

# SISTEMAS OPERACIONAIS

## AULA 03: PROCESSOS E THREADS

23 de outubro de 2024

Prof. Me. José Paulo Lima

IFPE Garanhuns



**INSTITUTO  
FEDERAL**  
Pernambuco

# SUMÁRIO I



## Processos

- Programa x Processo

- Definição

- Estrutura

- Características

- Criação de processos

- Término de Processos

- Hierarquia

- Estados

# SUMÁRIO II



## *Threads*

Definição

*Threads* x Processos

Características

Modelos de *Threads*

Clássico

*Multithread*

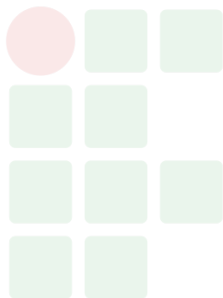
Implementação de *Threads*

*Threads* de usuário

*Threads* de núcleo

## Referências

**PROCESSOS**



**INSTITUTO  
FEDERAL**  
Pernambuco

# PROCESSOS

## PROGRAMA X PROCESSO



- ▶ Um programa é:
  - ▶ Arquivo executável;
    - ▶ Sem atividade.
  - ▶ Uma sequência finita de instruções;
  - ▶ Uma entidade passiva;
    - ▶ Não se altera com o tempo.
  - ▶ Armazenado em disco;
  - ▶ Um programa ou partes de um programa podem ser compartilhados por diversos processos.
    - ▶ Exemplo: biblioteca compartilhadas.
- ▶ Um processo é:
  - ▶ Uma abstração que representa um programa em execução;
    - ▶ É um objeto do SO que suporta a execução dos programas;
    - ▶ Um processo pode executar diversos programas.
  - ▶ Uma entidade dinâmica;
    - ▶ Seu estado se altera conforme for executando.
  - ▶ Armazenado na memória;
  - ▶ Um programa único pode instanciar mais de um processo.

# PROCESSOS

## DEFINIÇÃO



### Definição:

“Um processo é basicamente um programa em execução.” (TANENBAUM; BOS, 2016, p. 27).

- ▶ O processo é a abstração mais importante para o Sistema Operacional;
  - ▶ Abstração de um programa em execução.

# PROCESSOS

## DEFINIÇÃO: PAPEL DO SISTEMA OPERACIONAL



- ▶ Gerenciamento de operações (pseudo)concorrentes mesmo em uma única CPU disponível;
- ▶ Transformam uma única CPU em múltiplas CPUs virtuais;
- ▶ Multiprogramação:
  - ▶ Execução, em paralelo, de múltiplos programas na mesma máquina;
  - ▶ Necessita do suporte a múltiplos processos;
  - ▶ Considerando um grau de tempo fino, o paralelismo não é real;
    - ▶ Pseudoparalelismo ou pseudoconcorrência - implementação de sistemas multiprogramados sobre um computador com um único processador.
- ▶ Paralelismo de *hardware* dos sistemas multiprocessadores.

# PROCESSOS

## DEFINIÇÃO

### ► Processos explícitos:



► Exemplos: Navegador, Programa Processador de texto.

### ► Processos Implícitos:



► Exemplos: Varredura do antivírus, Placa de vídeo.



# PROCESSOS

## ESTADOS

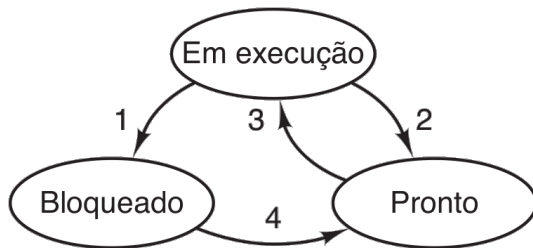


Figura extraída de Tanenbaum e Bos (2016, p. 64).

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

# PROCESSOS

## DEFINIÇÃO

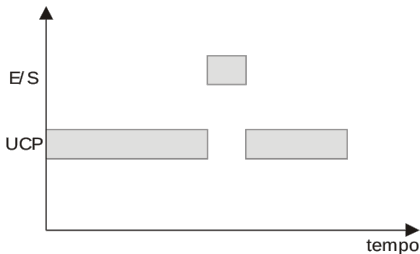


Figura extraída de Machado e Maia (2017, p. 76).

- ▶ *CPU-bound* (ligado à UCP) está maior parte do tempo em execução ou pronto;
  - ▶ Poucas operações de leitura e gravação.

