Sistemas Operativos

Licenciatura Engenharia Informática
Licenciatura Engenharia Computacional

Ano letivo 2023/2024

Nuno Lau (nunolau@ua.pt)

Escalonador do CPU

Processos a querer executar >> Número de CPU's

- Selecciona de entre os processos *Ready* qual o que irá ser executado no(s) CPU(s) → Entre os processos que estão Ready escolher um para ficar Running
- Escalonador é activado quando o processo:
  - Muda do estado de *running* para *waiting* → bloquear semáforo/bloquear variável de condição/ler dados do disco
  - Muda do estado *running* para *ready* → Pode ser por exemplo Thread.yield / O SO considerou que o processo ocupou o CPU demasiado tempo ...
  - Muda do estado *waiting* para *ready*
  - Termina → Preciso de escolher outro...
- Os escalonadores que usam apenas 1 e 4 são designados *non preemptive* → Running para waiting / → Termina } Só mudam quando ele pede para mudar ...
- Escalonadores que usam 2 e 3 são *preemptive*

→ Retirar do CPU um processo que queria continuar em execução

Escalonador do CPU

- *Dispatcher* encarrega-se de colocar o processo selecionado pelo escalonador em execução no CPU
  - Mudança de contexto
  - Alterar CPU para modo de utilizador
  - Saltar para instrução do programa que permite continuar a execução do processo selecionado

Restaurar o contexto de utilização

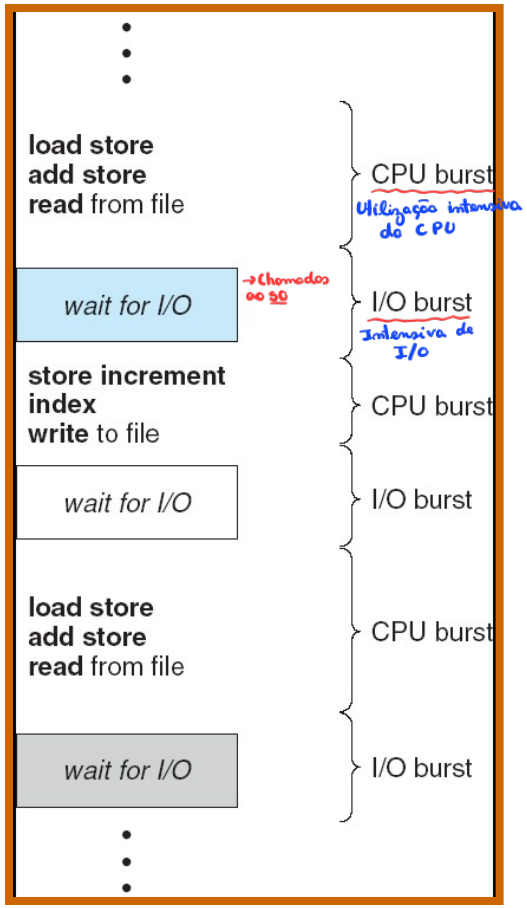
- *Dispatch latency* – tempo que o *Dispatcher* demora entre parar um processo e reiniciar o processo selecionado pelo escalonador

