Sistemas Operacionais

Processos e threads.

Prof. Felipe Leite Lobo
felipe.lobo@ufrr.br

Processos

- Processo é a chave do Sistema Operacional;
- Eles mantêm a capacidade das operações pseudo concorrentes;
- Em qualquer sistema multiprogramado a CPU chaveia de programa para programa, executando cada um deles por dezenas ou centenas de milissegundos;

Processos

- No espaço de um segundo, a CPU pode trabalhar sobre vários programas;
- Modelo de Processo:
- Todos os softwares são organizados em vários processos sequênciais;
- Todo processo possui sua CPU virtual;

Processos

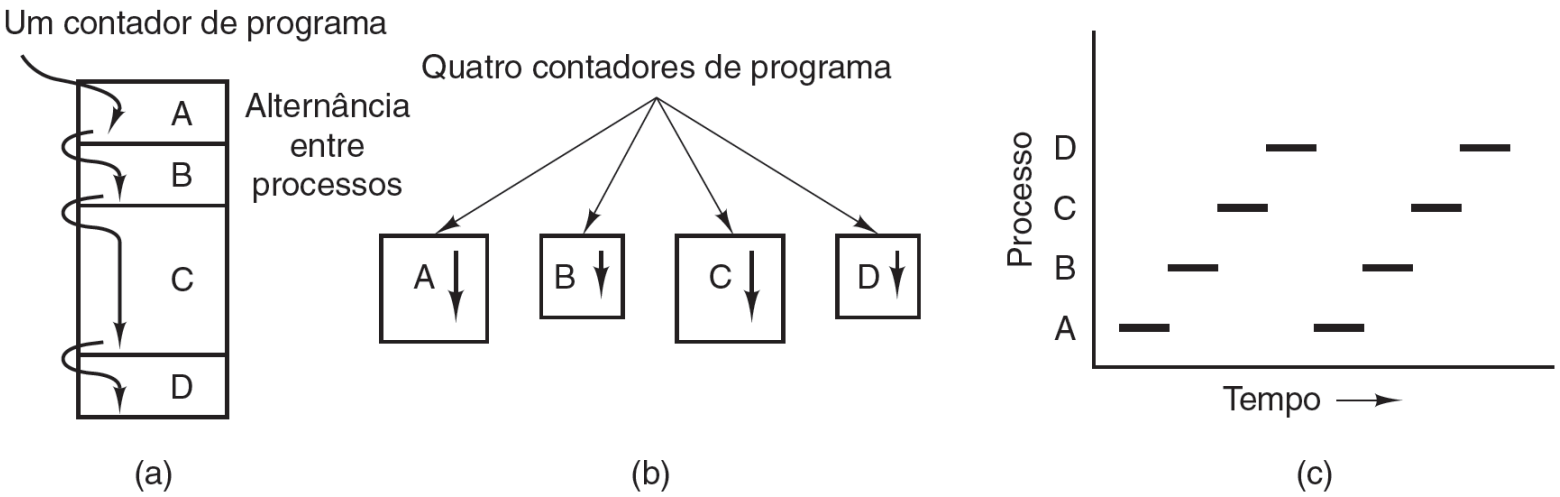


Figura 2.1 (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Somente um programa está ativo a cada momento.

Processos

- **Um processo é caracterizado por um programa em execução;**
- Cada processo possui:
  - Um espaço de endereço;
  - Uma lista de alocação de memória (mínimo, máximo);
  - Um conjunto de registradores (contador de programa, ponteiro de pilha e outros);
- O Sistema Operacional controla todos os processos;

Processos

- Criação de processos: Há quatro eventos principais que fazem os processos serem criados:

  - **Início do sistema:** Processos em primeiro plano – Interagem com o usuário; Processos em segundo plano – *Deamons*.

  - **Execução de uma chamada de sistema de criação do processo por um processo em execução:** Um processo em execução fará chamadas de sistema para criar um ou mais processos, para ajudar em seu trabalho.

