Sistemas Operacionais

Threads e Processos

Threads

- Uma *thread* é uma linha de execução de código que executa em paralelo com outras linhas do mesmo processo, compartilhando seu espaço de memória.
- Na prática uma thread é equivalente a um “mini-processo” dentro de um processo
- Isto permite que várias ações sejam executadas em paralelo por um mesmo processo

Threads

- Em um programa muitas vezes é necessário executar mais de uma atividade ao mesmo tempo
  - ex.: aguardar a entrada de dados do usuário e reproduzir um som enquanto aguarda
- Uma thread é muito mais leve que um processo comum.
- Ganho de performance na criação e destruição de threads se comparada a processos (10 a 100x)
- Quando uma aplicação tem atividade *I/O bound* e *CPU bound* as threads podem acelerar a execução, pois não concorrerão por recurso.

Threads

- O uso de Threads pode também garantir um uso máximo dos vários processadores existentes em uma CPU

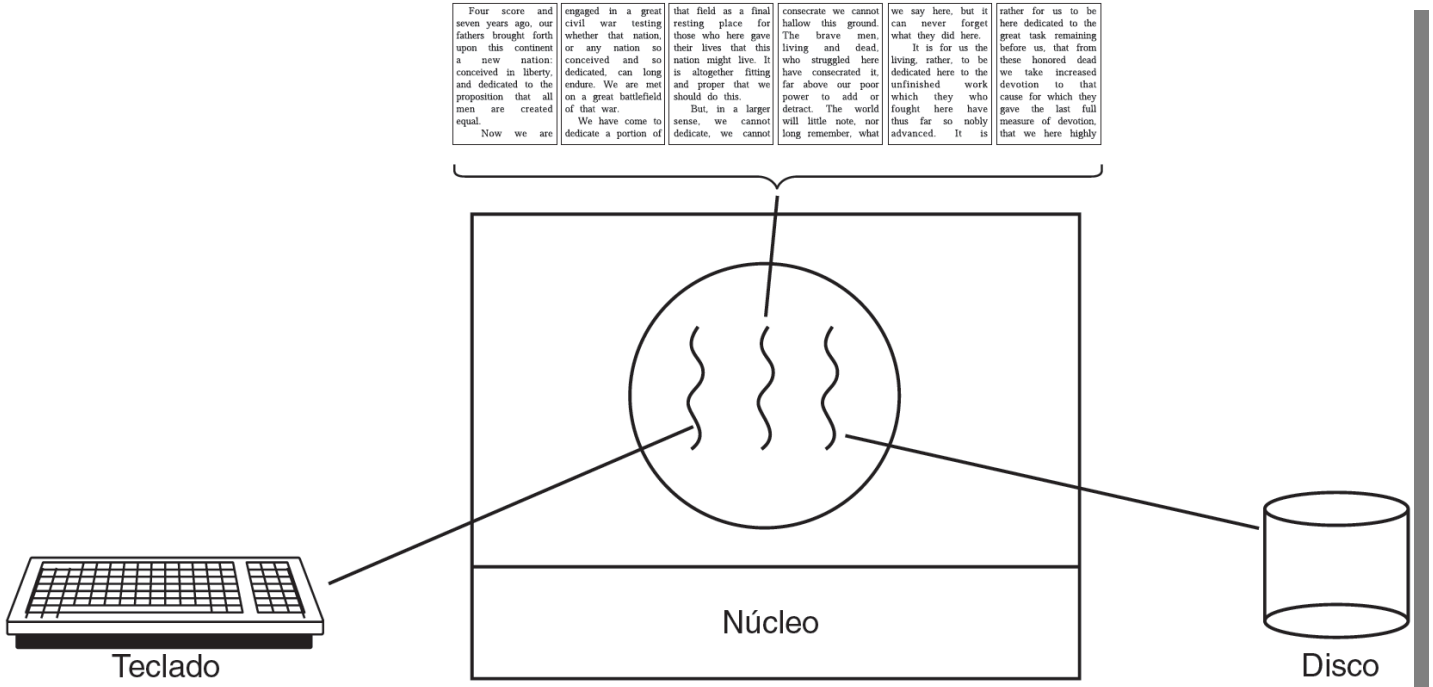
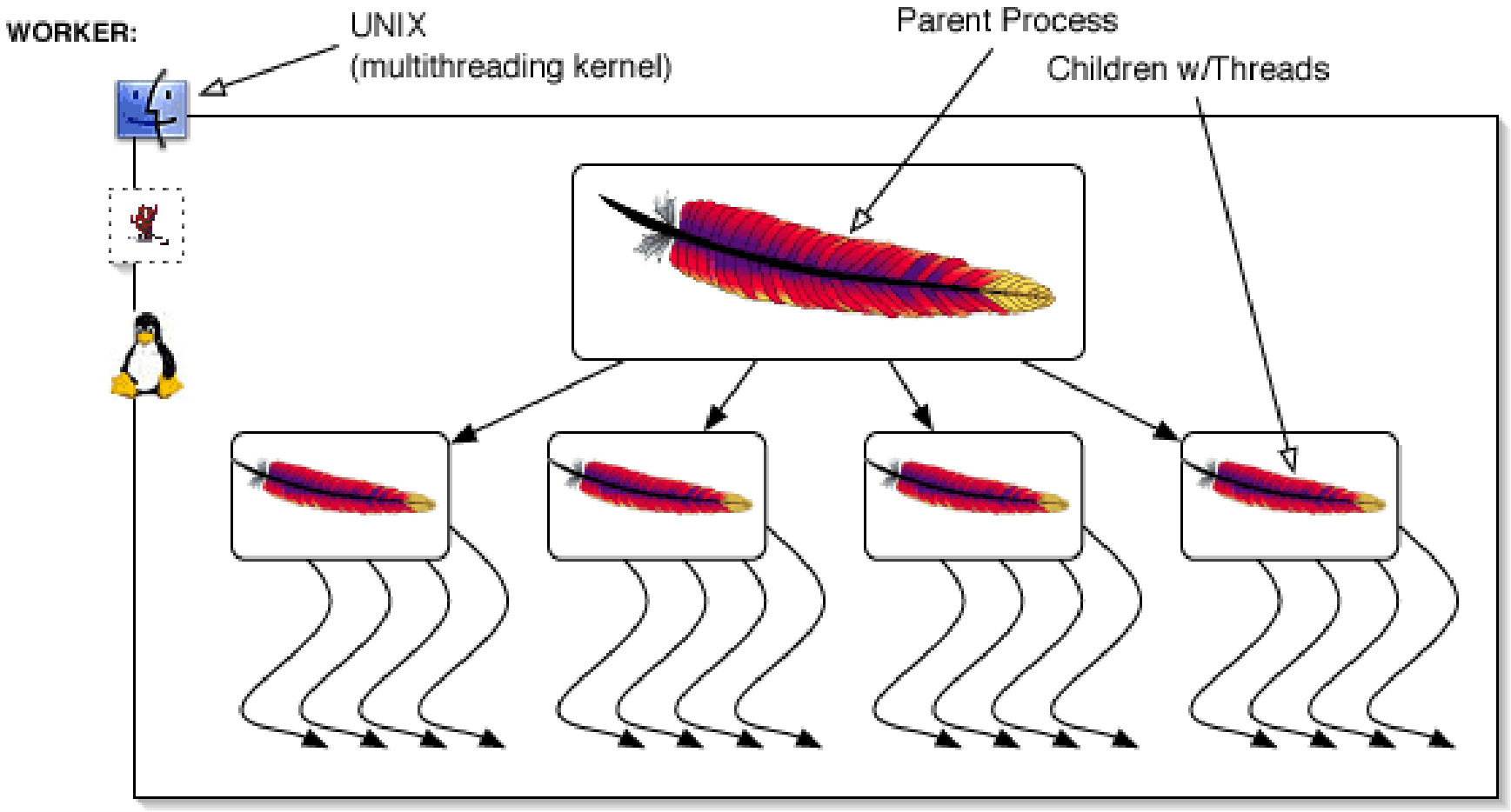


Figura 2.5 Um processador de textos com três threads.

