Processos e Threads

LICENCIATURA EM COMPUTAÇÃO

Processos

- **É uma abstração de um programa em execução** (está na RAM).
  - Conceito mais central em qualquer SO.

- Como processo, um programa pode:
  - Alocar recursos
  - Compartilhar dados
  - Trocar informações e
  - Sincronizar sua execução.

- Gerência de processos:
  - Função básica do SO.
  - Essência da multiprogramação.

Processos

- **Relembrando...**
  - Pseudoparalelismo: é ilusão de que se pode executar dois ou mais programas ao mesmo tempo em uma **máquina monoprocessada**.
  - Multiprogramação: divisão da memória para que ocorra uma troca rápida de processos na CPU.
  - Multiprocessadores: quando se tem mais de um processador para realizar um processamento paralelo verdadeiro.
- **Para ambos, a necessidade é de executar mais de um programa (processo) por vez.**

Processos

- SO cria **um contexto de execução** para rodar um programa:
  - Área da RAM a ser usada.
  - Abertura de arquivos acessados.
  - Privilégios de segurança.
  - *Quantum* de tempo.
  - Prioridade (entre outros).

Processos

- **Um processo constitui uma atividade;**
  - Possui programa, entrada, saída e um estado.
- **Um único processador pode ser compartilhado por vários processos.**
- **O escalonador (algoritmo) determina quando trocar um processo por outro.**

- **Processos necessários:**
  - Em sistemas simples (apenas uma aplicação – micro-ondas): todos os processos são necessários quando o sistema é ligado.
  - Em sistemas de propósito geral: é necessário criar e terminar processos durante a operação.

Criação de Processos

- **Eventos que levam a criação de processos:**
  - Início do sistema;
  - Execução de chamada ao sistema de criação de um processo por um processo em execução;
  - Solicitação do usuário para criar um novo processo;
  - Início de um tarefa em lote (computadores de grande porte): quando SO tem todos os recursos disponíveis, cria novo processo.

Criação de Processos

- Alguns processos criados ficam em **primeiro plano** (interagindo com o usuário) e outros em segundo (apresentam função específica).
- Para ver a lista de processos:
  - Unix e Linux: digite *ps* na linha de comando.
  - Windows: CTRL+ALT+DEL
- Nos dois sistemas é possível ter **várias janelas abertas ao mesmo tempo e cada uma executando um processo;**
- Em um sistema em lote (computadores de grande porte), quando o SO tiver todos os recursos, ele criará um novo processo e executará a próxima tarefa da fila.

Criação de Processos

- Todo processo é criado por um **processo existente** executando uma chamada ao sistema de criação de processo.
- Tanto no Windows quanto no Unix, quando um **processo filho é criado, o pai e o filho tem seus próprios e distintos espaços de endereçamento.**
- É possível o **compartilhamento de recursos** entre o pai e o filho.

Término de Processos

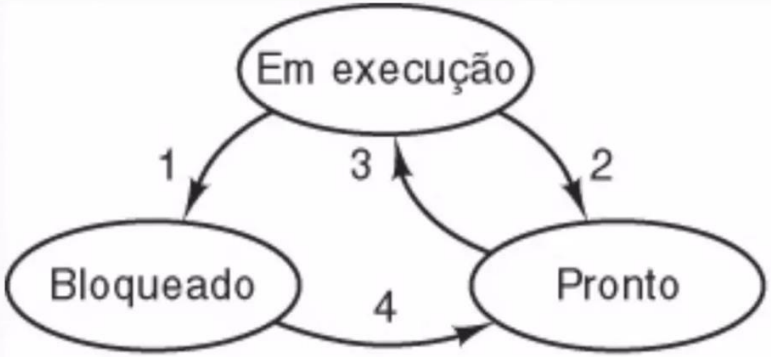
- **Condições que levam ao término de processos:**
  - Saída normal (voluntária)
    - Ex.: Quando finaliza o que tinha que fazer.
  - Saída por erro (voluntária): processo descobre erro fatal.
    - Ex.: Tenta compilar um programa e o arquivo não existe.
  - Erro fatal (involuntário) causado pelo processo, erro de programa.
    - Ex.: Referência a memória inexistente ou divisão por zero.
  - Cancelamento por um outro processo (involuntário): processo executa uma chamada de sistema dizendo ao SO para matar outro processo. Processo precisa de autorização.