Processos

- **Uma requisição do usuário para criar um novo processo:** O usuário ao clicar em algum ícone ou executar um comando, cada uma dessas ações criam um processo.

- **Início de uma tarefa em lote:** Usuários podem submeter, mesmo que remotamente, tarefas em lote.

Processos

- Término de Processos: Normalmente processos terminam por quatro motivos:

  - **Saída normal:** Processo termina porque terminou seu trabalho, através de uma chamada de sistema para dizer ao S.O. que ele terminou.
  - **Saída por erro:** Ao compilar um programa e o arquivo não existe, o compilador termina a execução.
  - **Erro Fatal:** Geralmente erro do programa devido a alguma instrução ilegal.
  - **Cancelamento por outro processo:** Um processo executa uma chamada de sistema dizendo ao S.O. para cancelar outro processo.

Processos

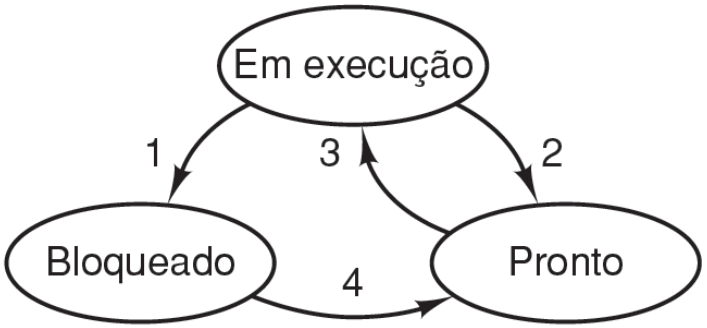
- Hierarquia de processos:
  - Linux X Windows;
  - Linux: Todos os processos filhos e descendentes formam um grupo de processos;
    - Quando um sinal é enviado, esse é entregue a todos os membros do grupo;
    - Individualmente, cada processo pode tomar uma ação;
    - Ex: *init,* lê um arquivo e indica quantos terminais existem. Então, um processo para cada terminal é criado;

Processos

- Hierarquia de processos:
  - Windows:
    - Não há a noção de hierarquia;
    - Ao criar um processo, ao pai é dado um identificador especial *(handle);*
    - O que invalida a hierarquia é que o processo é livre para passar o identificador para outros processos,

Processos

- Estados de Processos:
  - Em execução;
  - Pronto;
  - Bloqueado;



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 Um processo pode estar nos estados em execução, bloqueado ou pronto. As transições entre esses estados são mostradas.

Processos

Gerenciamento de processo	Gerenciamento de memória	Gerenciamento de arquivo
Registros	Ponteiro para informações sobre o segmento de texto	Diretório-raiz
Contador de programa		Diretório de trabalho
Palavra de estado do programa	Ponteiro para informações sobre o segmento de texto	Descritores de arquivo
Ponteiro da pilha		ID do usuário
Estado do processo	Ponteiro para informações sobre o segmento de texto	ID do grupo
Prioridade		
Parâmetros de escalonamento		
ID do processo		
Processo pai		
Grupo de processo		
Sinais		
Momento em que um processo foi iniciado		
Tempo de CPU usado		
Tempo de CPU do processo filho		
Tempo do alarme seguinte		

Tabela 2.1 Alguns dos campos de um processo típico de entrada na tabela.

Processos

1. O hardware empilha o contador de programa etc.
2. O hardware carrega o novo contador de programa a partir do arranjo de interrupções.
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.