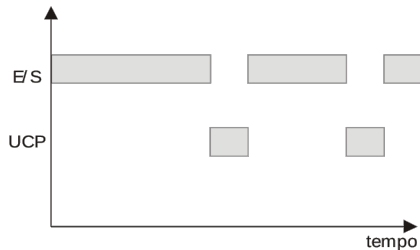


Figura extraída de Machado e Maia (2017, p. 76).

- ▶ *I/O-bound* (ligado à E/S) passa a maior parte do tempo no estado de espera;
  - ▶ Realiza um elevado número de operações e E/S.

# PROCESSOS

## DEFINIÇÃO



### Outra definição:

“Processos são uma das mais antigas e importantes abstrações que os sistemas operacionais proporcionam. Eles dão suporte à possibilidade de haver operações (pseudo) concorrentes mesmo quando há apenas uma CPU disponível, transformando uma única CPU em múltiplas CPUs virtuais.” (TANENBAUM; BOS, 2016, p. 59).

# PROCESSOS

## ESTRUTURA

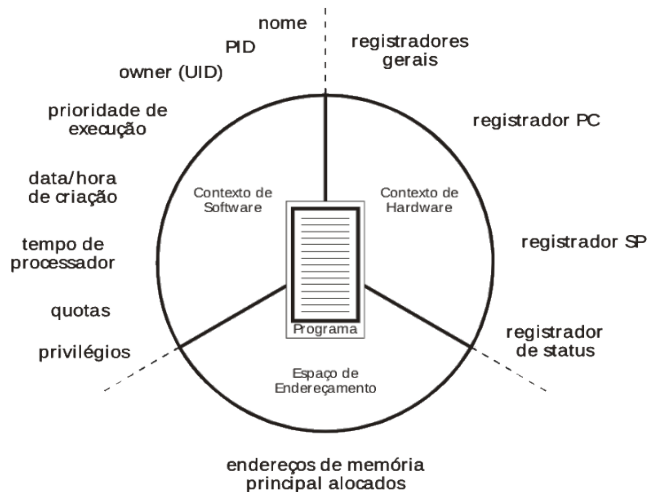


Figura extraída de Machado e Maia (2017, p. 64).

# PROCESSO

## ESTRUTURA



- ▶ Espaço de endereçamento (virtual):
  - ▶ Conjunto de posições de memória acessíveis;
  - ▶ Código, dados, e pilha;
  - ▶ Dimensão variável.
- ▶ Contexto de *software*:
  - ▶ As instruções do processador executáveis em modo usuário;
  - ▶ As funções do sistema operacional.
- ▶ Contexto de *hardware* (estado interno):
  - ▶ Valor dos registradores do processador;
  - ▶ Toda a informação necessária para retomar a execução do processo;
  - ▶ Memorizado quando o processo é retirado de execução.

# PROCESSOS

## CARACTERÍSTICAS

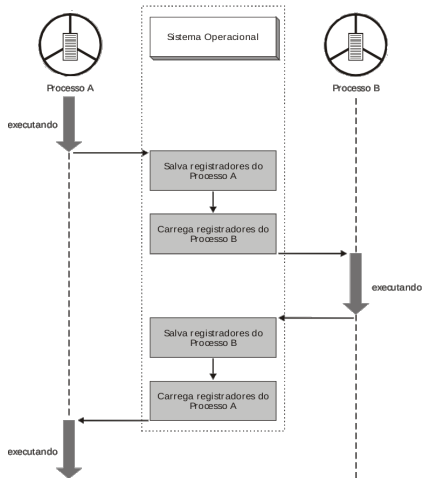


Figura extraída de Machado e Maia (2017, p. 64).

### ► Propriedades:

- Identificador;
- Programa;
- Endereçamento;
- Prioridade;
- Processo pai;
- Canais de E/S, Arquivos;
- Quota de uso de recursos;
- Contexto de Segurança.

### ► Operações:

- Funções que atuam sobre os processos:
  - Criar;
  - Eliminar;
  - Esperar pelo término de subprocessos.

# PROCESSOS

## CARACTERÍSTICAS



- ▶ Imagem de um programa:
  - ▶ Segmento de código.
- ▶ Conjunto de recursos de *hardware* alocados pelo SO:
  - ▶ Registradores (PC, *Stack Pointer*...);
  - ▶ Espaço de endereçamento (memória);
  - ▶ Espaço no disco (arquivos de E/S).
- ▶ Unidade de escalonamento:
  - ▶ Estado;
  - ▶ Algoritmos de escalonamento para otimizar o uso do *hardware*;
  - ▶ Alocar a CPU a um processo implica em uma troca de contexto.

