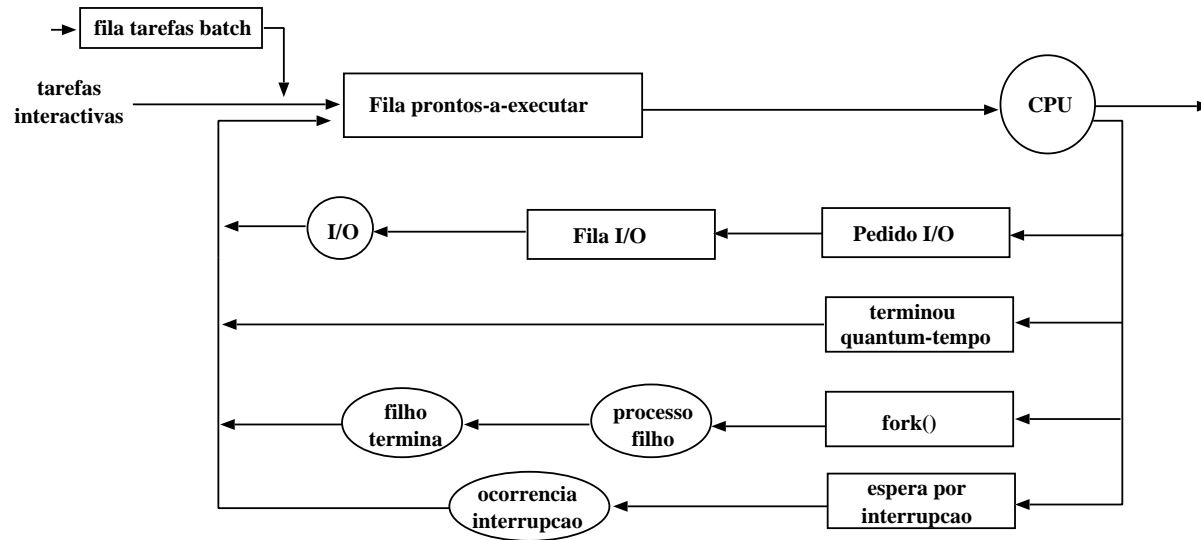


Temporização (Scheduling) de Processos

→ Tem por objetivo maximizar o uso da CPU, i.e. ter sempre um processo a executar.

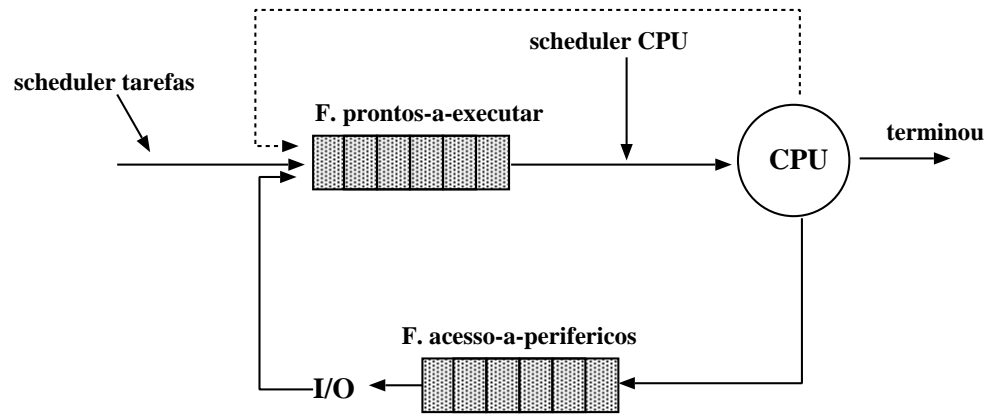
Filas de processos usadas em scheduling:

- *Fila de tarefas*: processos submetidos para execução, à espera de serem carregados para memória. Podemos separar *tarefas batch* de outras.
- *Fila dos prontos-a-executar*: processos já em memória, prontos para executar.
- *Filas de acesso-a-periféricos*: cada periférico tem uma fila de processos à espera de vez de acesso.
- Um processo migra entre as varias filas.



