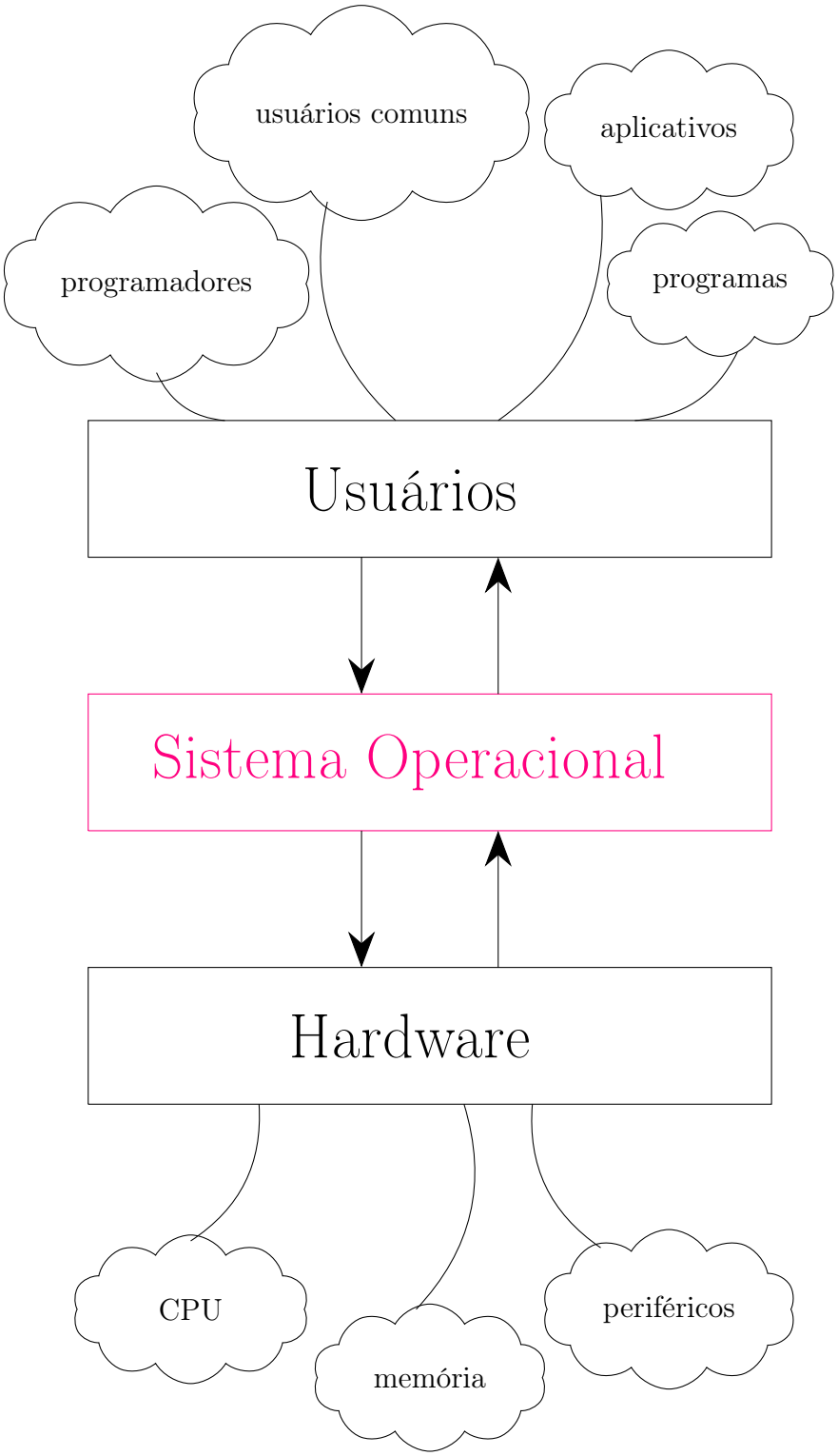
SO - Sistemas Operacionais

Conteúdo

1 Introdução a sistemas operacionais

Sistema operacional atua como um intermediário entre o usuário de um computador e o hardware do mesmo.



Definição: é um conjunto de rotinas executado pelo processador, de forma semelhante aos programas de usuário.

Principal função: controlar o funcionamento de um computador, gerenciando a utilização e o compartilhamento dos seus diversos recursos, como processadores, memória

e dispositivos de entrada e saída.

OBS: um SO não é executado de forma linear como a maioria das aplicações, com início, meio e fim. Suas rotinas são executadas concomitantemente/concorrentemente em função de eventos assíncronos, ou seja, eventos que podem ocorrer a qualquer momento.

Exemplos:

- PC: windows, linux, macOS
- Celular: android, iOS

Observações:

- **Periféricos:** mouse, teclado, monitor, impressora, caixa de som, microfone...
- **Dispositivos de entrada:** mouse, scanner, teclado, microfone...
- **Dispositivos de saída:** monitor, impressora, caixa de som...
- **Sícrono:** que acontece ao mesmo tempo. Exemplo com metrônomo.
- **Assíncrono:** que pode acontecer a qualquer momento. Entradas de mouse e teclado não têm momento certo para ocorrer, o SO tem que estar preparado para essas entradas.
- **Histórico:** até a década de 80, cada fabricante tinha seu próprio SO (como ocorre com a Apple hoje em dia). A partir de 80, com o windows e o linux, houve essa 'massificação'/'homogeneização' dos SOs.

2 Papel do sistema operacional

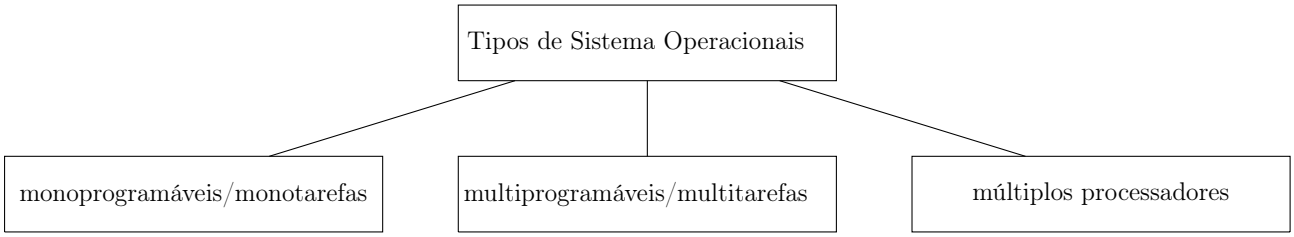
Do ponto de vista do computador (*visão bottom-up*):

