Gestão de processos

Programa: entidade estática que corresponde ao código executável de um programa. Um programa em si é apenas código em texto que está armazenado em disco.

Processo: entidade dinâmica que corresponde ao programa em execução. Um processo não é apenas o código do programa mas também todo o ambiente necessário para a sua execução. Um processo contém o *program counter,* conteúdo dos registos do *CPU, stack, heap, etc.*

Nota: duas invocações do mesmo programa resultam em dois processos distintos

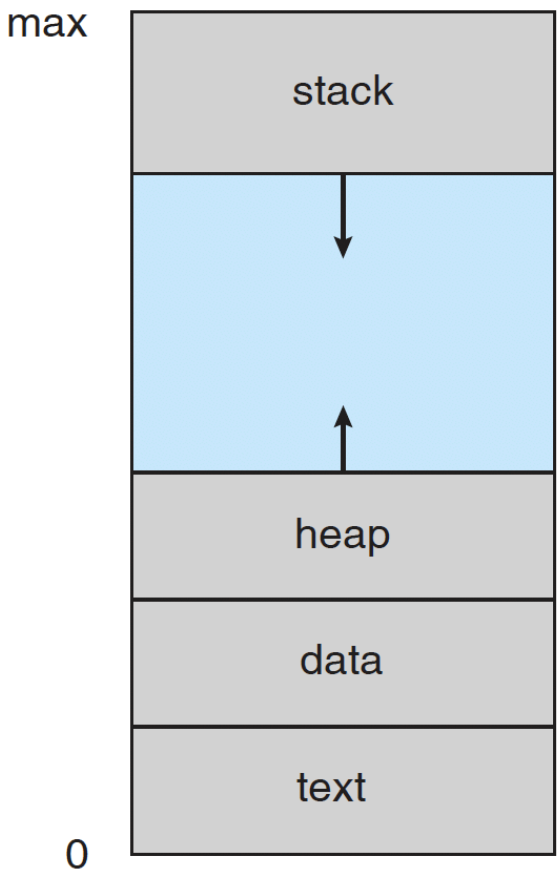
---

À medida que um programa executa vai variando o seu estado, sendo este definido pela sua atividade atual.

Conceitos de processo

O processo

Como já foi referido, um programa não é um processo. Um processo é mais do que o código de um programa, pois necessita de um ambiente onde possa ser executado. Um programa torna-se um processo quando é carregado para memória principal onde o sistema operativo prepara esse ambiente para um processo ser executado.



A imagem acima demonstra o conteúdo de um processo na memória principal:

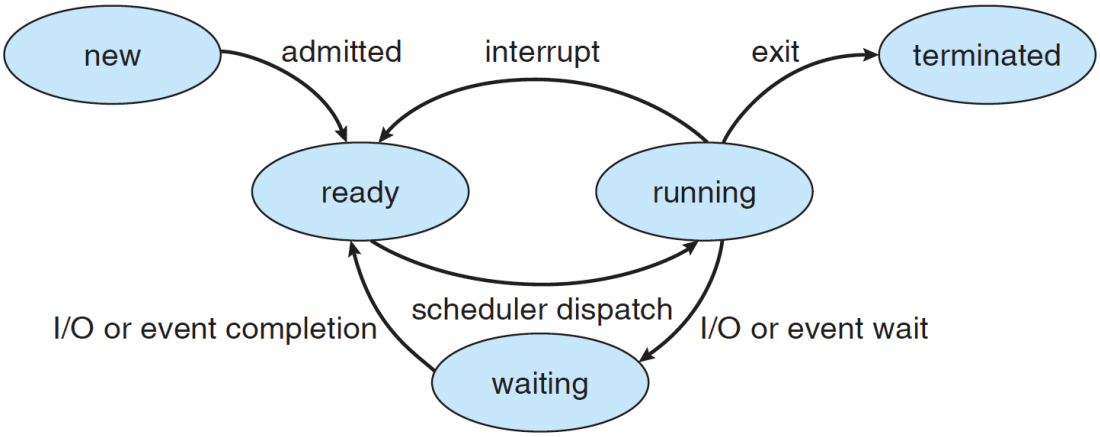
- *Stack:* contém os parâmetros das funções, endereços de retorno, variáveis locais, *etc.*
- *Heap:* memória alocada dinamicamente.
- *Data:* contém variáveis globais, variáveis estáticas, *etc.*
- *Text:* código fonte do programa em execução

