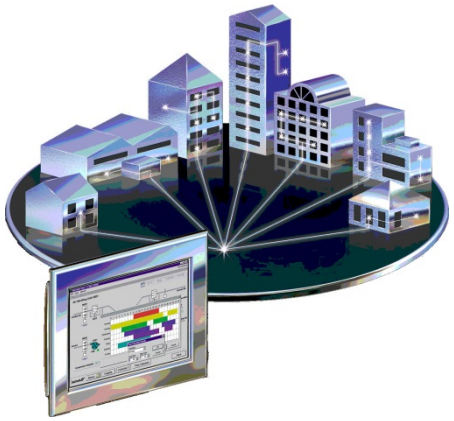
Revisão de Processos



Programação Paralela Avançada - PPA

Mestrado em Computação Aplicação – MCA
Programa de Pós-Graduação em Computação Aplicada – PPGCA
Centro de Ciências Tecnólogicas - CCT
Universidade do Estado de Santa Catarina – UDESC

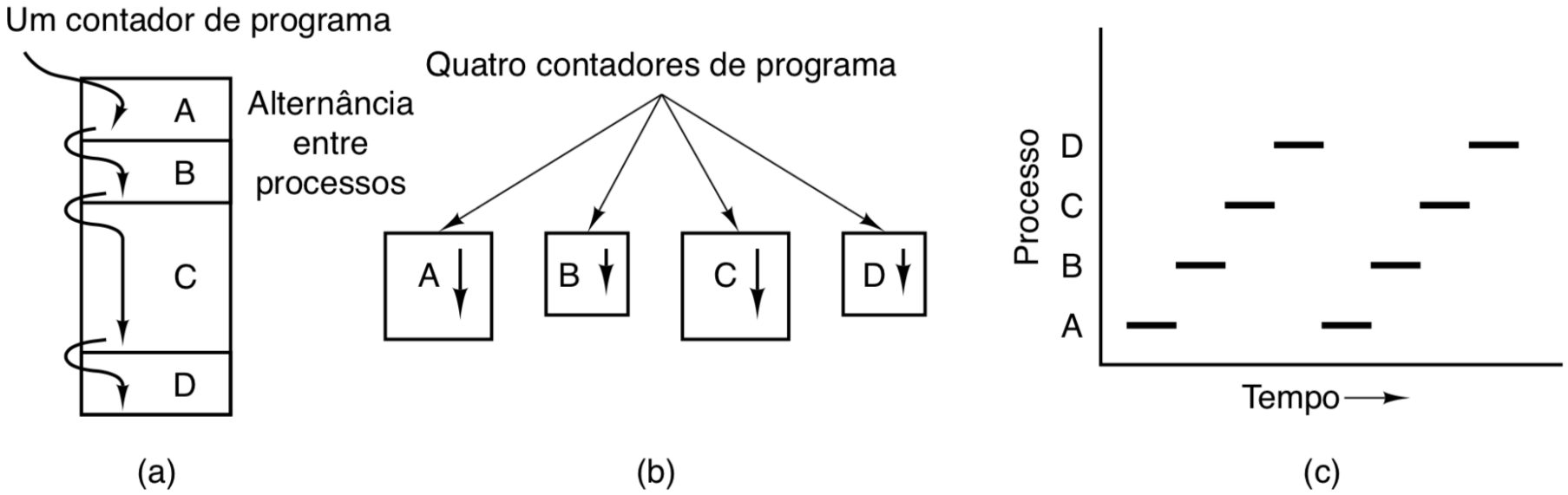
Profs Maurício A. Pillon e Guilherme P. Koslovski
Linha de Sistemas Computacionais
Grupo de Pesquisa de Redes de Computadore e Sistemas Distribuídos
Laboratório de Pesquisa LabP2D

Processos

Definição de Processos

- Um **processo** é um **programa em execução**
  - registradores, contador de programa, pilha
  - cada processo enxerga uma CPU virtual
- **Multiprogramação**: vários processos carregados na memória ao mesmo tempo
  - máquinas monoprocessadas: apenas um processo executa de cada vez
    - **pseudoparalelismo**
  - máquinas multiprocessadas: paralelismo real

Multiprogramação de quatro processos



- Processos não devem fazer hipóteses temporais ou sobre a ordem de execução
  - primitivas de sincronização

Criação de processos no Unix

UDESC

- No UNIX, processos são criados através da chamada *fork*
- O processo filho é idêntico ao processo pai:
  - código e dados são copiados
  - diferença está no valor de retorno da função
  - a chamada *exec* substitui o processo corrente

```
pid = fork();
if (pid == 0)              /* processo filho */
   execv("/bin/prog", args);
else                       /* processo pai */
   w = waitpid(pid, &status, 0);
```

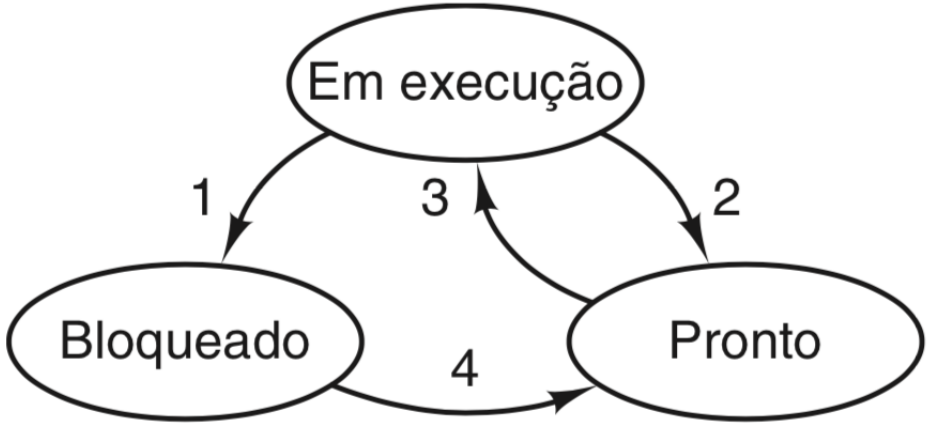
Hierarquia de processos

- Processos "procriam" por várias gerações
  - um processo pai cria processos filhos, que por sua vez também criam seus filhos, *ad nauseam*
- Leva à formação de **hierarquias** de processos
- Chamadas "grupos de processos" no UNIX
  - sinalizações de eventos se propagam através do grupo, e cada processo decide o que fazer com o sinal (ignorar, tratar ou "ser morto")
  - todos os processos UNIX descendem de *init*
- Windows não possui hierarquias de processos
  - todos os processos são criados iguais

Estado de um processo

- Um processo pode assumir diversos estados no sistema
  - **em execução**: processo que está usando a CPU
  - **pronto**: processo temporariamente parado enquanto outro processo executa
    - fila de prontos (aptos)
  - **bloqueado**: esperando por um evento externo

Transições de estado de um processo



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 entram no sistema na fila de prontos
- Transições dependem de interrupções para sinalizar condições
  - término de operações de E/S, passagem do tempo, ...

