O **escalonamento de processos** ou **agendador de tarefas** (em inglês *scheduling*) é uma atividade organizacional feita pelo escalonador (*scheduler*) da [CPU](https://pt.wikipedia.org/wiki/CPU) ou de um [sistema distribuído](https://pt.wikipedia.org/wiki/Sistema_distribu%C3%ADdo), possibilitando executar os [processos](https://pt.wikipedia.org/wiki/Processo_(inform%C3%A1tica)) mais viáveis e concorrentes, priorizando determinados tipos de processos, como os de [I/O Bound](https://pt.wikipedia.org/wiki/I/O_Bound) e os [CPU Bound](https://pt.wikipedia.org/wiki/CPU_bound).

O escalonador de processo é um processo que deve ser executado quando da mudança de contexto (*troca de processo*), ao passo que ele escolhe o processo que será executado pela CPU, sendo o escalonamento realizado com o auxílio do hardware.

São utilizados algoritmos de escalonamento para determinar qual processo será executado em determinado momento e por quanto tempo.

O escalonador de processos de 2 níveis escolhe o processo que tem mais prioridade e menos tempo e coloca-o na memória principal, ficando os outros alocados em disco; com essa execução o processador evita ficar ocioso.

O escalonador deve ainda se preocupar com a eficiência da CPU, pois o chaveamento de processos é complexo e custoso, uma vez que ele afeta o desempenho do sistema e por sua vez a satisfação do usuário.

**Índice**

* [1](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Tipos_b%C3%A1sicos)
* [Tipos básicos](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Tipos_b%C3%A1sicos)
  + [1.1](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_curto_prazo)
  + [Escalonador de curto prazo](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_curto_prazo)
  + [1.2](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_m%C3%A9dio_prazo)
  + [Escalonador de médio prazo](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_m%C3%A9dio_prazo)
  + [1.3](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_longo_prazo)
  + [Escalonador de longo prazo](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonador_de_longo_prazo)
* [2](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Defini%C3%A7%C3%A3o)
* [Definição](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Defini%C3%A7%C3%A3o)
* [3](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Objetivos_do_Escalonamento)
* [Objetivos do Escalonamento](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Objetivos_do_Escalonamento)
* [4](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Quando_%C3%A9_necess%C3%A1rio_o_uso_do_algoritmo_de_escalonamento?)
* [Quando é necessário o uso do algoritmo de escalonamento?](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Quando_%C3%A9_necess%C3%A1rio_o_uso_do_algoritmo_de_escalonamento?)
* [5](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Algoritmos_de_escalonamento)
* [Algoritmos de escalonamento](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Algoritmos_de_escalonamento)
* [6](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonamento_em_Sistemas_em_Batch)
* [Escalonamento em Sistemas em Batch](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Escalonamento_em_Sistemas_em_Batch)
* [7](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Estados_de_processos)
* [Estados de processos](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Estados_de_processos)
  + [7.1](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Diagrama_de_Estados_de_Processos)
  + [Diagrama de Estados de Processos](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Diagrama_de_Estados_de_Processos)
* [8](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Distribui%C3%A7%C3%A3o_de_Prioridades)
* [Distribuição de Prioridades](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Distribui%C3%A7%C3%A3o_de_Prioridades)
  + [8.1](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Alterando_prioridades_no_Windows)
  + [Alterando prioridades no Windows](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Alterando_prioridades_no_Windows)
* [9](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Trocas_de_contexto)
* [Trocas de contexto](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Trocas_de_contexto)
* [10](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Threads)
* [Threads](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Threads)
* [11](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Refer%C3%AAncias)
* [Referências](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Refer%C3%AAncias)
* [12](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Liga%C3%A7%C3%B5es_externas)
* [Ligações externas](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#Liga%C3%A7%C3%B5es_externas)

## Tipos básicos

### **Escalonador de curto prazo**

Seleciona entre os processos em estado de pronto que estão na [memória](https://pt.wikipedia.org/wiki/Mem%C3%B3ria), para serem executados pelo [processador](https://pt.wikipedia.org/wiki/Processador), após a interrupção de um ciclo, uma interrupção de E/S, uma [chamada de sistema](https://pt.wikipedia.org/wiki/Chamada_de_sistema) ou outra forma de sinal. Assim o escalonador de curto prazo faz decisões de escalonamento muito mais frequentemente que os de médio e longo prazo. Uma decisão de escalonamento deve ser feita no mínimo a cada fatia de tempo, e estas são bem curtas.