Estado de um processo

À medida que um processo executa vai variando de estado sendo os estados possíveis os seguintes:

- *New:* o processo está a ser criado

- *Running:* o processo está em execução
- *Waiting:* o processo está à espera que um evento ocorra, como por exemplo, uma operação de *I/O*.
- *Ready:* o processo está à espera de ser atribuído ao *CPU.*
- *Terminated:* o processo acabou a sua execução



Process Control Block (PCB)

Todos os processo são representados no sistema operativo por um *PCB.* Esta estrutura contém peças de informação associada a um específico processo servindo como um repositório que varia de processo para processo.

O *PCB* contém informação relativa a:

- Estado do processo: pode ser um dos estados referidos anteriormente.
- *Program Counter:* endereço da próxima instrução a ser executada para um determinado processo.
- Registos de *CPU:* variam conforme a arquitectura do computador contém acumuladores, índices dos registos, *stack pointers, etc*. Em conjunto com o *program counter,* esta é a informação que é guardada quando uma interrupção ocorre, para permitir que, quando o processo volta a ter tempo de *CPU,* volte ao estado em que estava.

- Informação sobre o escalonamento de *CPU:* contém informação sobre a prioridade do processo, apontadores para as *queres* de escalonamento, e qualquer outro tipo de informação sobre escalonamento.
- Informação sobre a gestão de memória: pode conter informação sobre os valores da base e limite dos registos, assim como informação sobre a paginação.
- *Accounting information:* tempo de *CPU* usado, tempo real usado, limites de tempo, números do processo, *etc.*
- Informação sobre o estado *I/O:* número de dispositivos *I/O* alocados para o processo, lista de ficheiros abertos, *etc.*

Escalonamento de processos

- Escalonamento cooperativo(*non-preemptive*): uma vez atribuído o *CPU* ao processo, só lhe é retirado se entrar no estado de *wait* ou *terminated.*

- Escalonamento por desafetação forçada(preemptive): o *CPU* é retirado ao processo ao fim de um *time-slice* ou porque surgiu um de maior prioridade.

Deve-se usar desafectação forçada para evitar que interações longas monopolizem o *CPU.* Desta maneira, interações curtas terminam dentro de uma *time-slice* e as interações longas executam durante uma *time-slice* e, após este tempo, passam a estar no estado de *ready* dando lugar a outro processo. Mais tarde será-lhe atribuído novamente uma fatia de tempo, e assim sucessivamente.