Hierarquia de Processos

- Pai cria um processo filho e o processo filho pode criar seu próprio processo.
- Formam uma hierarquia.
  - Unix chama isso de “grupo de processos”.
- Windows não possui o conceito de hierarquia de processos.
  - Todos os processos são criados **iguais**.

Estados do Processo

- **Possíveis estados:**
  - Em execução.
  - Bloqueado.
  - Pronto.
- **Transições entre os estados:**



1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Processos CPU-bound x I/O bound

- CPU-*bound* (ligado à CPU)
  - Maior parte do tempo em estado de **execução.**
  - Ou seja: usando o processador.
- I/O-*bound* (ligado à E/S)
  - Maior parte do tempo em estado de **bloqueado.**
  - Ou seja: fazendo E/S
- Pode iniciar CPU-*bound* e tornar-se I/O-*bound* (e vice-versa).

Gerenciamento de Processos

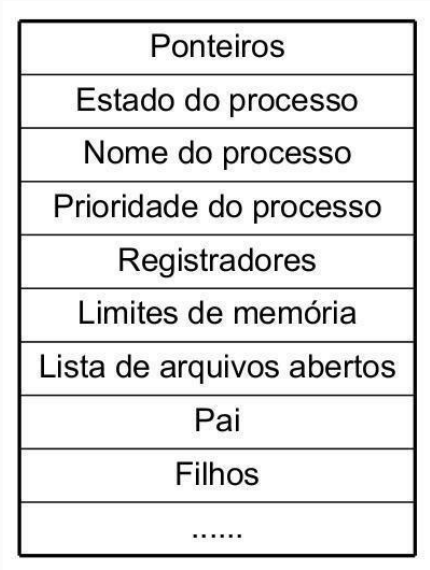
- SO organiza os processos da **fila de prontos.**
- Periodicamente o ESCALONADOR de processos:
  - Escolhe um processo da fila para executar.
  - Critério de escolha pode variar.
- Alguns **critérios** usados para escalonamento:
  - Ordem de chegada dos processos.
  - Fatia de tempo demandada pelo processo.
  - Prioridade do processo.

Gerenciamento de Processos

- Outros nomes para ESCALONADOR/ESCALONAMENTO:
  - Despachante/despacho
  - Dispatcher/dispatch
- Quando um processo solicita uma operação de E/S:
  - Ele é **interrompido** e desviado para a fila de **bloqueado.**
  - Posteriormente, ele retornará para o **fim da fila de pronto.**
  - Então, será submetido ao critério de escalonamento da fila.
- **Preempção** – capacidade do SO de:
  - **Interromper** um processo a qualquer instante (a seu critério).
  - Retomará a execução a partir do ponto de interrupção.

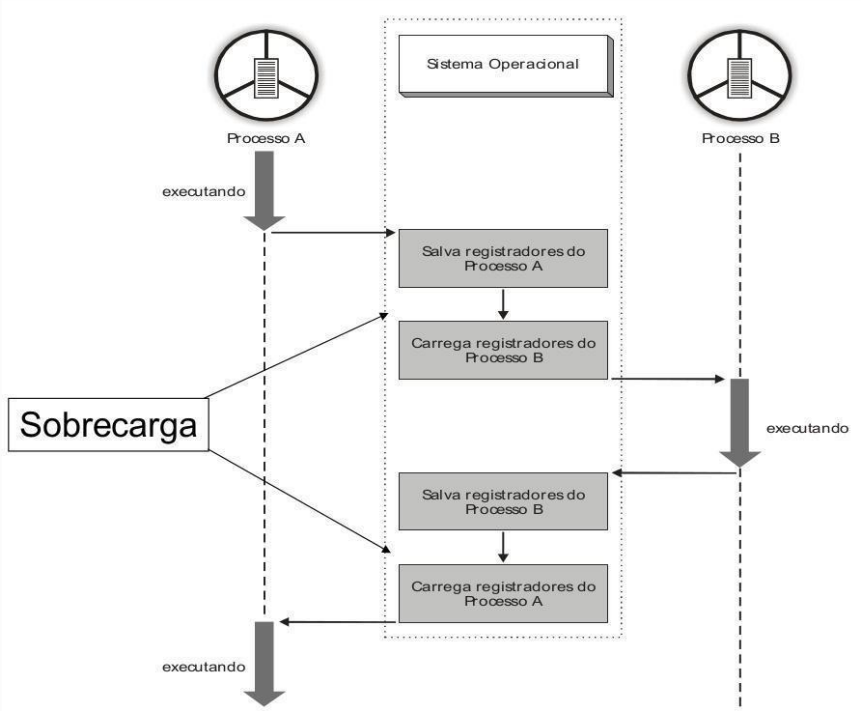
Bloco de Controle de Processos (PCB)