# PROCESSOS

## MODELO DE PROCESSO

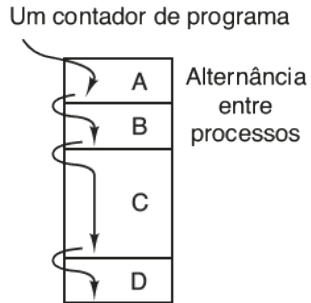


- ▶ Os *softwares* que podem ser executados são organizados em vários processos sequenciais (processos);
- ▶ O processo é um programa em execução, acompanhado:
  - ▶ Dos valores atuais do contador do programa;
  - ▶ Dos registradores;
  - ▶ E das variáveis;
- ▶ Multiprogramação é a troca rápida de um processo por outro realizado pela CPU.



# PROCESSOS

## MODELO DE PROCESSO



Na figura vemos um computador multiprogramando quatro programas na memória.

Figura extraída de Tanenbaum e Bos (2016, p. 60).

# PROCESSOS

## MODELO DE PROCESSO

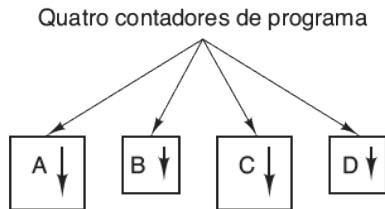


Figura extraída de Tanenbaum e Bos (2016, p. 60).

Na figura vemos quatro processos, cada um com seu próprio fluxo de controle e sendo executado independente dos outros.

# PROCESSOS

## MODELO DE PROCESSO

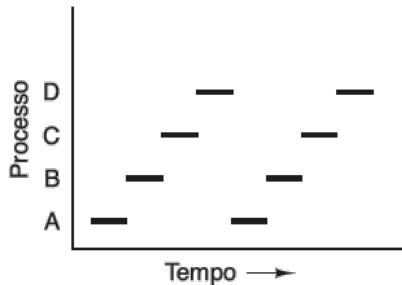


Figura extraída de Tanenbaum e Bos (2016, p. 60).

Na figura vemos que, analisados durante um intervalo longo o suficiente, todos os processos tiveram progresso, mas a qualquer dado instante apenas um está sendo de fato executado.

# PROCESSOS

## CRIAÇÃO DE PROCESSOS



- ▶ Sistemas simples ou *hard realtime systems*:
  - ▶ Todos os processos que serão necessários são criados quando o sistema é ligado;
  - ▶ Exemplo: Forno de micro-ondas
- ▶ Sistemas de propósito geral
  - ▶ É necessário algum mecanismo para criar e terminar processos durante a operação;
  - ▶ Processos ficam em primeiro plano interagindo com o usuário e realizam tarefas;
  - ▶ Outros processos ficam em segundo plano que apresentam alguma função específica;
    - ▶ Exemplo: Solicitação de serviço.

# PROCESSOS

## CRIAÇÃO DE PROCESSOS



Quatro eventos principais fazem com que os processos sejam criados:

1. Inicialização do sistema;
  - ▶ Um processo em execução fará chamadas do sistema (*System Calls*) para criar novos processos.
2. Execução de uma chamada de sistema de criação de processo por um processo em execução;
  - ▶ Sistemas multiprocessadores, permite que cada processo execute em uma CPU diferente.
3. Solicitação de um usuário para criar um novo processo;
  - ▶ Criação de processos em sistemas interativos ocorre digitando comandos ou clicando em ícones.
4. Início de uma tarefa em lote (*batch*).
  - ▶ Em sistemas de Lote, o sistema operacional criará um novo processo quando julgar que tem recursos para executar, caso tenha, executará a próxima tarefa da fila de entrada.

# PROCESSOS

## TÉRMINO DE PROCESSOS



Condições para término de um processo:

1. Saída normal (Voluntária);
  - ▶ Exemplos:
    - ▶ `Exit` (Linux);
    - ▶ `ExitProcess` (Win32);
    - ▶ O ícone ✕ das janelas dos programas.
2. Saída por erro (Voluntária)
  - ▶ Erro de compilação.
  - ▶ Não abriu um arquivo.
3. Erro fatal (Involuntária)
  - ▶ Divisão por zero.
4. Cancelamento por outro processo (Involuntário).

