Sistemas Operacionais

CAPITULO 2
(PROCESSOS)

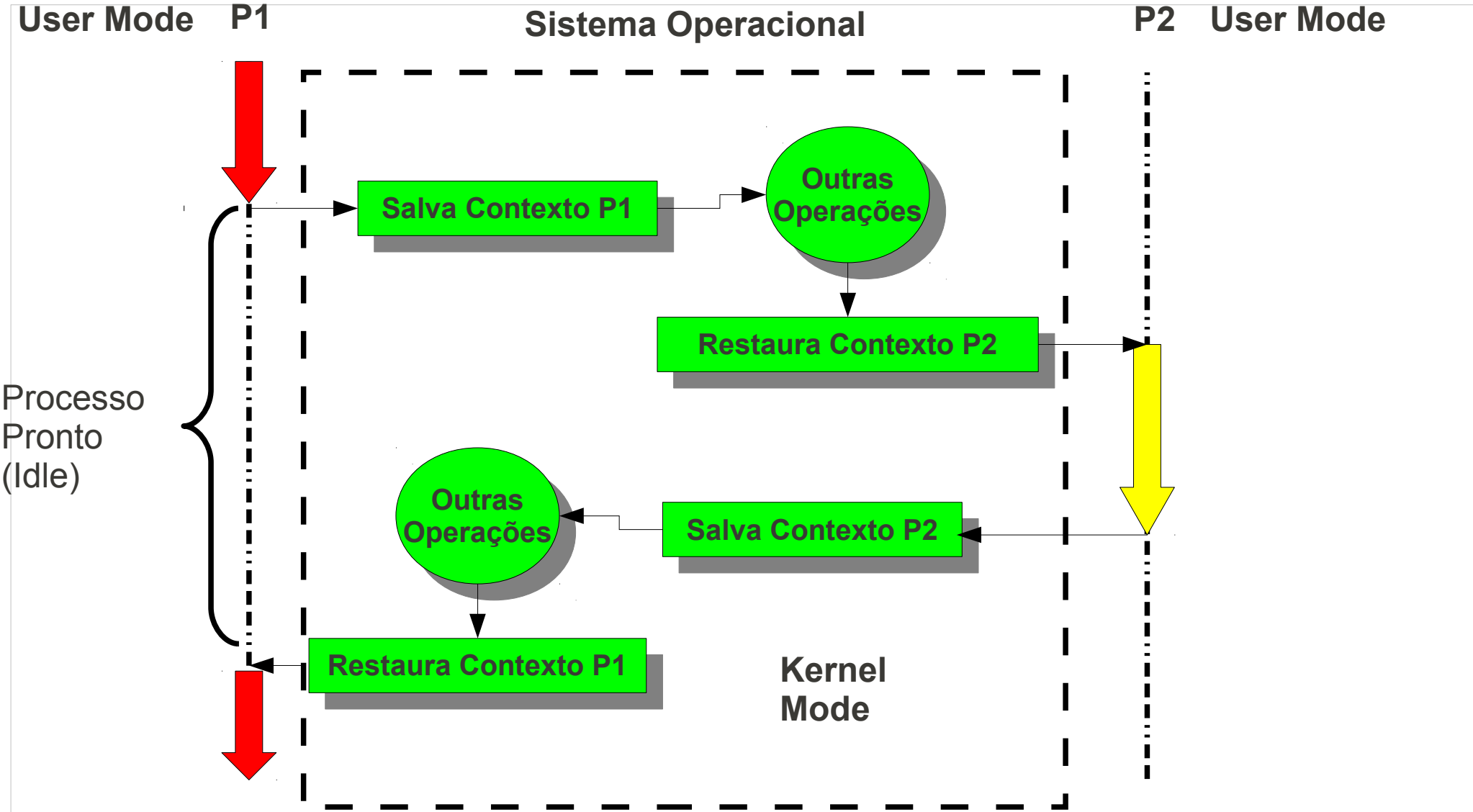
Sistemas Operacionais
Processos - Roteiro

- Conceitos
- Threads
- Escalonamento
- Concorrência

Sistemas Operacionais
Processos - Conceitos

- Multiprogramação
  - Otimização do uso dos recursos
  - Pseudo Paralelismo
  - Multiprocessamento(SMP)
  - Mudança de Contexto
    - Salvamento (Registradores, Program Counter)
    - Bloco de Controle de Processo

Sistemas Operacionais
Processos – Mudança de Contexto



Sistemas Operacionais
Processos - Conceitos

- Processo x Programa
  - Programa (Estático)
  - Processo (Dinâmico)
  - 1:N (Instâncias de Programa)
- CPU-bound
  - Uso intensivo de CPU
- I/O-bound
  - Uso não intensivo de CPU

Sistemas Operacionais
Processos

- Criação de Processos
  - Inicio do Sistema
    - Daemons (serviços)
  - Chamada de Sistema por um processo
    - Linux (fork(), execve())
    - Windows (CreateProcess())
  - Requisição de Usuário

Sistemas Operacionais Processos

- Término de Processos
  - Saída Normal
  - Saída por Erro
    - Linux (exit()), Win (ExitProcess())
  - Erro Fatal
    - Erro de Proteção, Estouro de Buffer, divisão por zero
  - Cancelamento
    - Linux (kill()) ou Win (TerminateProcess())

Sistemas Operacionais
Processos

- Hierarquia de Processos
  - Processo pai cria processo Filho
  - Arvore de processos
  - Windows não possui hierarquia, todos os processos estão no mesmo nível

Sistemas Operacionais
Processos

- Estados
  - Em execução: Processo com a CPU
  - Pronto: Aguardando CPU
  - Bloqueado: Aguardando SO

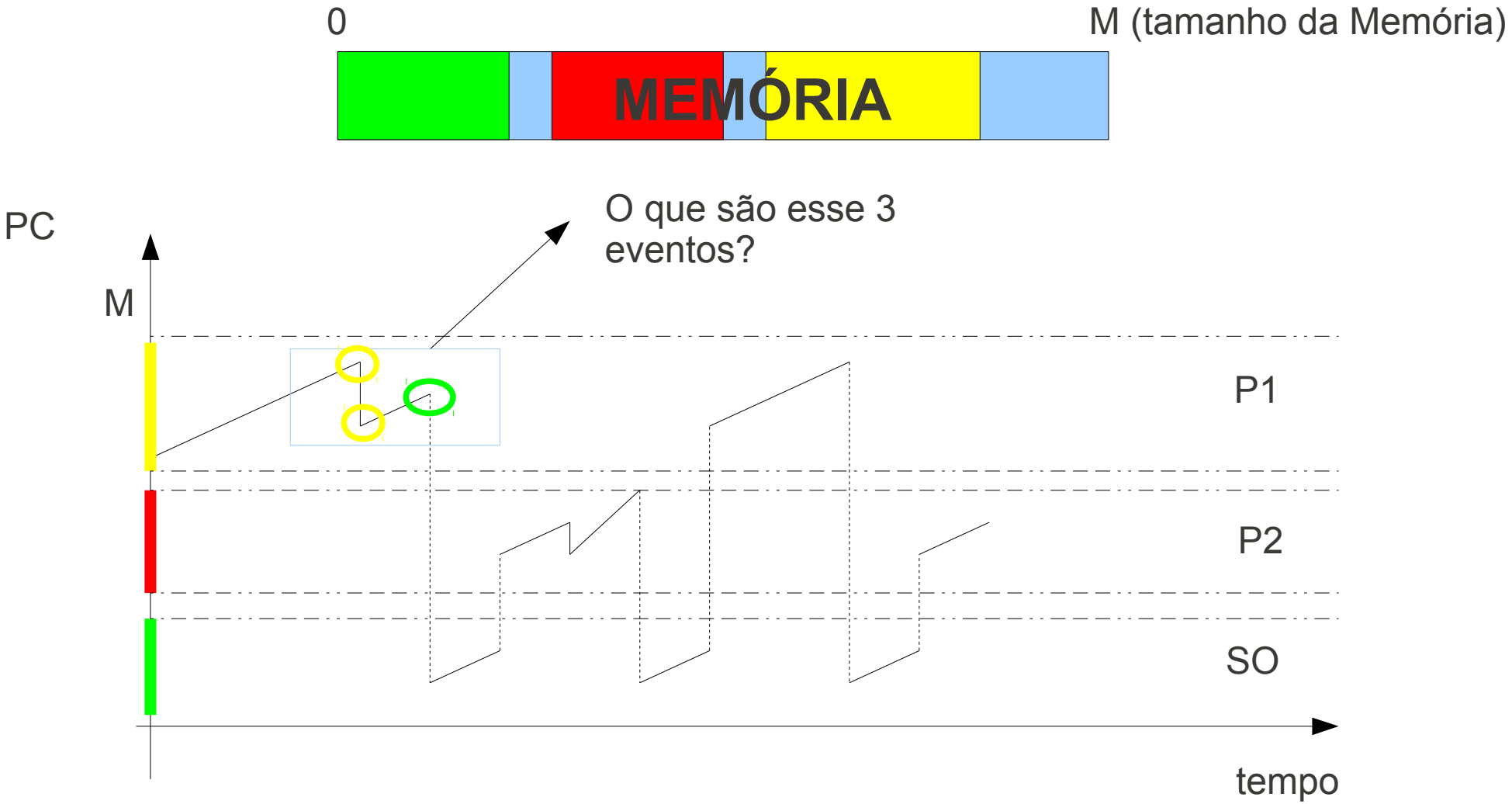
1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Figura 2.2 Estados dos Processos
Fonte:Tenenbaum