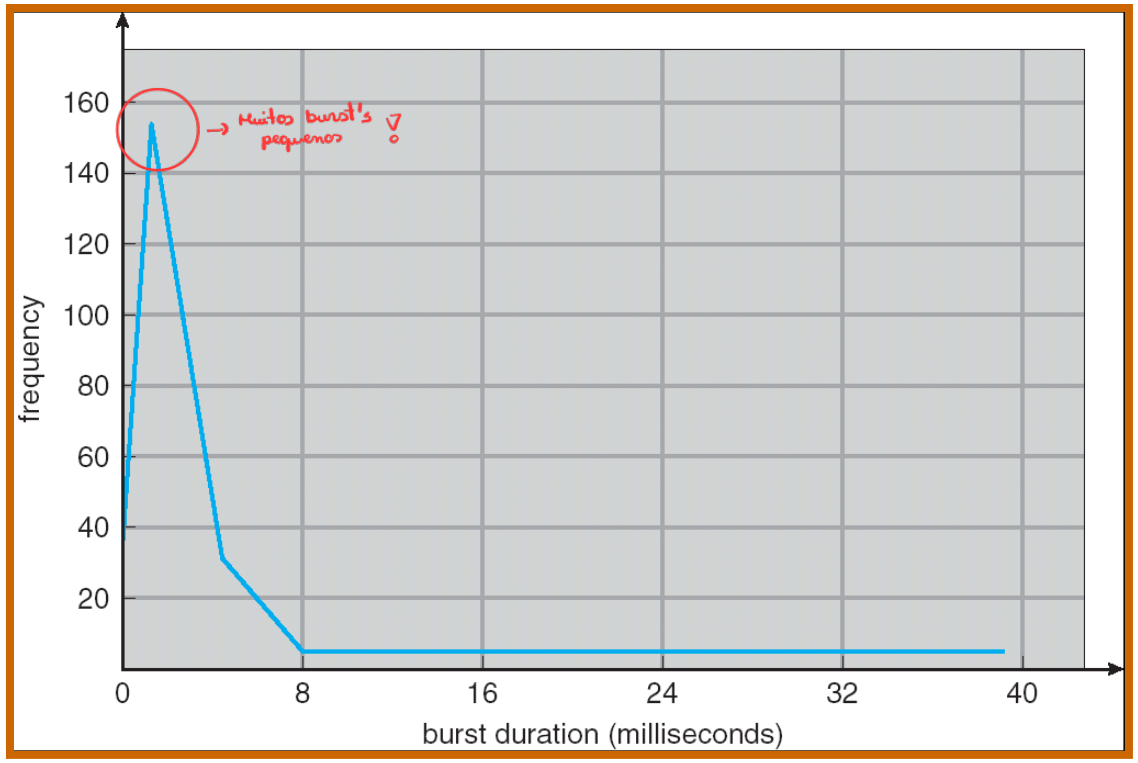
↳ tempo perdido ...

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
    → eles não querem lá estar...
- Tempo de resposta ⊖
  - Tempo entre pedido e primeira resposta (eventualmente parcial) a esse pedido
    ↳ e.g.: clica numa tecla e ela aparecer no ecrã

Execução de um processo





Escalonamento FCFS

=> Pela ordem que aparecem!

- *First-Come, First-Served*

Process	Burst Time
P_1	24 ms
P_2	3 ms
P_3	3 ms

Chegaram todos no início =>

Burst time = Execution time

First-Come, First-Served:

Processos	Burst time
P1	24 ms
P2	3 ms
P3	3 ms

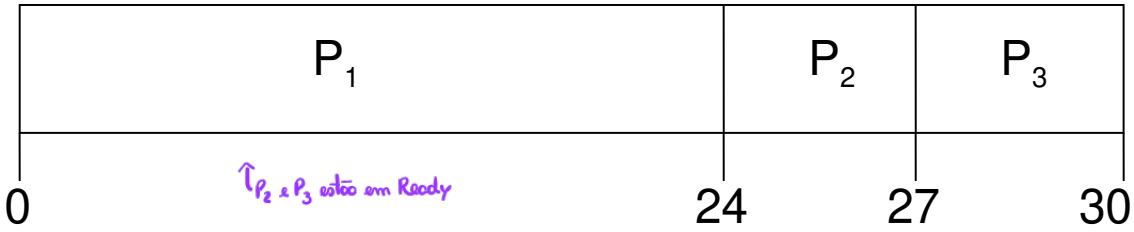
→ 

Tempo de espera: $T_1 = 0$ ms; $T_2 = 24$ ms; $T_3 = 27$ ms

Tempo total de espera: $T_{total} = T_0 + T_1 + T_2 = 51$ ms

Tempo médio de espera: $\frac{51}{3} = 17$ ms

- 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**

Escalonamento FCFS

- 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 !!!** ← Muito menos tempo perdido!