# PROCESSOS

## HIERARQUIA



- ▶ Um processo cria um ou mais processos, formando uma hierarquia entre os processos;
  - ▶ Cada um tem seus próprios espaços de endereçamento distintos;
  - ▶ Linux:
    - ▶ É possível que um processo filho compartilhe algum de seus recursos com o processo que o criou.
  - ▶ Windows:
    - ▶ Pai e filhos separados desde o início;
    - ▶ Não há hierarquia de processos, todos os processos são criados iguais;
    - ▶ O processo Pai tem um identificador (Handle) que pode usar para controlar o filho;
    - ▶ O processo Pai passa esse identificador para outros processos.

# PROCESSOS

## BLOCO DE CONTROLE DE PROCESSO

<b>Ponteiros</b>
<b>Estado do processo</b>
<b>Nome do processo</b>
<b>Prioridade do processo</b>
<b>Registradores</b>
<b>Limites de memória</b>
<b>Lista de arquivos abertos</b>
⋮

Figura extraída de Machado e Maia (2017, p. 71).

“O processo é implementado pelo sistema operacional através de uma estrutura de dados chamada bloco de controle do processo (*Process Control Block* - PCB). A partir do PCB, o sistema operacional mantém todas as informações sobre o contexto de *hardware*, contexto de *software* e espaço de endereçamento de cada processo.” (MACHADO; MAIA, 2017, p. 66).

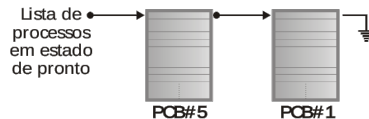


# PROCESSOS

## BLOCO DE CONTROLE DE PROCESSO: ESTRUTURA

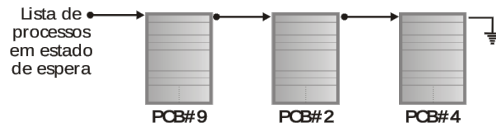
- ▶ Contém as informações necessárias:
  - ▶ Registradores, memória, disco (arquivos);
  - ▶ Prioridade;
  - ▶ Estado;
  - ▶ Histórico (contabilidade);
  - ▶ Ponteiro para um outro PCB (lista encadeada).
- ▶ O SO deve manter listas de processos:

- ▶ Listas encadeadas;



- ▶ A estrutura PCB é usada para tal;

- ▶ Mantém-se um ponteiro sobre o primeiro e/ou o último PCB.



# PROCESSOS

## ESTADOS



Os processos podem ter os seguintes estados:

1. Criado (*new*);

- ▶ “Um processo é dito no estado de criação quando o sistema operacional já criou um novo PCB, porém ainda não pode colocá-lo na lista de processos do estado de pronto.” (MACHADO; MAIA, 2017, p. 71).

2. Pronto (*ready*);

- ▶ “Um processo está no estado de pronto quando aguarda apenas para ser executado.” (MACHADO; MAIA, 2017, p. 67).

3. Execução (*running*);

- ▶ “Um processo é dito no estado de execução quando está sendo processado pela UCP. [...] Os processos se alternam na utilização do processador seguindo uma política estabelecida pelo sistema operacional.” (MACHADO; MAIA, 2017, p. 67).

# PROCESSOS

## ESTADOS



Os processos podem ter os seguintes estados:

4. Bloqueado ou Em espera (*wait*);

- ▶ “Um processo no estado de espera aguarda por algum evento externo ou por algum recurso para prosseguir seu processamento.” (MACHADO; MAIA, 2017, p. 68).

5. Encerrado ou Término (*exit*);

- ▶ “Um processo neste estado não é considerado mais ativo, mas como o PCB ainda existe, o sistema operacional pode recuperar informações sobre a contabilização de uso de recursos do processo [...]”. (MACHADO; MAIA, 2017, p. 71).

# PROCESSOS

## ESTADOS



### 6. Suspenso (*suspend*).

- ▶ “Um processo em estado de pronto ou de espera pode não se encontrar na memória principal. Esta condição ocorre quando não existe espaço suficiente para todos os processos na memória principal e parte do contexto do processo é levado para memória secundária”. (MACHADO; MAIA, 2017, p. 74–75).
  - ▶ A técnica conhecida como *swapping*, na condição citada, retira processos da memória principal (*swap out*) e os traz de volta (*swap in*) seguindo critérios de cada sistema operacional.
  - ▶ Neste caso, os processos em estados de espera e pronto podem estar residentes ou não residentes (*outswapped*) na memória principal.