Sistemas Operacionais
Processos

- Interrupções
  - Sinal
    - Desvia a Execução do Programa
    - Mudança de Contexto
  - Tratador de interrupção
    - Vetor de Interrupção
  - Tipos de Interrupção
    - Software e Hardware
  - Interrupções de Software (Syscalls)

Sistemas Operacionais
Processos



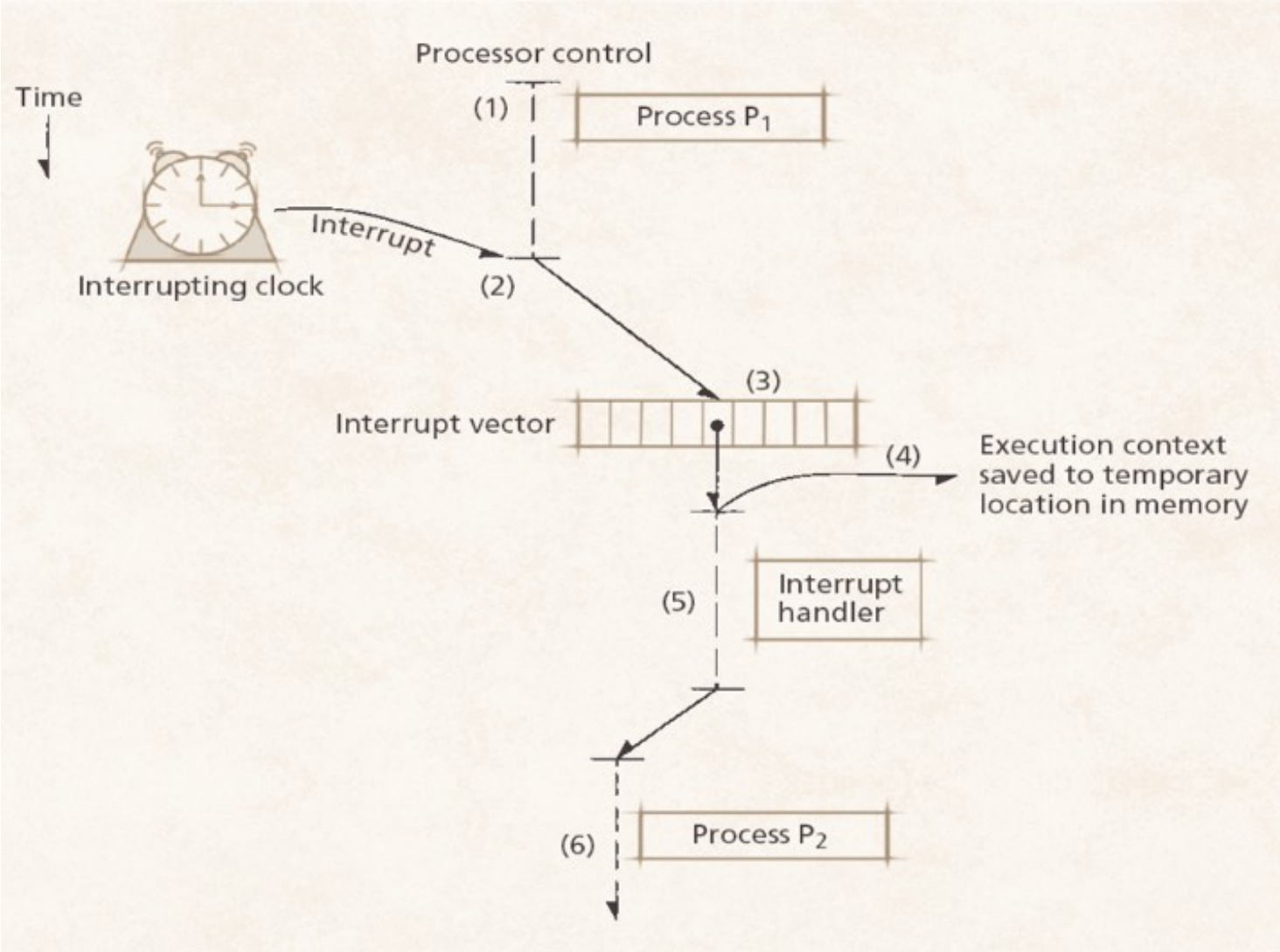
Sistemas Operacionais
Processos

- Interrupções

1. O hardware empilha o contador de programa etc.
2. O hardware carrega o novo contador de programa a partir do vetor de interrupção.
3. O procedimento em linguagem de montagem salva os registradores.
4. O procedimento em linguagem de montagem configura uma nova pilha.
5. O serviço de interrupção em C executa (em geral lê e armazena temporariamente a entrada).
6. O escalonador decide qual processo é o próximo a executar.
7. O procedimento em C retorna para o código em linguagem de montagem.
8. O procedimento em linguagem de montagem inicia o novo processo atual.

Figura 2.5 Tenenbaum

Sistemas Operacionais
Processos - Interrupções



Sistemas Operacionais
Processos

- Implementação de Processos
  - Tabela de Processos
    - Bloco de Controle de Processos

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		

Figura 2.4 Tenenbaum

Sistemas Operacionais
Processos - Threads

- Threads
  - Linhas de Execução
  - Compartilhamento de Recursos
    - Arquivos Abertos (descritores)
    - Espaço de endereçamento (memória de dados e programa
  - Menor overhead na mudança de contexto

Sistemas Operacionais
Processos - Threads

- Modelo de Thread
  - Processo como Agrupamento de Recursos
    - Arquivos abertos, espaço de endereçamento, tratadores de sinais, etc.
  - Thread
    - Fluxo de Execução
    - Contador de Programa, pilha de execução, registradores, etc.
    - Thread executa de fato (ocupam CPU)

Sistemas Operacionais
Processos- Threads

- Multithread
  - Processos leves (*Lightweight Process*)