Threads

- Cenários de uso



Uso de Threads

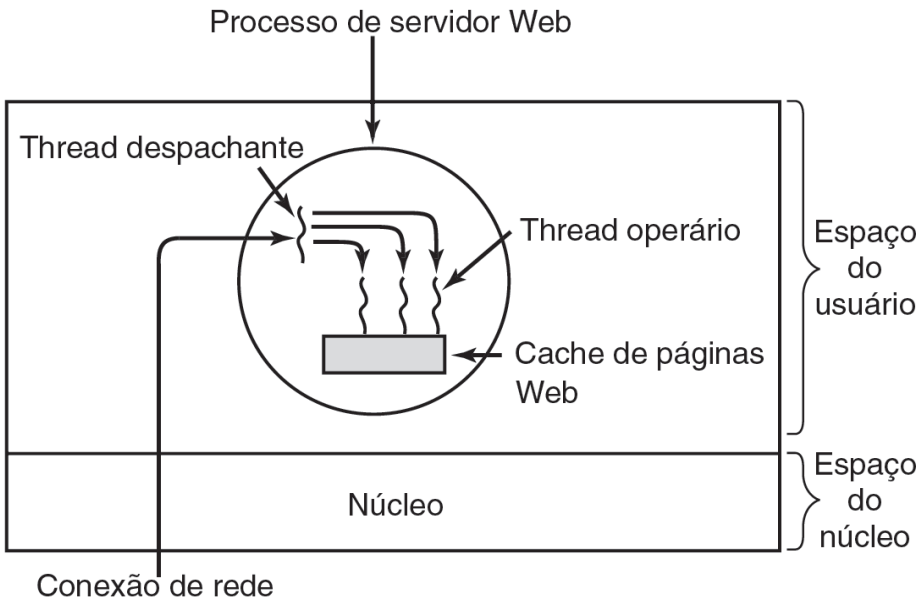


Figura 2.6 Um servidor Web multithread.

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b)