# PROCESSOS

## ESTADOS

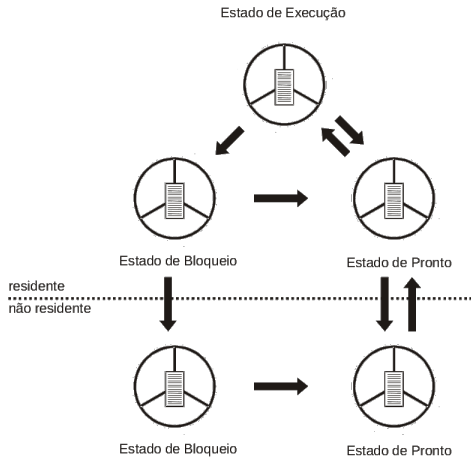


Figura extraída de Machado e Maia (2017, p. 74).

- ▶ **Bloqueado/Suspenso:**
  - ▶ O processo está na memória secundária, esperando por um evento.
- ▶ **Pronto/Suspenso:**
  - ▶ O processo está na memória secundária, mas está disponível para a execução.

# PROCESSOS

## ESTADOS

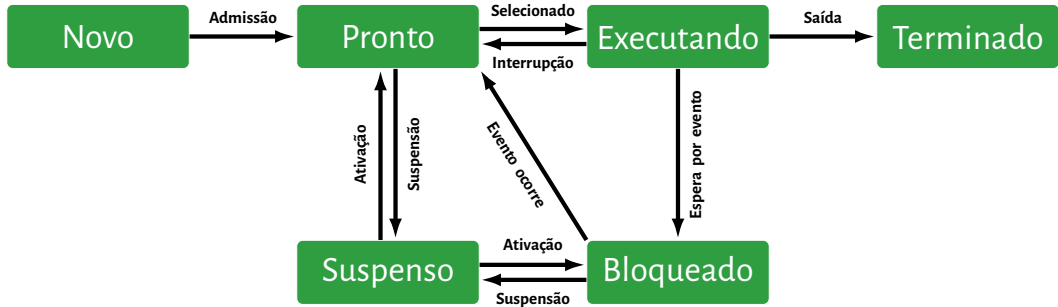


Figura desenvolvida pelo professor com base em Machado e Maia (2017) e Tanenbaum e Bos (2016).

# PROCESSOS

## ESTADOS

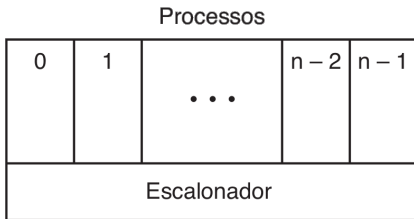
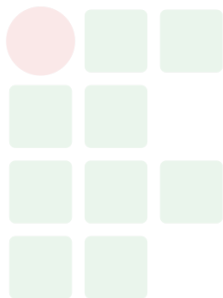


Figura extraída de Tanenbaum e Bos (2016, p. 65).

- ▶ Troca de processos é executada por um escalonador;
  - ▶ Exceção: processo solicita bloqueio de si mesmo ao SO;
  - ▶ Exemplo: pause (Linux).

***THREADS***



**INSTITUTO  
FEDERAL**  
Pernambuco



# THREADS

## DEFINIÇÃO



### Definição

“De forma simplificada, um *thread* pode ser definido como uma sub-rotina de um programa que pode ser executada de forma assíncrona, ou seja, executada paralelamente ao programa chamador.” (MACHADO; MAIA, 2017, p. 86).

# THREADS

## THREADS X PROCESSOS



- ▶ *Threads* são “linhas” de execução de um processo (programa em execução);
- ▶ *Threads* podem ser executadas em paralelo, assim como processos;
  - ▶ *Threads* compartilham o mesmo espaço da memória, o que torna possível a fácil comunicação entre elas.
    - ▶ Isto não é possível com processos.
- ▶ *Threads* são muito mais rapidamente criadas;
  - ▶ Não possuem recursos associados.
    - ▶ Demanda dinâmica de *threads*.
- ▶ Escalonamento de *threads* é menos custoso.
  - ▶ Muitas requisições de E/S.