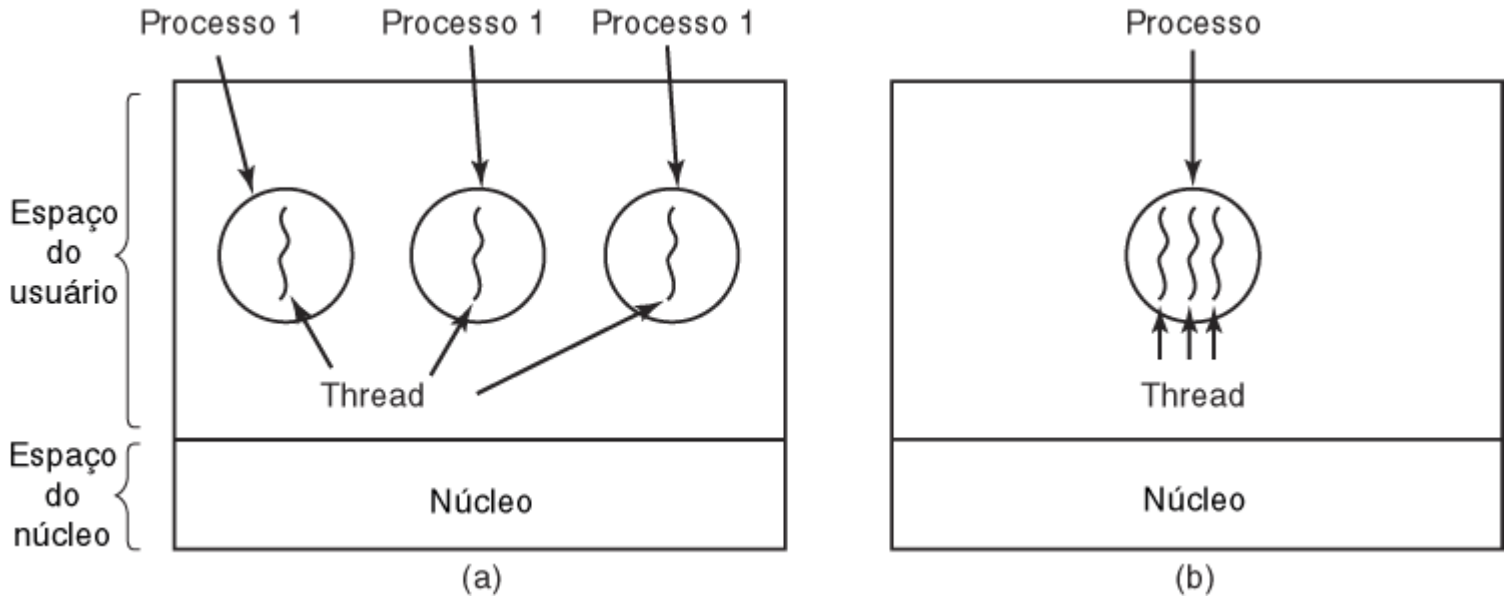


Figura 2.6 Tenenbaum

Sistemas Operacionais Processos -Threads

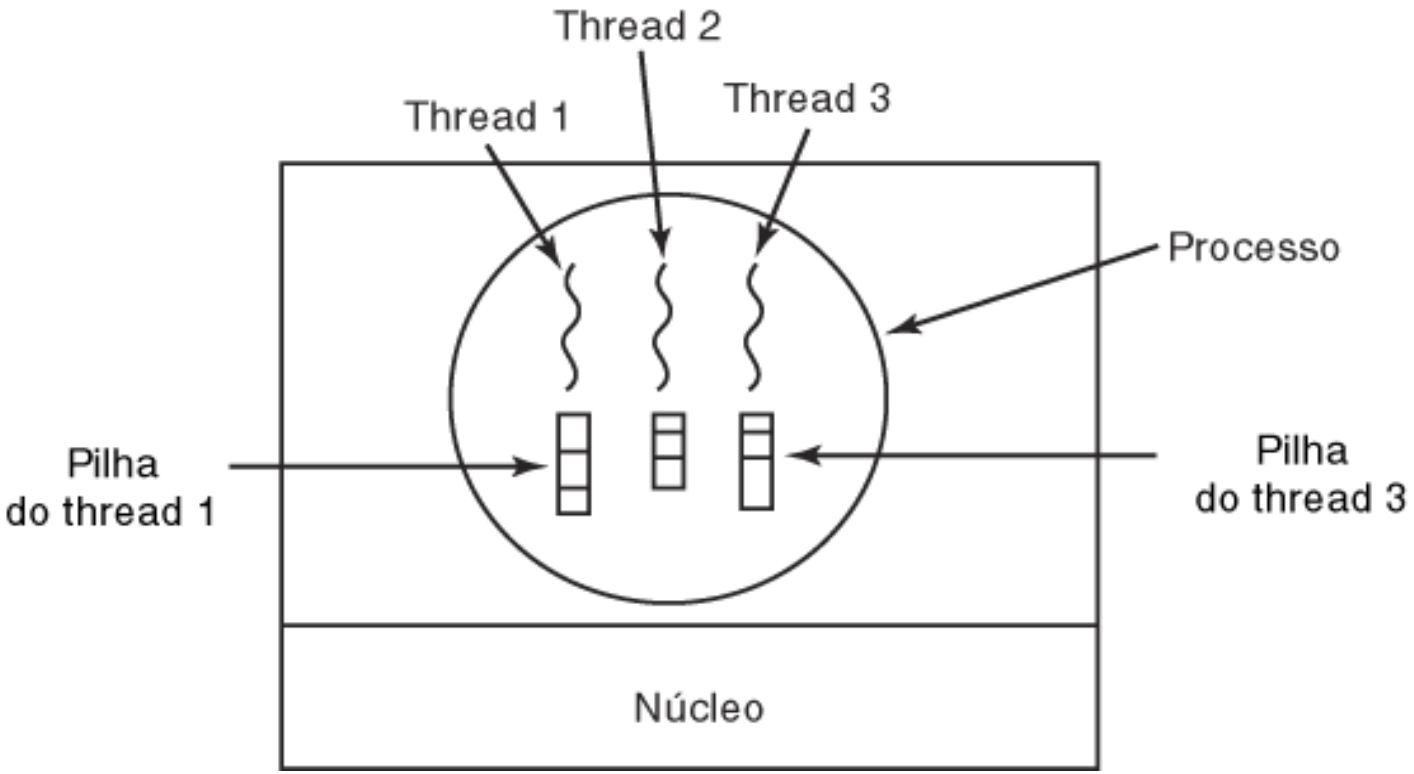
- Multithread
  - Falta de proteção
  - Cooperação
  - Mesmos estados do Processo

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	

Figura 2.7 Tenenbaum

Sistemas Operacionais
Processos - Threads

- Multithread
  - Pilhas exclusivas

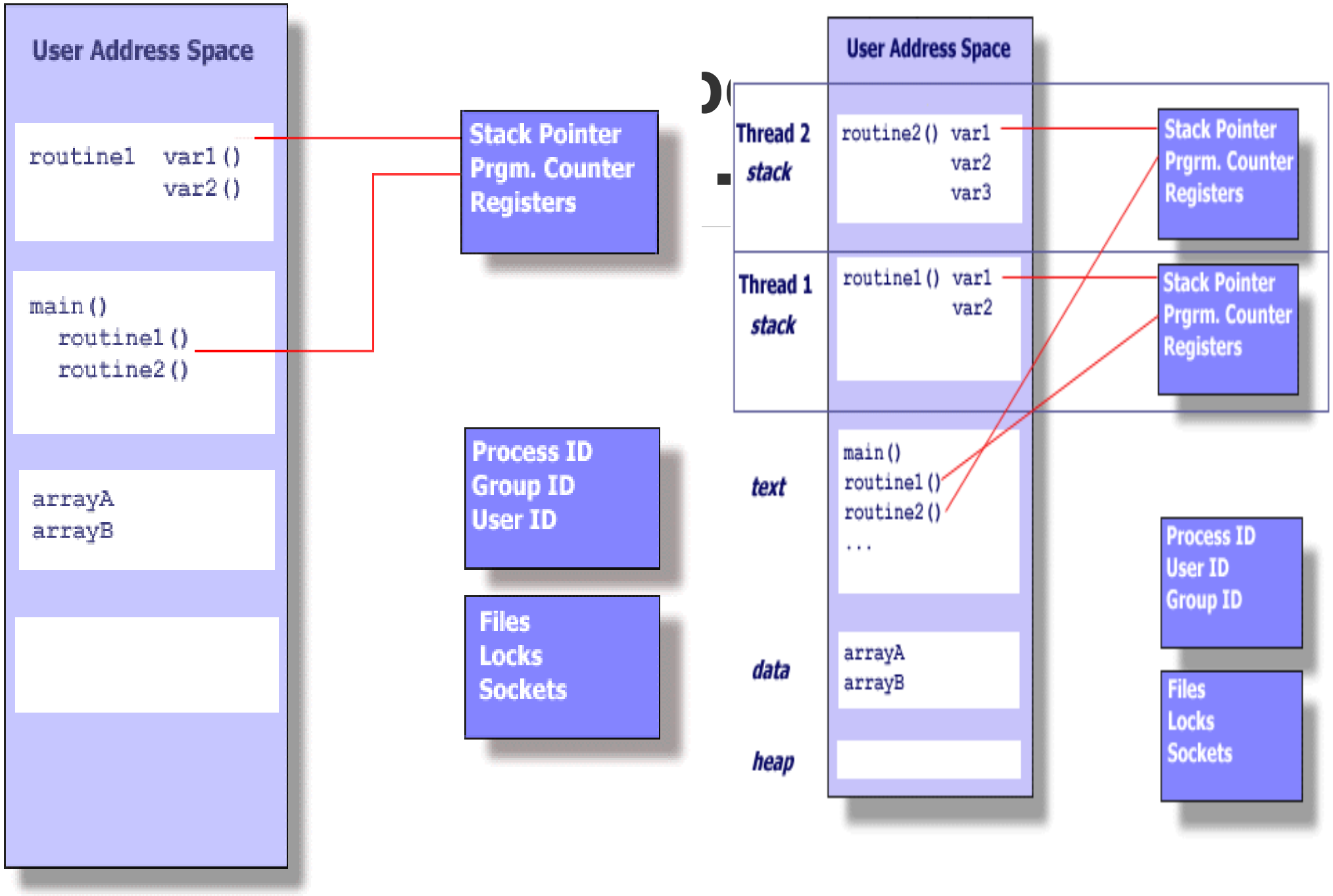


Sistemas Operacionais
Processos - Threads

- Multithread
  - funções
    - Tid = Thread_create (procedimento)
    - Thread_exit
    - Thread_wait
      - Espera outro thread terminar
    - Thread_yield
      - Libera CPU (Abre mão do escalonamento)

Sistemas Operacionais
Processos - Threads

- Uso de Threads
  - Aplicações com diversos fluxos (atividades)
  - Aproveitando os bloqueios (desperdício de tempo)
  - Mais fácil de criar e destruir
  - Aceleram aplicações de I/O-Bound
  - Uteis em Sistemas com múltiplas CPUs
    - Paralelismo Real



fonte:https://computing.llnl.gov/tutorials/pthreads/#PthreadsAPI

Sistemas Operacionais
Processos - Threads

Uso de Threads

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *function(void *param)
{
      int id = (int)param;
      int i, loops = 100;
      for(i = 0; i < loops; i++)
      {
        printf("thread %d: loop %d\n", id, i);
      }
      //sleep(10);
      pthread_exit(NULL);
}
```