Implementação de processos

- As informações sobre os processos do sistema são armazenadas na **tabela de processos**
  - cada entrada é chamada de **descritor de processo** ou **bloco de controle de processo**

Gerenciamento de processos	**Gerenciamento de memória**	**Gerenciamento de arquivos**
Registradores	Ponteiro para o segmento de código	Diretório-raiz
Contador de programa	Ponteiro para o segmento de dados	Diretório de trabalho
Palavra de estado do programa	Ponteiro para o segmento de pilha	Descritores de arquivos
Ponteiro de pilha		Identificador (ID) do usuário
Estado do processo		Identificador (ID) do grupo
Prioridade		
Parâmetros de escalonamento		
Identificador (ID) do processo		
Processo pai		
Grupo do processo		
Sinais		
Momento em que o processo iniciou		
Tempo usado da CPU		
Tempo de CPU do filho		
Momento do próximo alarme		

Processos Leves (*Threads*)

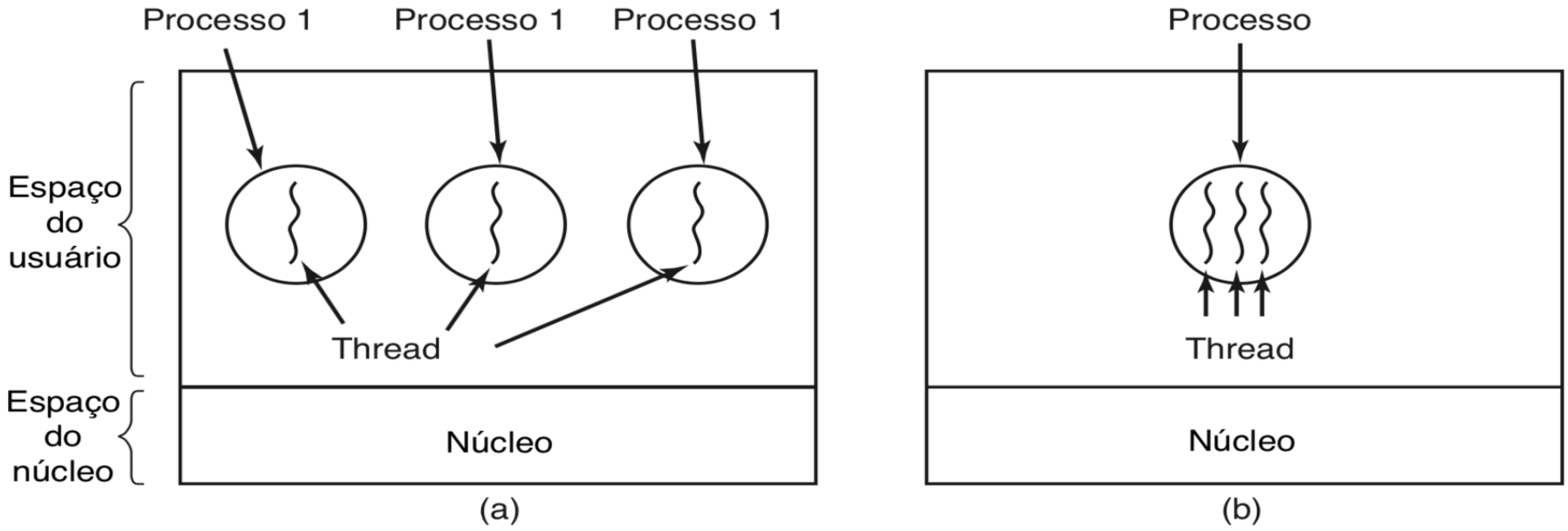
Threads

UDESC

- Processos possuem
  - um espaço de endereçamento
  - uma thread de execução ou fluxo de controle
- Processos agrupam recursos
  - espaço de endereçamento (código+dados), arquivos, processos filhos, alarmes pendentes, ...
  - esse agrupamento facilita o gerenciamento
- A thread representa o estado atual de execução
  - contador de programa, registradores, pilha
- A unificação é uma conveniência, não um requisito

Threads

- Múltiplas threads em um processo permitem execuções paralelas sobre os mesmos recursos
  - análogo a vários processos em paralelo
- Processos leves ou multithread