- Programa mais intimamente envolvido com o hardware
- **Alocador de recursos:** administra e aloca recursos (tempo de CPU, espaço de memória, espaço de armazenamento em disco, dispositivos de I/O – input/output, entre outros) necessários a resolução de um problema
- **Programa de controle:** gerencia a execução dos programas de usuário para evitar erros e o uso impróprio do computador. Se preocupa principalmente com a operação e o controle de dispositivos de I/O

Do ponto de vista do usuário (*visão top-down*):

- **PC projetado para um único usuário:** SO projetado para facilidade de uso
- **Terminal conectado a um mainframe, no qual, outros usuários acessam o mesmo computador por intermédio de outros terminais:** SO projetado para maximizar o uso de recursos
  - Assegurar que todo o tempo de CPU, memória e I/O disponíveis seja utilizados eficientemente e que nenhum usuário individual ocupe mais do que sua cota
- **Estações de trabalho conectadas a rede com outras estações de trabalho e servidores:** SO projetado para estabelecer um compromisso entre usabilidade individual e utilização de recursos

3 Tipos de Sistemas Operacionais



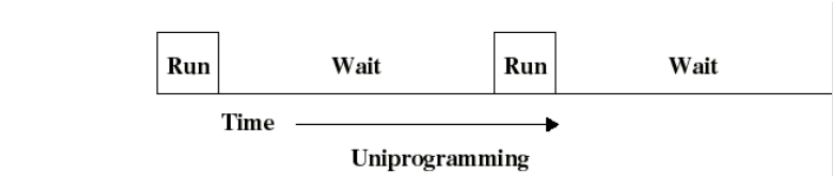
3.1 Sistemas monoprogramáveis/monotarefas:

- Sistema voltado a execução de um único programa/tarefa: são os primeiros SOs (décadas de 60 e 70). Qualquer outra aplicação, para ser executada, deve aguardar o término da corrente.
- Processador, memória e periféricos exclusivamente dedicados a execução de um único programa.
- Tarefa do SO passa a ser unicamente transferir o controle de um *job* (programa e dados) para outro.
- Desvantagem: memória subutilizada, processador ocioso.

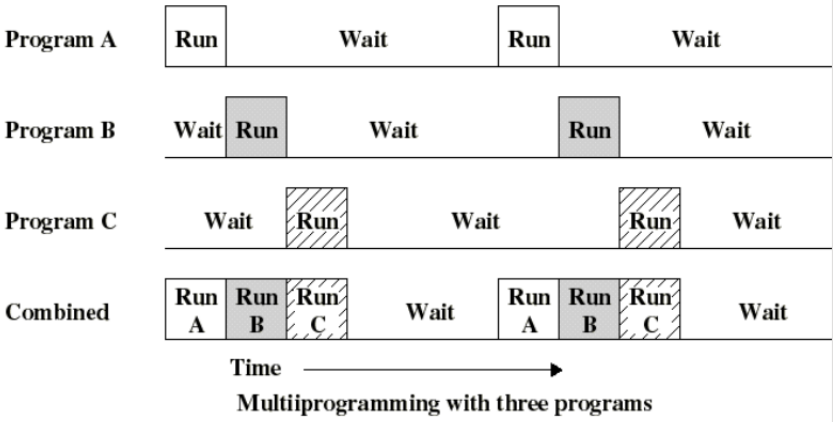
3.2 Sistemas multiprogramáveis/multitarefas:

- SO mantém vários *jobs* na memória simultaneamente, e a CPU é dividida entre eles.
  - Parte deles fica em uma fila de *jobs* no disco (todos os processos residentes em disco aguardando alocação na memória principal).
- SO seleciona e começa a executar um dos *jobs* na memória.
  - Se *job* pode ter de aguardar que alguma tarefa seja concluída
  - SO passa para um novo *job* e o executa
  - Se *job* tem que aguardar, CPU seleciona outro *job* e assim por diante → CPU nunca ficará ociosa
- Fornecem um ambiente em que os diversos recursos do sistema (por exemplo: CPU, memória e dispositivos periféricos) são utilizados eficientemente.

Sistemas Monoprogramados



Sistemas Multiprogramados

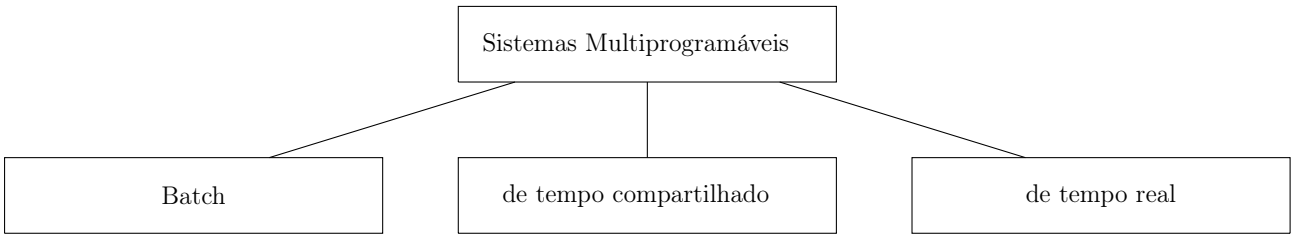


A partir do número de usuários que interagem com o sistema, os sistemas multiprogramáveis são classificados como:

- Monousuário
- Multiusuário

	Um usuário	**Dois ou mais usuários**
Sistema Monoprogramado	monousuário	não possui
Sistema Multiprogramado	monousuário	multiusuário

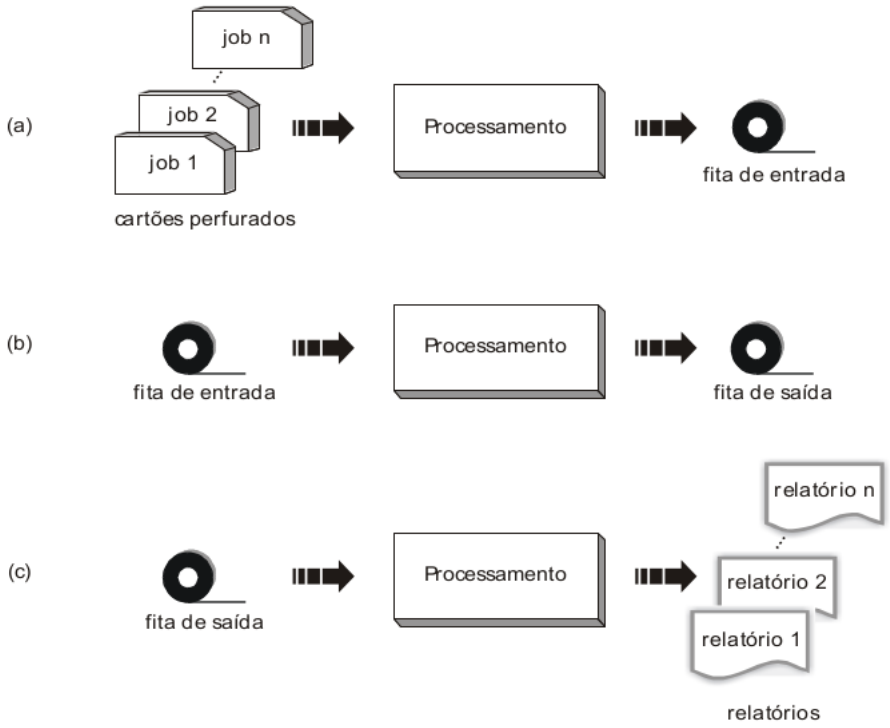
Os sistemas multiprogramáveis podem ser classificados pela forma com que suas aplicações são gerenciadas:



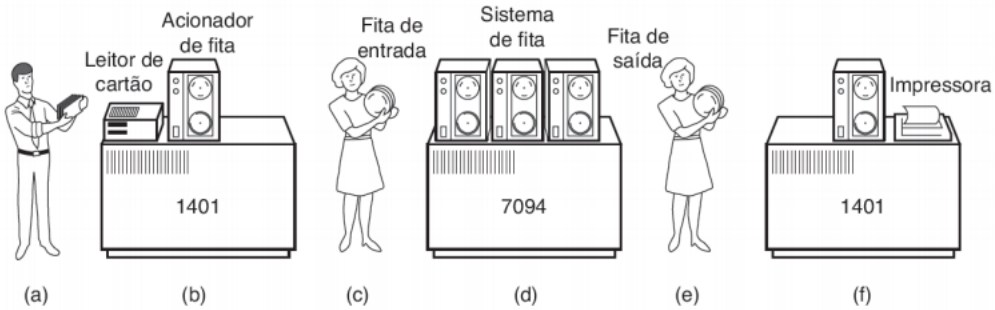
Sistemas Batch

- Implementado a partir da década de 60.
- Todos os programas a executar eram colocados em uma fila.
- O processador recebia um programa após o outro, processando-os em sequência, o que permitia um alto grau de utilização do sistema.
- O termo lote ainda é usado para definir um conjunto de comandos que rodam sem interferência do usuário.

- Todas a entradas e saídas de dados da aplicação são implementadas por algum tipo de memória secundária.
- Exemplos de aplicações:
  - Programas envolvendo cálculo numérico
  - Compilações
  - Backups
  - Outras que não exigem interação com o usuário



Um sistema em lotes (batch) antigo:



(a) Os programadores levam os cartões para o 1401.
(b) O 1401 grava os lotes de tarefas nas fitas.
(c) O operador leva a fita de entrada para o 7094.
(d) 7094 executa o processamento.
(e) O operador leva a fita de saída para o 1401.
(f) 1401 imprime as saídas

Sistemas de Tempo Compartilhado

- Também chamados de *Time-sharing*, começaram a ser implementados na década de 1970.
- Permitem que diversos programas sejam executados a partir da divisão do tempo do processador em pequenos intervalos, denominados fatia de tempo (*time-slice*)
- Caso fatia de tempo insuficiente para conclusão do programa:
  - Programa interrompido pelo SO e substituído por outro
  - Enquanto isso (...) aguarda por nova fatia de tempo
- Sistema cria um ambiente de trabalho próprio, dando a impressão de que todo o sistema está dedicado, exclusivamente para cada usuário
- Permitem a interação do usuário com o sistema através de terminais que incluem vídeo, teclado e mouse
  - Usuário interage com sistema através de comandos. É possível verificar arquivos armazenados em disco ou cancelar a execução de um programa
  - Sistema responde em poucos segundo a execução dos comandos
- Aplicações comerciais utilizam esses sistemas

Sistemas de Tempo Real

- Implementados de forma semelhante aos Sistemas de tempo compartilhado, exceto que não existe a ideia de fatia de tempo, o programa utiliza o processador o tempo que for necessário ou até que apareça outro mais prioritário (definida pela aplicação)
- Sistemas operacionais de tempo real ou RTOS (Real Time Operating Systems) são uma categoria especial de sistemas operacionais. Eles são voltados para aplicações onde é essencial a confiabilidade e a execução de tarefas em prazos compatíveis com a ocorrência de eventos externos
- Sistemas presentes em aplicações de controle de processos
  - Monitoramento de refinarias de petróleo
  - Controle de tráfego aéreo
  - Controle de usinas termoelétricas e nucleares
  - Qualquer aplicação onde tempo de resposta é fator fundamental
- Estruturas comuns em sistemas de tempo real:
  - Semáforos ou flags: são definidos bits ou palavras para a sinalização do tipo **booleano** (binário 0 ou 1) para a troca de mensagens entre duas rotinas;
  - Áreas de troca de mensagens, filas ou **buffers**: memórias temporárias que com auxílio dos semáforos permitem a transferência de estruturas de dados maiores entre as rotinas.

3.3 Trabalho

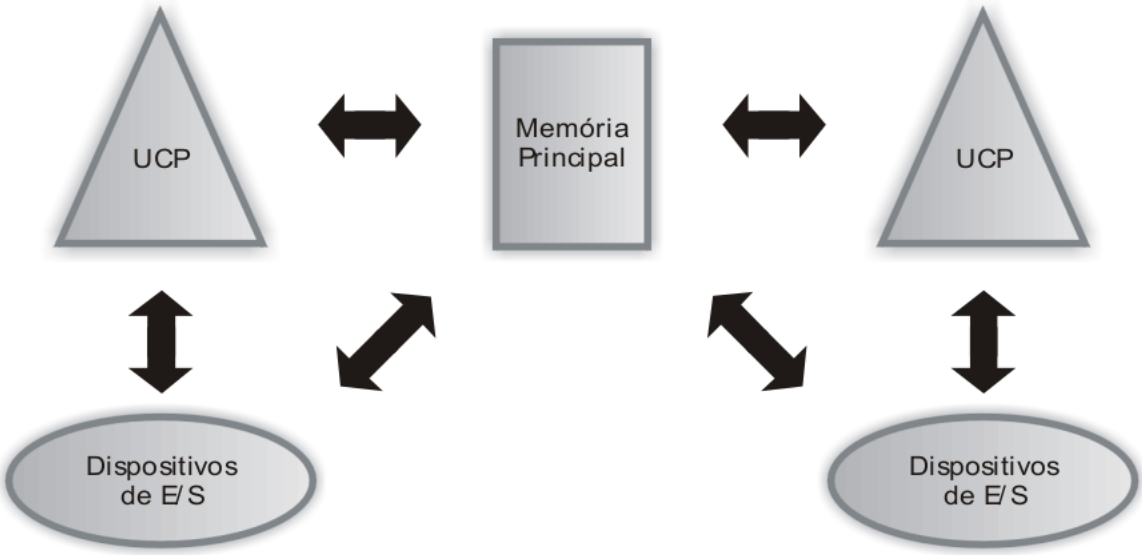
Dividir a turma em grupos, cada grupo fica responsável por pesquisar sobre um dos três tipos de Sistemas Multiprogramados, com imagens ou vídeos (e usos ou exemplos).

