0-139) presente na fila de *ready*
  - 0-99 prioridades estáticas para os processo de tempo-real
  - 100-139 prioridades dinâmicas para processos convencionais
- Notas:
  - As prioridades estáticas são sempre superiores às dinâmicas
  - Os processos de tempo-real tem sempre prioridade relativamente aos processo convencionais

Linux

- Para processo convencionais
  - O tempo da CPU é dividido em *epochs* (épocas)
  - No início de cada epoch o escalonador calcula os *time quantums* de cada processo
  - Um processo é substituído por outro quando:
    - utiliza o seu *time quantum*
    - passa para o estado de *waiting*
    - é lançado um processo mais prioritário
  - O *epoch* termina quando todos os processo na fila de *Ready*, tiverem esgotado o seu *time quantum*

Linux

- Para processos convencionais, o Linux usa um algoritmo para calcular o *time quantum* baseado em créditos e prioridades
  - A regra de creditação

$$\text{credits} := \frac{\text{credits}}{2} + \text{priority}$$

  - Os créditos são transformados em tempo
  - A regra leva em conta a história do processo e a sua prioridade
  - Este sistema de creditação automaticamente dá mais prioridade a processos interactivos ou *I/O-bound* em detrimento de processos *CPU-bound*

Linux

- O Linux implementa as classes de escalonamento de tempo-real FIFO (ou FCFS) e Round-Robin; em ambos os casos, cada processo tem uma prioridade e uma classe de escalonamento próprios
  - O escalonador executa o processo com a prioridade mais alta; em caso de empate, executa o processo que tem mais tempo de espera
  - Os processos FCFS executam até terminar ou bloquear
  - Os processos *Round-Robin* são interrompidos ao fim do seu *time quantum* e colocados no fim da fila de *ready*, por forma a garantir equidade no tratamento (entre eles)

Sistemas Operativos I

Escalonamento

Luis Lino Ferreira

Abril de 2008