- Ao criar um processo, o SO cria um BCP (PCB)
  - É uma **tabela com informações** relativas ao processo.
  - Reside na **RAM** enquanto o processo existe.
  - Acesso **exclusivo** do SO.



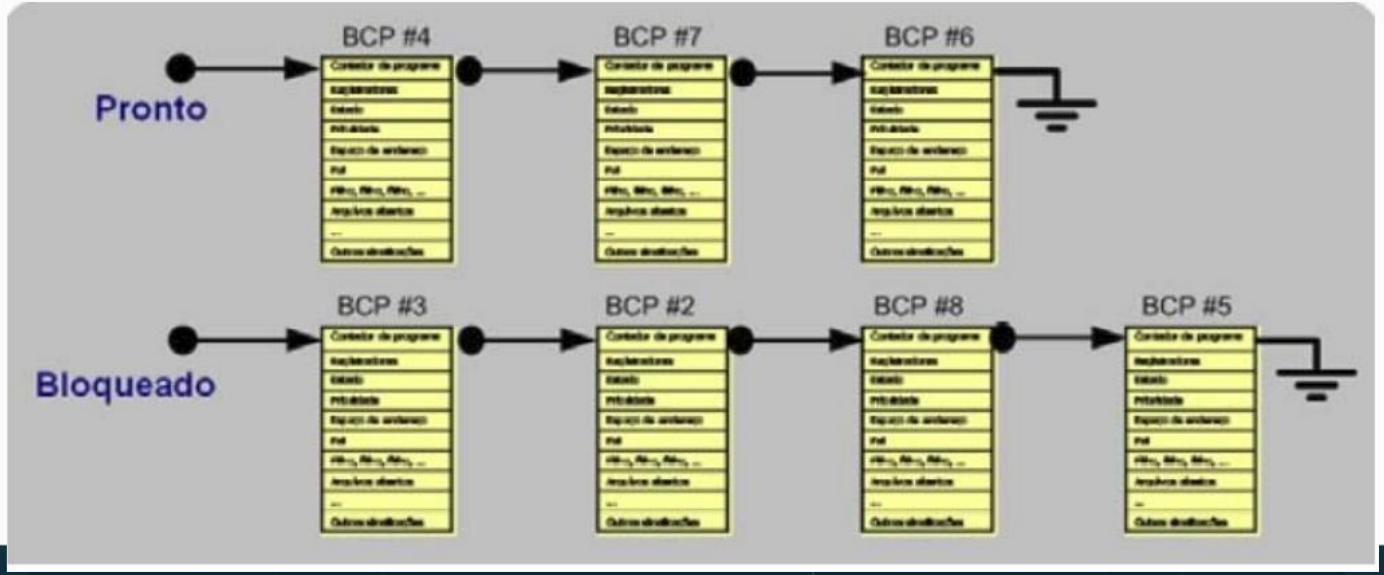
Chaveamento de Contexto (Alternância de Processos)

- SO ao **interromper** um processo:
  - Armazena no BCP as informações de contexto (*hardware*).
  - Permite continuar, mais tarde, exatamente de onde parou.
- SO também **responde a interrupções**:
  - Sinais enviados pelo *hardware* através do barramento.
  - Verificadas após execução de cada instrução (atomicidade).
  - Requerem uma rotina de tratamento apropriada.
  - Interrompem o processo em execução.