(a) 3 processos com uma thread

(b) Um processo com 3 threads

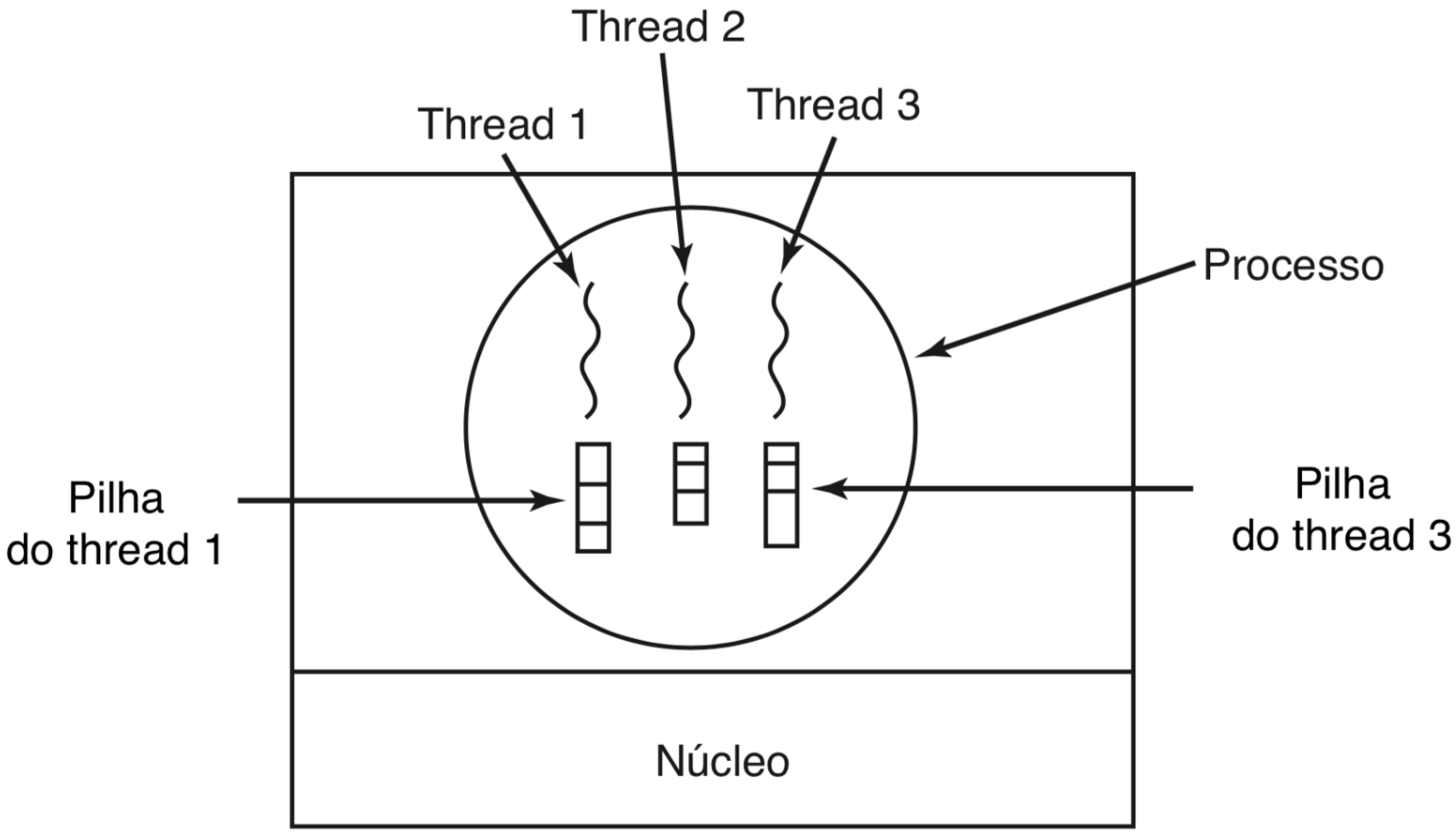
Threads: compartilhamento de recursos

- As várias threads de um processo compartilham muitos dos recursos do processo
  - **não existe proteção entre threads**

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	

Threads: compartilhamento de recursos

- Cada thread precisa da sua própria pilha
  - mantém suas variáveis locais e histórico de execução



Threads: Problemas

UDESC

- Complicações no modelo de programação
  - um processo filho herda todas as threads do processo pai?
  - se herdar, o que acontece quando a thread do pai bloqueia por um entrada de teclado?
- Complicações pelos recursos compartilhados
  - e se uma thread fecha um arquivo que está sendo usado por outra?
  - e se uma thread começa uma alocação de memória e é substituída por outra?