3.4 Sistemas com Múltiplos Processadores

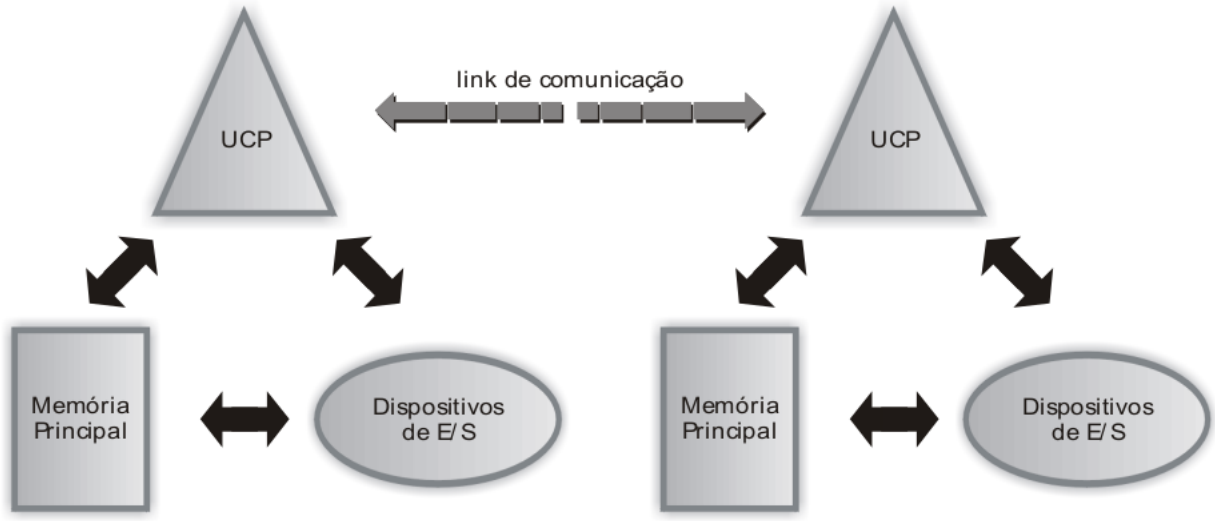
- Caracterizam-se por possuir dois ou mais processadores interligados e trabalhando em conjunto
- **Vantagens:**
  - Vários programas executando ao mesmo tempo
  - Mesmo programa subdividido em partes para serem executadas simultaneamente em mais de um processador
- Possibilidade de implementação de aplicações voltadas para processamento científico
  - Simulações
  - Processamento de imagens
  - Desenvolvimento aeroespacial
- Características:
  - Multiprogramação
  - Escalabilidade: Capacidade de ampliar o poder computacional do sistema adicionando novos processadores
  - Disponibilidade: Capacidade de manter o sistema em operação mesmo diante de falhas
  - Balanceamento de carga: Possibilidade de distribuir o processamento entre os diversos processadores

Os sistemas com múltiplos processadores podem ser classificados quanto a forma de comunicação entre os processadores ou quanto ao grau de compartilhamento da memória e os dispositivos de entrada e saída:

- Sistemas fortemente acoplados



- Sistemas fracamente acoplados



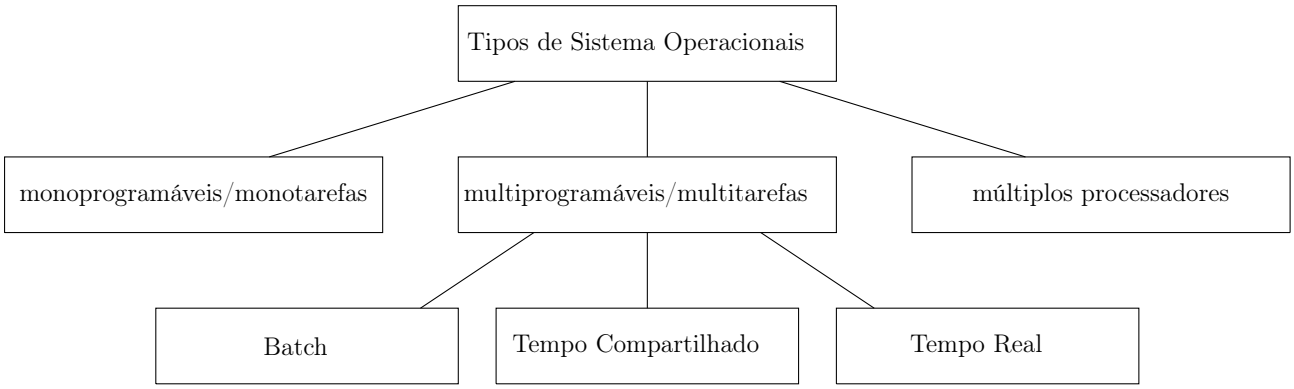
Exemplos de Sistemas Operacionais com Múltiplos Processadores Fracamente Acoplados:

- Sistemas Operacionais de Rede
  - Permitem que um computador (host) compartilhe seus recursos (p.e., impressora, diretório) com os demais hosts da rede
  - Usados em redes locais (estação oferece serviços de impressão e arquivos para as demais estações da rede, entre outros)

  - Exemplo: Windows 2000, Novell Netware, Linux

- Sistemas Operacionais Distribuídos
  - Sistema operacional esconde os detalhes dos hosts individuais e passa a tratá-los como um conjunto único
  - Exemplo: Amoeba → Tanembaum, 1991

3.5 Resumo



3.6 Trabalho 2

Lista de Exercícios: usar parte para um segundo trabalho, e parte para revisão.