```
int main(void){
      pthread_t threads[THREADS_MAX];
      int i;
      printf("pre-execution\n");
      for (i = 0; i < THREADS_MAX; i++) {
pthread_create(&threads[i], NULL, function, (void *)i);
      }
      printf("mid-execution\n");
      for (i = 0; i < THREADS_MAX; i++) {
            pthread_join(threads[i], NULL);
            printf("waiting-end %d\n",i);
      }
      printf("post-execution\n");
      return EXIT_SUCCESS;
}
```

Sistemas Operacionais
Processos - Threads

- Implementação de Threads
  - Em espaço de Usuário
  - No kernel
  - Híbridas

Sistemas Operacionais
Processos - Threads

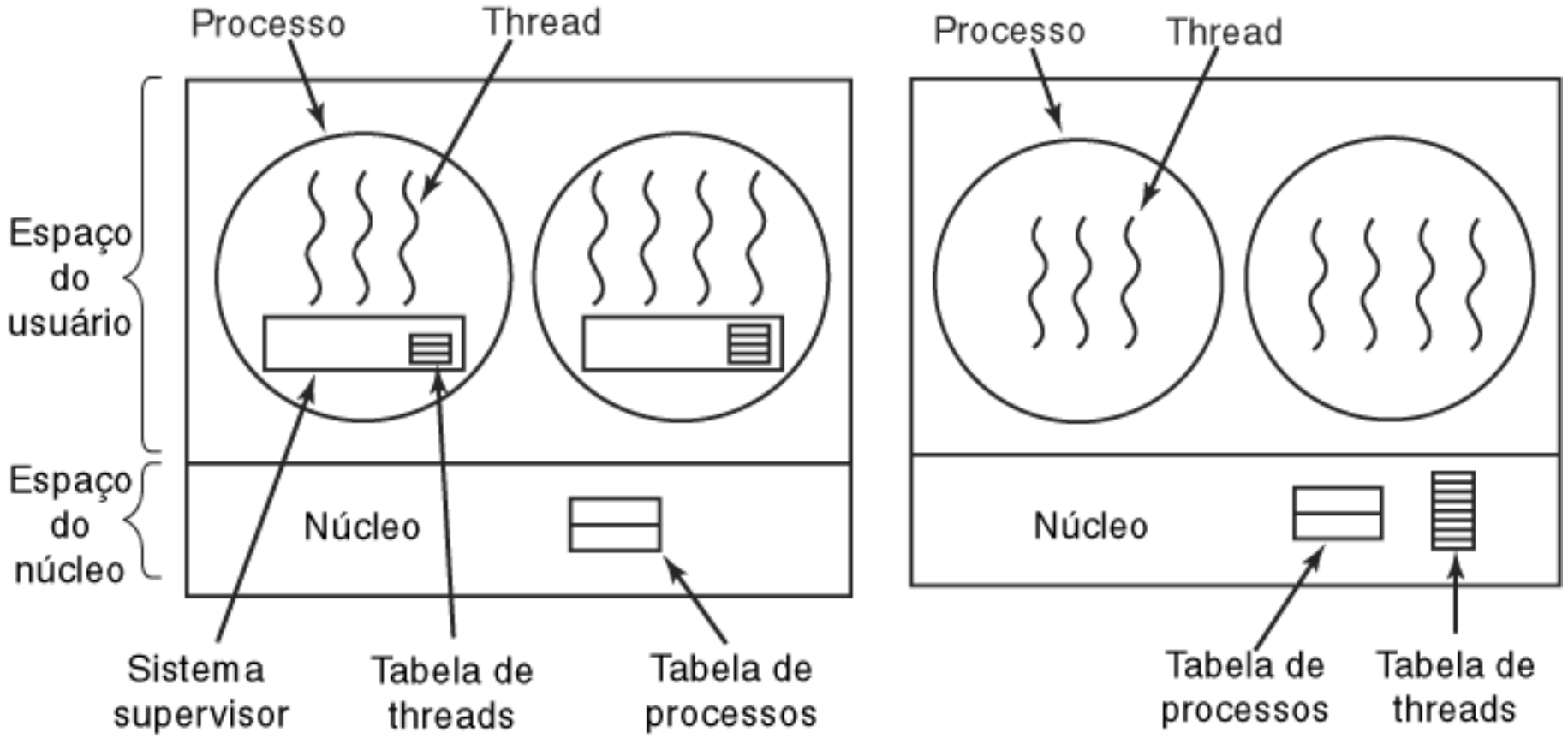
- Threads de usuário (ULT)
  - Tabela de Threads no processo
  - Processo Supervisor
    - Kernel não precisa suporte a thread
  - Algoritmos de escalonamento específico
    - Ex: *Garbage Collector – JVM*
  - Chamadas de Sistemas Bloqueiam o processo
  - Alternativas:
    - Chamadas não blocantes
    - Jacket ou wrapper (Verifica se a chamada vai blocar)

Sistemas Operacionais
Processos - Threads

- Threads de Sistema (KLT)
  - Tabelas de Threads no SO
  - Não existe Supervisor
  - Syscalls não bloqueiam outras threads
  - Operações com threads são mais caras
    - Criar, bloquear, destruir
    - Reciclagem (otimização)

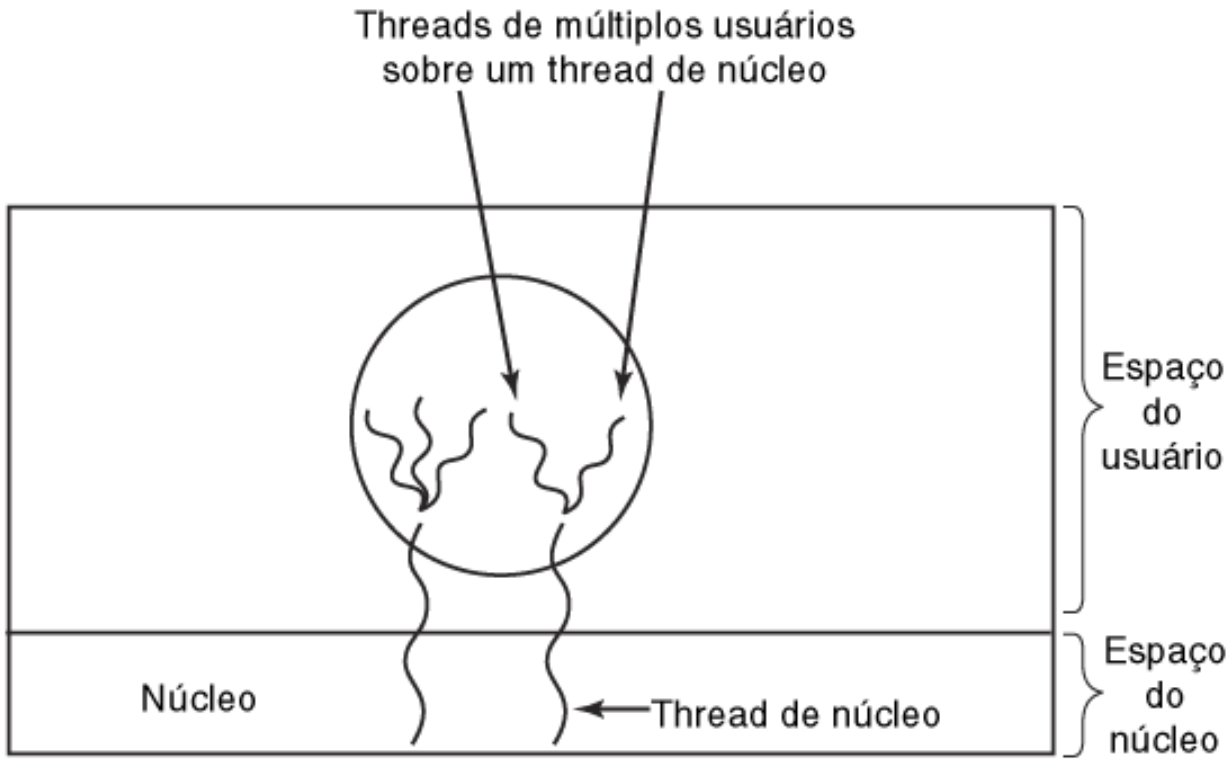
Sistemas Operacionais
Processos -Threads

- ULT e KLT



Sistemas Operacionais
Processos - Threads

- Threads Híbridas



Sistemas Operacionais
Processos - Threads

- Threads POP-UP
  - Geradas Sob Demanda
  - Abordagem Tradicional
    - Processo bloqueado em receive
  - Requisição gera criação de nova Thread
    - Vantagens
      - Não precisa chavear contexto

Sistemas Operacionais
Processos - Threads

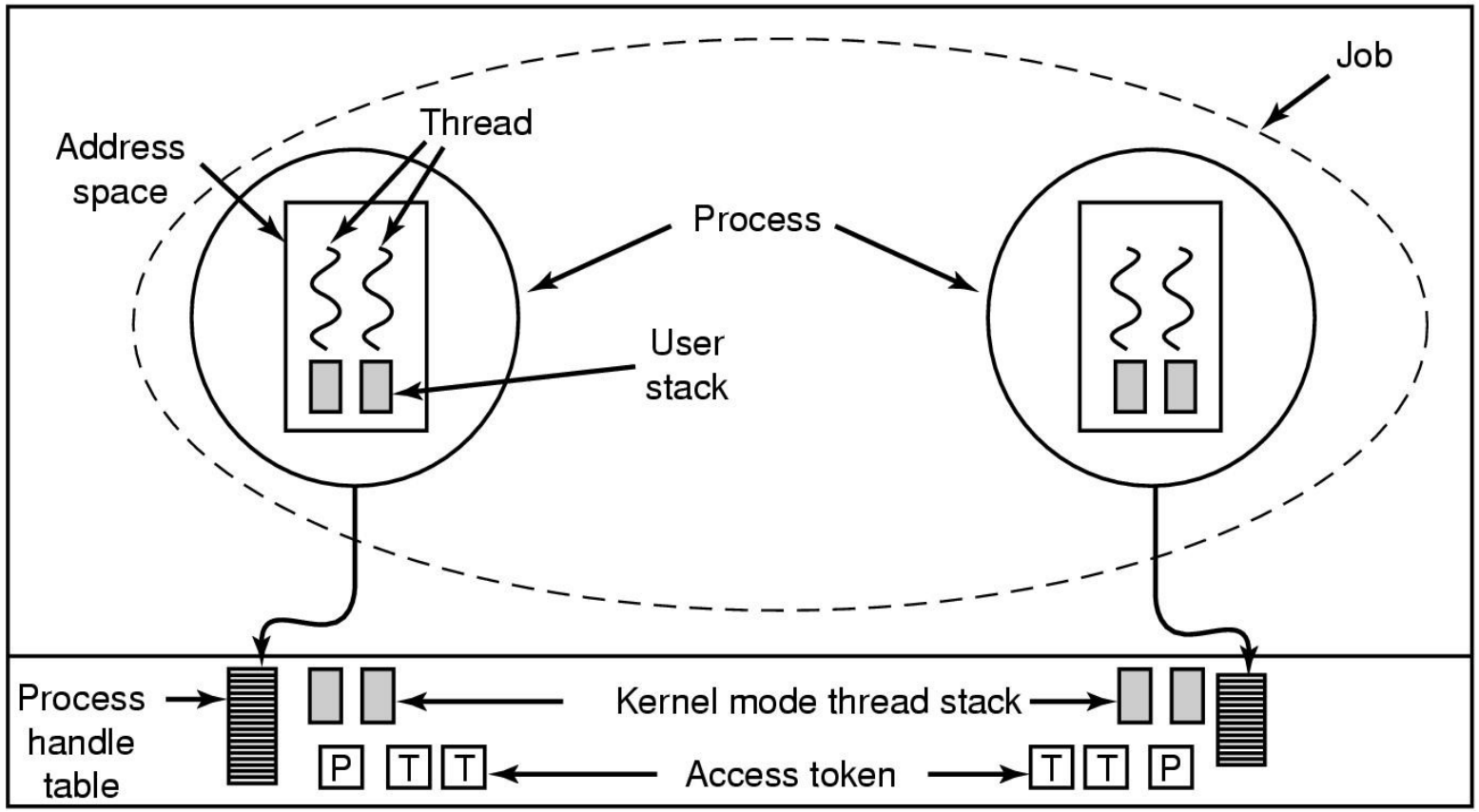
- Threads - Linux
  - Processo
    - API
      - Fork
      - Exec
      - exit
  - Thread
    - API
      - Clone(função, Pilha, flags, argumentos)

Sistemas Operacionais
Processos - Threads

- Threads – Windows
  - Trabalho
    - quotas
  - Processo
    - CreateProcess (10 parâmetros) ; ExitProcess
  - Thread
    - CreateThread (6 Argumentos) ;ExitThread
  - Filamento (Thread Peso Leve)
    - CreateFIber; ExitFiber

Sistemas Operacionais
Processos

- Threads - Windows

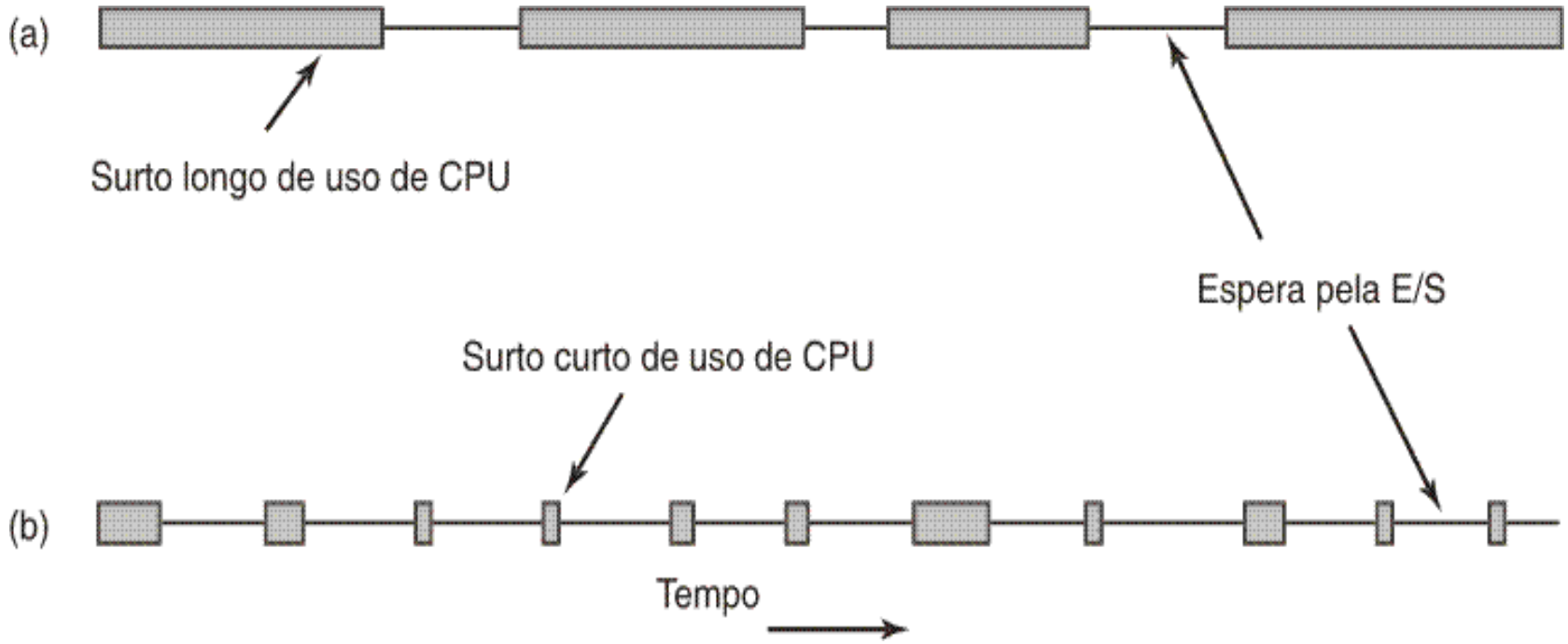


Sistemas Operacionais
Processos - Escalonamento

- Revisão
  - Multiprogramação
    - Vários processos em memória
  - Multitarefa (timesharing)
    - Compartilhamento do tempo de CPU (Quantum)
  - Ciclo de Vida do Processo
    - Estados (Em execução, pronto, bloqueado)
  - Preemptivo X Não Preemptivo
    - Fatia de Tempo para execução: Quantum ou Quanta

Sistemas Operacionais
Processos - Escalonamento

- Comportamento do Processo



Tanembaum – Fig 2.37

Sistemas Operacionais
Processos - Escalonamento

- Escalonador
  - Decide quem vai usar a CPU (Curto Prazo)
  - Sempre na memória
  - Executa várias vezes por segundo
    - Age sobre processos prontos
  - Separação de Política e Mecanismo
    - Garantir a satisfação geral e a execução eficiente de tarefas essenciais do sistema.
  - Quando o escalonador deve entrar em ação?
    - Resp: Próximo Slide

Sistemas Operacionais
Processos - Escalonamento

- Atuação do Escalonador
  - Criação de um processo
  - Término do processo
  - Bloqueio (syscall blocante)
  - Interrupção de HW