Schedulers (temporizadores)

- scheduler de tarefas (longa-duração): seleciona quais os processos que devem ser carregados para a fila dos prontos-a-executar.
Ativado menos frequentemente (segundos, minutos); rapidez não é crucial.
- scheduler do CPU (curta-duração): seleciona o processo que irá ser executado a seguir e atribui-lhe o CPU.
Ativado com muita frequência (milissegundos), tem de ser muito rápido.



Scheduler de CPU

- Entre os processos que estão em memória prontos-a-executar, seleciona um e atribui-lhe o CPU.
- O scheduler de CPU toma decisões quando um processo:
 1. passa do estado em-execução é suspenso (estado em-espera).
 2. passa do estado em-execução para pronto-a-executar.
 3. passa do estado em-espera para pronto-a-executar.
 4. termina
- Um scheduler diz-se *não-interruptível* (*non-preemptive*) se apenas intervém nas situações 1. e 4.
- Caso contrário, o scheduler diz-se *interruptível* (*preemptive*).

Scheduler de CPU (cont.)

- Dispatcher: módulo que dá o controlo do CPU ao processo seleccionado pelo scheduler; passos envolvidos:
 - troca de contexto de processos
 - passar para modo-utilizador
 - re-iniciar a execução do programa
- *Dispatch Latency* - tempo que o *dispatcher* demora para parar um processo e iniciar outro.

Características de um bom algoritmo de scheduling.

- **Uso Eficiente de CPU** – manter o CPU o mais ocupado possível.
40% = levemente ocupado; 90% = fortemente ocupado.
- **throughput** – nº de processos que terminam a sua execução por unidade de tempo.
- **tempo de turnaround** – qtd. tempo para executar um processo em particular (inclui tempo na fila dos prontos-a-executar, a usar CPU e a realizar I/O).
- **tempo de espera** – qtd. tempo que um processo esteve à espera na fila dos prontos-a-executar.
- **tempo de resposta** – qtd. de tempo entre a submissão de um pedido e a primeira resposta produzida (não é output). Importante para time-sharing.

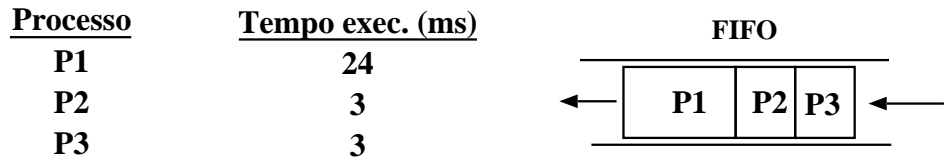
- **equitativo (justo)** – garantir que cada processo obtém a sua parte de CPU (não fica esquecido).

É desejável:

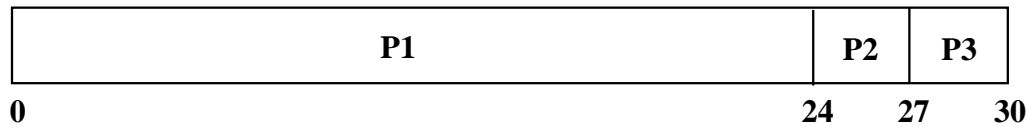
- Maximizar uso de CPU
- Maximizar throughput
- Minimizar tempo de turnaround
- Minimizar tempo de espera
- Minimizar tempo de resposta

Algoritmos de Scheduling

→ 1º a chegar, 1º a ser selecionado (FCFS: First-Come, First-Served), é um algoritmo extremamente simples e fácil de implementar com uma fila (FIFO).



Obdecendo a ordem de chegada dos processos, o diagrama de Gantt para o escalonamento e’:



- Tempo de espera para $P_1 = 0$; $P_2 = 24$; $P_3 = 27$.
- Tempo médio de espera: $(0 + 24 + 27)/3 = 17\text{milisecs}$.

Algoritmos de Scheduling (cont.)

→ Menor tarefa primeiro (SJF: Shortest-job First): o CPU é atribuído ao processo que se pensa demorar menos tempo a executar.