1. Dê exemplo de Sistemas Operacionais.
2. Os Sistemas Operacionais podem ser divididos em quantos tipos? Quais são eles?
3. Cite três características dos sistemas monoprogrmáveis.
4. Cite três características dos sistemas multiprogramáveis.
5. Os Sistemas Operacionais Multiprogramáveis podem ser divididos em quantos tipos? Quais são eles?
6. Cite uma diferença entre os tipos de Sistemas Operacionais Multiprogramáveis.
7. Cite três características dos Sistemas Operacionais com Múltiplos Processadores.

4 Processos

4.1 O que é um Processo Computacional

Um processo computacional ou simplesmente processo pode ser entendido como uma atividade que ocorre em meio computacional, usualmente possuindo um objetivo definido, tendo duração finita e utilizando uma quantidade limitada de recursos computacionais.

Esta definição traz algumas implicações: apenas as atividades que acontecem num sistema computacional são compreendidas como sendo processos computacionais. Outro ponto importante é a duração finita, pois isto implica que um processo computacional,

por mais rápido ou curto que possa ser tem sempre uma duração maior que zero, ou seja, não existem processos instantâneos. Além disso, um processo utiliza ao menos um dos recursos computacionais existentes para caracterizar seu estado.

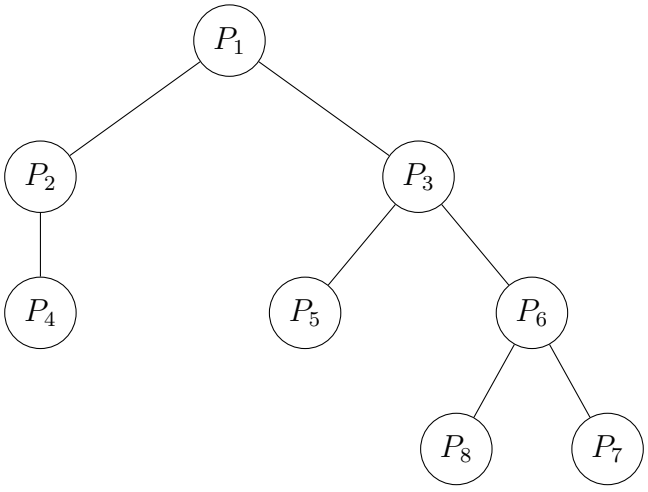
O termo processo (*process*) é muitas vezes substituído pelo termo tarefa (*task*) e pode assumir um dos seguintes significados

- um programa em execução
- uma atividade assíncrona
- o espírito ativo de um procedimento
- uma entidade que pode utilizar um processador ou
- uma unidade que pode ser despachada para execução.

4.2 Subdivisão de Processos

Outro ponto importante é que os processos computacionais podem ser divididos em sub-processos, ou seja, podem ser decompostos em processos componentes mais simples que o processo como um todo, o que permite um detalhamento da realização de sua tarefa ou do seu modo de operação. Esta análise aprofundada dos processos através de sua decomposição em sub-processos pode ser feita quase que indefinidamente, até o exagerado limite das micro-instruções do processador que será utilizado. O nível adequado de divisão de um processo é aquele que permite um entendimento preciso dos eventos em estudo, ou seja, depende do tipo de problema em questão e também da solução pretendida.

Processos tipicamente também podem criar novos processos. O processo criador é chamado de processo-pai (*parent process*) enquanto os processos criados são denominados de processos filhos (*child process*). Um processo-filho também pode criar novos processos, permitindo a criação de árvores de processos hierarquicamente relacionados, como exemplificado abaixo:



4.3 Ocorrência de Processos

Como cada processo precisa de recursos para ser executado e concluído, a ocorrência de processos significa a utilização de recursos do computador. Um critério muito importante de análise dos processos computacionais é aquele que considera os processos segundo sua

ocorrência, isto é, a observação de seu comportamento considerando o tempo. Neste caso teríamos os seguintes tipos de processos:

- **Sequenciais:** são aqueles que ocorrem um de cada vez, um a um no tempo, serialmente, como que de forma exclusiva.
- **Paralelos:** aqueles que, durante um certo intervalo de tempo, ocorrem simultaneamente, ou seja, aqueles que no todo ou em parte ocorrem ao mesmo tempo.

4.3.1 Processos sequenciais

São aqueles que ocorrem um de cada vez no tempo, como numa série de eventos, temos que para um dado processo, todos os recursos computacionais estão disponíveis, ou seja, como só ocorre um processo de cada vez, os recursos computacionais podem ser usados livremente pelos processos, não sendo disputados entre processos diferentes, mas apenas utilizados da maneira necessária por cada processo.

Problemas:

- Como é muito improvável que um processo utilize mais do que alguns poucos recursos do sistema, todos os demais recursos não utilizados ficarão ociosos por todo o tempo de execução deste processo.
- A ociosidade dos diversos recursos computacionais é muito alta, sugerindo que sua utilização é pouco efetiva, ou, em outros termos, inviável economicamente.

4.3.2 Processos paralelos

São aqueles que, durante um certo intervalo de tempo, ocorrem simultaneamente. Então existe a possibilidade de que dois ou mais destes processos passem, a partir de um dado momento, a disputar o uso de um recurso computacional particular.

Considerando tal possibilidade de disputa por recursos e também sua natureza, os processos paralelos podem ser classificados nos seguintes tipos:

- **Independentes:** quando utilizam recursos completamente distintos, não se envolvendo em disputas com outros processos.
- **Concorrentes:** quando pretendem utilizar um mesmo recurso, dependendo de uma ação do sistema operacional para definir a ordem na qual os processos usarão o recurso.
- **Cooperantes:** quando dois ou mais processos utilizam em conjunto um mesmo recurso para completarem uma dada tarefa.

Como não se pode prever quais os tipos de processos que existirão num sistema computacional, o sistema operacional deve estar preparado para administrar a ocorrência de processos paralelos concorrentes em quantidade.

Apesar da maior complexidade, a existência de processos paralelos permite o melhor aproveitamento dos sistemas computacionais e mais, através do suporte oferecido pelo sistema operacional passa a ser possível a exploração do processamento paralelo e da computação distribuída.

4.4 Estados dos Processos

Dado que um processo pode ser considerado como um programa em execução, num sistema computacional multiprogramado poderíamos identificar três estados básicos de existência de um processo:

- **Pronto (ready):** o processo está apto a utilizar o processador quando este estiver disponível. Isto significa que o processo pode ser executado quando o processador estiver disponível.
- **Execução (running):** o processo está utilizando um processador para seu processamento. Neste estado o processo tem suas instruções efetivamente executadas pelo processador.
- **Bloqueado (blocked):** o processo está esperando ou utilizando um recurso qualquer de E/S (entrada e saída). Como o processo deverá aguardar o resultado da operação de entrada ou saída, seu processamento fica suspenso até que tal operação seja concluída.