Conclusão:
- O escalonamento "First-Come, First-Served" pode vir a ter muito tempo de espera, dependo da ordem como são executados!

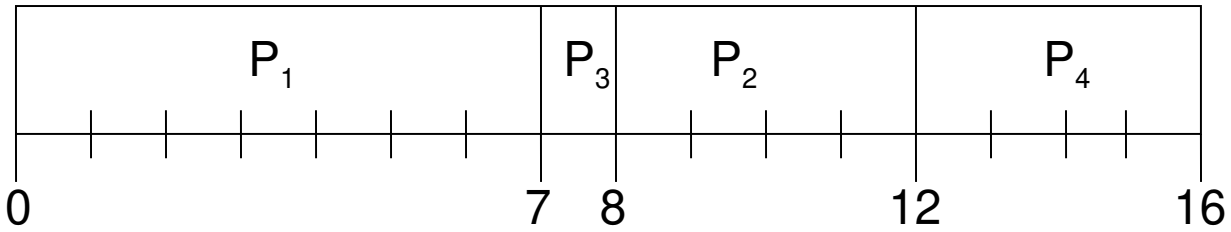
Escalonamento SJF

- *Shortest Job First*
- Ordena os processos considerando a duração do próximo CPU burst. Executa primeiro os processos com CPU burst mais curtos
- Duas opções
  - *Nonpreemptive* – 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-Remaining-Time-First* (SRTF)
- SJF é óptimo do ponto de vista do tempo médio de espera de um conjunto de processos

Escalonamento SJF

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)*

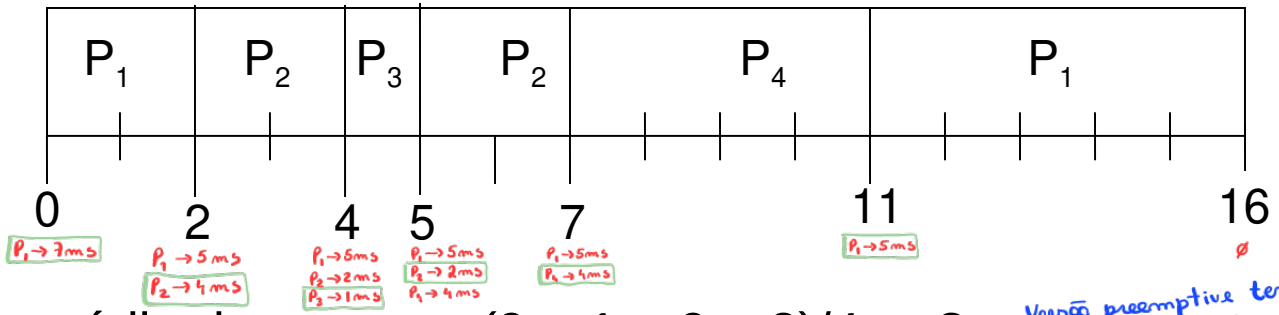


- Tempo médio de espera= (0 + 6 + 3 + 7)/4 = **4**

Escalonamento SJF

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** ⇐ Versão preemptive tem um tempo médio de espera menor

Determinar o tempo do próximo CPU *Burst*

Tudo muito bonito mas...