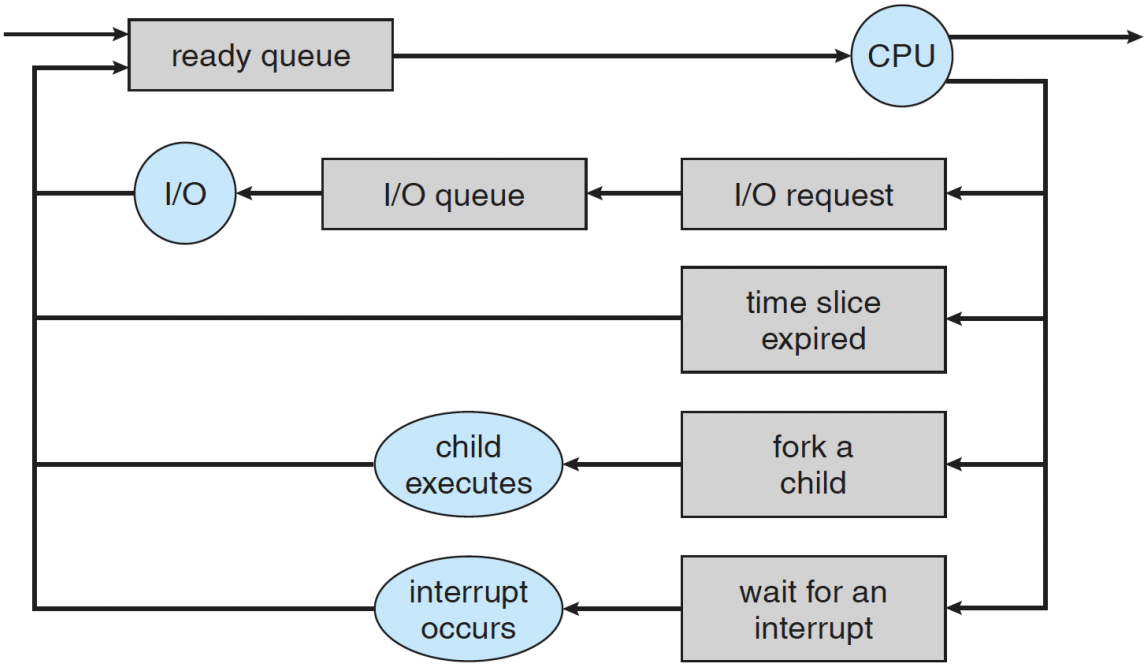
- *Process scheduler:* escolhe um processo, de um conjunto de

processos, para ser executado pelo *CPU.* Num sistema s*ingle-processor,* nunca vai existir mais do que um processo a ser executado ao mesmo tempo. Se existirem mais processos, estes vão ter que esperar até que o *CPU* esteja livre.

Filas de escalonamento

À medida que os processo entram no sistema, são colocados numa *job queue,* que consiste em todos os processos do sistema. Os processos que residem na memória principal e que estão prontos ou à espera para serem executados são guardados numa *ready queue,* geralmente guardada como uma lista ligada onde a cabeça da lista aponta para o primeiro e último *PCB.*

Quando um processo está à espera de uma operação de *I/O,* e uma vez que existem vários processos que possam estar à espera do mesmo, existe também uma fila para os processos que esperam por um determinado dispositivo *I/O,* designada por *divise queue.* Cada dispositivo tem a sua própria fila.



Um processo quando chega à memória principal é colocado na *ready queue* e fica à espera até ser seleccionado para execução. Quando um processo é alocado para o *CPU* podem acontecer os seguintes eventos:

- O processo pode pedir uma operação de *I/O* e é colocado na *I/O queue.*
- O processo pode criar um sub-processo e espera que este acabe.
- O processo pode ser removido forçosamente do *CPU* através de uma interrupção e ser colocado novamente na *ready queue.*

Um processo mantém-se neste ciclo até acabar a sua execução. Uma vez acabada, é removido de todas as filas e o seu *PCB* e recursos desalocados.

Em resumo, as filas e os respetivos escalonadores são:

- *Job queue:* processos à espera de serem escolhidos para irem para memória principal para serem executados.
  - *job scheduler (long-term scheduler) : executado com menos frequência, controla o nível de multiprogramação*
- *Ready queue:* processos em memória principal que estão no estado de *ready* ou *waiting,* à espera que sejam escolhidos para serem executados pelo *CPU.*
  - *cpu scheduler (short-term scheduler) : executado com muita mais frequência, tipicamente uma vez a cada 100ms*
- *Device queue:* processos à espera de serem atendidos por um dispositivo *I/O.*

Existem sistemas, como os *Unix* e *Microsoft Windows,* que não têm o *job-scheduler*, mas têm o designado *medium-term scheduler* que

usa uma técnica designado por *swapping* onde os programas vão entrando e saindo da memória principal