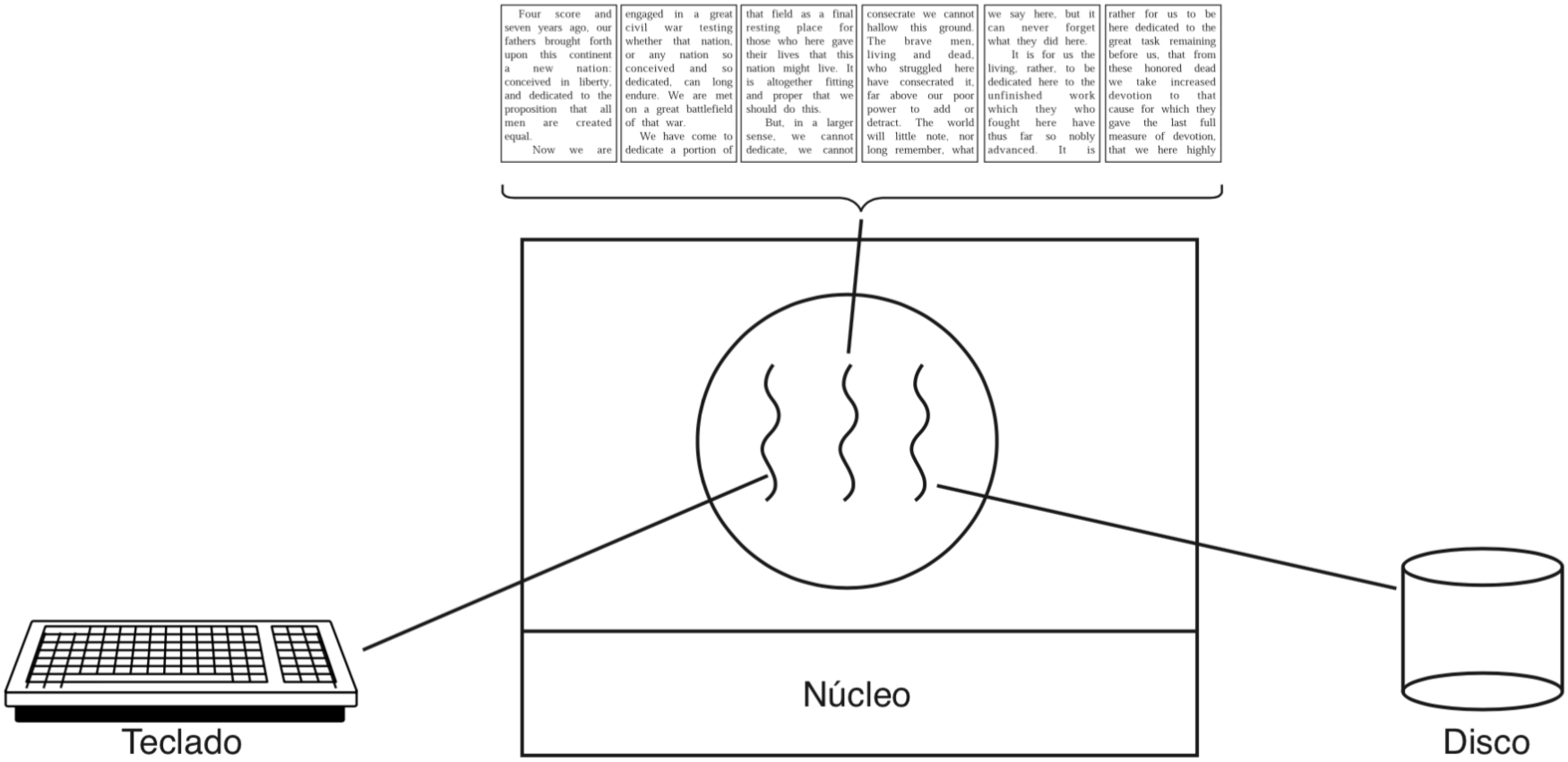
Threads: Vantagens

UDESC

- Possibilitar soluções paralelas para problemas
  - cada thread sequencial se preocupa com uma parte do problema
  - interessante em aplicações dirigidas a eventos
- Desempenho
  - criar e destruir threads é mais rápido
  - o chaveamento de contexto é muito mais rápido
  - permite combinar threads I/O-bound e CPU-bound