<u>Processo</u>	<u>Tempo chegada</u>	<u>Tempo exec. (ms)</u>
P1	0.0	7
P2	0.2	4
P3	4.0	1
P4	5.0	4

	P1	P3	P2	P4	
0		7	8	12	16

- Tempo de espera para $P_1 = 0$; $P_2 = 8$; $P_3 = 7$; $P_4 = 12$.
- Tempo médio de espera: $(0 + 7 + 8 + 12)/4 = 6.75 \text{ milisecs}$.
- Tempo médio de espera (FCFS): $(0 + 7 + 11 + 12)/4 = 7.5 \text{ milisecs}$.

Menor tarefa primeiro: vantagens e inconvenientes

- é *ótimo* pois dá o menor tempo médio de espera para um dado conjunto de processos.
- *pouco praticável!* Dificuldade em determinar a menor tarefa, pois não é possível saber-se antecipadamente qual o tempo que um dado processo precisa de CPU.
- permite *inanição* de processos (os que têm tempo de execução grande).
- Uma solução (com *preemption*– SRTF: Shortest Remaining Time First) para processos interativos seria usar estimativas do tempo de execução e aplicar uma *técnica de envelhecimento* de modo a que os valores estimados à mais tempo tenham menor peso na nova estimativa.

Técnica de envelhecimento

Considere-se:

$$E_{n+1} = \alpha T_n + (1 - \alpha)E_n = \\ \alpha T_n + (1 - \alpha)\alpha T_{n-1} + (1 - \alpha)^2 \alpha T_{n-2} + \dots + (1 - \alpha)^n E_1$$

onde,

T_i é o tempo de execução estimado para a execução do processo no instante i .

E_i valor previsto para o instante i .

E_1 valor previsto para o instante inicial.

tomando $\alpha = 0.8$ teríamos:

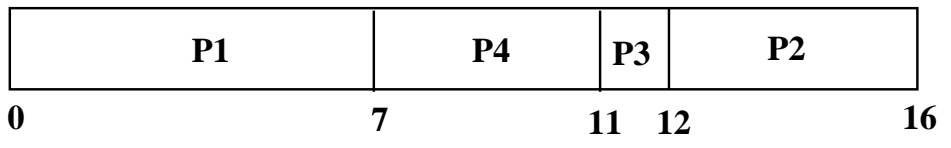
$$E_{n+1} = 0.8T_n + 0.16T_{n-1} + 0.032T_{n-2} + 0.0064T_{n-3} + \dots$$

Quanto mais antiga for a observação menor peso na estimação atual.

Escalonamento com Prioridades

- Neste algoritmo associa-se a cada processo um valor de prioridade. O processo com maior prioridade é escolhido pelo scheduler para aceder ao CPU.
- Valores menores de prioridade \Rightarrow maior prioridade.
- Um exemplo para uma versão não-interruptível do algoritmo:

<u>Processo</u>	<u>Tempo chegada</u>	<u>Tempo exec. (ms)</u>	<u>Prioridade</u>
P1	0.0	7	3
P2	0.2	4	4
P3	4.0	1	3
P4	5.0	4	1



• Tempo médio de espera: $(0 + 7 + 11 + 12)/4 = 7.5 \text{ milisecs}$.

Escalonamento com Prioridades

- *Problema:* pode levar à inanição de processos. Um processo com prioridade baixa pode chegar a não ser escolhido para executar.
- *Solução:* técnica de envelhecimento, em que a prioridade de um processo aumenta com o tempo de espera, acabando por vir a ser selecionado.

Round-Robin (distribuição circular de tempo)

- *time quantum* ou *time-slice* = intervalo mínimo de tempo de CPU atribuído a um processo (10-100 milisegundos).
- a cada processo é atribuído um quantum de tempo para executar. Decorrido esse tempo o processo é interrompido (*preempted*) e adicionado no fim da fila dos prontos-a-executar.
- quando um processo esgota o seu quantum, ou quando bloqueia ou termina a sua execução antes de esgotar o seu quantum, outro processo é escolhido para tomar o seu lugar, normalmente é o que está há mais tempo na fila.
- O quantum de tempo deve ser maior que o que se perde na troca de contexto entre os processos, senão não seria compensador!

