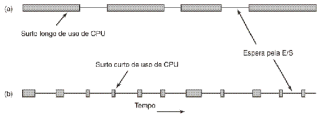
Escalonamento

Geral: A parte do Sistema Operacional (SO) que faz a escolha de qual processo executar é chamada de escalonador, e o algoritmo que ele usa é o *algoritmo de escalonamento.*
Além de escolher o processo certo para executar, o escalonador também deve se preocupar em fazer um uso eficiente da CPU, pois alternar processos é muito caro. De início, deve ocorrer uma alternância do modo de usuário para o modo núcleo. Depois o estado atual do processo deve ser salvo, armazenando-se inclusive seus registradores na tabela de processos, para que possam ser recarregados posteriormente (em muitos sistemas, o mapa de memória também deve ser salvo). Em seguida, um novo processo precisa ser selecionado pela execução do algoritmo de escalonamento. Depois disso, a MMU tem de ser recarregada com o mapa de memória do novo processo. Por fim, o novo processo precisa ser iniciado. Além disso tudo, a alternância do processo normalmente invalida toda a memória cache, forçando-a a ser dinamicamente recarregada da memória principal por duas vezes.
Comportamento dos processos: Em geral, a CPU executa indefinidamente e então é feita uma chamada ao sistema para ler de um arquivo ou escrever nele. Quando a chamada ao sistema termina, a CPU computa novamente até que ela requisite ou tenha de escrever mais dados e assim continua.
O que é importante notar é que alguns processos gastam a maior parte do tempo computando, enquanto outros, passam a maior parte de seu tempo esperando I/O. Isto nós leva a dois tipos de processos:

- ***orientados à CPU:*** Aqueles que passam a maior parte do seu tempo computando. Os processos orientados a CPU apresentam, em geral, longos surtos de uso da CPU e esporádicas esperas por I/O.
- ***orientados a I/O:*** Passam grande parte do seu tempo esperando I/O. Os processos orientados a I/O têm pequenos surtos de uso da CPU e esperas frequentes por I/O.



Note que o fator principal é o tamanho do surto de CPU, não o tamanho do surto de I/O. Convém observar que, à medida que as CPUs se tornam mais rápidas, os processos tendem a ficar mais orientados a I/O. Este efeito ocorre por que as CPUs estão ficando muito mais rápidas.
Quando escalonar: Segue alguns momentos que o escalonamento deve ocorrer:

1. Quando se cria um novo processo, é necessário tomar uma decisão entre executar o processo pai ou o processo filho. Como ambos os processos estão no estado pronto, essa é uma decisão normal de escalonamento e pode levar à escolha de um ou de outro – isto é, o escalonador pode escolher legitimamente executar o pai ou o filho.
2. Uma decisão de escalonamento deve ser tomada ao término de um processo.
3. Quando ocorre uma interrupção de I/O, sobre um semáforo ou por alguma outra razão, outro processo precisa ser selecionado para executar.
4. Quando ocorre uma interrupção de I/O, pode-se tomar uma decisão de escalonamento.

Uma decisão de escalonamento podem ser divididos em duas categorias quanto ao modo como tratam essas interrupções:

- ***Não preemptivo:*** escolhe um processo para executar e então o deixa executar até que seja bloqueado (à espera de I/O ou de um outro processo) ou até que ele voluntariamente libere a CPU. Mesmo que ele execute por horas, não será compulsoriamente suspenso.
- ***Preemptivo:*** escolhe um processo e o deixa em execução por um tempo máximo fixado. Se ainda estiver executando ao final desse intervalo de tempo, o processo será suspenso e o escalonador escolherá outro processo para executar.