---

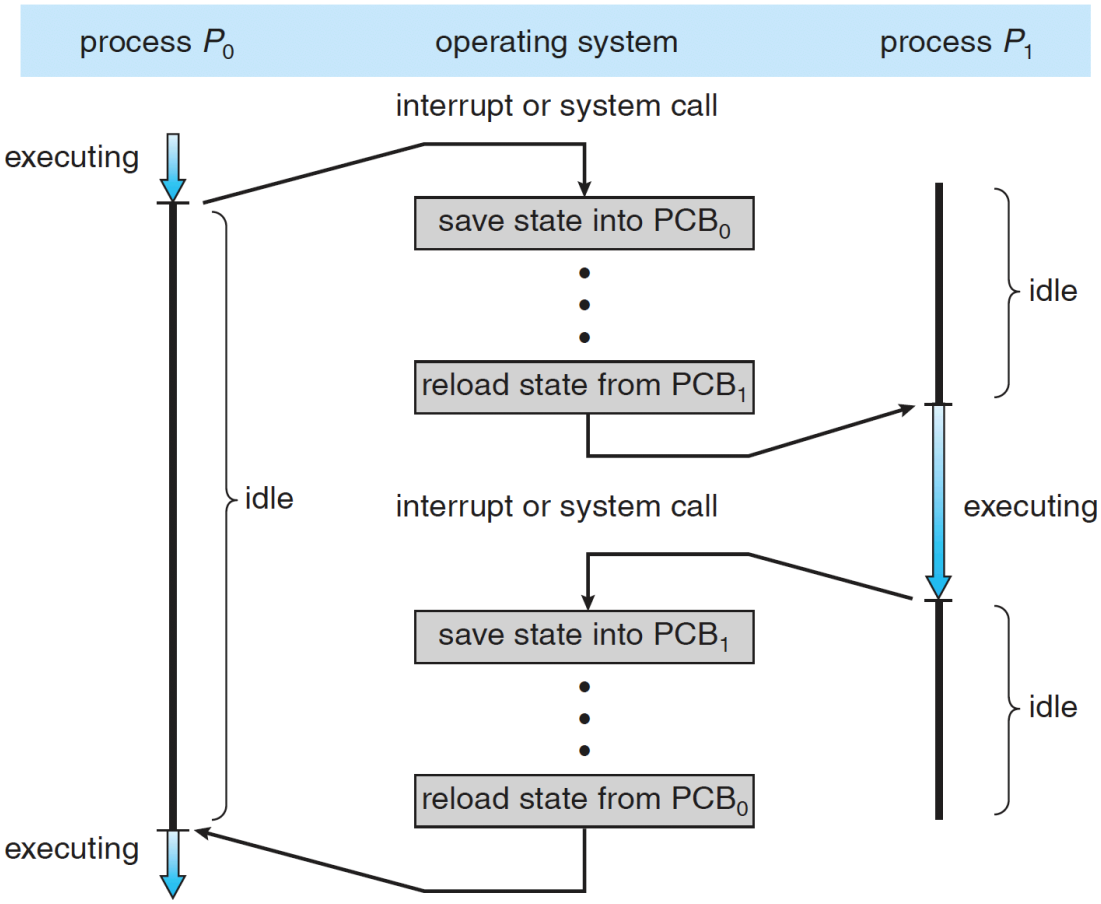
Critérios de escalonamento

Existem vários critérios de escalonamento:

- *IO-Bound / CPU-Bound:* dizem respeito a processos que, respetivamente, precisam de mais tempo de leituras e escritas e processos que precisam mais de poder computacional. O sistema é balanceado se forem escalonados uma mistura dos dois.
- Interativo ou não.
- Urgência de resposta.
- Comportamento recente (utilização de memória, *CPU*)
- Necessidade de periféricos especiais.
- 'Pagou' para ir à frente dos outros.