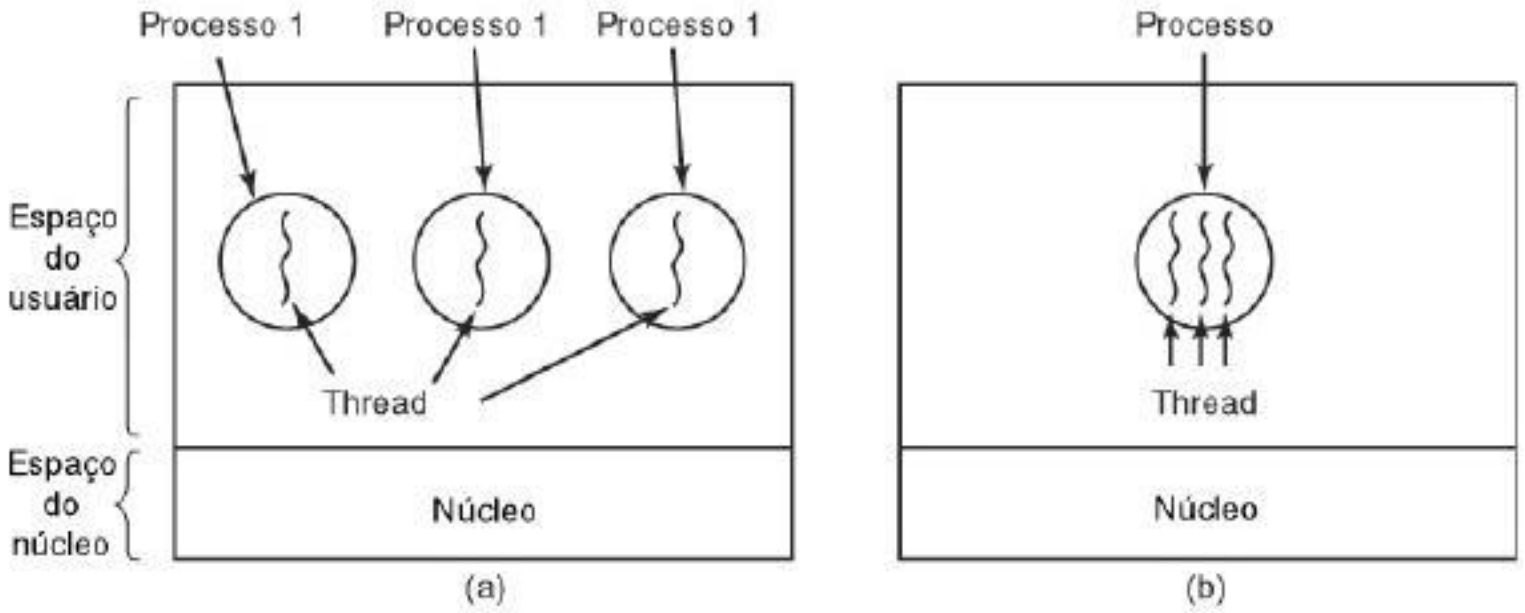
Tabela 2.2 O esqueleto do que o nível mais baixo do sistema operacional faz quando ocorre uma interrupção.

Processos e Threads

- Sistemas Operacionais tradicionais:
  - Cada processo possui:
    - Espaço de endereçamento;
    - Um único thread de controle;
    - Às vezes é desejável ter múltiplos threads de controle no mesmo espaço de endereçamento.

- Threads – São os microprocessos dentro de um processo, executados quase em paralelos.