    - Interrupções de Relógio em sistemas preemptivos

Sistemas Operacionais Processos - Escalonamento

- Dispatcher
  - Porção final do Escalonador
  - Dá o controle da CPU ao processo
    - Restaura o contexto
    - Muda o processador para o modo usuário
    - Inicia o Program Counter (PC) com o endereço adequado
  - Latência
    - Tempo entre a parada de um processo e o início de outro

Sistemas Operacionais Processos - Escalonamento

- Categorias de Escalonadores
  - Lote
    - Não existe iteração com o usuário.
    - Em geral não precisam de tempo de resposta.
    - Grande quantum pode reduzir o overhead.
  - Iterativo
    - Usuários aguardando uma resposta
    - Preempção é fundamental
  - Tempo Real
    - Acordo de tempo de resposta

Sistemas Operacionais
Processos - Escalonamento

- Objetivos do escalonamento
  - **Garantir Justiça:** Cada Processo ganha porção igual da CPU
  - **Aumentar a Eficiência:** Escolhendo processo que vão garantir uso de 100% da CPU.
  - **Menor Tempo de Resposta** (Processos interativos)
  - **Menor tempo de Retorno** (para processos batchs)
  - **Maximizar o número de processos atendidos** em um intervalo de tempo (vazão)

Sistemas Operacionais
Processos - Escalonamento

- Categoria de Escalonadores X Objetivos
  - **Todos os Sistemas:** Justiça, Aplicação da Politica, Equilíbrio no uso dos recursos
  - **Sistemas Batches:** Vazão e Ocupação da CPU
  - **Sistema Iterativos:** Tempo de Resposta e Proporcionalidade (percepção do usuário)
  - **Sistema de Tempo Real:** Cumprimento dos prazos, Previsibilidade (evitar degradação em sistemas multimídia).

Sistemas Operacionais Processos - Escalonamento

- Algoritmos para sistemas de Lote
  - FCFS
  - Mais Curto Primeiro
  - Próximo de menor tempo restante
  - Três Níveis

Sistemas Operacionais
Processos - Escalonamento

- FCFS
  - Não Preemptivo
  - FIFO
  - Implementação através de Fila
  - Criado para sistemas de lote
  - Ruim para sistemas de tempo compartilhado
    - Tempo de resposta ruim

Sistemas Operacionais
Processos - Escalonamento

- Mais Curto Primeiro
  - Não Preemptivo
  - Processo com menor tempo para Terminar é escolhido
    - Dificuldade em coletar esse dado
      - Usa-se normalmente o tempo da ultima execução
      - FIFO para desempate
  - Reduz o tempo médio de espera
    - Ex: 20,8,15,4,12

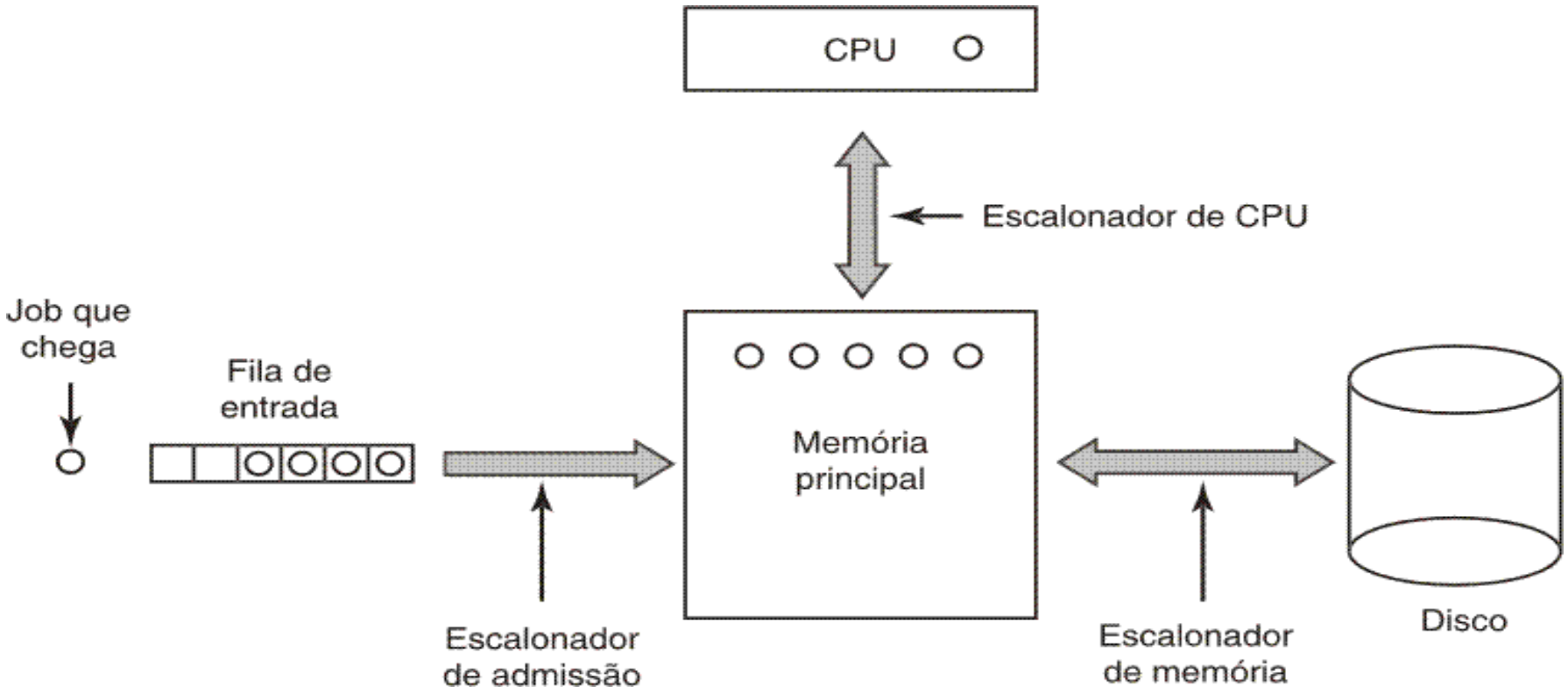
Sistemas Operacionais
Processos - Escalonamento

- Próximo de menor tempo restante
  - Preemptivo
  - Se aparecer um processo na fila de prontos com menor tempo restante que o processo que ocupa a CPU, esse será escolhido.

Sistemas Operacionais
Processos - Escalonamento

- Escalonamento em 3 Níveis
  - Admissão (longo prazo)
    - Entrada (quem vai para a memória)
  - Memória (médio prazo)
    - SWAP (quem vai pra memória)
  - CPU (curto prazo)
    - Quem vai usar a CPU

Sistemas Operacionais
Processos - Escalonamento



Tenenbaum – Fig 2.40

Sistemas Operacionais
Processos - Escalonamento

- Escalonamento em Sistemas Iterativos
  - Round-Robin
  - Por Prioridades
  - Escalonamento por Loteria
  - Fração Justa

Sistemas Operacionais
Processos - Escalonamento

- Round-Robin (RR)
  - Fração de Tempo de execução
    - Time Slice ou Quantum
  - Fila de Pronto é Circular
  - Timer
    - Gera interrupções a cada 1 Quantum