Mudança de contexto

As interrupções causam o sistema operativo a mudar a tarefa que o *CPU* estava a correr e correr uma rotina do *kernel.* Quando ocorre uma interrupção, o sistema precisa de guardar o contexto atual do processo em execução no *CPU* para depois ser restaurado uma vez que a rotina do *kernel* acabe, ficando o processo que estava a ser executado suspendido. O contexto é guardado no *PCB* do processo que estava a correr para depois ser restaurado. A mudança de contexto é puro *overhead* uma vez que não está a ser feito trabalho útil e a sua velocidade varia de sistema para sistema.



Algoritmos de escalonamento de processos

Os algoritmos de escalonamento de processos têm objectos diferentes, uns pretendem diminuir o tempo de resposta(diminuindo o tempo de espera para determinados processos) e outros tendem maximizar a utilização de *CPU.*

Os algoritmos a estudar são os seguintes:

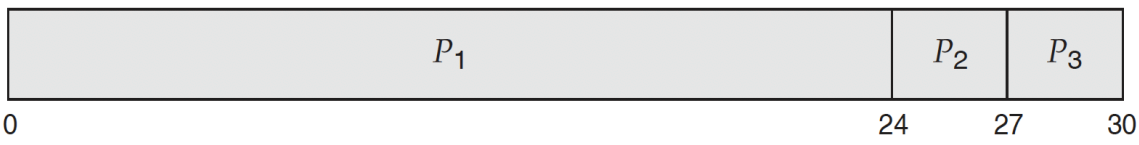
- FCFS - *First Come, First Served*
- SJF - *Shortest Job First*

- SRTF - *Shortest Remaining Time First*
- PPS - *Preemptive Priority Scheduling*
- RR - *Round Robin*
- MFQ - *Multilevel Feedback Queue Scheduling*

Um conceito importante é o de ***CPU-Burst*** que representa o tempo consecutivo que um processo aguenta só a computar dados até necessitar de auxilio de *I/O.*

First-Come, First Served Scheduling

- **Descrição** : O primeiro processo a pedir *CPU* é o primeiro processo a adquirir o mesmo. Quando um processo entra na *ready queue,* o seu *PCB* é colocado na cauda da fila, e então, a *ready queue* funciona como uma *FIFO queue.* Quando o *CPU* está livre, o processo que está à cabeça da fila é alocado.
- **Tipo** : cooperativo
- **Pontos positivos** : código fácil de escrever e de fácil leitura
- **Pontos negativos** : o tempo médio de espera com esta política de escalonamento é demorado. Problemático para sistemas de *time-sharing* e interativos. Tempo de espera com grandes flutuações dependendo da ordem de chegada.
- **Nota** : teria melhores resultados se houverem muitos mais processos *I/O-bounded* do que *CPU-bounded.*



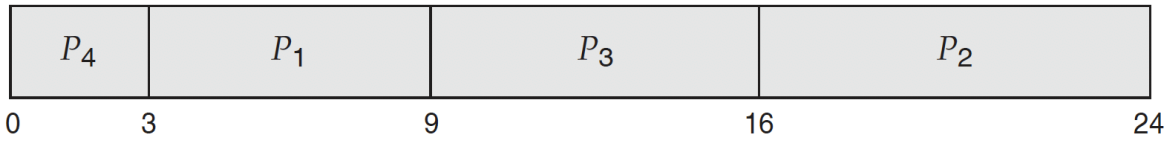
Se a ordem de chegada for P1, P2 e P3, e uma vez que neste algoritmo são executados por ordem, o processo P1 tem tempo de

espera 0, o P2 tempo de espera 24ms e o P3 27ms. O tempo médio de espera é de 17ms. É obvio que se a ordem não for esta e se for, por exemplo, P2, P3 e P1 o tempo médio de espera é de 3ms. No entanto, o tempo médio de espera não tende para valores mínimos.