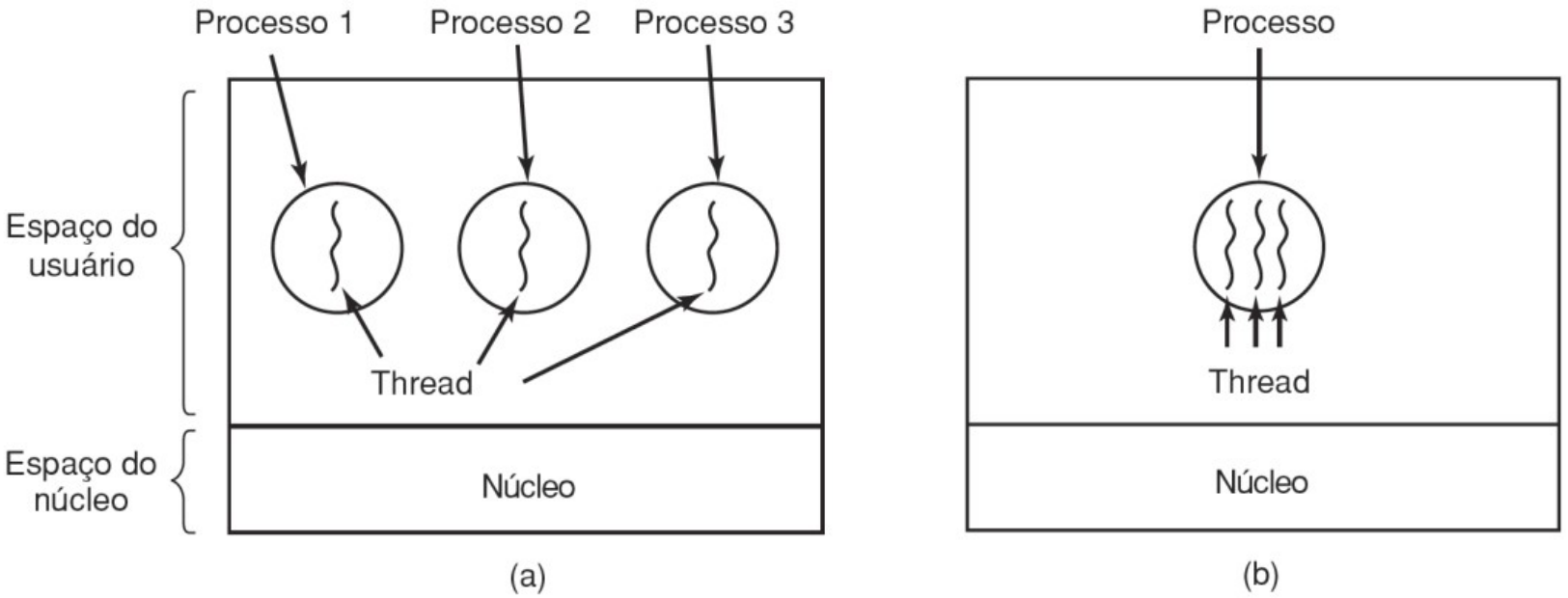
Figura 2.7 Uma simplificação do código para a Figura 2.6. (a) Thread despachante. (b) Thread operário.

Comparativo

Modelo	Características
Threads	Paralelismo, chamadas de sistema bloqueante
Processo monothread	Não paralelismo, chamadas de sistema bloqueantes
Máquina de estados finitos	Paralelismo, chamadas não--bloqueantes, interrupções

Tabela 2.3 Três modos de construir um servidor.

Modelo de implementação pelo S.O.



Recursos de 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 manipuladores de sinais	
Informação de contabilidade	

Tabela 2.4 A primeira coluna lista alguns itens compartilhados por todos os threads em um processo. A segunda lista alguns itens específicos a cada thread.

Comandos comuns para Threads

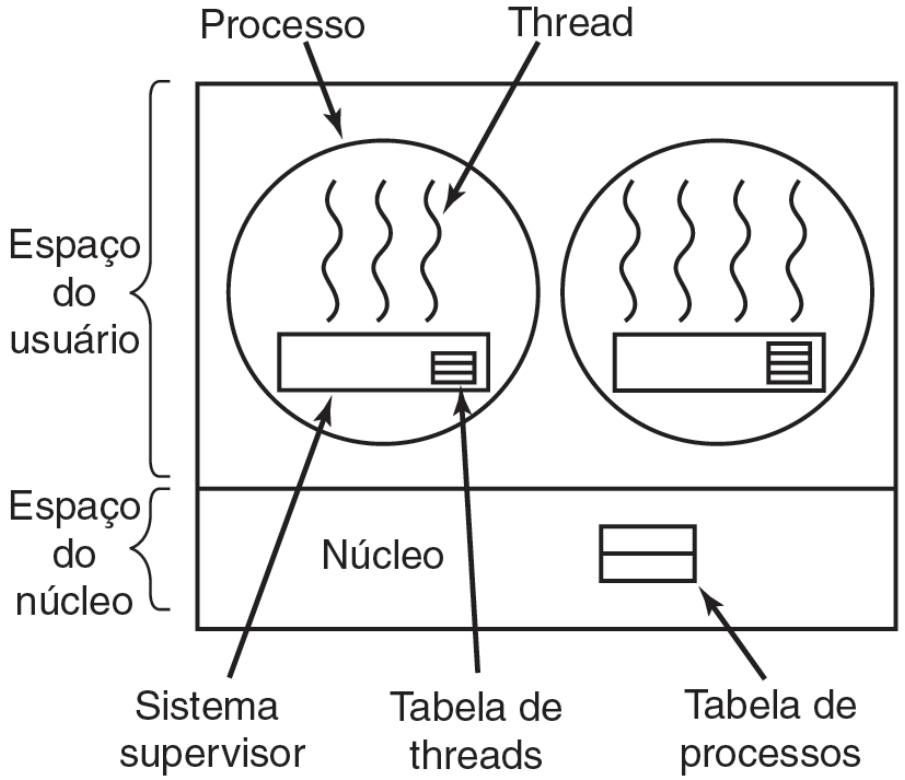
- `Create()` - Cria uma nova thread comumente passando como parâmetro um procedimento/método que esta irá exeutar
- `Exit()` - Encerra uma thread liberando os recursos alocados para esta
- `Join()` - Aguarda que uma outra thread termine para continuar a execução, útil quando uma thread necessita de dados de outra
- `Yield()` ou `Sleep()` - Libera a CPU e volta para a fila de pronto, comum quando a atividade da thread não é necessária no momento