      - Faz a mudança de Contexto (Próximo Processo)
  - Feito para sistemas de tempo compartilhado
  - Espera Máxima: (n-1)q

Sistemas Operacionais
Processos - Escalonamento

- Round-Robin (RR) - cont...
  - Premptivo
  - Quantum → Infinito
    - FIFO
  - Quantum → Zero
    - Overhead das mudanças de contexto
  - Tempo médio de espera é alto

Sistemas Operacionais
Processos - Escalonamento

- Por Prioridade
  - Prioridade associada a cada processo
  - Pode ser ou não preemptivo
  - Job Menor Primeiro é um exemplo
    - Prioridade maior para processo menor *burst*
  - Definição da Prioridade
    - Dinâmica: Pelo Sistema (I/O Primeiro por exemplo)
    - Estática: Externamente (Fatores Políticos, importância)

Sistemas Operacionais
Processos - Escalonamento

- Por Prioridade - cont...
  - Problema: Postergação indefinida de processo
    - Sempre tem alguém com maior prioridade
    - Solução
      - Aumentar progressivamente a prioridade dos processos que estão aguardando
  - Agrupa Processos por filas de prioridade
    - Processos de prioridade mais alta ganham mais Quantums

Sistemas Operacionais
Processos - Escalonamento

- Escalonamento por Loteria
  - Processos escalonados por sorteio
  - Prioridade implementada através quantidade de bilhetes extra, quanto maior a importância mais bilhetes extras

Sistemas Operacionais
Processos - Escalonamento

- Fração Justa
  - Noção de Justiça
    - Quota de CPU por usuário e não por processo

Sistemas Operacionais
Processos - Escalonamento

- Escalonamento em Sistemas de Tempo Real
  - Acordo para tempo de resposta
  - TR Crítico e TR Não Crítico
  - Periódicos e Aperiódicos
    - C = Tempo
    - P = Período

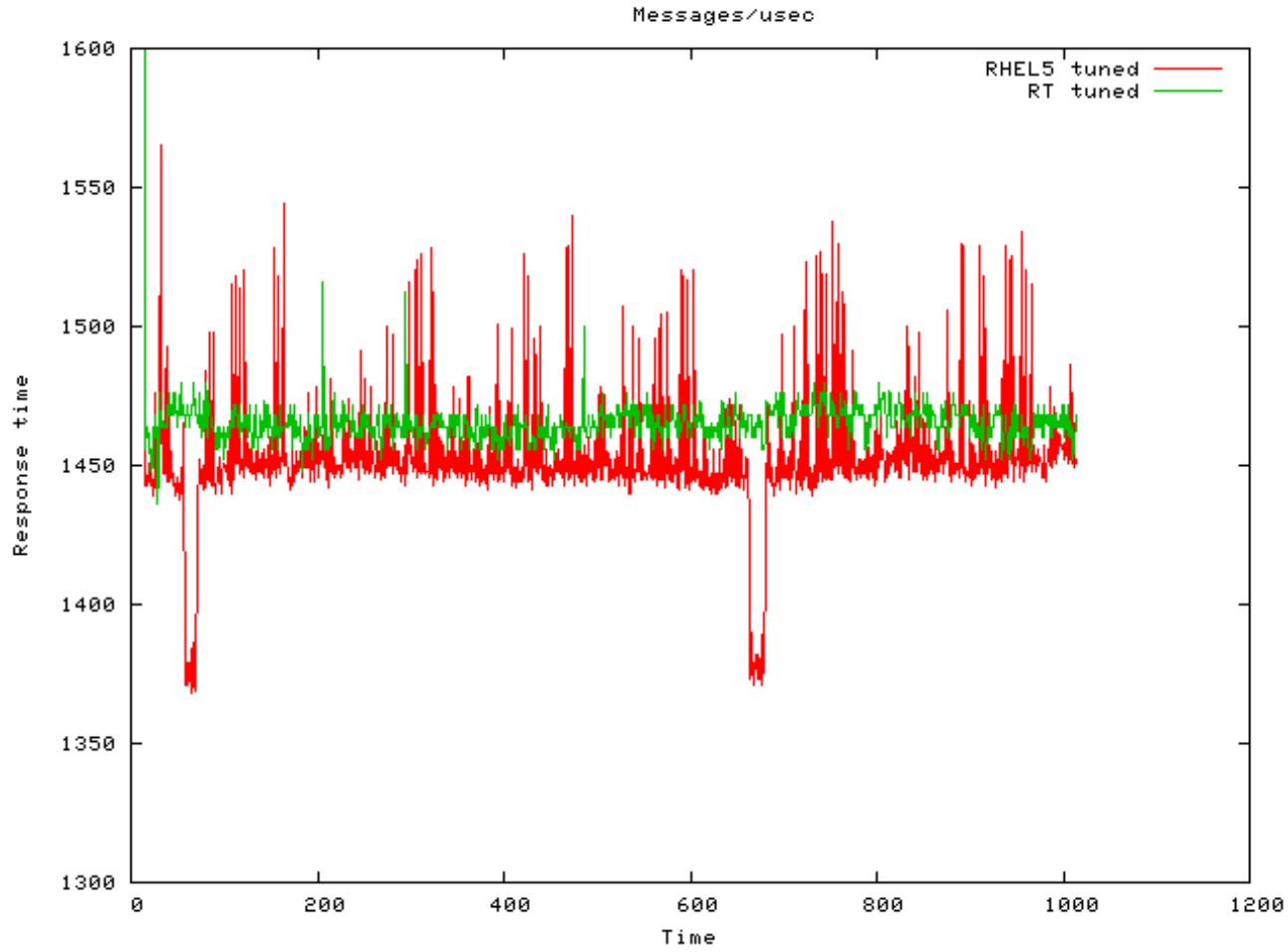
$$\sum_{i=0}^{m} \frac{C_i}{P_i} \leq 1$$

Sistemas Operacionais
Processos - Escalonamento

- Exemplo
  - P = {100, 200, 500}ms
  - C = {50, 30, 100}ms
    - 0,5 + 0,15 + 0,2 < 1

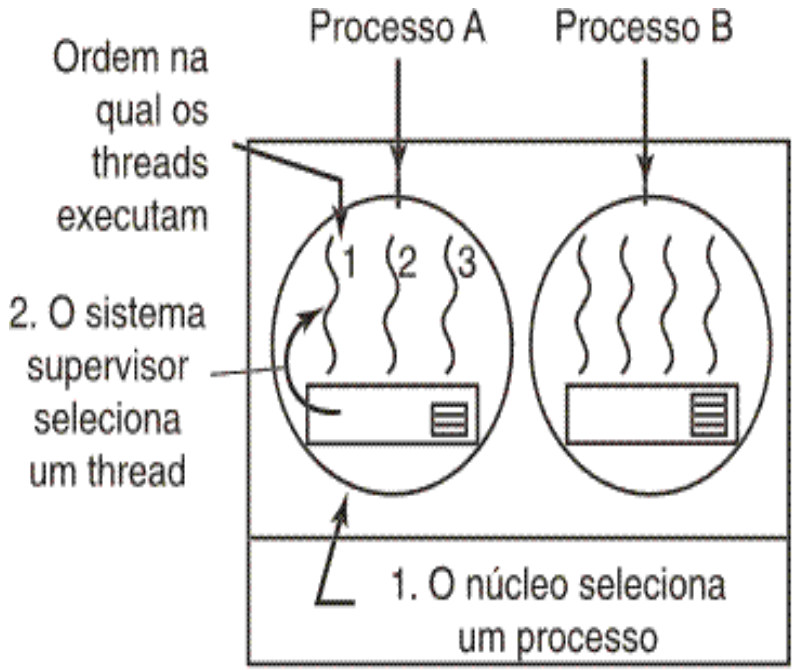
Sistemas Operacionais
Processos - Escalonamento

- Escalonamento em Sistemas de Tempo Real



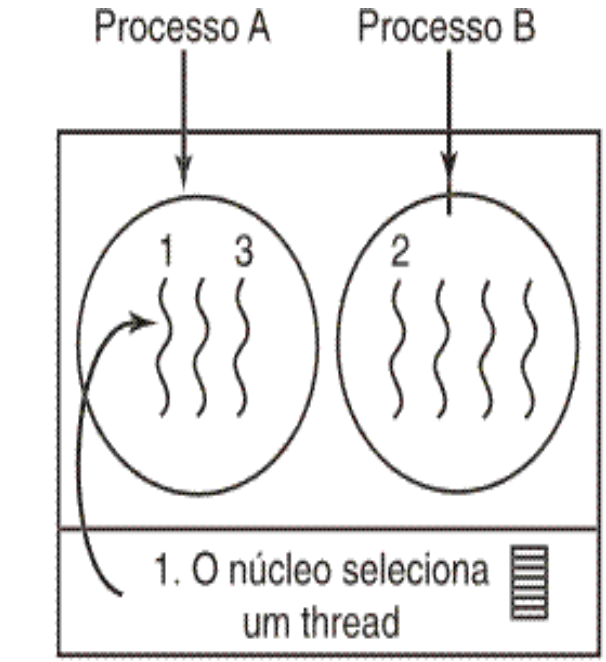
Sistemas Operacionais
Processos - Escalonamento

- Escalonamento de Threads



Possível: A1, A2, A3, A1, A2, A3
Impossível: A1, B1, A2, B2, A3, B3

(a)



Possível: A1, A2, A3, A1, A2, A3
Também possível: A1, B1, A2, B2, A3, B3

(b)

Sistemas Operacionais
Processos - Concorrência

- Programação Concorrente
  - Processos sequenciais executando paralelamente.
  - Consequência
    - Disputa por Recursos (Variáveis, dispositivos, etc)
  - Difícil depuração
    - Ordem em que roda os processos afeta o resultado.
    - Não determinístico

Sistemas Operacionais
Processos - Concorrência

- Condições de Corrida (*Race Conditions*)
  - 2 ou mais processos acessam um dado e o valor final depende da ordem de execução
  - Ordem de execução determinada pelo escalonador normalmente aleatória(Não deterministica)
  - Evitando condições de Corrida
    - Sincronização – Estabelece a ordem de execução
    - Mecanismos de Exclusão mutua
      - Um só processo acessará o dado compartilhado por vez

Sistemas Operacionais Processos - Concorrência

- Região Crítica (Seção Crítica)
  - Parte do código que em que os dados compartilhados são acessados
  - Precisa ser protegida de acessos simultâneos
    - Exclusão mutua

Sistemas Operacionais
Processos - Concorrência

- Exemplo 1