Processos e Threads



Processos e Threads

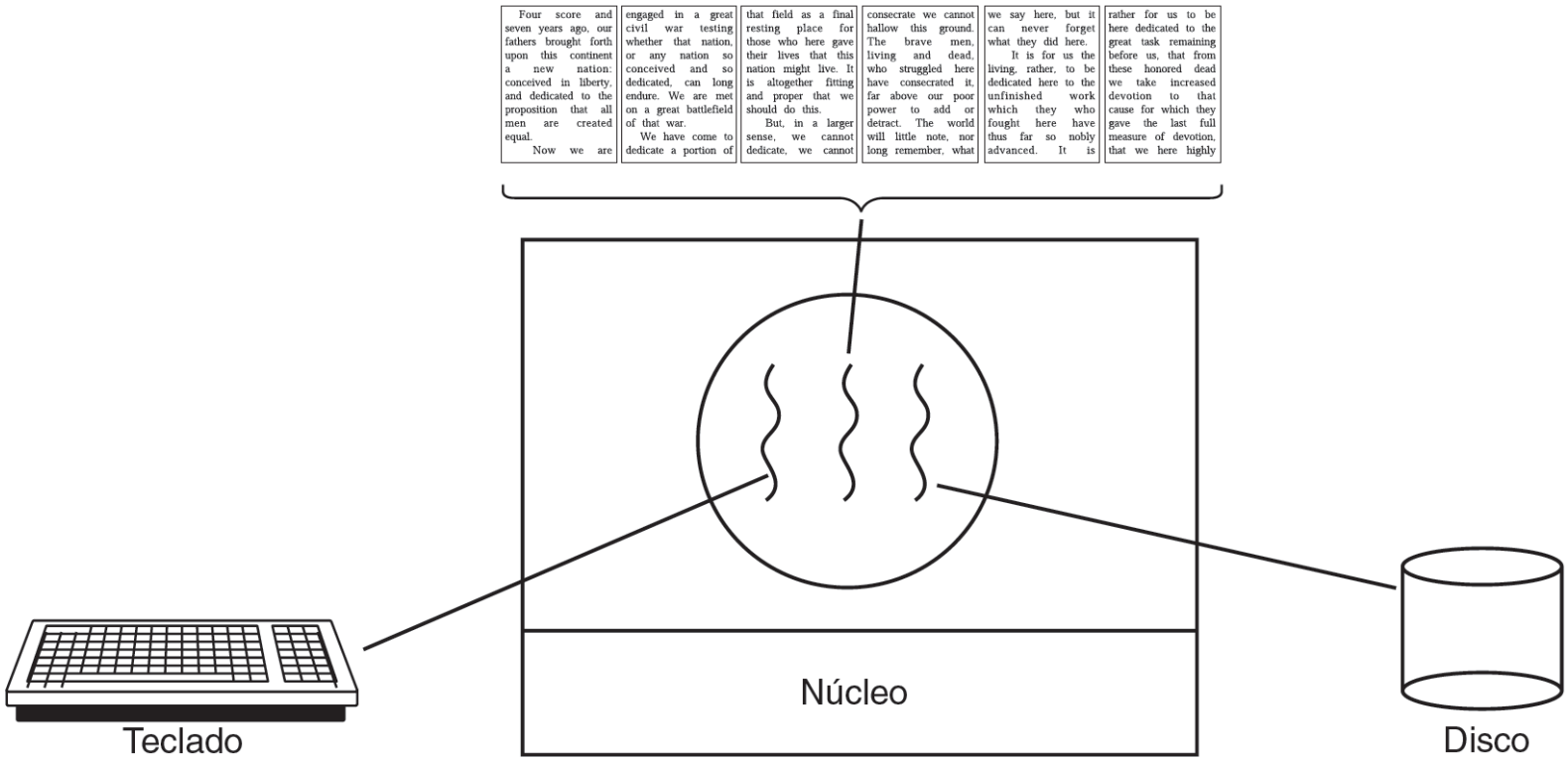


Figura 2.5 Um processador de textos com três threads.

Threads

- Argumentos para threads existirem:
  - Entidades paralelas compartilharem o mesmo espaço de endereçamento e todos os seus dados entre elas.
  - São mais rápidas de criar e destruir que os processos.