- Os tempos dos CPU *Burst* não são, em geral, conhecidos
- Solução: tentar obter boas estimativas

- Como?
  - Usar histórico do processo para prever o futuro
  - Exemplo: Média exponencial

- Se $\alpha = 1 \Rightarrow \tau_{n+1} = t_n$ ... deve ser um meio termo
- Se $\alpha = 0 \Rightarrow \tau_{n+1} = \tau_n$

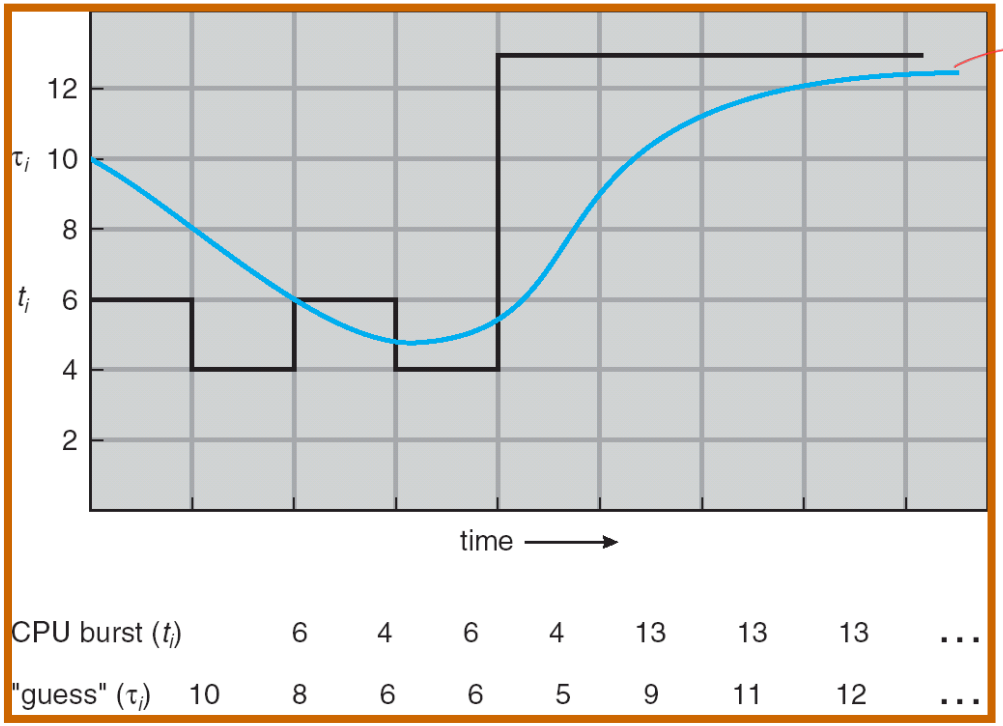
α é alterável!

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n .$$

τ_{n+1}: estimativa do próximo burst; t_n: tempo do burst anterior; τ_n: estimativa do burst anterior

  - t_n é o tempo do último CPU *Burst*, τ é a estimativa do CPU burst

Determinar o tempo do próximo CPU Burst



Escalonamento por prioridades

- *Priority sheduling*
- É associado um nível de prioridade (inteiro) com cada processo
  - Não existe acordo sobre se a prioridade mais alta corresponde a valores baixos ou altos do nível de prioridade
  - Iremos assumir que números baixos representam maior prioridade
- O CPU é atribuido ao processo com maior prioridade
  - Preemptive
  - Nonpreemptive
- SJF é um caso particular de escalonamento por prioridades
- Problema: Adiamento indefinido
  - Processos com prioridade baixa podem nunca executar
- Solução: Contar com o tempo de espera (*aging*)
  - Aumentar a prioridade dos processos em espera à medida que o tempo passa

⇒ No SJF a prioridade depende do Burst time !!!

↳ Se pensarmos bem isto é MUITO importante...
→ tem pouca prioridade mas tem de ser executado!