Shortest-Job-First Scheduling

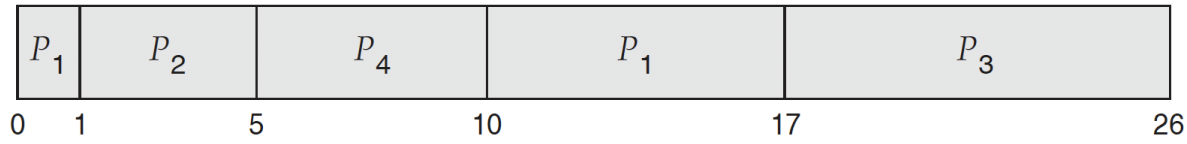
- **Descrição** : A cada processo é associado o tamanho do próximo *CPU-Burst.* Quando o *CPU* está livre é associado ao processo mais curto. Caso haja empate, é usado o *FCFS* para decidir.
- **Tipo** : pode ser cooperativo ou desafectação forçada. A escolha é feita no momento em que o processo chega à *ready queue.* Quando é usado em modo de desafectação forçada é designado por *Shortest-Remaining-Time-First Scheduling.*
- **Pontos positivos** : é provado que seja ótimo, na medida em que fornece o tempo médio de espera mínimo para um dado conjunto de processos.
- **Pontos negativos** : não se consegue adivinhar o tamanho do próximo pedido do *CPU,* apenas se podem fazer estimativas.
- **Nota** : Não é usado no *CPU-Scheduler* pois não há maneira ao certo de saber quanto é o comprimento do próximo *CPU-Burst* mas é possível saber uma aproximação baseada nos valores anteriores. É usado no *Job-scheduler.*

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



Em modo cooperativo, a fila estaria organizada como na imagem acima, por ordem de tempo, e a média de tempo de espera é 3ms enquanto se fosse usado o *FCFS* o tempo médio seria de 10.25ms. Ao mover os processos curtos para a frente dos processos mais longos, o tempo médio de espera baixa.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

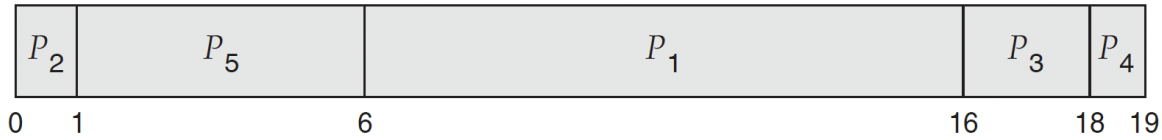


Em modo de desafectação forçada, se aparecer um processo com o tempo mais curto, é retirado o *CPU* ao processo que está a ser executado e atribuído ao de tempo com menor duração.
No exemplo acima, o processo P1 é o primeiro a chegar e como não existe mais nenhum começa a ser executado. No entanto chega um processo P2 com o tempo mais curto do que o tempo atual do processo (que passou para 7 por exemplo), e por isso será atribuído o *CPU* a esse processo. De seguida, são executados por ordem crescente de tempo. O tempo médio de espera, neste caso, é de 6.5ms enquanto que se fosse cooperativo demoraria 7.75ms.

Priority Scheduling

- **Descrição** : A cada processo é associado uma prioridade e o *CPU* é alocado ao processo com maior prioridade. O algoritmo anterior é um caso especial deste algoritmo. Processos com a mesma prioridade são escalonados com o algoritmo *FCFS.*
- **Tipo** : pode ser cooperativo ou desafectação forçada.
- **Pontos negativos** : pode haver *starvation* se um processo com pouca prioridade for sempre ultrapassado por processos com maior prioridade. A técnica para resolver isto designa-se por *aging,* e o objectivo é aumentar a prioridade à medida que o tempo passa.
- **Nota** : As prioridades podem ser definidas interna ou externamente. As internas usam quantidades mensuráveis, como por exemplo, limites de tempo, requisitos de memória, número de ficheiros abertos ou a relação entre *CPU-Burst* e *I/O-Burst.* As externas não são definidas pelo sistema operativo e baseiam-se na importância do processo, ou a quantidade de dinheiro que se está a pagar para o uso de computador.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