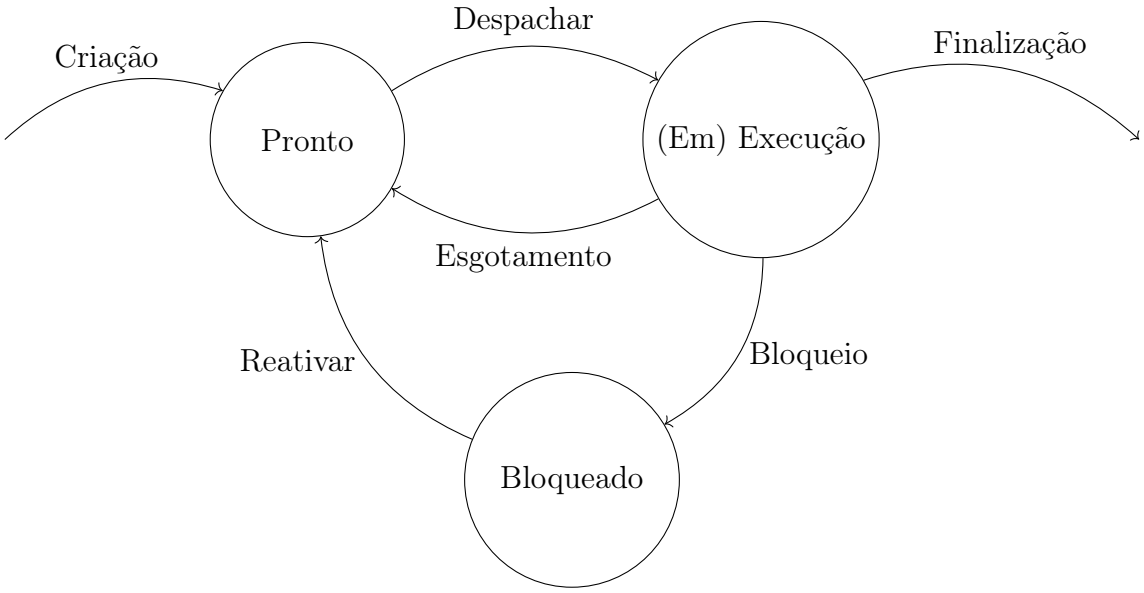
Além disso, entre os três estados básicos existem quatro transições possíveis:

- Despachar (Dispatch)
- Esgotamento (Time Run Out)
- Bloqueio (Block)
- Reativar (Awake)

Existem também outras duas transições, mas estas não entre estados:

- Criação (Create)
- Finalização (Terminate)

Diagrama de estados de processos:



PID: Quando solicitamos a execução de um programa, o sistema operacional cria (Create) um processo atribuindo a este um número de identificação ou seu PID (Process Identifier), um valor inteiro que servirá para distinguir este processo dos demais.

4.5 PCB

PCB (Process Control Block ou Process Descriptor) é uma estrutura de dados que mantêm a representação de um processo para o sistema operacional, com as informações necessárias de cada processo. Apesar de ser dependente do projeto e implementação particulares do sistema operacional, geralmente o PCB contém as seguintes informações:

- identificação do processo (PID);
- estado corrente do processo;
- ponteiro para o processo pai (parent process);
- lista de ponteiros para os processos filho (child processes);
- prioridade do processo;
- lista de ponteiros para as regiões alocadas de memória;
- informações sobre horário de inıcio, tempo utilizado do processador;
- estatısticas sobre uso de memória e periféricos;
- cópia do conteúdo do contador de programa (program counter);
- cópia do conteúdo dos registradores do processador;
- identificador do processador sendo utilizado;
- informações sobre diretórios raiz e de trabalho;
- informações sobre arquivos em uso;
- permissões e direitos.

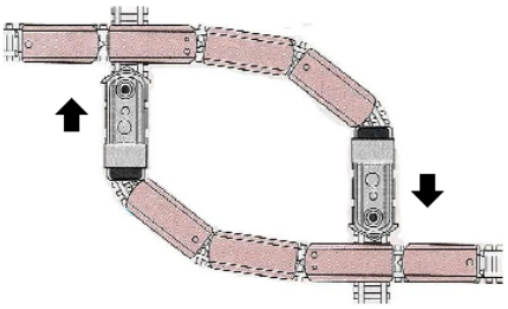
4.6 Deadlocks ou Impasse

Em sistema multiprogramado, o termo deadlock, ou seja, bloqueio perpétuo ou impasse, significa um evento que jamais irá ocorrer [DEI92, TAN92, SGG01]. Dizemos que um processo está em deadlock quando espera por um evento particular que jamais acontecerá. Igualmente dizemos que um sistema está em deadlock quando um ou mais processos estão nesta situação.

Segundo Tanenbaum:

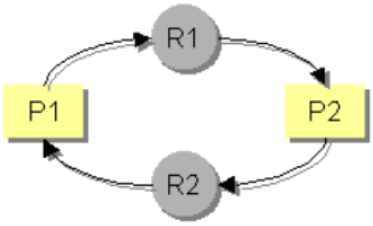
Um conjunto de processos está num bloqueio perpétuo quando cada processo do conjunto está esperando por um evento que apenas outro processo do conjunto pode causar. [TAN92, p. 242]

Observemos a Figura abaixo, onde temos ilustrado um deadlock que pode ocorrer em duas linhas de trens cujas interseções não são compartilháveis. Problemas semelhantes podem ocorrer no trânsito de uma metrópole.



Um bloqueio perpétuo pode ocorrer de diferentes maneiras:

- Quando um processo é colocado em espera por algo e o sistema operacional não inclui qualquer previsão para o atendimento desta espera dizemos que ocorreu o bloqueio perpétuo de um processo único (one-process deadlock).
- Quando se forma uma cadeia sucessiva de solicitações de recursos que culminam num arranjo circular, onde um processo A, que detêm um recurso R1, solicita um recurso R2 alocado para um processo B, que por sua vez está solicitando o recurso R1, em uso por A (veja a Figura abaixo). Como nenhum processo de dispõe a, voluntariamente, liberar o recurso que aloca, configura-se uma situação de bloqueio perpétuo.



4.7 Condições para ocorrência de Deadlocks

Condições de Coffman: existem quatro condições para a ocorrência de Deadlocks.

1) **Exclusão mútua:** Cada recurso está atualmente associado a exatamente um processo ou está disponível.