### **Escalonador de médio prazo**

Seleciona entre os processos que estão na [memória virtual](https://pt.wikipedia.org/wiki/Mem%C3%B3ria_virtual). Ele temporariamente remove o processo da [memória principal](https://pt.wikipedia.org/wiki/Mem%C3%B3ria_principal) e o coloca na memória secundária ([swap](https://pt.wikipedia.org/w/index.php?title=Swapping&action=edit&redlink=1)) fazendo as operações de *swapping in* e *swapping out*. O escalonador a médio prazo pode realizar a operação swap out em vários casos, como: um processo que não está mais ativo após um tempo, um processo que tem baixa prioridade, um processo que tem tido [falta de página](https://pt.wikipedia.org/wiki/Falta_de_p%C3%A1gina) frequentemente, um processo que está ocupando uma larga quantidade de memória que precisa ser removido da memória principal para outros processos serem carregados.

### **Escalonador de longo prazo**

Seleciona os processos que estão na memória secundária e que serão levados para a memória principal. Isto é, quando uma tentativa é feita de executar um programa, sua admissão ao conjunto de processos sendo executados é autorizada ou atrasada pelo escalonador de longo prazo. Assim, este escalonador dita quais processos serão executados em um sistema, e o degrau de concorrência do sistema, ou seja quantos processos serão executados em concorrência, e como a divisão entre processos I/O bound(orientado à E/S) e CPU bound(orientados à CPU) deve ser feita. Geralmente este escalonador é responsável pelo Grau de Multiprogramação.

## Definição

Para que a [CPU](https://pt.wikipedia.org/wiki/CPU) não fique muito tempo sem executar [tarefa](https://pt.wikipedia.org/w/index.php?title=Tarefa&action=edit&redlink=1) alguma, os sistemas operacionais utilizam técnicas para escalonar os processos que estão em execução na máquina.

O escalonamento de processos é uma atividade complicada já que nenhum algoritmo é totalmente eficiente e a prova de falhas, principalmente em se tratando de sistemas interativos, como o [Microsoft Windows](https://pt.wikipedia.org/wiki/Windows). Nos sistemas interativos a interação com o usuário é fundamental já que quem o utiliza busca respostas rápidas e, a todo o momento, [processos](https://pt.wikipedia.org/wiki/Processo) são interrompidos pelo usuário.

Escalonamento é muito atuante nos sistemas em tempo real, onde o tempo é um fator extremamente crítico, como: aviões, hospitais, usinas nucleares, bancos, multimídia, etc. Em ambientes como estes, quando um sistema tem respostas com atraso, é tão ruim quanto não obter. Tipos de sistemas em tempo real:

* Hard Real Time onde atrasos não são tolerados (aviões, usinas nucleares, hospitais);
* Soft Real Time quando atrasos são tolerados (Bancos, Multimídia).

O escalonador do SO utiliza alguns critérios de escalonamento, como:

* a taxa de utilização de CPU, que é a fração de tempo durante a qual ela está sendo ocupada;
* [*throughput*](https://pt.wikipedia.org/wiki/Throughput) que são números de processos terminados por unidade de tempo;
* *turnaround* que é o tempo transcorrido desde o momento em que o software entra e o instante em que termina sua execução;
* tempo de resposta: intervalo entre a chegada ao sistema e inicio de sua execução;
* tempo de espera: soma dos períodos em que o processo estava no seu estado pronto.

Os responsáveis por essa tarefa são algoritmos de escalonamento. Os sistemas operacionais utilizam combinações deles para melhor escalonar os processos.