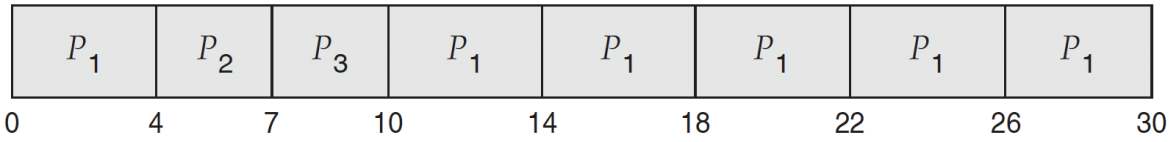
Se os processo P1, …, P5 chegarem todos no instante zero ao sistema, os processos serão escalonados como na imagem acima. Se o algoritmo for cooperativo, acaba o processo que está a executar e,

de seguida, passa para o próximo com mais prioridade e, se entretanto chegar um ao sistema com maior prioridade é posto à cabeça da lista. Caso seja de desafetação forçada, o *CPU* abandona o processo que estava a executar e é alocado de imediato ao de maior prioridade.

Round-Robin Scheduling

- **Descrição** : É semelhante ao *FCFS* mas é *preemptive* para permitir que o sistema troque entre processos. Uma unidade pequena de tempo, designada por *time quantum* ou *time slice,* é definida, normalmente entre os 10-100ms. A *ready queue* é tratada como um fila circular *FIFO*. O *CPU Scheduler* anda à volta da fila alocando o *CPU* para cada processo durante um *time slice.* Novos processos são adicionados à cauda da fila e o escalonador escolhe o processo que está à cabeça da mesma. Pode acontecer uma de duas coisas, ou o processo acaba antes de 1 *time slice* e o processo liberta o *CPU* e é retirado o próximo processo da cabeça da lista, ou o processo não acaba dentro deste intervalo de tempo e uma interrupção é lançada ao sistema operativo. É executada uma troca de contexto, o processo é colocado na cauda da lista e o *CPU* é alocado para o processo que está na cabeça da lista.
- **Tipo** : desafectação forçada
- **Pontos positivos** : Fácil de implementar
- **Pontos negativos** : O tempo de espera é normalmente longo
- **Nota** : Se o *time slice* for muito grande então o algoritmo tem um comportamento semelhante ao *FCFS.* Se for muito pequeno tem-se *overhead* de mudanças de contexto, degradando os níveis de utilização do *CPU.* Cada um dos *n* processos *CPU-Bound* terá *1/n* do tempo disponível no *CPU.*