2) **Posse e espera:** Processos atualmente de posse de recursos que foram concedidos antes podem solicitar novos recursos.

3) **Não preempção:** Recursos concedidos antes não podem ser tomados à força de um processo. Eles precisam ser explicitamente liberados pelo processo que os têm.

4) **Espera circular:** Deve haver uma lista circular de dois ou mais processos, cada um deles esperando por um processo de posse do membro seguinte da cadeia.

Todas essas quatro condições devem estar presentes para que um impasse de recurso ocorra. Se uma delas estiver ausente, nenhum impasse de recurso será possível.

4.8 Estratégias para lidar com Deadlocks

4.8.1 Algoritmo do Avestruz

A abordagem mais simples é o algoritmo do avestruz: enfie a cabeça na areia e finja que não há um problema, ou seja, ignora o problema, supondo que a ocorrência de Deadlocks é baixa.

4.8.2 Detecção e Recuperação de Deadlocks

A detecção é feita de acordo com as condições de Coffman e usando estruturas como: vetor de recursos, matriz de alocação, matriz de requisição ou grafo de recursos (para sistemas mais simples, que possuem apenas um recurso de cada tipo).

A recuperação pode ser feita das seguintes formas:

1) Mediante Preempção: um recurso é tomado de um processo que está em deadlock. Em alguns casos, pode ser necessária a intervenção manual.

2) Mediante Retrocesso (roll-back): os procesos geram **checkpoints** (pontos de salvaguarda) regularmente, dessa forma, em um impasse, um processo é retrocedido até esse checkpoint, mesmo assim, parte do seu andamento é perdido.

3) Mediante a eliminação de processos: é a maneira mais simples de se resolver um deadlock, mata-se um ou mais processos. Uma possibilidade é matar um processo no ciclo. Com um pouco de sorte, os outros processos serão capazes de continuar. Se isso não ajudar, essa ação pode ser repetida até que o ciclo seja rompido.

4.8.3 Evitando Deadlocks

- **Trajetórias de recursos:** os processos e suas respectivas demandas por recursos são analisados ao longo do tempo. Caso haja uma intersecção de recursos entre dois processos, ou seja, a possibilidade de dois processos precisarem de um mesmo recurso ao mesmo tempo, estes recursos são colocados como regiões inseguras e apenas um processo será escolhido para utilizar esses recursos, cabendo ao outro esperar que o primeiro termine.

- **Estados seguros e inseguros:** um estado é dito seguro se existir alguma ordem de escalonamento na qual todos os processos puderem ser executados até sua conclusão mesmo que todos eles subitamente solicitem seu número máximo de recursos imediatamente. A diferença entre um estado seguro e um inseguro é que a partir de um seguro o sistema pode garantir que todos os processos terminarão; a partir de um estado inseguro, nenhuma garantia nesse sentido pode ser dada.

- **Algoritmo do Banqueiro:** (desenvolvido por Dijkstra - 1965) supõe que a quantidade de recursos é limitada e só libera um processo de ser iniciado se esse limite não for atingido no pior caso (processo utilizar todos os recursos que precisa ao mesmo tempo). O algoritmo confere se conceder a solicitação leva a um estado inseguro. Se afirmativo, a solicitação é negada. Se conceder a solicitação conduz a um estado seguro, ela é levada adiante.

4.8.4 Prevenção de Deadlocks

Atacar uma das quatro condições de Coffman:

- **Atacar a condição de exclusão mútua:** se nunca acontecer de um recurso ser alocado exclusivamente para um único processo, jamais teremos impasses. Uma ideia é evitar alocar um recurso a não ser que seja absolutamente necessário, e tentar certificar-se de que o menor número possível de processos possa, realmente, requisitar o recurso. Por exemplo, para dados, o método mais simples é tornar os dados somente para leitura, de maneira que os processos podem usá-los simultaneamente.

- **Atacar a condição de posse e espera:** se pudermos evitar que processos que já possuem recursos esperem por mais recursos, poderemos eliminar os impasses. Uma maneira de atingir essa meta é exigir que todos os processos solicitem todos os seus recursos antes de iniciar a execução. Se tudo estiver disponível, o processo terá alocado para si o que ele precisar e pode então executar até o fim. Uma maneira ligeiramente diferente de romper com a condição de posse e espera é exigir que um processo que solicita um recurso primeiro libere temporariamente todos os recursos atualmente em suas mãos.**Problemas:**

  - muitos processos não sabem de quantos recursos eles precisarão até começarem a executar.
  - os recursos não serão usados de maneira otimizada com essa abordagem.

- **Atacar a condição de não preempção:** uma solução é virtualizar os recursos, dessa forma um processo consegue tomar a força (furar a fila de um recurso) determinado recurso, enquanto o outro processo fica na parte virtualizada do recurso. **Problema:** poucos recursos podem realmente ser virtualizados.

- **Atacar a condição de espera circular:** uma solução é: ordenar os recursos disponíveis, dessa forma, um processo pode solicitar novos recursos (sem liberar os que está utilizando) em ordem crescente apenas, por exemplo, se um processo A está utilizando o recurso 5, ele pode apenas solicitar recursos que tem seu identificador maior que 5, ou precisa liberar o recurso 5 para solicitar algum recurso de identificador menor. Com essa regra, o grafo de alocação de recursos jamais pode ter ciclos.

4.9 Escalonamento de Processos

Definição:

- Escalonar: dividir ou dispor em um período de tempo.

Ou seja, escalonar processos significa organizar os processos que ocorrem concomitantemente em um processador ao longo do tempo.

Note que: se estamos falando de processos que ocorrem ao mesmo tempo, estamos nos atendo aos sistemas multiprogramáveis.

Já vimos as subdivisões dos sistemas multiprogramáveis: em lote (Batch), interativos (de tempo compartilhado), de tempo real; além disso temos os sistemas com uso de Threads.

Teremos algoritmos mais indicados a depender do tipo de sistema.