Categorias de algoritmo de escalonamento: O que deve ser otimizado pelo escalonador não é o mesmo para todos os sistemas. Três ambientes merecem distinção:

- ***Lote:*** Neste sistema não há, em seus terminais, usuários esperando impacientes por uma resposta rápida. Consequentemente, os algoritmos não preemptivos ou preemptivos com longo intervalo de tempo para cada processo são em geral aceitáveis. Essa tática reduz a alternância entre processos e assim melhora o desempenho.
- ***Interativo:*** A preempção é essencial para evitar que um processo se aposse da CPU e com isso negue serviço aos outros.
- ***Tempo real:*** A preempção é, estranhamente, desnecessária algumas vezes, pois os processos sabem que não podem executar por longos períodos e em geral fazem seus trabalhos e bloqueiam rapidamente. A diferença com relação aos sistemas interativos é que os de tempo real executam apenas programas que visam ao progresso da aplicação.

Objetivos do algoritmo de escalonamento: Alguns objetivos dependem do ambiente (lote, interativo ou tempo real), mas há também aqueles que são desejáveis para todos os casos. Em qualquer circunstância, justiça é importante. Processos semelhantes devem ter serviços semelhantes.
De alguma maneira relacionada à justiça está o cumprimento das políticas do sistema.
Se a CPU e os demais dispositivos de I/O puderem ser mantidos em execução o tempo todo, mais trabalho por segundo será feito do que se algum dos componentes estiver ocioso. Em um sistema em lote, por exemplo, o escalonador tem o controle de quais *jobs* são trazidos para a memória para executar. É melhor manter o sistema todo executando de uma vez formulando-se cuidadosamente essa mistura de processos.

- ***Todos os sistemas***
  - Justiça – dar a cada processo uma porção justa da CPU.
  - Aplicação da política – verificar se a política estabelecida é cumprida.
  - Equilíbrio – manter ocupada todas as partes do sistema.
- ***Sistemas em lote***
  - Vazão (*throughput*) – maximizar o número de *jobs* por hora.
  - Tempo de retorno – minimizar o tempo entre a submissão e o término.
  - Utilização da CPU – manter a CPU ocupada o tempo todo.
- ***Sistemas interativos***
  - Tempo de resposta – responder rapidamente às requisições.
  - Proporcionalidade – satisfazer às expectativas dos usuários.
- ***Sistemas de tempo real***
  - Cumprimento dos prazos – evitar a perda de dados
  - Previsibilidade – evitar a degradação da qualidade em sistemas multimídia

Escalonamento em sistemas em lote

Primeiro a chegar, primeiro a ser servido: Com esse algoritmo, a CPU é atribuída aos processos na ordem em que eles a requisitam. Basicamente, há uma fila única de processos prontos. Quando o primeiro *job* entra no sistema, logo quando chega de manhã, é iniciado imediatamente e autorizado a executar por quanto tempo queira. A medida que chegam os outros *jobs*, eles são encaminhados para o fim da fila. Quando o processo em execução é bloqueado, o primeiro processo na fila é o próximo a executar. Quando um processos bloqueado fica pronto, ele é posto no fim da fila.
A grande desvantagem reside em um possível processo que consome muito tempo de CPU enquanto outros precisam de muito I/O.
Job mais curto primeiro: Quando vários *jobs* igualmente importantes estiverem postados na fila de entra à espera de serem iniciados, o escalonador escolhe o ***job mais curto primeiro***.