Process	Burst Time
P_1	24
P_2	3
P_3	3

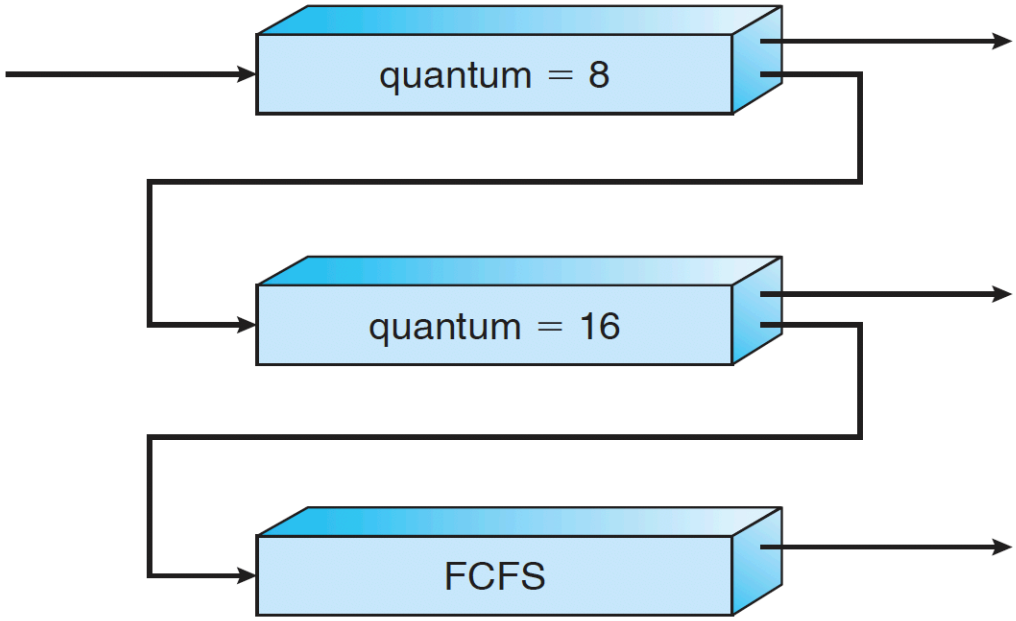


Definindo o *time slice* em 4ms, o processo P1 corre durante 4s, esgotando o tempo sobrando ainda 20ms de execução. O processo P1 é colocado na cauda da fila e, de seguida, à mudança de contexto para o processo P2. Tanto o processo P2 e P3, acabam dentro das *time silves* correspondentes. Por último, o processo P1 é finalizado em 5 *time slices,* mas, como é o único processo que sobra, não há troca de contexto. O tempo médio de espera é de 5.66ms.

Multilevel Feedback Queue Scheduling

- **Descrição** : Classe de algoritmos de escalonamento em que os processos podem ser classificados em diferentes grupos, por exemplo, processos a correr em *background*(*Batch)* ou processos a correr em *forground(interactive).* Parte a *ready queue* em várias filas separadas. A ideia da separação dos processos pelas filas tem como base o *CPU-Burst*. Se um processo necessita de muito tempo de *CPU* é adicionado a uma fila com baixa prioridade, deixando processos interativos e *I/O-Bounded* em filas com maior prioridade. Um processo que esteja muito tempo numa fila com baixa prioridade é movido para uma fila com maior prioridade usando a técnica anteriormente mencionada, *aging.*
- **Tipo** : desafectação forçada ou cooperativo
- **Pontos positivos** : prioridade de processos

- **Pontos negativos** : complexa implementação
- **Nota** : Dá mais prioridade aos processos com *CPU-Burst* <= 8ms



Imaginemos um *multilevel feedback queue scheduler* com três filas numeradas de 0 a 2. O escalonador primeiro executa todos os processos contidas na fila 0. Só quando a primeira fila de encontra vazia é que vai para a segunda e, só quando as duas primeiras se encontram vazias é que vai para a terceira. Um processo que chegue à fila 0 vai antecipar um processo que chegue à fila 1, e um processo que chegue à fila 1 vai antecipar um processo que chegue à fila 2.

Um processo quando chega ao sistema é colocado na fila 0 e é-lhe dado um *time-slice* de 8ms. Se o processo não terminou dentro desse intervalo de tempo é colocado na cauda da fila 1. Se a fila 0 estiver vazia, o *CPU* é alocado ao processo que está à cabeça da fila 1 e é-lhe dado um *time-slice* de 16ms. Se não terminar, é colocado na cauda da fila 2. Os processos na fila 2 são executados com base no *FCFS.*

Tipicamente, este tipo de algoritmos é definido com os seguintes parâmetros:

- Número de filas
- Algoritmo de escalonamento para cada fila
- Método usado para quando um processo sobe onde desce de nível
- Método usado para determinar a fila a que um processo pertence