Implementação de threads pelo S.O.

- Threads podem ser implementadas em nível de usuário ou em nível de kernel.
- As duas alternativas são válidas e tem vantagens e desvantagens.

Threads em nível de usuário

- Nesse modelo uma biblioteca de threads é responsável por gerenciar a multiprogramação e escalonamento das threads



Threads em nível de usuário

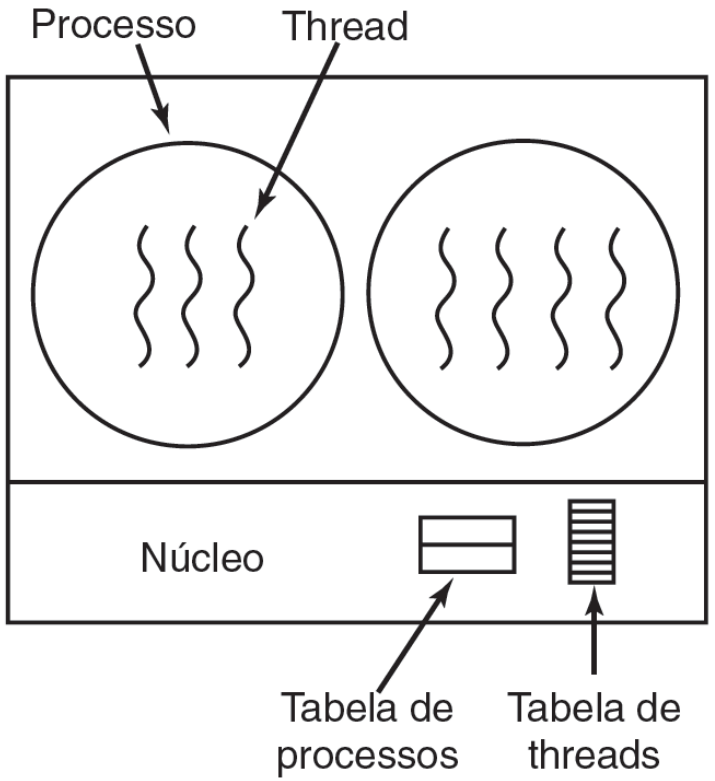
- Nesse modelo a mudança de uma trhread para outra é rápida pois não envolve necessariamente uma *system_call*
- A troca entre esses tipos de threads é da ordem de nanosegundos.
- Permitem que cada processo use o algoritmo de escalonamento que achar mais adequado
- São mais economicas quanto ao espaço de memória (no kernel) necessário para sua implementação.