## Objetivos do Escalonamento

O projeto de um escalonador adequado deve levar em conta uma série de diferentes necessidades, ou seja, o projeto de uma política de escalonamento deve contemplar os seguintes objetivos[[*carece de fontes*](https://pt.wikipedia.org/wiki/Wikip%C3%A9dia:Livro_de_estilo/Cite_as_fontes)]:

* Ser justo: Todos os processos devem ser tratados igualmente, tendo possibilidades idênticas de uso do processador, devendo ser evitado o adiamento indefinido.
* Maximizar a produtividade ([throughput](https://pt.wikipedia.org/wiki/Throughput)): Procurar maximizar o número de tarefas processadas por unidade de tempo.
* Ser previsível: Uma tarefa deveria ser sempre executada com aproximadamente o mesmo tempo e custo computacional.
* Minimizar o tempo de resposta para usuários interativos.
* Maximizar o número possível de usuário interativos.
* Minimizar a sobrecarga ([overhead](https://pt.wikipedia.org/w/index.php?title=Overhead&action=edit&redlink=1)): Recursos não devem ser desperdiçados embora algum investimento em termos de recursos para o sistema pode permitir maior eficiência.
* Favorecer processos "bem comportados": Processos que tenham comportamento adequado poderiam receber um serviço melhor.
* Balancear o uso de recursos: o escalonador deve manter todos os recursos ocupados, ou seja, processos que usam recursos sub- utilizados deveriam ser favorecidos.
* Exibir degradação previsível e progressiva em situações de intensa carga de trabalho.

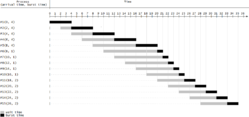
Como pode ser visto facilmente, alguns destes objetivos são contraditórios, pois dado que a quantidade de tempo disponível de processamento (tempo do processador) é finita, assim como os demais recursos computacionais, para que um processo seja favorecido outro deve ser prejudicado.

O maior problema existente no projeto de algoritmos de escalonamento está associado à natureza imprevisível dos processos, pois não é possível prevermos se um dado processo utilizará intensamente o processador, ou se precisará grandes quantidades de memória ou se necessitará numerosos acessos aos dispositivos e [E/S](https://pt.wikipedia.org/wiki/E/S).

## Quando é necessário o uso do algoritmo de escalonamento?

Existem situações nas quais um processo de escalonamento é necessário. Dentre elas estão ocasiões onde um novo processo é criado; quando um processo terminou sua execução e um próximo processo pronto deve ser executado; quando um processo é bloqueado (semáforo, dependência de E/S), resultando que outro processo deve ser executado. Quando uma interrupção de E/S ocorre, o escalonador deve optar por: executar o processo que estava esperando essa interrupção; continuar executando o processo que já estava sendo executado; ou executar um terceiro processo que esteja pronto para ser executado.

## Algoritmos de escalonamento



Execução do [escalonamento FIFO](https://pt.wikipedia.org/wiki/FIFO_(escalonamento)).

Existem os algoritmos [preemptivos](https://pt.wikipedia.org/wiki/Preemp%C3%A7%C3%A3o) e os não preemptivos. Os preemptivos são algoritmos que permitem que um processo seja interrompido durante sua execução, que seja por força de uma interrupção de [entrada/saída](https://pt.wikipedia.org/wiki/Entrada/sa%C3%ADda), quer seja em decorrência da politica de escalonamento adotada e aplicada por parte do escalonador de processos ou simplesmente por força do término da execução do processo. Após a interrupção deste processo, ocorre o que se chama de [troca de contexto](https://pt.wikipedia.org/wiki/Troca_de_contexto), que consiste em salvar o conteúdo dos registradores e a memória utilizada pelo processo e conceder a outro processo o privilégio de executar na CPU, restaurando assim o contexto deste ultimo processo. Cabe ressaltar que nos algoritmos não preemptivos, por serem utilizados exclusivamente em sistemas monoprocessados, esse fato não ocorre, sendo cada programa executado até o fim.

Exemplos de Algoritmos:

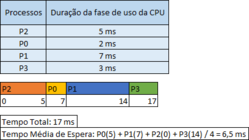
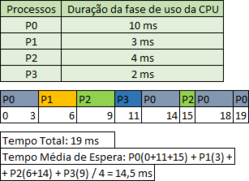