# THREADS

## CARACTERÍSTICAS



- ▶ Muitas aplicações ocorrem múltiplas atividades simultaneamente e algumas dessas atividades podem ser bloqueadas de tempos e tempos;
  - ▶ Exemplo - Leitor de *feed*:
    - ▶ Fazendo *download* de imagens (*thumbnails*) enquanto exibe notícia escolhida.
- ▶ O modelo de programação se torna mais simples;
  - ▶ Decompondo-se a aplicação em múltiplos *threads* sequenciais que executam em paralelo.
- ▶ São mais fáceis de criar e destruir que os processos:
  - ▶ Não tem recursos associados a eles;
  - ▶ Chamadas de processos leves (*Light weight Process*);
  - ▶ Em alguns sistemas, criar *thread* é até cem vezes mais rápido do que criar um processo.
- ▶ *Threads* são úteis em sistemas com múltiplas CPUs, onde, o paralelismo real é possível.

# THREADS

## EXEMPLO: EDITOR DE TEXTO

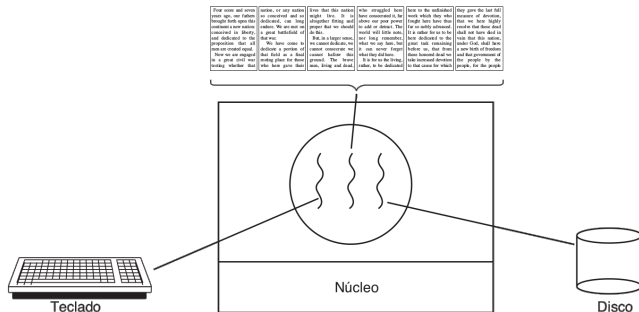


Figura extraída de Tanenbaum e Bos (2016, p. 68).

- Se o programa tivesse apenas uma *thread*, então, sempre que um *backup* de disco se iniciasse, os comandos do teclado e do mouse seriam ignorados enquanto o *backup* não terminasse.

# THREADS

## MODELO DE *THREAD*: CLÁSSICO

- ▶ O modelo de processos é baseado em:
  - ▶ Agrupamento de recursos;
    - ▶ Mais fácil de gerenciar;
    - ▶ Arquivos;
    - ▶ Processos filhos (hierarquia).
  - ▶ Execução;
    - ▶ Cada *thread* tem seus Registradores e Pilha;
    - ▶ Execução independente das demais.

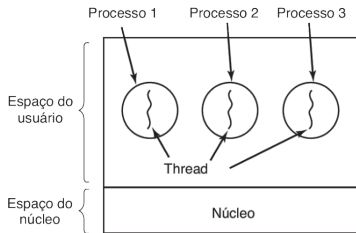


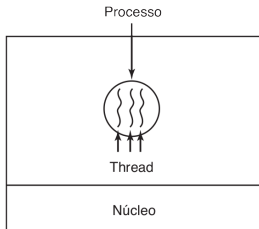
Figura extraída de Tanenbaum e Bos (2016, p. 71).

- ▶ Três processos tradicionais;
- ▶ Cada processo tem seu espaço de endereçamento e um único *thread* de controle;
- ▶ Cada um deles opera num espaço de endereçamento.

# THREADS

## MODELOS DE *THREAD*: *MULTITHREAD*

- ▶ Processo com mais de uma *thread*:
  - ▶ É iniciado com uma *thread* e, a partir desta, outras são criadas.
    - ▶ Normalmente, não existe uma relação de pai e filho entre *threads*.
  - ▶ Podem ser encerradas numa chamada de rotina de biblioteca:
    - ▶ A execução de uma *thread* pode depender do encerramento de outra *thread* dependente.
  - ▶ *Threads* podem ceder tempo na CPU voluntariamente para outras *threads* (não existem interrupções para *threads*).
- ▶ Problemas inerentes dos processos *multithread*: Concorrência.



- ▶ Um único processo com três *threads* de controle;
- ▶ Todas as três compartilham o mesmo espaço de endereçamento.

# THREADS

## MODELOS DE THREAD: MULTITHREAD

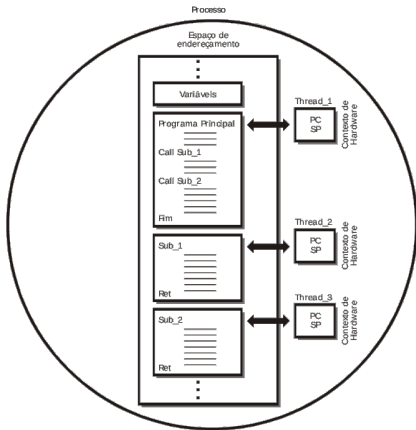


Figura extraída de Machado e Maia (2017, p. 86).