Gerenciamento de Filas

- SO gerencia filas de BCPs:
  - Fila de PRONTOS.
  - Fila de BLOQUEADOS.
- Não existe fila de processos em EXECUÇÃO.



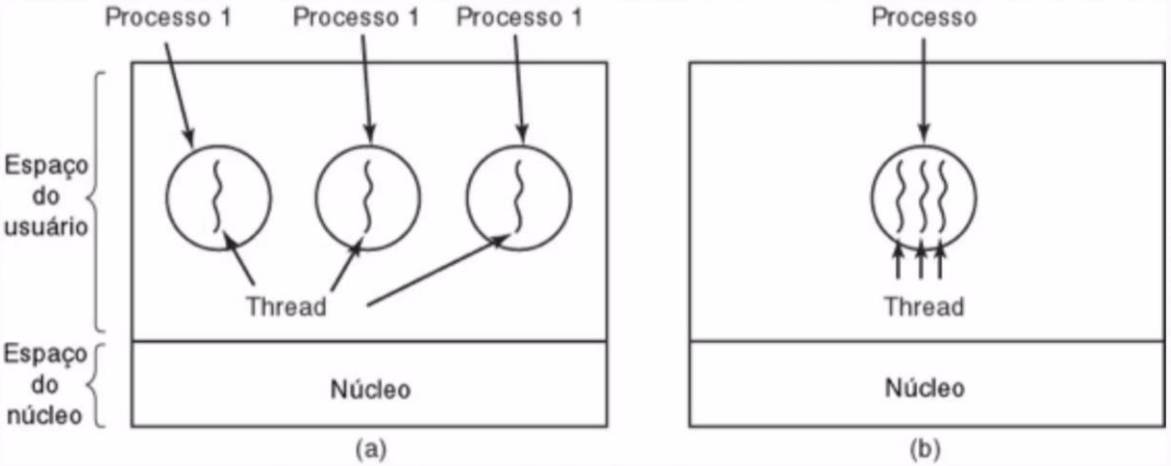
Threads (processos leves)

- Primeiros SO **multiprogramados**: um processo de cada vez.
- SO multiprocessados:
  - Execução simultânea de processos.
  - Em processadores distintos.
- **Antes** (mesmo com multiprocessamento):
  - Em cada processo, **somente uma instrução por vez.**
  - Linguagens não tinham estruturas de paralelismo (C e C++).
- **Hoje:**
  - SO e linguagens de programação evoluíram.
  - **Suportam: execução simultânea em um mesmo processo.**
  - Estruturas de execução paralela: *threads* (processos leves).
- Suporte às tecnologias "*multithreading*":
  - SO modernos (Windows e Linux).
  - Linguagens de última geração (Java e C#).

Threads

- Um processo pode gerar **vários *threads*** (ou *pool de threads*).
- Conhecidos como “processos leves” porque compartilham:
  - Espaços de **endereçamento.**
  - Contextos de **software.**
  - Porém: cada *thread* tem seu próprio contexto de **hardware.**
- Comunicação entre *threads* é mais simples:
  - Criar *threads* gasta menos recursos do que subprocessos.
  - SO pode escalonar processos ou *threads* (para execução).