Diagrama de Gantt do algoritmo FIFO

* [FIFO](https://pt.wikipedia.org/wiki/FIFO)[[1]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-1)[[2]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-2)[[3]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-3) (First in, first out) ou [FCFS](https://pt.wikipedia.org/w/index.php?title=FCFS&action=edit&redlink=1) (First come, first served), em português:"primeiro que entra, primeiro que sai": Onde como seu próprio nome já diz, o primeiro que chega será o primeiro a ser executado, não-preemptivo, ou seja, executa o processo como um todo do início ao fim não interrompendo o processo executado até ser finalizado, apenas uma fila, processos que passam para o estado de pronto vão para o final da fila e são escalonados quando chegam no início. Vantagens: o mais simples entre os processos de escalonamento, até mais do que o Round-Robin, todos os processos tendem a serem atendidos. Desvantagens: muito sensível a ordem de chegada, se processos maiores chegarem primeiro aumentarão o tempo médio de espera, não garante um tempo de resposta rápido.
* [SJF](https://pt.wikipedia.org/wiki/SJF) (Shortest Job First): Onde o menor processo ganhará a CPU e atrás do mesmo formar uma fila de processos por ordem crescente de tempo de execução, não-preemptivo. Desvantagem: baixo aproveitamento quando se tem poucos processos prontos para serem executados.
* [SRT](https://pt.wikipedia.org/wiki/SRT) (Shortest Remaining Time): Neste algoritmo é escolhido o processo que possua o menor tempo restante, mesmo que esse processo chegue à metade de uma operação, se o processo novo for menor ele será executado primeiro, preemptivo. Desvantagem: processos que consomem mais tempo de execução podem demorar muito para serem finalizados se muitos processos com curto tempo de execução chegarem.
* [Algoritmo Loteria](https://pt.wikipedia.org/w/index.php?title=Algoritmo_Loteria&action=edit&redlink=1): O Sistema Operacional distribui tokens (fichas), numerados entre os processos, para o escalonamento é sorteado um numero aleatório para que o processo ganhe a vez na CPU, processos com mais tokens têm mais chance de receber antes a CPU;
* Algoritmo de Prioridade: Como o próprio nome já diz, é um algoritmo onde cada processo no estado de pronto recebe uma prioridade, os processos com maiores prioridades são executados primeiro, prioridades que podem ser atribuídas dinâmica ou estaticamente. É um Algoritmo [preemptivo](https://pt.wikipedia.org/wiki/Preemptividade).  
  
* Diagrama de Gantt do algoritmo RR
* [Escalonamento garantido](https://pt.wikipedia.org/wiki/Escalonamento_garantido): Este algoritmo busca cumprir promessas de alocação de CPU o mais preciso possível. Uma forma completamente diferente de tratar a questão do escalonamento é fazer certas promessas ao usuário a respeito da performance, e cumpri-las de alguma forma.Uma promessa bem realista e muito fácil de cumprir é a de que se houver N usuários ativos na rede, cada um vai receber em torno de 1/N da capacidade de processamento que um usuário usou para todos os seus processos desde o momento em que tal usuário tornou-se ativo.Como o tempo que cada usuário gastou até o momento é conhecido, é fácil calcular a razão entre o tempo realmente concedido ao usuário e o tempo prometido. A ideia do algoritmo é por para rodar o processo com razões mais baixas, diminuindo, em consequência, as razões mais altas.
* [RR](https://pt.wikipedia.org/wiki/Round-robin_(algoritmo))[[4]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-4)[[5]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-5)[[6]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-6) (Round-Robin): Inspirado na história de Robin Hood onde, na procura de justiça, Robin roubava dos ricos para entregar aos pobres, fazendo assim com que todos no seu reino tivesse o mesmo tanto de bens. Uma das mais simples e robustas entre as atuais técnicas utilizadas para problemas de distribuição de carga, nesse escalonamento o [sistema operacional](https://pt.wikipedia.org/wiki/Sistema_operacional) possui um timer, chamado de [quantum](https://pt.wikipedia.org/wiki/Quantum), onde todos os processos ganham o mesmo valor de quantum para rodarem na [CPU](https://pt.wikipedia.org/wiki/CPU), depois que o quantum acaba e o processo não terminou, ocorre uma preempção e o processo é inserido no fim da fila. Se o processo termina antes de um *quantum*, a CPU é liberada para a execução de novos processos. Em ambos os casos, após a liberação da CPU, um novo processo é escolhido na fila. Novos processos são inseridos no fim da fila.Quando um processo é retirado da fila para a CPU, ocorre uma *troca de contexto*, o que resulta em um tempo adicional na execução do processo.Esta técnica remove a necessidade de criar sistemas para monitoração dinâmica e são obviamente construídas de forma muito mais rápida e prática das que fazem balanceamento através de medições de recursos. Esta técnica foi criada antes mesmo de existirem computadores e é até hoje utilizada em larga escala por inúmeros sistemas com diferentes propósitos. . Com exceção do algoritmo RR, FIFO e escalonamento garantido, todos os outros sofrem do problema de [Inanição](https://pt.wikipedia.org/wiki/Inani%C3%A7%C3%A3o_(computa%C3%A7%C3%A3o)) (*starvation*), preemptivo;
* [Múltiplas Filas](https://pt.wikipedia.org/wiki/M%C3%BAltiplas_Filas)[[7]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-7)[[8]](https://pt.wikipedia.org/wiki/Escalonamento_de_processos#cite_note-8): São usadas várias filas de processos prontos para executar, cada processo é colocado em uma fila, e cada fila tem uma política de escalonamento própria e outra entre filas, preemptivo, cada fila tem um determinado nível de prioridade, sendo um dos mais antigos agendadores de prioridade, estava presente no CTSS (Compatible Time-Sharing System - Sistema Compatível de Divisão por Tempo).No algoritmo de Múltiplas Filas, também pode ser aplicado particularmente, em cada fila, diferentes algoritmos como por exemplo, o algoritmo RR ou FCFS.
* Algoritmo Fair-Share: O escalonamento é feito considerando o dono dos processos, onde cada usuário recebe uma fração da CPU e processos são escalonados procurando garantir essa fração. Se um usuário A possui mais processos que um usuário B e os dois têm a mesma prioridade, os processos de A serão mais demorados que os do B.