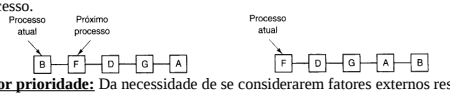
Próximo de menor tempo restante: Uma versão preemptiva do *job* mais curto primeiro é o próximo de menor tempo restante (*shortest remaining time next*). Com esse algoritmo, o escalonador sempre escolhe o processo cujo tempo de execução restante seja o menor. Novamente, o tempo de execução deve ser previamente conhecido.
Escalonamento em três níveis: à medida que chegam ao sistema, os *jobs* são inicialmente colocados em uma fila de entrada armazenada em disco. O escalonador de admissão decide qual *job* será admitido no sistema. Os outros são mantidos na fila de entrada até que sejam selecionados. Um algoritmo típico para controle de admissão pode ser pensado como uma mescla de *jobs* orientados a computação e *jobs* orientados a I/O. Por outro lado, *jobs* curtos poderiam ser admitidos mais rapidamente, ao passo que os *jobs* mais longos teriam de esperar.
Assim que um *job* é admitido no sistema, um processo pode ser criado para ele, que passa então a competir pela CPU. Contudo, pode acontecer de o número de processos ser tão grande que não haverá lugar suficiente para todos na memória. Nesse caso, alguns dos processos devem ser levados para o disco (*swapped out*). O segundo nível de escalonamento é que decide quais processos deverão ser mantidos na memória e quais permanecerão no disco. Chamaremos esse escalonador de ***escalonador de memória***, pois ele determina quais processos ficarão na memória ou permanecerão no disco.
Essa decisão precisa ser revista com frequência para permitir que os processos em disco obtenham algum serviço. No entanto, como trazer o processo do disco é caro, essa revisão não ocorrerá, provavelmente, mais que uma vez a cada segundo, talvez muito menos que isso. Se o conteúdo da memória principal estiver muito desorganizado, uma grande parcela da largura de banda de disco será gasta, tornando a I/O de arquivos lenta.
Para otimizar o desempenho do sistema como um todo, o escalonador de memória deve decidir, com parcimônia, quantos processos ele quer manter na memória (ou seja, o ***grau de multiprogramação***) e que tipo de processos.
Segue alguns dos critérios para a tomada de decisão:

- Há quanto tempo o processo passou por uma troca entre o disco e a memória (*swapped out*)?
- Quanto tempo de CPU o processo teve da última vez?
- Qual o tamanho do processos? (Os pequenos não entram)
- Qual a importância do processo?

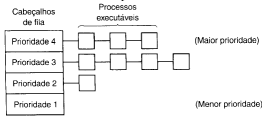
O terceiro nível é chamado de ***escalonador de CPU*** e é nele que se associa o 'escalonador'.

Escalonamento em sistemas interativos

Escalonamento por alternância circular (*round-robin*): A cada processo é atribuído um intervalo de tempo, o seu ***quantum***, no qual ele é permitido executar. Se, ao final do quantum o processo estiver ainda executando, a CPU sofrerá preempção e será dada a outro processo. Se o processo foi bloqueado ou terminou antes que o quantum tenha decorrido, a CPU é alternada para outro processo. O escalonador só precisa manter uma lista de processos executáveis.
O que interessa para o escalonamento circular é o tamanho do quantum, pois a alternância de um processo para outro requer uma certa quantidade de tempo para sua administração.
Vale observar também que se o quantum for maior que o surto médio de CPU, a preempção raramente ocorrerá. Na verdade, a maior parte dos processos bloqueará antes que o quantum acabe, causando uma alternância de processo.



Escalonamento por prioridade: Da necessidade de se considerarem fatores externos resulta o ***escalonamento por prioridade***. A ideia básica é simples: a cada processo é atribuída uma prioridade e ao processo executável com a prioridade mais alta é permitido executar.
Para evitar que processos de alta prioridade executem indefinidamente, o escalonador pode reduzir a prioridade do processo em execução a cada tique de relógio (isto é, a cada interrupção do relógio). Se isso fizer com que sua prioridade caia abaixo da prioridade do próximo processo com prioridade mais alta, então ocorrerá uma alternância de processo. Prioridades podem ser atribuídas ou estaticamente ou dinamicamente aos processos. O sistema também pode atribuir dinamicamente as prioridades para atingir certos objetivos.
Fazer o processo orientado a I/O esperar um longo tempo pela CPU significa tê-lo ocupado a memória desnecessariamente por tempo demais. Um algoritmo simples e que funciona bem para processos orientados a I/O é atribuir $1/f$ à prioridade, sendo f a fração do último quantum que o processo usou.
Muitas vezes é conveniente agrupar processos em classes de prioridade e usar o escalonamento por prioridade entre classes – contudo, dentro de cada classe, usar o escalonamento circular. Enquanto houver processos executáveis na classe de prioridade 4, execute apenas um por quantum em alternância circular e nunca perca tempo com classes de baixa prioridade.