Threads



(a) Três processos cada um com um thread
(b) Um processo com três threads

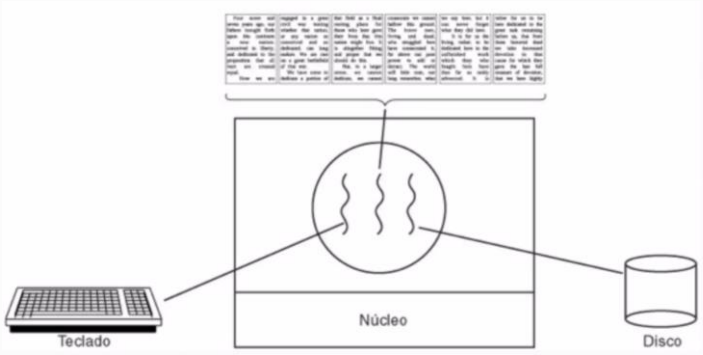
Threads

- Alguns itens são **propriedades de processos**:
  - **Exemplo:** se um *thread* abre um arquivo, esse arquivo fica visível aos outros *threads* no processo e eles podem ler e escrever nele.
  - O processo, e não o *thread*, é a unidade de gerenciamento de recursos.
  - Se cada *thread* tivesse o seu próprio espaço de endereçamento, arquivos abertos, alarmes pendentes e assim por diante, seria um processo em separado.

Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e tratadores de sinais	
Informação de contabilidade	

Aplicações típicas de *Threads*

- Servidor WEB:
  - Cada solicitação de cliente: ativa um *thread* (operária).
- Servidor de BD:
  - Cada consulta de usuário: ativa um *thread* (operária).
- Processador de texto:
  - Uma rotina básica de captura de teclas (principal).
- Com menor prioridade, pode implementar *threads* para:
  - Formatar o texto.
  - Fazer revisão ortográfica.
  - Salvar automaticamente o texto (a cada 15 minutos).



Um processador de texto com três threads

Tipos de Threads

- *Threads* podem ser implementadas de 3 modos distintos:
  - De usuário (muitos para um).
  - De núcleo (um para um).
  - Híbridas (muitos para muitos ou m x n, onde $m \geq n$).
- **De usuário:**
  - Usam bibliotecas carregadas em tempo de execução.
  - São totalmente transparentes para o SO.
  - Escalonamento é feito pela aplicação (modo usuário).
- **De núcleo:**
  - Suportadas diretamente pelo núcleo do SO.
  - Usam uma API para gerenciamento de *threads*.
  - Têm acesso ao *hardware* e às instruções privilegiadas.
- Híbridas: combinam aspectos positivos dos tipos acima.

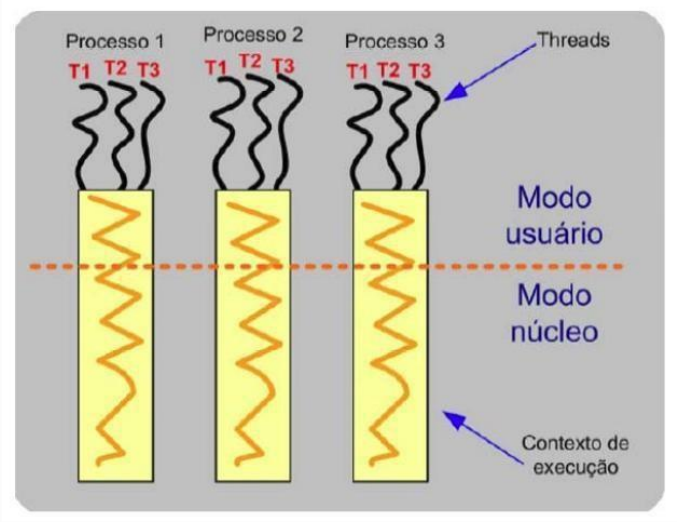
Threads de usuário (não suportadas pelo núcleo)

- Coloca o pacote de **threads** inteiramente no **espaço do usuário**.
  - O núcleo não sabe nada a respeito deles. Até onde o núcleo sabe, ele está gerenciando processos comuns de um único thread.
- Cada processo precisa da sua própria **tabela de threads** privada para controlá-los naquele processo.

- **Vantagens:**
  - Pacote de *threads* no nível do usuário pode ser implementado em um sistema operacional que não dá suporte aos *threads*.
  - Cada processo pode ter seu próprio **algoritmo de escalonamento** customizado.