```
Procedure echo();
   var out, in: character;
begin
   input (in, keyboard);
   out := in;
   output (out, display)
end.
```

Sistemas Operacionais
Processos - Concorrência

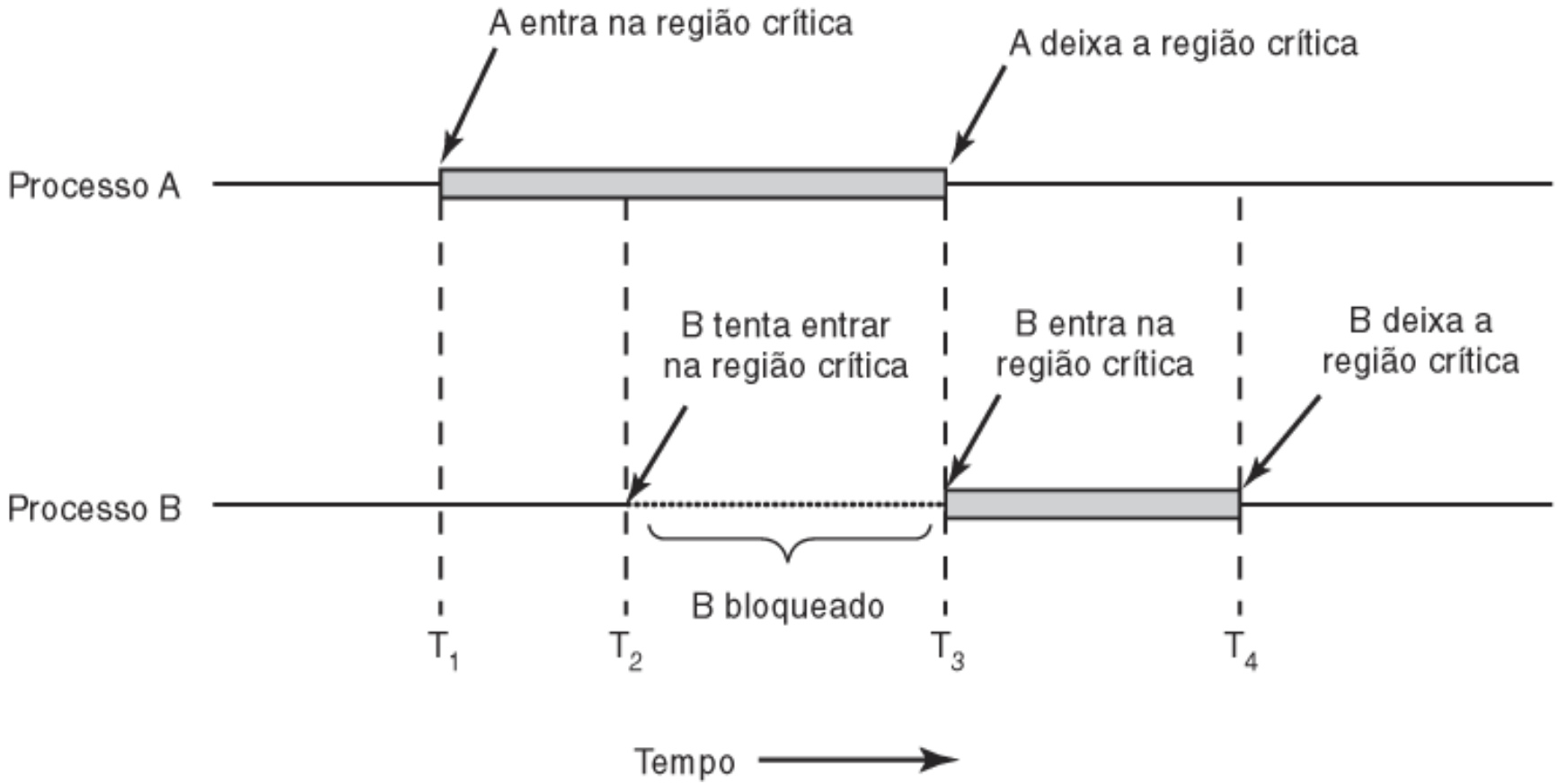
- P1 invoca echo()
  - e é interrompido imediatamente após a conclusão da função input(). Suponha que x tenha sido o caractere digitado, que agora está armazenado na variável in.
- P2 também invoca echo().
  - Suponha que y seja digitado (in recebe y), sendo então exibido no dispositivo de saída.
- P1 retoma a posse do processador.
  - O caractere exibido não é o que foi digitado (x), pois ele foi sobreposto por y na execução do processo P2.
- Conclusão: o caractere y é exibido duas vezes.
- Causa do problema: o compartilhamento da variável global in.

Sistemas Operacionais
Processos - Concorrência

- Exclusão Mutua
  - Requisitos para uma boa solução:
    - A apenas um processo pode estar em sua região critica em um dado instante
    - Nenhum processo que executa fora de sua região crítica pode bloquear outro processo
    - Não se dever supor as velocidades relativas dos processos ou sobre o número de CPUs no sistema.
    - Nenhum processo pode ter que esperar eternamente para entrar em sua região critica ou lá ficar eternamente

Sistemas Operacionais
Processos - Concorrência

- Exclusão Mutua



Sistemas Operacionais
Processos - Concorrência

- Soluções para Exclusão Mutua
  - Baseadas em HW
    - Desabilitar interrupções, Instruções TSL
  - Soluções Baseadas em SW com espera ocupada
    - *Busy-wait*
    - Variáveis de Impedimento, Alternância Obrigatória, Peterson
  - Soluções de SW com bloqueio
    - Sleep/Wakeup, Semáforos, mutex e Monitores

Sistemas Operacionais Processos - Concorrência

- Desabilitar Interrupções
  - Desabilitar a Troca de Contexto
    - Interrupções de Relógio (Preempção)
    - Interrupções de Hardware
    - Interrupções de Software (*Trap*)
  - Problemas
    - Usuário pode ganhar CPU indefinidamente
    - Não garante a exclusão mutua em ambiente multiprocessado
  - Muito usado no nível do sistema operacional

Sistemas Operacionais Processos - Concorrência

- Variáveis de Impedimento
  - Variável compartilhada lock
  - Ao entrar na região critica liga-se a variável.
  - Outro thread que quiser entrar deverá verifica se lock==0.
  - Não funciona (duas operações).
    - Checar se lock==0
    - E Fazer lock=1

Sistemas Operacionais
Processos - Concorrência

- Instrução TSL
  - *Test and Set Lock*
    - TSL RX, LOCK
    - Lê o conteúdo da posição de memória apontada por LOCK no registrador RX, se diferente de ZERO, então armazena um valor diferente de zero em LOCK.
  - Usa espera ociosa (*Busy Wait)*

Sistemas Operacionais
Processos - Concorrência

- Instrução TSL