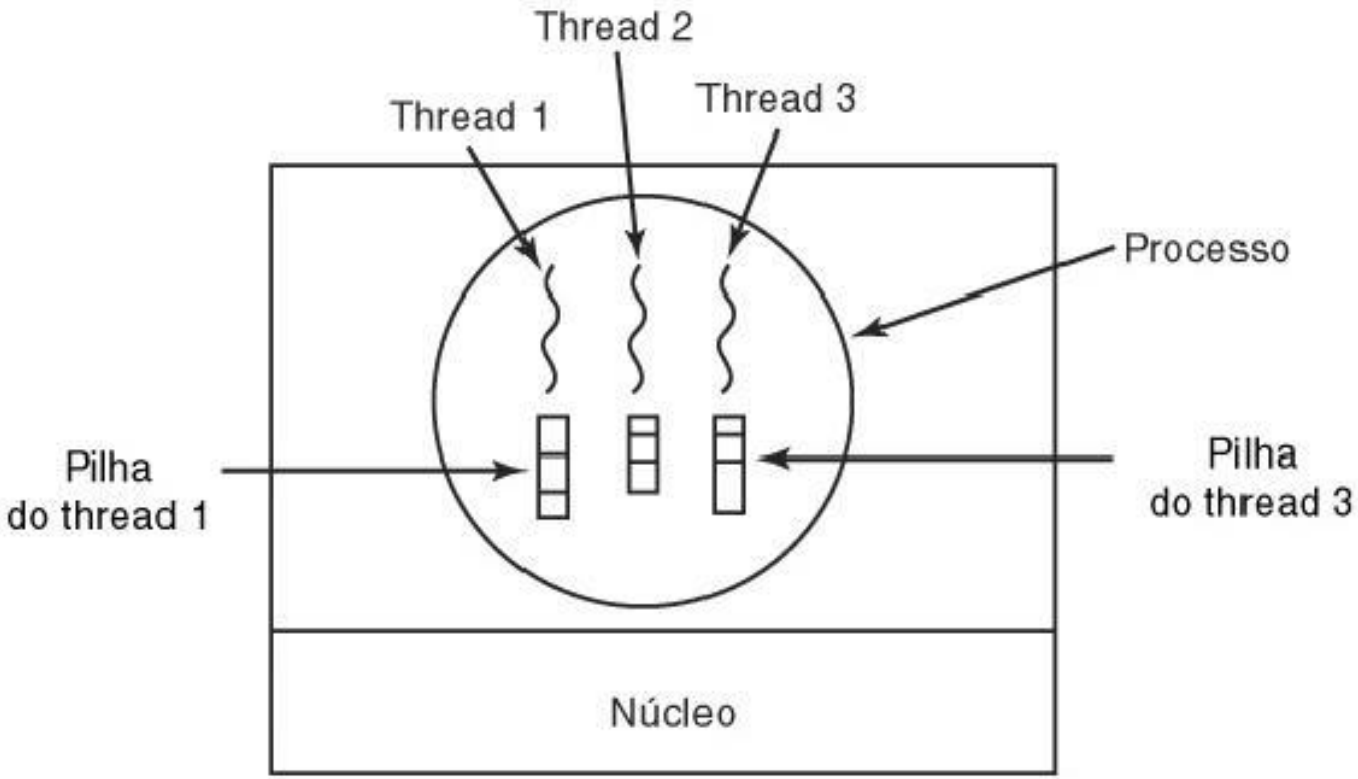
Threads

- Importante:
  - Múltiplos threads executando em paralelo em um processo é análogo a múltiplos processos executando em paralelo em um único computador.
  - As CPU's que tem suporte de *hardware* direto a *multithread*, permitem a ocorrência de chaveamento de threads em uma escala de tempo de nanossegundos.

Threads

- Threads têm exatamente o mesmo espaço de endereçamento;
- Cada thread:
  - Tem acesso a qualquer endereço de memória dentro do espaço do endereçamento do processo.
  - Tem sua própria pilha, que contém estrutura para cada rotina chamada (chamadas de sistema).

Threads


Thread 2
Thread 1
Thread 3
Processo
Pilha
do thread 1
Pilha
do thread 3
Núcleo

Threads

- Complicações que um thread pode causar:
  - No Linux, processos filhos tantos threads quanto o pai.
  - Threads compartilham muitas estruturas de dados. (ex: um thread fechar um arquivo enquanto outro ainda estiver lendo o mesmo)

Tipos de Threads

- POSIX: Suportada pela maioria dos sistemas UNIX;

- Chamadas de thread:
  - pthread_create;
  - pthread_exit;
  - pthread_join;
  - pthread_yield;

Tipos de Threads

- Threads no espaço do usuário:
  - Pode ser implementado em S.O. que não suporte threads;
  - São executados no topo de um sistema de tempo de execução (*runtime*).
  - Cada processo possui sua tabela de threads.