Todos os algoritmos classificam os processos em estados: Iniciando, Pronto, Executando, Entrada/ Saída e Terminado.

## Escalonamento em Sistemas em Batch

● **Algoritmo First-Come First-Served**

* Estratégia de permitir que o processo sendo executado continue sendo executado até ser bloqueado por alguma razão (Não-preemptivo);
* Processos são executados na CPU seguindo a ordem de requisição;
* Facilidade de entender e se programar;
* Desvantagem: Ineficiente quando se tem processos que demoram na sua execução.

● **Algoritmo Shortest Job First**

* Estratégia de permitir que o processo sendo executado continue sendo executado até ser bloqueado por alguma razão (Não-preemptivo);
* Possível prever o tempo de execução do processo;
* Menor processo é executado primeiro;
* Uma das suas principais desvantagens é o baixo aproveitamento quando se tem poucos processos prontos para serem executados.

● **Algoritmo Shortest Remaining Time Next**

* Estratégia de suspender o processo sendo executado (Preemptivo);
* Quando se tem processos com menor tempo de execução, são executados primeiro;
* Se um processo novo chega e seu tempo de execução é menor do que do processo corrente na CPU, a CPU suspende o processo corrente e executa o processo que acabou de chegar;
* A desvantagem desse processo é que consome mais tempo podendo demorar muito para serem finalizados, caso comece a chegar processos menores que ele.

## Estados de processos

Para o sistema operacional organizar os processos que serão atendidos eles são atribuídos estados para os mesmos. São eles:

* Pronto: O processo pode ser executado, está temporariamente parado para que outro processo execute;
* Executando: Processo está sendo servido pela CPU nesse momento, Se ocorrer durante a execução uma requisição de E/S o processo é colocado no estado de espera e outro processo da fila de prontos poderá então concorrer a CPU;
* Espera: É o processo que foi colocado na fila de espera de E/S devido ao dispositivo de E/S ser mais lento que a CPU principal. Enquanto o processo está em espera outros em estado pronto irão concorrer pela CPU;
* Terminado: O processo concluiu sua execução;

### **Diagrama de Estados de Processos**

Quem armazena essas informações como os [estados de processos](https://pt.wikipedia.org/wiki/Processo_(inform%C3%A1tica)) e outras como: tempo e execução, por exemplo, é o [bloco de controle de processo](https://pt.wikipedia.org/wiki/Bloco_de_controle_de_processo) (**PCB** - *Process Control Block*).

## Distribuição de Prioridades

Para melhorar essa distribuição da CPU entre os processos, alguns algoritmos utilizam diferentes prioridades. Com intuito de gerenciar melhor as prioridades de processo, o sistema operacional cria filas de processos. Em cada fila existem processos de mesma prioridade, e existe também fila para processos de entrada e saída. Prioridades podem ser mudadas pelo usuário, ou atribuídas automaticamente pelo sistema operacional em questão.