Filas múltiplas: Definir classes de prioridade. Os processos na classes de prioridade mais alta eram executados por um quantum. Os processos na classe seguinte de prioridade mais alta executavam por dois quantum... Se um processo terminasse, todos os quanta a ele alocados iriam para uma classe abaixo. Além disso, à mediada que o processo se aprofundasse mais nas filas de prioridade, ele seria cada vez menos frequentemente executado, liberando a CPU para processos interativos e rápidos.
Próximo processo mais curto *(shortest process next)*: Processos interativos geralmente seguem o padrão de esperar por comando, executar comando... Se víssemos a execução de cada comando como um '*job*' isolado, então poderíamos minimizar o tempo de resposta geral executando o *job mais curto primeiro.* O único problema é saber qual dos processos atualmente executáveis é o mais curto. Uma saída é realizar uma estimativa com base no comportamento passado e, então, executar o processo cujo tempo de execução estimado seja o menor. A técnica de estimar o valor seguinte da série, tomando a média ponderada do valor sendo medido e a estimativa anterior, é algumas vezes chamada de ***aging***.
Escalonamento garantido: Fazer promessas reais sobre o desempenho aos usuários e então satisfazê-los. O sistema deve manter o controle da quantidade de CPU que cada processo recebe desde sua criação. Ele então calcula a quantidade de CPU destinada a cada um ou simplesmente o tempo desde a criação dividido por n.
Escalonamento por loteria: A ideia básica é dar bilhetes de loteria aos processos, cujo prêmios são vários recursos do sistema, como tempo de CPU. Se houver uma decisão de escalonamento, um bilhete de loteria será escolhido aleatoriamente e o processo que tem o bilhete conseguirá o recurso. "Todo os processos são iguais, mas alguns são mais iguais que os outros". Aos processos mais importantes podem ser atribuídos bilhetes extras para aumentar suas probabilidades de vitória. Se houver cem bilhetes extras e um processo detiver vinte deles, esse processo terá uma chance de 20% de vencer cada loteria.
Se aparece um novo processo e a ele são atribuídos alguns bilhetes, já no próximo sorteio da loteria sua probabilidade de vencer, será proporcional ao número de bilhetes que ele tiver. Em outra palavras, o escalonamento por loteria é altamente responsivo. Os processos cooperativos podem trocar bilhetes entre si, se assim desejarem.
Escalonamento por fração justa (fair-share): Alguns sistemas consideram a propriedade do processo antes de escaloná-lo. Nesse modelo, a cada usuário é alocada fração da CPU, e o escalonador escolhe os processos de modo que garanta essa fração.

Escalonamento em sistemas de tempo real

Tempo real é aquele no qual o tempo de uma função essencial. Em geral, um ou mais dispositivos físico externos ao computador geram estímulos, e o computador deve reagir apropriadamente a eles dentro de um dado intervalo de tempo. Sistemas de tempo real são em geral categorizados como *tempo real crítico*, isto é, há prazos absolutos que devem ser cumpridos ou, então, como ***tempo real não crítico***, no qual o descumprimento ocasional de um prazo é indesejável, contudo tolerável. Quando é detectado um evento externo, o trabalho do escalonador é escalonar os processos de tal maneira que todos os prazos sejam cumpridos. Os eventos aos quais um sistema de tempo real pode precisar responder podem ser categorizados como ***periódicos*** (ocorrem em intervalos regulares) ou ***aperiódicos*** (acontece de modo imprevisível). Os algoritmos de escalonamento de tempo real podem ser estáticos ou dinâmicos. Os primeiros tomam suas decisões de escalonamento antes de o sistema começar a executar. Os últimos o fazem em tempo de execução.