Threads de usuário (não suportadas pelo núcleo)

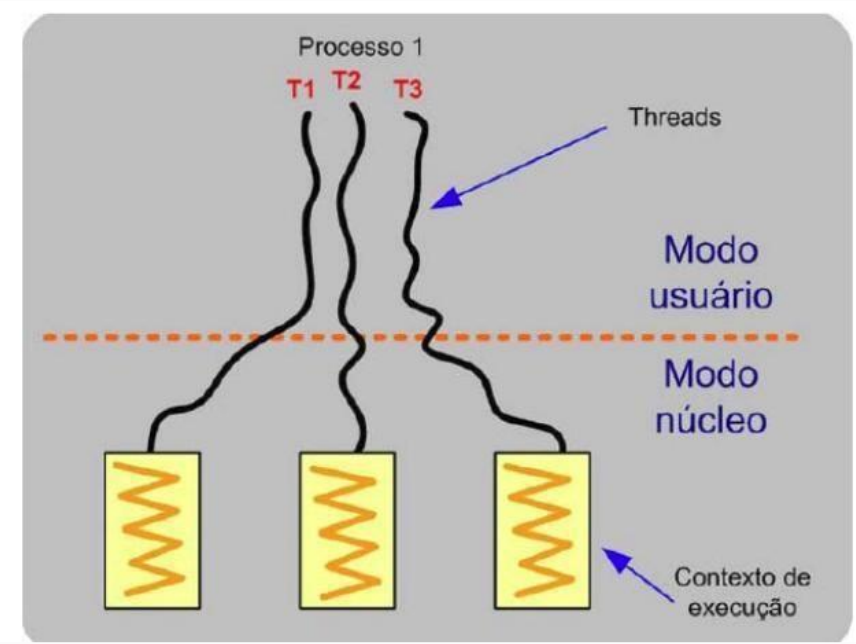
- **Programadores** decidem a rotina de escalonamento.
- Não executam instruções privilegiadas (modo usuário).
- Não podem ser despachadas para outras UCP.
- E/S bloqueante de um *thread* paralisa outras *threads*.



Threads de núcleo (suportadas pelo núcleo)

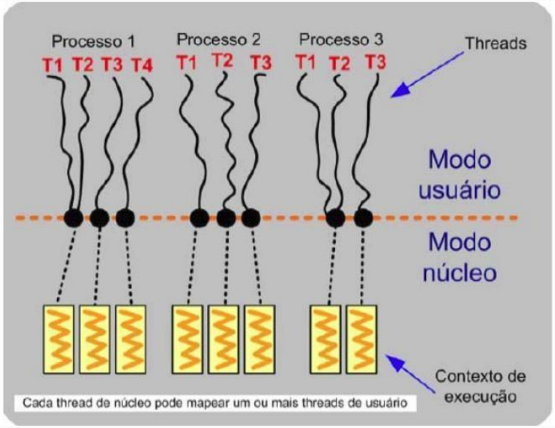
- Núcleo tem uma **tabela** que controla todos os *threads* no sistema.
- Podem executar **instruções privilegiadas** (modo núcleo).
- Podem ser despachadas para outras UCP.
- Ideal para escalonamento por prioridades (SO tempo real).
- **E/S bloqueante de um *thread* não paralisa outros *threads*:** Quando um thread é bloqueado, o núcleo tem a opção de executar outro thread do mesmo processo (se um estiver pronto) ou algum de um processo diferente.
- Aplicativos dependem do suporte pelo SO: menor portabilidade.
- **Gerenciar muitos *threads*:** consumo intenso de recursos.
    - O custo de uma chamada de sistema é substancial, então se as operações de *thread* (criação, término etc.) forem frequentes, ocorrerá uma sobrecarga muito maior.

Threads de núcleo (suportadas pelo núcleo)



Threads Híbridas

- Visa combinar as vantagens dos modos anteriores.
- O programador pode determinar quantos *threads* de núcleo usar e quantos *threads* de usuário multiplexar para cada um.
  - Flexibilidade máxima.
  - Cada *thread* de núcleo tem algum conjunto de *threads* de usuário que se revezam para usá-lo.
- Implementação complexa, sem padronização: menor portabilidade.
- *Threads* com baixo paralelismo: único *thread* (modo núcleo).



Ler!!!

- Capítulo 2 do Livro Texto.

Atividade

- **Atividade no AVA.**