Round-Robin (distribuição circular de tempo)

<u>Processo</u>	<u>Tempo chegada</u>	<u>Tempo exec. (ms)</u>	
P1	0.0	7	quantum=3
P2	0.2	4	
P3	4.0	1	
P4	5.0	4	

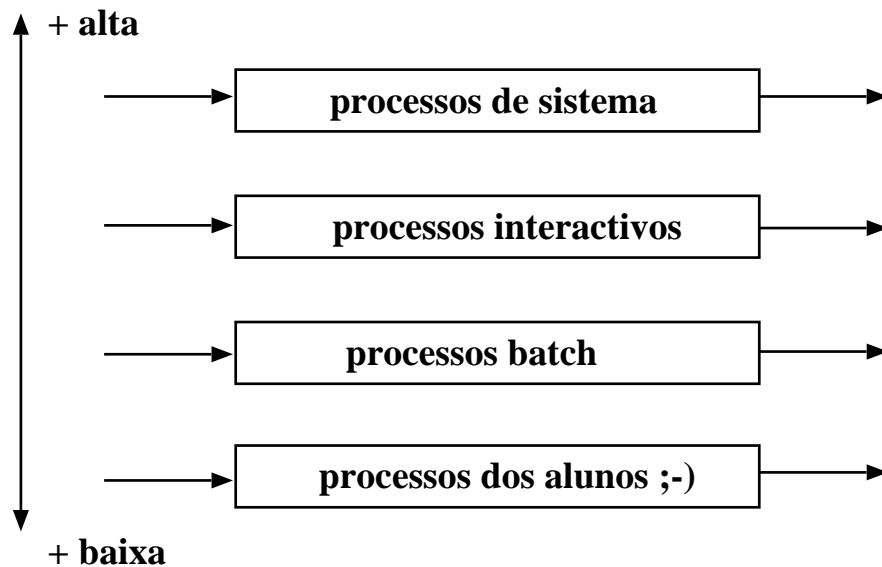
P1	P2	P1	P3	P4	P2	P1	P4	
0	3	6	9	10	13	14	15	16

- Normalmente, tem um tempo médio de espera maior que SJF (SRTF), mas melhor tempo de resposta.

Filas de Níveis Múltiplos

- A fila dos prontos-a-executar é subdividida em várias filas, consoante os requisitos: *foreground* (interativos) ou *background* (batch).
- cada fila tem o seu algoritmo de scheduling, e.g. interativos é RR e os batch (FCFS).

Prioridade



- um scheduler mais geral coordena os schedulers associados a cada fila, atribuindo uma fatia de tempo de CPU para execução de processos de uma dada fila, e.g.

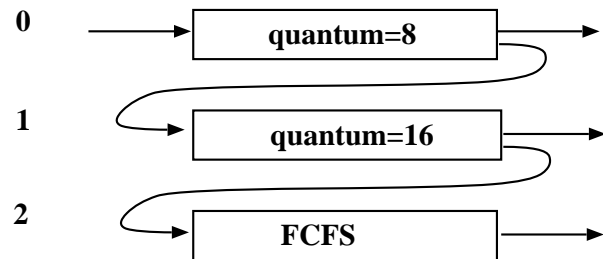
80% do tempo para processos interativos,
20% do tempo para processos batch.

Filas de Níveis Múltiplos com Realimentação

- Estende o algoritmo anterior, permitindo que os processos possam ser deslocados de umas filas para outras, consoante o uso que vão fazendo do CPU.
- Assim se um processo usar demasiado tempo de CPU, é deslocado para uma fila de mais baixa prioridade.
- Processos que estejam há muito tempo em filas de prioridade mais baixa vão sendo deslocados para filas de prioridade mais alta (*técnica do envelhecimento*), evitando-se inanição de processos.