Tipos de Threads

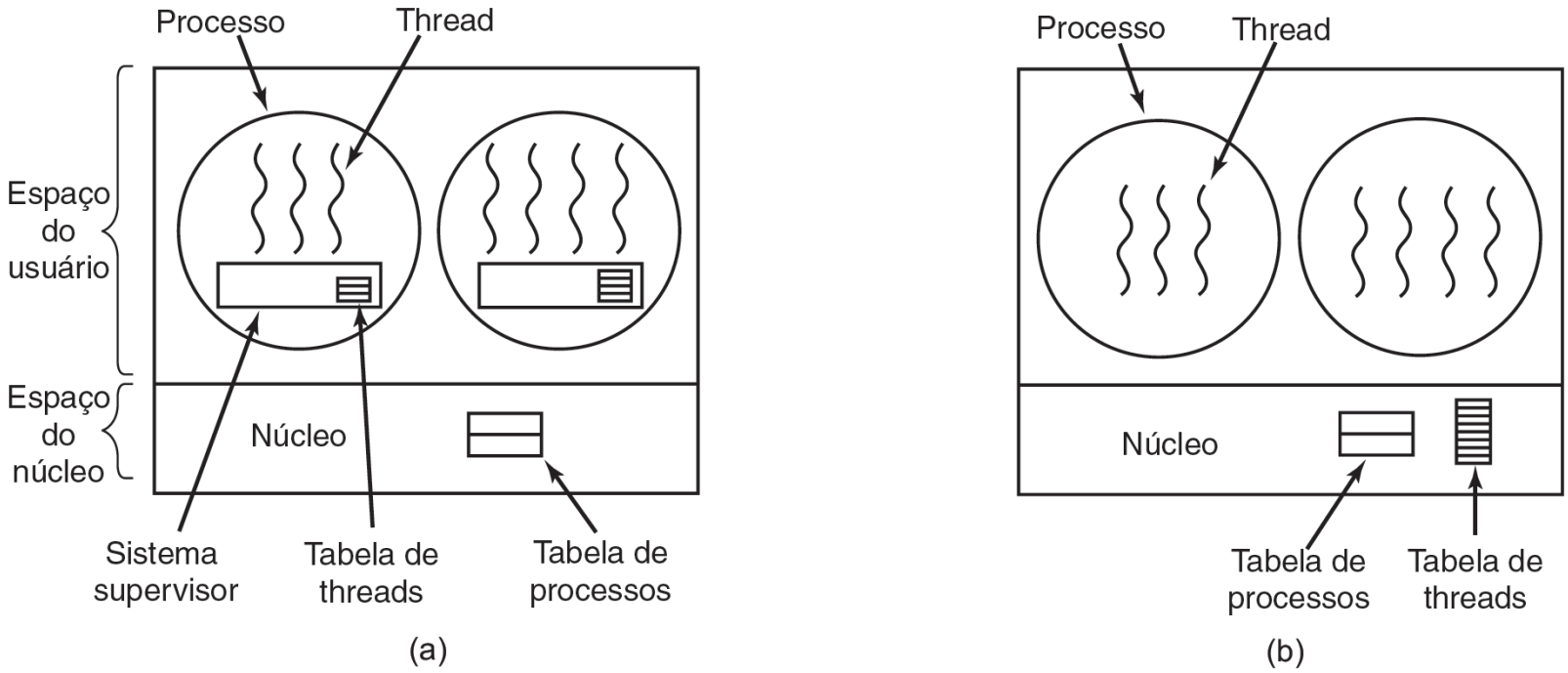


Figura 2.11 (a) Um pacote de threads de usuário. (b) Um pacote de threads administrado pelo núcleo.

Tipos de Threads

- Importante:
  - Todas as chamadas que possam bloquear um thread, são implementadas como chamadas de sistema (custo maior que uma chamada de rotina do sistema de tempo de execução).

Tipos de Threads

- Threads no espaço núcleo:
  - Não necessita de um sistema de tempo de execução;
  - Não há tabela de threads em cada processo;
  - O núcleo tem a tabela de threads de todo o sistema.