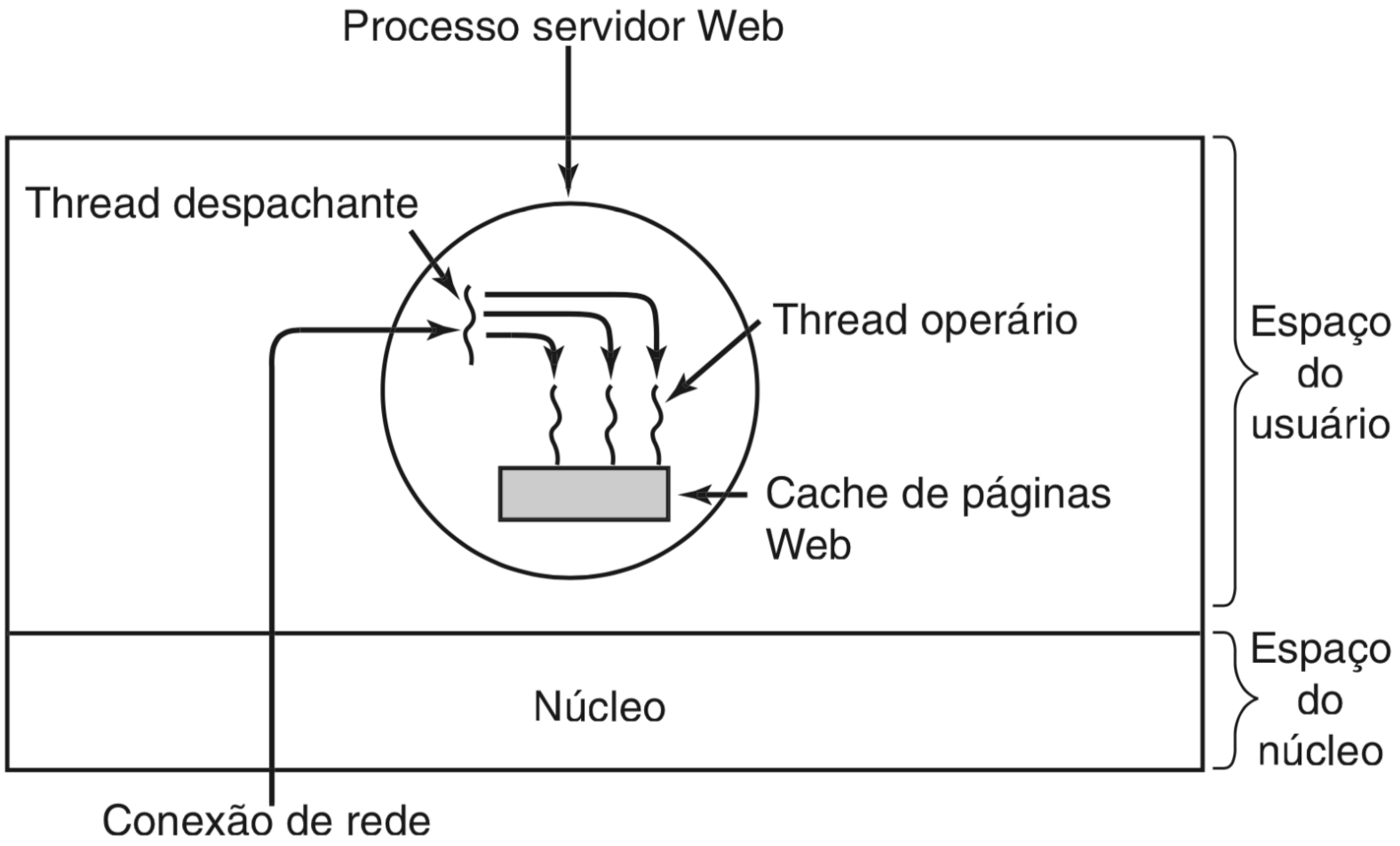
Threads: Exemplos

- Processador de texto com 3 threads
  - considere a implementação monothread



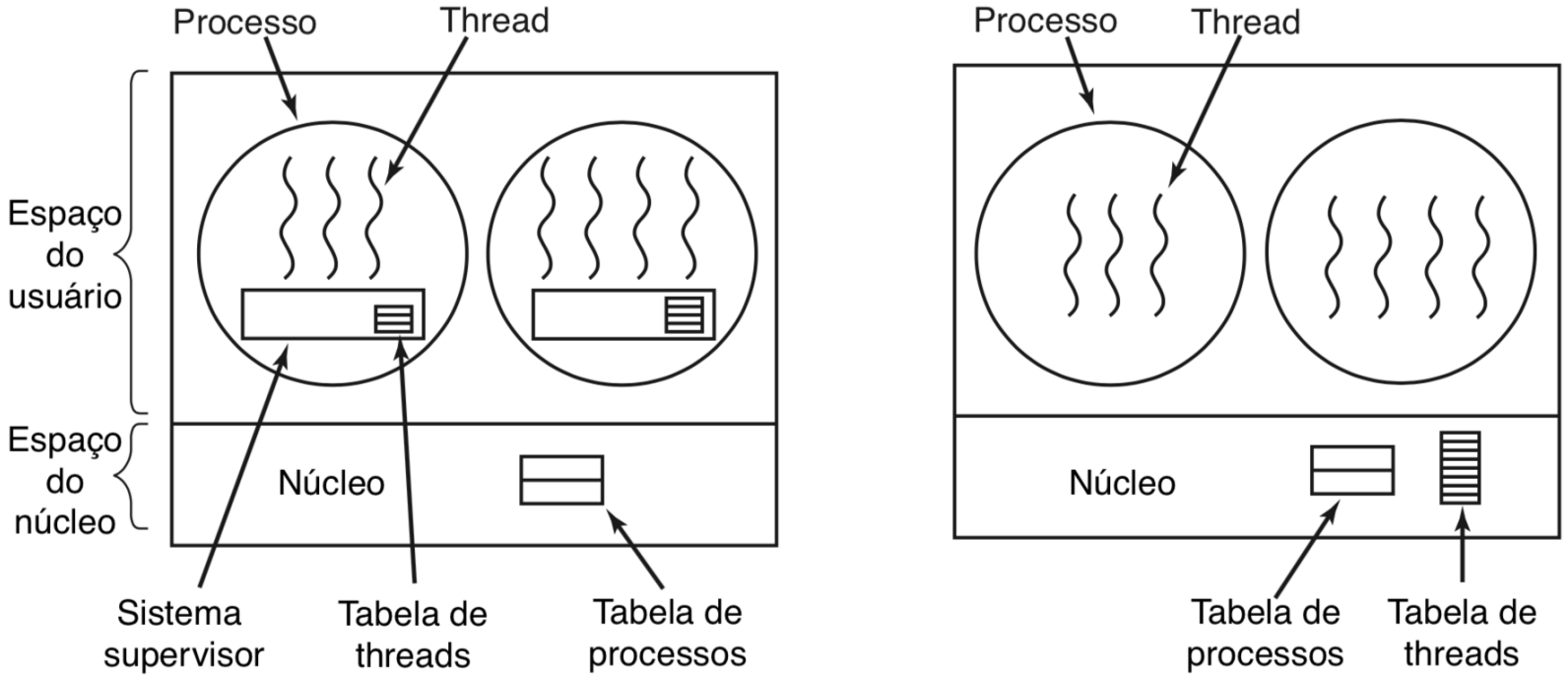
Threads: Exemplos

- Servidor web multithread



Threads: Implementação

- Existem dois modos principais de se implementar threads
  - (a) threads no espaço do usuário (N:1)
  - (b) threads no espaço do núcleo (1:1)



- Implementações híbridas também são possíveis

Threads de usuário: Implementação

- As threads são implementadas por uma biblioteca, e o núcleo não sabe nada sobre elas
  - N threads são mapeadas em um processo (N:1)
  - núcleo escalona processos, não threads
  - o escalonamento de threads é feito pela biblioteca
- Vantagens
  - permite usar threads em SOs que não têm suporte
  - chaveamento de contexto entre threads não requer chamada de sistema $\rightarrow$ desempenho
- Desvantagens
  - tratamento de chamadas bloqueantes
  - preempção por tempo é complicada