```
enter_region:
        TSL REGISTER,LOCK       | copia lock para o registrador e põe lock em 1
        CMP REGISTER,#0         | lock valia zero?
        JNE enter_region        | se fosse diferente de zero, lock estaria ligado,
                                  portanto continue no laço de repetição
        RET | retorna a quem chamou; entrou na região crítica

leave_region:
        MOVE LOCK,#0            | coloque 0 em lock
        RET | retorna a quem chamou
```

Sistemas Operacionais
Processos - Concorrência

- Alternância Obrigatória
  - Evolução do método de Variável de Impedimento
    - Spin-lock
  - Variável controla quem entra na região crítica
  - Ao sair da região crítica o processo passa a vez
  - Espera Ociosa (Ruim)
  - Viola a Condição
    - Nenhum processo que executa fora de sua região crítica pode bloquear outro processo

Sistemas Operacionais
Processos - Concorrência

- Alternância Obrigatória

```
while (TRUE) {
    while (turn !=0)          /* laço */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn !=1)          /* laço */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Sistemas Operacionais
Processos - Concorrência

- Solução de Peterson

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* número de processos */

int turn;                               /* de quem é a vez? */
int interested[N];                      /* todos os valores inicialmente em 0 (FALSE) */

void enter_region(int process);         /* processo é 0 ou 1 */
{
      int other;                        /* número de outro processo */

      other = 1 – process;              /* o oposto do processo */
      interested[process] = TRUE;       /* mostra que você está interessado */
      turn = process;                   /* altera o valor de turn */
      while (turn == process && interested[other] == TRUE) //* comando nulo */;
}

void leave_region(int process)          /* processo: quem está saindo */
{
      interested[process] = FALSE;      /* indica a saída da região crítica */
}
```

Sistemas Operacionais Processos - Concorrência

- Dormir e Acordar
  - Sleep e Wakeup
  - Não usa *busywait*
  - Não tem problemas de inversão de Prioridades
    - Com em TSL, e Peterson
  - Problema pode haver perda do sinal de acordar
    - Ex: Problema dos produtores e consumidores
      - Antes do consumidor executar o sleep() o produtor é escalonado

Sistemas Operacionais Processos - Concorrência

- Semáforos
  - Primitivas
    - Down(s) → Se s!=0: s=s-1 , senão dorme
    - Up(s) → Se alguém estiver dormindo s=0 e acorda, senão s=s+1
  - Ações atômicas
    - Verificar, Alterar, Dormir
    - Chamadas de Sistema
      - Desabilitar interrupções em nível de SO

Sistemas Operacionais Processos - Concorrência

- Mutex
  - Versão Simplificada dos semáforos
  - Variável compartilhada (uso em threads)
  - mutex_lock
    - Para ter acesso a região critica
    - Dorme se estiver ocupado
  - Mutex_unlock
    - Escolha aleatória do thread que vai acordar.

Sistemas Operacionais
Processos - Concorrência

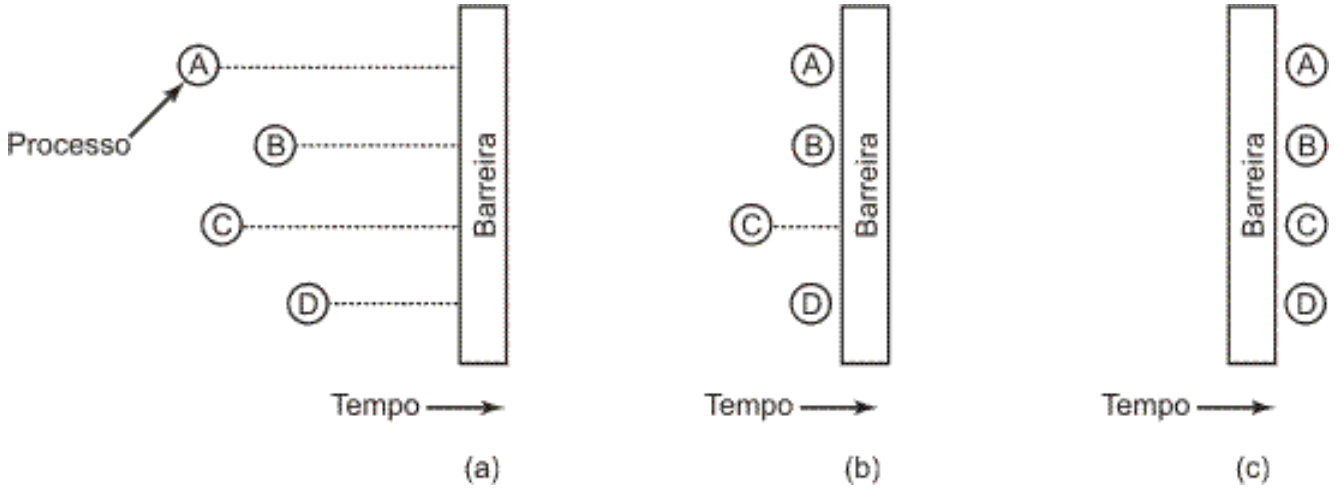
- Produtor e consumidor sem memória comum
  - Ex: Diferentes máquinas
  - Troca de Mensagens
    - Send(destino, mensagem)
    - Receive(destino, mensagem)
    - Mensagem de Reconhecimento
      - Vazia

Sistemas Operacionais
Processos - Concorrência

- Problemas clássicos
  - Barbeiro Dorminhoco
  - Jantar dos filósofos

Sistemas Operacionais
Processos - Concorrência

- Barreiras
  - Sincronização de Vários processos
    - Checkpoint



Sistemas Operacionais
Unidade 1

- Capítulos do Livro
  - Capítulo 1 completo.
  - Capítulo 2
    - 2.1 Processos
    - 2.2 Threads
    - 2.3 Concorrência
      - (até 2.3.6)
    - 2.5 Escalonamento