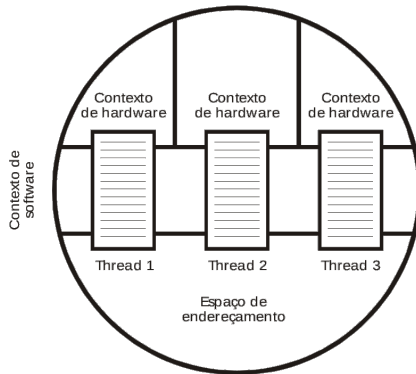


Figura extraída de Machado e Maia (2017, p. 85).

# IMPLEMENTAÇÃO DE *THREADS*

## *THREADS* DE USUÁRIO



- ▶ Executam no topo de um sistema denominado sistema de tempo de execução (*runtime*);
- ▶ Vantagem:
  - ▶ O pacote de *threads* pode ser implementado em um sistema operacional que não suportam *threads*.
- ▶ Para o gerenciamento das *threads* (mudança de estados), cada processo precisa de sua tabela de *threads* para controlar;
- ▶ Problemas:
  - ▶ Chamadas bloqueantes atuam em todas as *threads*;
    - ▶ *Threads* não devem fazer chamada de sistema.
    - ▶ A chamada de sistema pode ser “envolta” por um mecanismo de previsão de ocorrência ou não de bloqueio (*jacket* ou *wrapper*).
  - ▶ Não há interrupções para *threads* (como dito anteriormente);
  - ▶ Processos *multithreads* fazem, normalmente, muitas chamadas bloqueantes.



# IMPLEMENTAÇÃO DE *THREADS*

## *THREADS* DE USUÁRIO

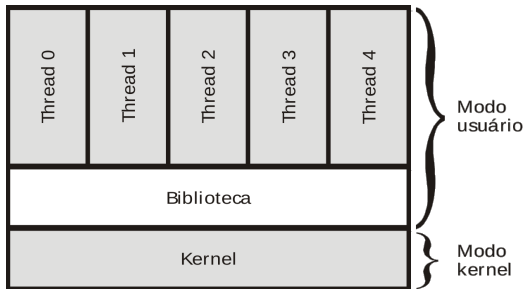


Figura extraída de Machado e Maia (2017, p. 91).

- ▶ Todos os *threads* são inseridas totalmente no espaço usuário;
  - ▶ O núcleo não é informado sobre eles;
    - ▶ Núcleo cuida apenas de processos *monothread*;
    - ▶ “Vê” apenas o processo.
- ▶ *Threads* implementadas por bibliotecas;
- ▶ *Threads* executam em um ambiente de execução.

# IMPLEMENTAÇÃO DE *THREADS*

## *THREADS* DE USUÁRIO

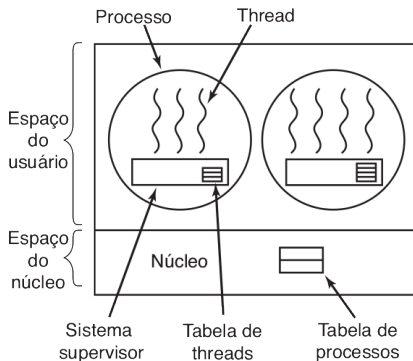


Figura extraída de Tanenbaum e Bos (2016, p. 75).

- ▶ Possuem uma tabela de *threads*;
- ▶ Manter controle das propriedades das *threads*;
- ▶ Chavear *threads* no espaço de usuário é mais rápido do que chavear no núcleo;
  - ▶ Mais rápido do que chaveamento de processos.
- ▶ Cada um pode ter sua própria rotina de escalonamento.

# IMPLEMENTAÇÃO DE *THREADS*

## *THREADS* DE NÚCLEO

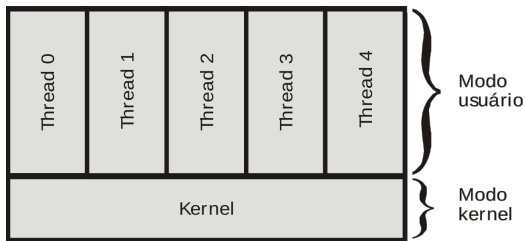


Figura extraída de Machado e Maia (2017, p. 92).