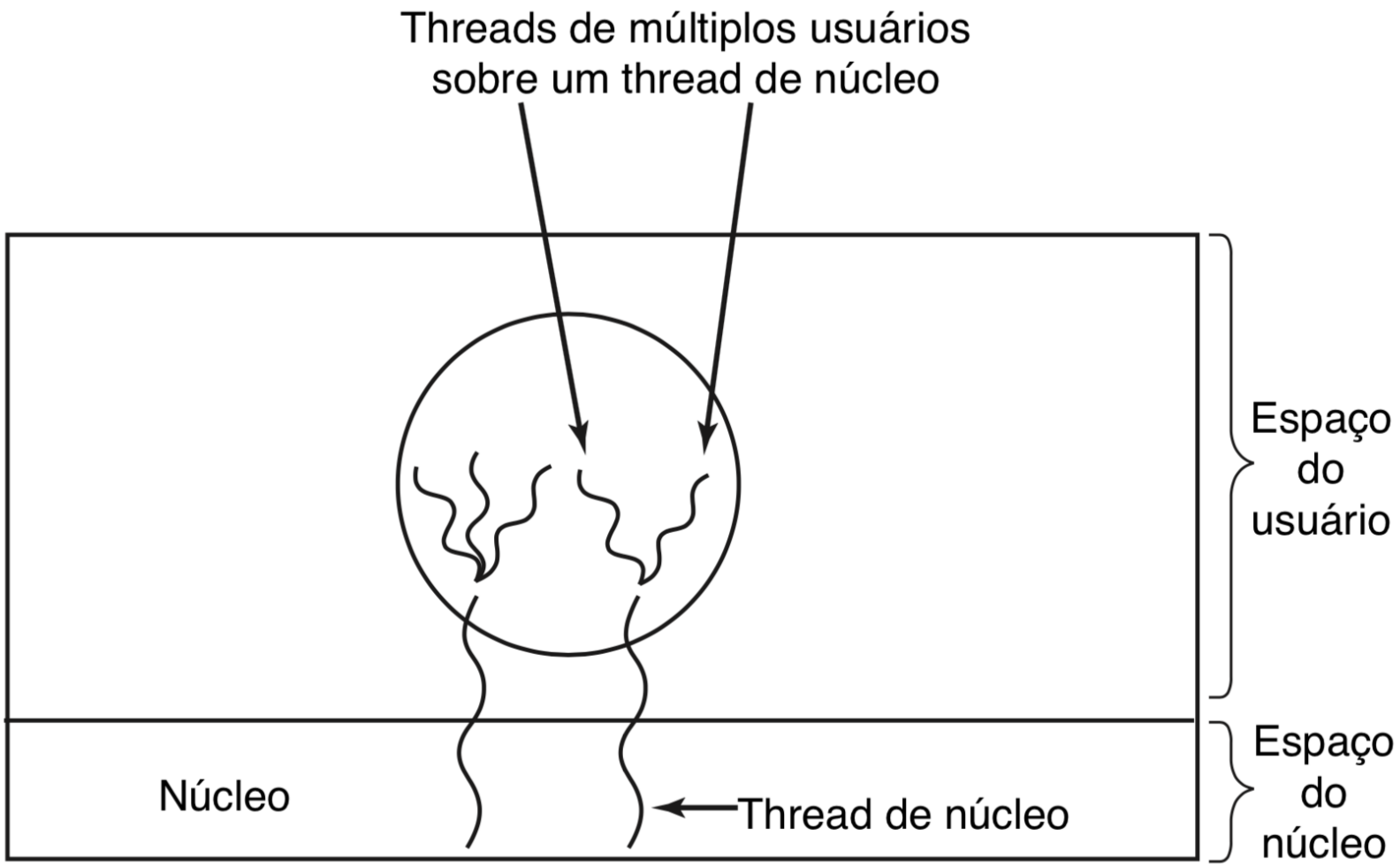
Threads de núcleo: Implementação

UDESC

- O núcleo conhece e escalona as threads
  - não há necessidade de biblioteca
  - modelo 1:1
- Vantagens
  - facilidade para lidar com chamadas bloqueantes
  - preempção entre threads
- Desvantagens
  - operações envolvendo threads têm custo maior
    - exigem chamadas ao núcleo

Threads híbridas: Implementação

- Combina os dois modelos anteriores



Threads: Preocupações ao converter códigos sequenciais

- Problemas em potencial
  - variáveis globais modificadas por várias threads
    - proibir o uso de variáveis globais
    - permitir variáveis globais privativas de cada thread
  - bibliotecas não reentrantes: funções que não podem ser executadas por mais de uma thread
    - permitir apenas uma execução por vez
  - sinais
    - quem captura? como tratar?
  - gerenciamento da pilha
    - o sistema precisa tratar o overflow de várias pilhas

Comunicação entre Processos

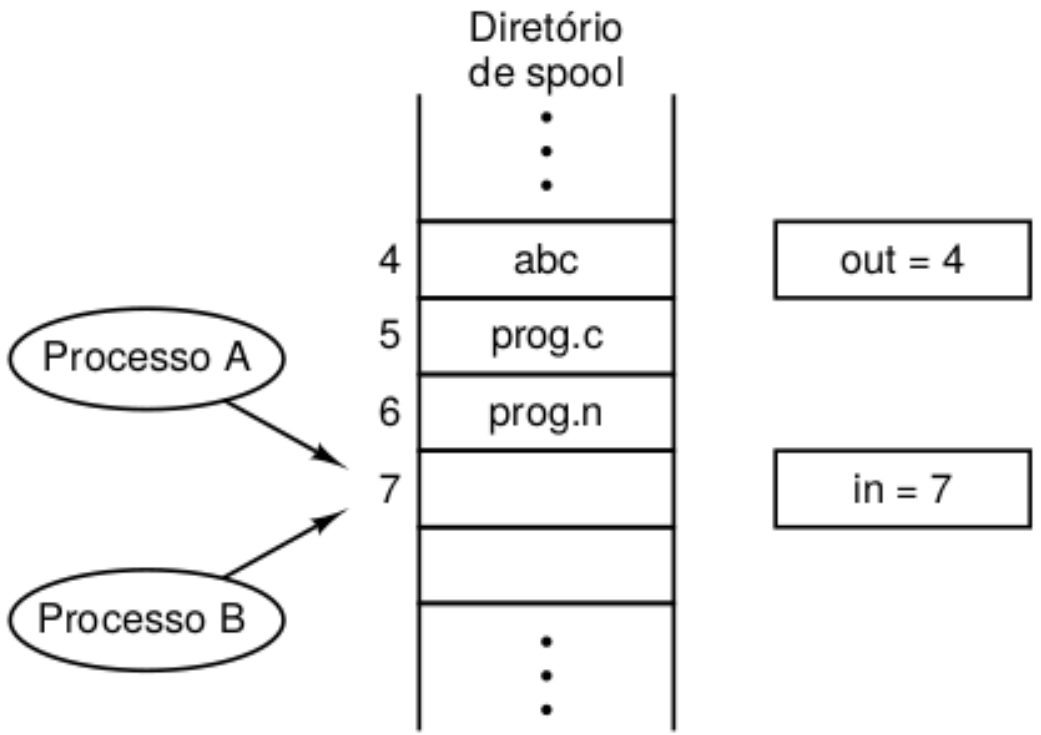
Conceitos

- Frequentemente, processos precisam se comunicar para trocar dados
  - exemplo: pipes
- Tópicos envolvidos
  - como processos trocam informações
  - como garantir que um processo não invada o outro quando envolvidos em atividades críticas
  - como determinar a sequência de execução de processos
- Valem igualmente para threads

Condições de Disputa

- Quando dois ou mais processos manipulam dados compartilhados simultaneamente e o resultado depende da ordem precisa em que os processos são executados
- Exemplo: spool de impressão
  - processos colocam trabalhos em uma fila
  - servidor de impressão retira trabalhos da fila e os envia para a impressora
  - fila de impressão é uma área de armazenamento compartilhada
    - diretório de spool