Round Robin (Muito utilizado)

- Versão *Time sharing* e *preemptive* de FCFS
- 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 lista de Ready
  - *Time quantum* varia, em geral, entre 10 e 100ms
- Se existem n processos na fila de *Ready* (nenhum em execução) e o *time quantum* é q então:
  - cada processo usa cerca de 1/n do processador
  - Um processo nunca espera mais do que (n-1).q unidades de tempo
- Desempenho
  - Q grande ▲ FCFS
  - Q pequeno ▲ o overhead da mudança de contexto pode ser significativo

→ dividimos o CPU por todos os processos (desconsiderando o tempo de mudança)

→ tempo de ele dar a volta e volta a ser executado !//

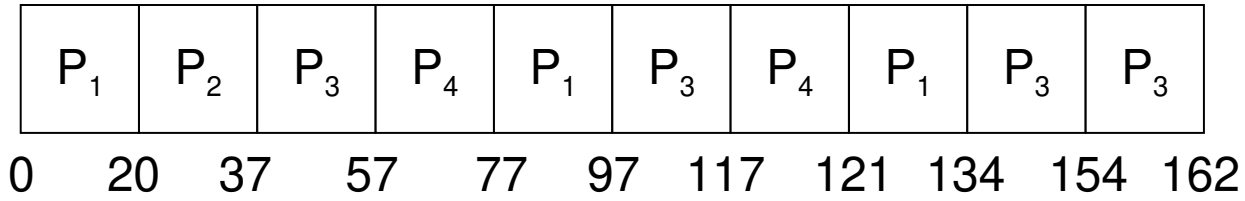
demora a mudar entre Processos => Process Block Control (PCB)

Round Robin com q=20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

- O escalonamento será:

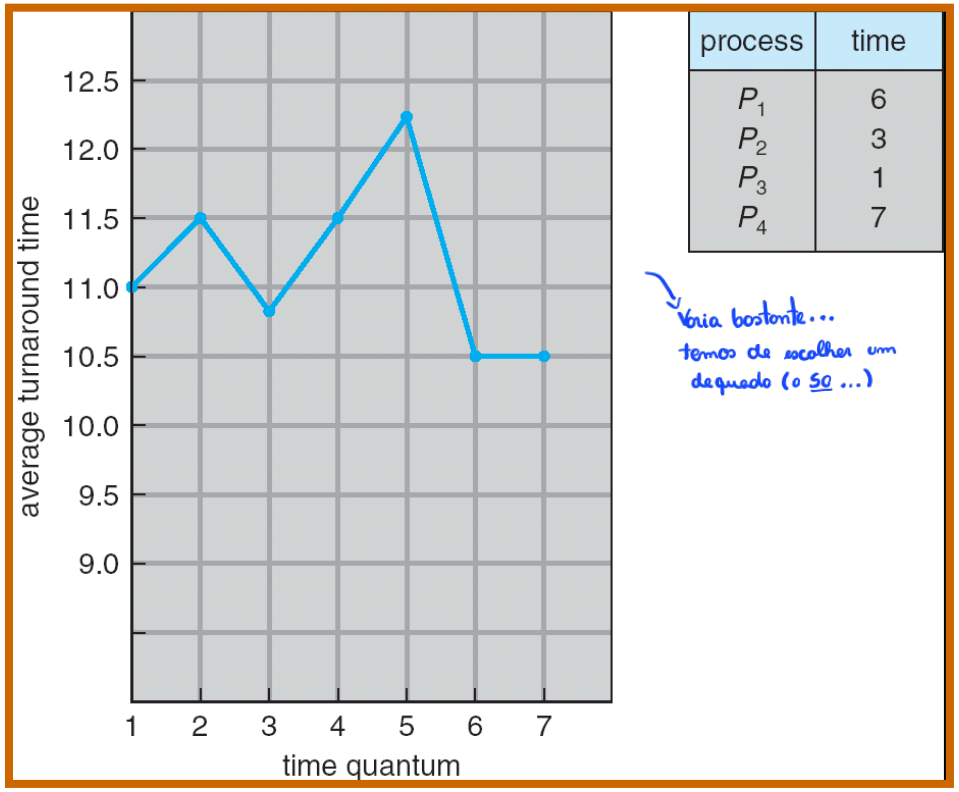
time quantum : 20ms



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

Tempo do processo médio vs. *Time quantum*

Tempo de processo : turnaround time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Varia bastante...
temos de escolher um
dequado (o SO ...)