- ▶ Não há necessidade um sistema de tempo de execução (*runtime*);
- ▶ Tabela de *threads* tem o mesmas informações que a tabela de *threads* no modo usuário;
- ▶ Abordagem mais comum.

# IMPLEMENTAÇÃO DE *THREADS*

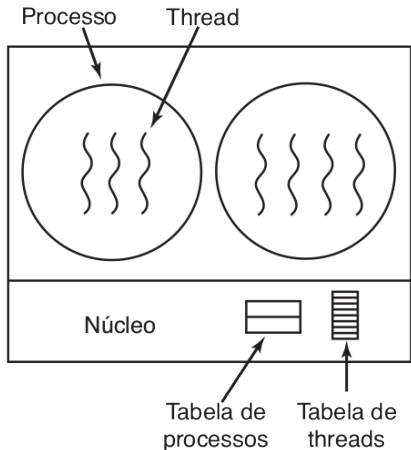
## *THREADS* DE NÚCLEO



- ▶ Não é necessário um sistema de tempo de execução para gerenciar as *threads*;
- ▶ Processo de criar e destruir *threads* pelo núcleo é caro:
  - ▶ Abordagem “ecologicamente correta”: reciclagem de *threads*!
- ▶ Não precisa se preocupar com chamadas bloqueantes:
  - ▶ Outras *threads* são escolhidas pelo núcleo, podendo ser do próprio processo.

# IMPLEMENTAÇÃO DE *THREADS*

## *THREADS* DE NÚCLEO



- ▶ Não precisa de sua tabela de *threads* para controle no processo;
- ▶ Existe uma tabela de *threads* que acompanha todas as *threads* no sistema;

# IMPLEMENTAÇÃO DE *THREADS*

## HÍBRIDA

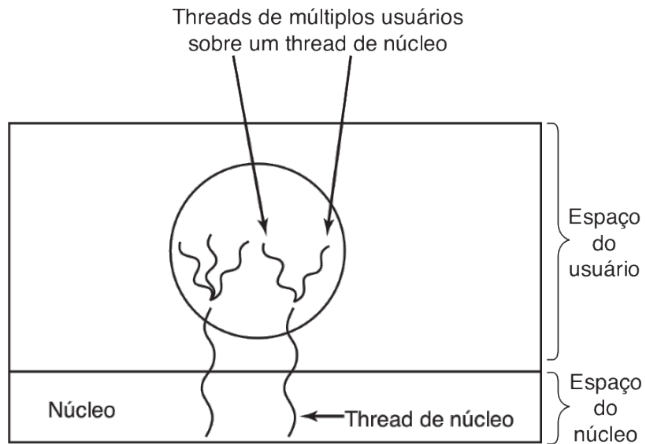


Figura extraída de Tanenbaum e Bos (2016, p. 78).

# IMPLEMENTAÇÃO DE *THREADS*

## HÍBRIDA

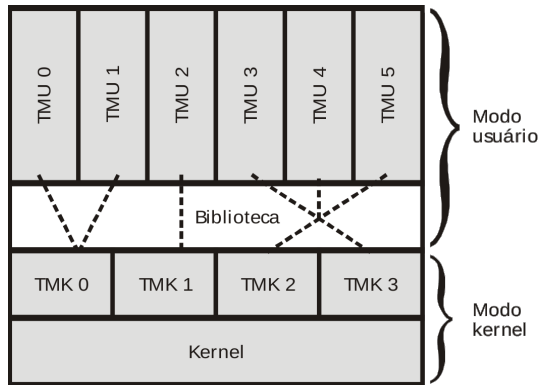
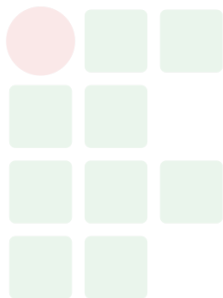


Figura extraída de Machado e Maia (2017, p. 93).

## REFERÊNCIAS




INSTITUTO  
FEDERAL  
Pernambuco

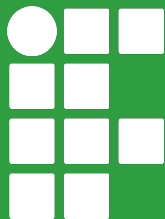


# REFERÊNCIAS I



 MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais**. 5. ed. Rio de Janeiro: LTC, 2017. ISBN 978-85-216-2210-9.

 TANENBAUM, Andrew S.; BOS, Herbert. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2016.



**INSTITUTO FEDERAL**

Pernambuco