Exemplo: fila de Impressão



- A e B inserem os seus trabalhos na posição 7

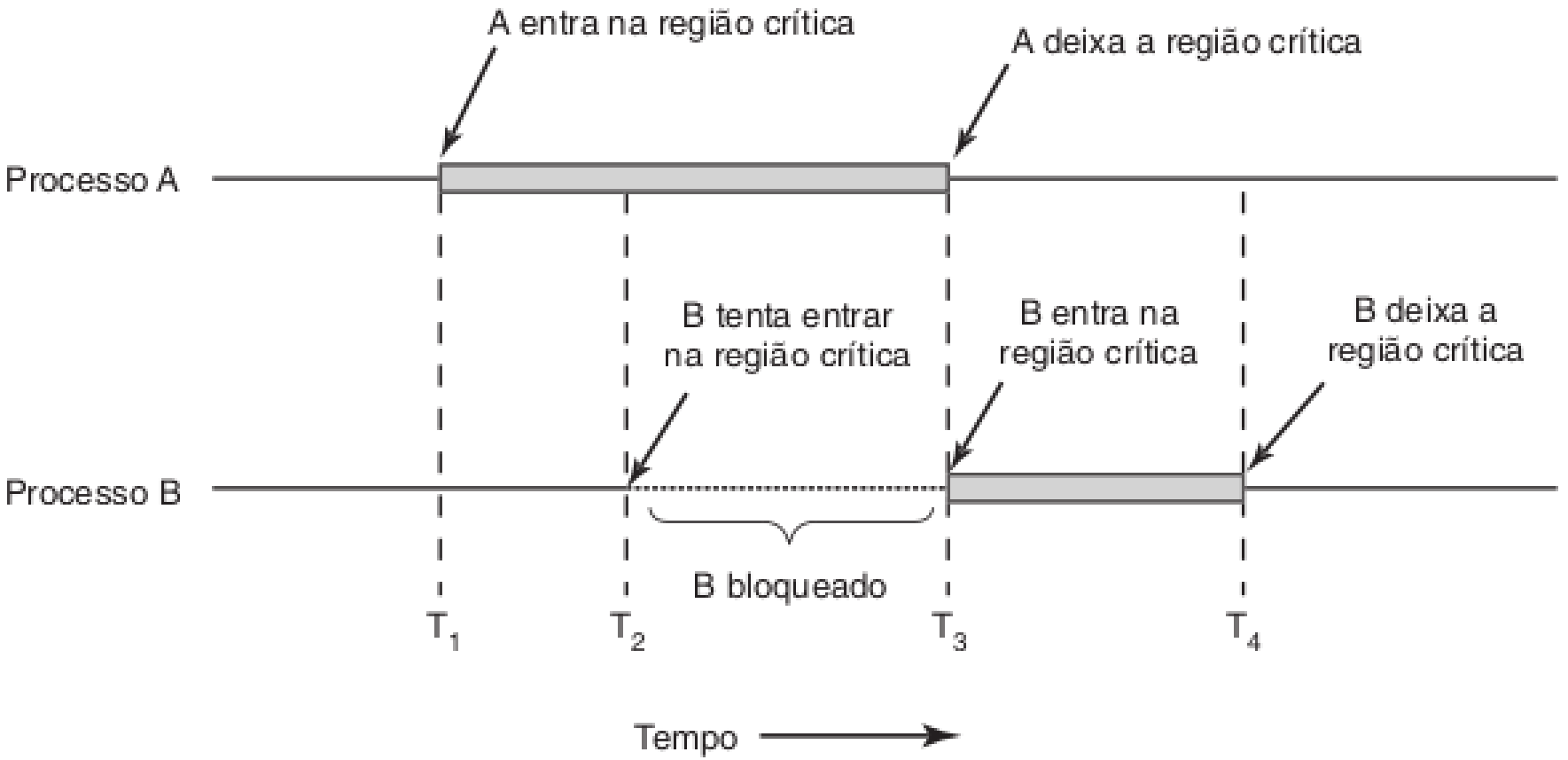
Região Crítica

- Partes do código em que há acesso a memória compartilhada e que pode levar a condições de disputa
- É necessário haver **exclusão mútua** entre os processos durante suas regiões críticas
- Também chamadas de seções críticas

Condições p/ exclusão mútua

- Quatro condições necessárias para prover exclusão mútua:
  1. Nunca dois processos podem estar simultaneamente em uma região crítica
  2. Nenhuma afirmação sobre velocidades ou números de CPUs
  3. Nenhum processo executando fora de sua região crítica pode bloquear outros processos
  4. Nenhum processo deve esperar eternamente para entrar em sua região crítica

Exclusão mútua em Regiões Críticas



Exclusão mútua com espera ocupada

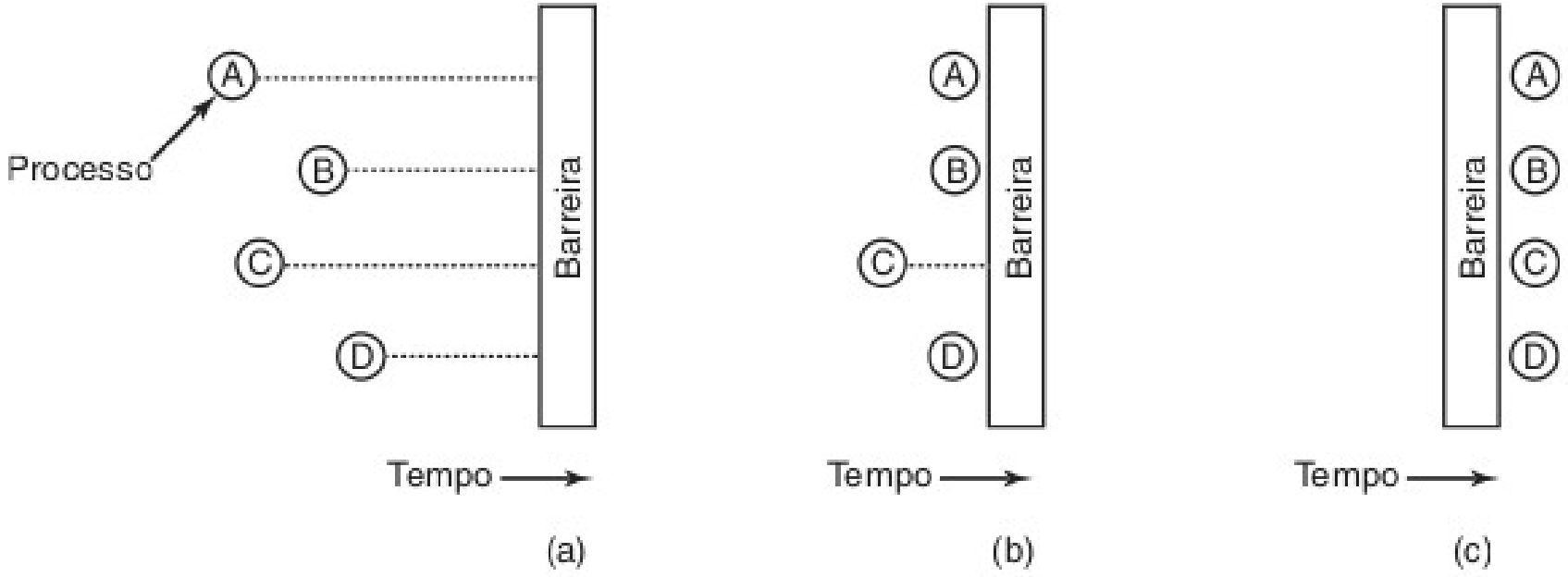
- Existem diversas soluções para o problema de exclusão mútua
- Algumas delas se baseiam em espera ociosa (ocupada)
  - o processo fica em loop até conseguir entrar na seção crítica
- Exemplos
  - desabilitação de interrupções
  - variáveis de impedimento (lock)
  - alternância obrigatória
  - solução de Peterson
  - instrução TSL