Threads em nível de usuário

- Apesar de sua versatilidade e velocidade as threads em nível de usuário tem suas desvantagens:
  - Uma chamada bloqueante pode bloquear todo o processo e não apenas a thread em questão
  - Uma *page_fault* pode ocorrer o que causará uma *system_call* e também parará todo o processo
  - Dentro de um processo não há como o escalonador contar o tempo de uso da CPU para retirar uma thread de execução a não ser que esta execute *yield.*
  - A maioria dos processos de usuário que usam threads o fazem pela característica I/O bound o que implica que bloquearão o sistema sempre que fizerem I/O

Threads em nível de kernel

- Nesse modelo o kernel cuida da criação e escalonamento das threads de todos os processos



Threads em nível de kernel

- As chamadas de criação e fim de threads são gerenciadas pelo kernel
- Isso significa um *overhead* para essas operações
- Este modelo de implementação resolve um problema grave das threads em nível de usuário:
  - Threads que bloqueiam um processo inteiro
- Uma estratégia para diminuir o *overhead* é reciclar threads

Estratégia híbrida

- Múltplas threads mapeadas em uma thread do kernel

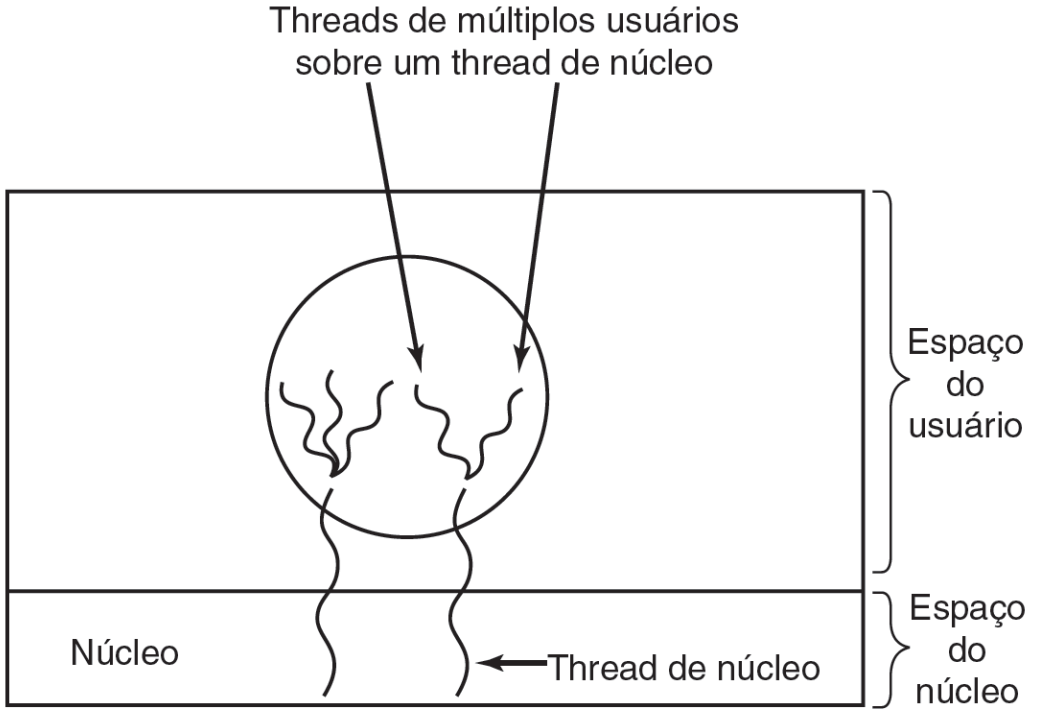


Figura 2.12 Multiplexando threads de usuários sobre threads de núcleo.

Atividade

- Defina Thread.
- Dê um exemplo de uso de threads em um sistema, diferente dos apresentados em sala.
- Por que cada thread deve ter sua pilha (*stack*)?
- Descreva uma vantagem do uso de threads de usuário.
- Descreva uma vantagem do uso de threads de kernel.

Endereço para entrega: https://goo.gl/Y0nClq