Filas de Níveis Múltiplos com Realimentação

- Parâmetros que determinam este scheduler:
 - número de filas
 - o algoritmo de scheduling de cada fila
 - método para promover um processo
 - método para despromover um processo
 - método para determinar qual a fila inicial onde o processo é colocado.
- Um exemplo:



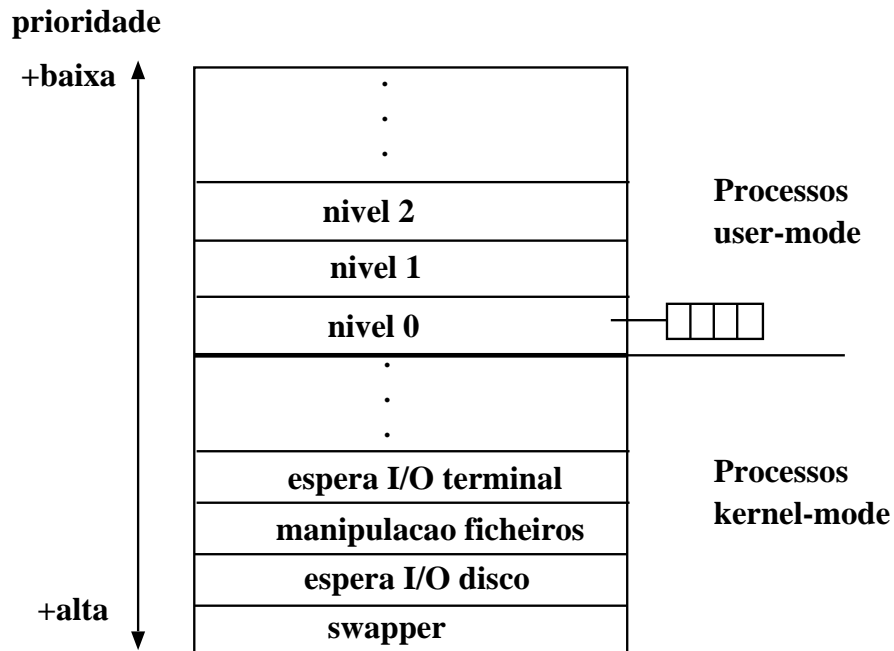
- Este é o algoritmo mais geral (e mais complexo) e pode facilmente ser configurado para um SO em vista.

Scheduling (tradicional) em Unix

Unix é um sistema time-sharing, pelo que o algoritmo procura garantir que processos interativos obtenham um bom tempo de resposta.

O algoritmo de scheduling tem dois níveis:

- *nível-baixo*: escolha do próximo processo a executar; usa filas múltiplas com um valor de prioridade associado a cada fila e um algoritmo Round-Robin dentro de cada fila.
- *nível-mais-alto*: partes dos processos são deslocados entre a memória e o disco, para que todos tenham a chance de estar em memória e serem executados.



- processos em execução em modo kernel têm prioridade mais alta (para que possam deixar o kernel o mais rápido possível!) do que em modo de utilizador.

Scheduling em Unix (cont.)

- as prioridades dos processos prontos-a-executar são atualizadas periodicamente, e.g. cada segundo, o que pode mudar o processo de nível.
- o objetivo é baixar a prioridade dos processos que mais usaram o CPU recentemente e fazer subir a prioridade dos processos que menos usaram o CPU.

Scheduling em Unix (cont.)

- prioridade de um processo P_j varia com um instante i do seguinte modo:

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + nice_j$$

$$CPU_j(i) = \frac{U_j(i)}{2} + \frac{CPU_j(i-1)}{2}$$

onde,

- $P_j(i)$ = prioridade do processo P_j no início do intervalo i ;
- $Base_j$ = prioridade de base do processo P_j ;
- $U_j(i)$ = tempo de CPU usado por P_j no intervalo i ;
- $CPU_j(i)$ = tempo médio de utilização do CPU por P_j até intervalo i ;
- $nice_j$ = valor de ajuste definido pelo utilizador;

- o contador de utilização de CPU para um dado processo é atualizado em cada tick do relógio, quando o processo está em execução.
- os processos interativos têm normalmente um $U_j(i)$ baixo, porque suspendem muito, pelo que a sua prioridade é habitualmente alta.

Exemplo de atualização de prioridades

tempo	P0		P1		P2	
	Pri.	CPU	Pri.	CPU	Pri.	CPU
0	60	0	60	0	60	0
1	75	⋮	60	0	60	0
		60				
2	67	15	75	⋮	60	0
				60		
4	63	7	67	15	⋮	30
					60	
5	76	⋮	63	7	67	15
		67				
6	68	16	76	⋮	63	7
				67		
	68	16	76	33	63	7