Exclusão mútua sem espera ocupada

- Soluções de exclusão mútua baseadas em espera ocupada são indesejáveis
  - um loop vazio ocupa o processador
- Isso evita que outros processos executem
  - incluindo um processo na seção crítica
- Pode causar **inversão de prioridade**
  - processo mais prioritário fica no loop e um menos prioritário não consegue liberar a seção crítica
- Melhor seria se o processo que encontra a seção crítica ocupada pudesse ficar bloqueado até que a seção crítica fosse liberada

Barreiras

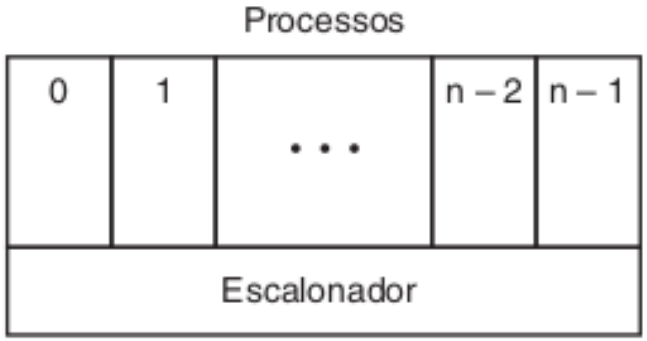
- Mecanismo usado para definir um ponto de sincronização para múltiplos processos/threads



Escalonamento

Conceito de Escalonamento

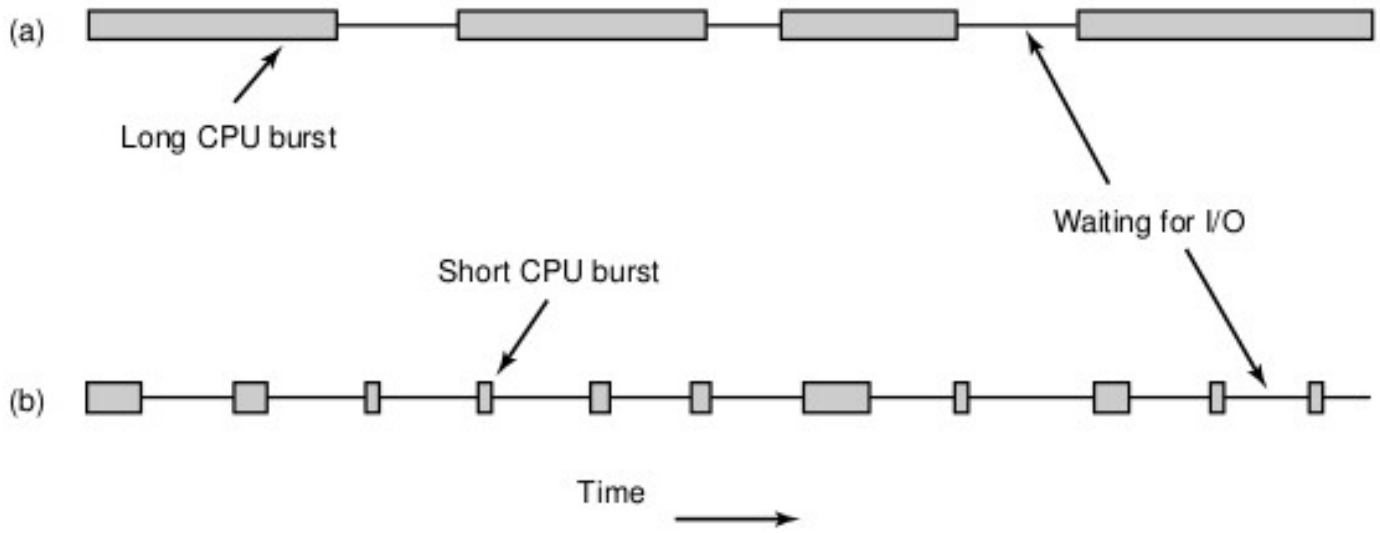
- Um requisito básico de sistemas multiprogramados é decidir qual processo deve executar a seguir, e por quanto tempo
  - o componente do SO que faz isso é o **escalonador** (*scheduler*)
  - o escalonador implementa um **algoritmo de escalonamento**
- Algoritmos de escalonamento podem diferir nos seus objetivos → o que se deseja priorizar?
  - todos visam a usar a CPU de modo eficiente
    - chaveamentos de contexto são caros
- Visão dos processos



Comportamento dos processos

- Em geral, processos alternam ciclos de uso de CPU com ciclos de requisição de E/S
  - o processo executa várias instruções de máquina e faz uma chamada de sistema solicitando um serviço do SO
- Existem duas grandes classes de processos
  - orientados a CPU (CPU-bound)
  - orientados a E/S (I/O-bound)
  - há processos que alternam essas características

Comportamento dos processos



(a) um processo orientado a CPU

(b) um processo orientado a E/S