Mesmo com a aplicação de prioridades e algoritmos melhor implementados, alguns processos ainda correm o risco de sofrer [*starvation*](https://pt.wikipedia.org/wiki/Starvation) (ficar muito tempo sem receber a CPU) por isso em determinando momento pode ocorrer o que chamamos de *aging* (O *aging* ocorre quando a prioridade de um processo vai se alterando com o "tempo de vida" do mesmo, controlando o *starvation*), que muda momentaneamente a prioridade de um processo que não é executado há muito tempo e joga sua prioridade para a mais alta possível para que ele seja atendido, logo após as prioridades voltam ao normal.

Outro caso em que prioridades são alteradas é quando um programa de baixa prioridade começou a fazer uso de algum periférico de entrada e saída antes de outro de prioridade alta. Neste caso processos de alta prioridade são obrigados a esperarem os de baixa terminar sua [E/S](https://pt.wikipedia.org/wiki/E/S) para poderem usar este periférico.

### **Alterando prioridades no Windows**

Existem ainda sistemas em que quando um processo inicia sua execução, o sistema garante que este processo vai ser terminado, são chamados sistemas garantidos. Nestes sistemas a intervenção do usuário é mínima, ao contrário do que ocorre em sistemas em tempo real como o Windows em que o usuário interrompe processos a todo instante por isso o sistema não garante que um processo vai ser Terminado.

## Trocas de contexto

Processos são interrompidos e retomados a todo tempo, para que o sistema operacional possa fazer esse tipo de ação, é necessária a troca de contexto. Para que o sistema operacional possa interromper um processo e retomar ele mais tarde, ele usa a [PCB](https://pt.wikipedia.org/wiki/Bloco_de_controle_de_processo) (*Process Control Block*) para guardar todas as informações que a CPU estava usando naquele momento e possa consultá-la mais tarde para que retome exatamente no ponto em que foi interrompido anteriormente.

Existem três motivos em potencial para realizar uma troca de contexto:

* Multitarefa: Em um esquema de escalonamento de processos, um processo deve ser substituído por outro na CPU. Em um sistema preemptivo, o escalonador permite que cada tarefa seja executada por um determinado tempo. Se um processo não indicar explicitamente a troca de contexto (por exemplo, ao realizar uma operação de E/S), uma interrupção de tempo é disparada, e o sistema operacional troca o contexto para outro processo. Isso assegura que a CPU não é monopolizada por um processo somente;
* Interrupção de hardware: Isso significa que se a CPU requisita dados de um disco, por exemplo, ela não precisa esperar a leitura em disco terminar, podendo continuar alguma outra tarefa na fila de execução. Quando a leitura em disco é terminada, a CPU é interrompida e o resultado da leitura é disponibilizado. Antes de realizar a interrupção, o contexto do processo em execução é armazenado para futura restauração;
* Troca do modo usuário para modo núcleo: Quando uma transição do modo usuário para o modo núcleo é requisitada pelo sistema operacional, a troca de contexto não é necessária; a transição entre os modos por si só *não* é uma troca de contexto. Entretanto, dependendo do sistema operacional, uma troca de contexto pode ser realizada neste momento;

## Threads

*Ver artigo principal:* [*Threads*](https://pt.wikipedia.org/wiki/Thread_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o))

Processos podem ser divididos em “pedaços” para que eles não deixem de responder por algum motivo externo, como isso poderia atrapalhar a sua execução, ou para agilizar a programação e execução. Quando programas são divididos em threads, podemos ter partes do processo rodando em paralelo, pois as threads também são escalonáveis e todas as threads dentro de um processo compartilham o mesmo espaço de endereçamento.

Vantagens:

* As bibliotecas de usuário podem escalonar seus threads para otimizar o desempenho;
* A sincronização é realizada fora do núcleo, e isso evita chaveamento de contexto;
* É mais portável;
* Threads são mais fáceis de serem criadas e destruídas do que um processo;

Desvantagens:

* O núcleo considera o processo multithread como um único thread de controle;
* Isso pode fazer com que o desempenho fique abaixo do ideal se um thread requisitar uma operação E/S;
* Não pode ser escalonado para executar em múltiplos processadores ao mesmo tempo;