4.9.1 Escalonamento em Sistemas em Lote (Batch)

- **Primeiro a chegar, primeiro a ser servido** (**FIFO:** *first in, first out*): é o algoritmo mais simples de ser implementado, utiliza uma Fila simples e não é preemptivo, ou seja, um recurso não será tomado a força de um processo. Ele é o mais simples de se compreender e também o mais simples de se implementar.
- **Tarefa mais curta primeiro** (*shortest job first*):
  - Não preemptivo
  - Assume que o tempo de execução de cada processo é conhecido previamente
  - Escolhe a tarefa (o processo) que demanda menos tempo do processador
- **Tempo restante mais curto em seguida** (*shortest remaining time next*):
  - Preemptivo
  - Assume que o tempo de execução de cada processo é conhecido previamente
  - Quando uma tarefa nova chega, seu tempo de execução é comparado com as tarefas que estão em andamento, se seu tempo for menor que de alguma outra tarefa, esta é suspensa para que a tarefa nova seja realizada
  - Permite que tarefas curtas tenham um bom desempenho

4.9.2 Escalonamento em Sistemas Interativos

- **Chaveamento circular** (*Round-robin*):
  - Preemptivo
  - Cada processo recebe um intervalo de tempo, um *quantum*, e ele pode executar durante esse período, acabado seu tempo, o processo vai para o fim da fila e aguarda sua vez novamente
  - Cada troca de processo, cada chaveamento, demanda algum tempo de CPU, não é automático. Por isso, determinar o tamanho do *quantum* é crucial, pois um *quantum* muito pequeno pode significar que uma porcentagem alta da CPU será utilizada apenas nessa troca de processos, no chaveamento.
- **Prioridades**:
  - Não preemptivo
  - A cada processo é designada uma prioridade, e o processo executável com a prioridade mais alta é autorizado a executar
  - Processos de baixa prioridade podem ter sua execução adiada indefinidamente pela chegada de processos de maior prioridade e sofrer de estagnação (*starvation*, ou seja, morrer famintas)
  - Uma solução para *starvation* é aumentar a prioridade dos processos que esperam na fila com o passar do tempo, o que é chamado de *aging*

- **Múltiplas filas**: Uma forma de considerar prioridades entre os processos. Estabelecem-se classes de prioridade para os processos, os na classe mais alta seriam executados por dois quanta, os na classe seguinte seriam executados por quatro quanta, etc. Sempre que um processo consumia todos os quanta alocados para ele, era movido para uma classe inferior. O processo executa pelo tempo que lhe foi dado de acordo com a fila em que estava e é colocado na próxima fila (fila inferior, com mais quanta).

- **Processo mais curto em seguida** (*Shortest Process First*):

  - Não preemptivo
  - Mesma ideia do algoritmo **tarefa mais curta primeiro**, mas pensada para sistemas interativos e não em lote
  - O algoritmo faz estimativas baseadas no comportamento passado e executar o processo com o tempo de execução estimado mais curto

- **Garantido**: em um sistema com n processos sendo executados, todos os fatores permanecendo os mesmos, cada um deve receber $\frac{1}{n}$ dos ciclos da CPU. Atribui-se um índice de uso de CPU para cada processo, um índice de 0,5 significa que o processo usou metade do tempo de CPU que tinha direito e um índice de 2 significa que o processo usou o dobro de tempo de CPU que tinha direito. Dessa forma, o algoritmo então executará o processo de menor índice para que este aumente e o processo aproxime de um índice 1.

- **Loteria**: a ideia básica é dar bilhetes de loteria aos processos para vários recursos do sistema, como o tempo da CPU. Sempre que uma decisão de escalonamento tiver de ser feita, um bilhete de loteria será escolhido ao acaso, e o processo com o bilhete fica com o recurso. Processos mais importantes podem receber bilhetes extras, para aumentar a chance de vencer.

  O algoritmo por loteria é altamente responsivo: se um novo processo aparece e ele ganha alguns bilhetes, no sorteio seguinte ele teria uma chance de vencer na proporção do número de bilhetes que tem em mãos.

- **Fração justa**: Até agora presumimos que cada processo é escalonado por si próprio, sem levar em consideração quem é o seu dono. Como resultado, se o usuário 1 inicia nove processos e o usuário 2 inicia um processo, com chaveamento circular ou com prioridades iguais, o usuário 1 receberá 90% da CPU e o usuário 2 apenas 10% dela. O algoritmo por fração justa leva em conta qual usuário é dono de um processo antes de escaloná-lo. Nesse modelo, a cada usuário é alocada alguma fração da CPU e o escalonador escolhe processos de uma maneira que garanta essa fração.

4.9.3 Escalonamento em Sistemas de Tempo Real

Um sistema de tempo real é aquele em que o tempo tem um papel essencial. Tipicamente, um ou mais dispositivos físicos externos ao computador geram estímulos, e o computador tem de reagir em conformidade dentro de um montante de tempo fixo.

Sistemas em tempo real são geralmente categorizados como tempo real **crítico**, significando que há prazos absolutos que devem ser cumpridos e tempo real **não crítico**, significando que descumprir um prazo ocasional é indesejável, mas mesmo assim tolerável.

Os eventos a que um sistema de tempo real talvez tenha de responder podem ser categorizados ainda como **periódicos** (significando que eles ocorrem em intervalos regulares) ou **aperiódicos** (significando que eles ocorrem de maneira imprevisível).

Algoritmos de escalonamento de tempo real podem ser **estáticos** ou **dinâmicos**.

- **Estáticos:** tomam suas decisões de escalonamento antes de o sistema começar a ser executado. Este funciona apenas quando há uma informação perfeita disponível antecipadamente sobre o trabalho a ser feito, e os prazos que precisam ser cumpridos.
- **Dinâmicos:** tomam suas decisões no tempo de execução, após ela ter começado.

Além disso, um sistema em tempo real pode ser **escalonável** ou não. Isto depende de uma métrica: a soma de todos os tempos que os eventos irão gastar deve ser menor que o tempo total da CPU. Ou seja, se o montante total de tempo de CPU que os processos querem coletivamente é maior do que a CPU pode proporcionar, o sistema é dito **não escalonável**.

4.9.4 Escalonamento de Threads

- **Threads de usuário:**
  - Pode utilizar qualquer um dos algoritmos de escalonamento descritos anteriormente. Na prática, o escalonamento circular e o de prioridade são os mais comuns.
  - Ausência de um relógio para interromper um thread que esteja sendo executado há tempo demais. Visto que os threads cooperam, isso normalmente não é um problema.
  - Chaveamento de thread com threads de usuário exige uma grande quantidade de instruções de máquina.
  - Um bloqueio de thread na E/S suspende todo o processo.
  - Podem empregar um escalonador de thread específico de uma aplicação, o que pode maximizar o paralelismo.
- **Threads de núcleo:**
  - O thread recebe um quantum e é suspenso compulsoriamente se o exceder.
  - O núcleo escolhe um thread em particular para executar, sem a necessidade de considerar a qual processo o thread pertence.
  - Chaveamento de threads de núcleo exige uma troca de contexto completo, mudar o mapa de memória e invalidar o cache, o que representa uma demora de magnitude várias ordens maior (comparado ao chaveamento de threads de usuário).