UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

Rus0013 - Sistemas Operacionais

Aula 04: Threads

Professor Pablo Soares

2022.2

Sumario

1. Threads
2. Uso de Threads
3. Modelo de Threads
4. Implementação de Threads
    1. Usuário
    2. Núcleo
    3. Híbridas

Threads

- Cada processo tem um espaço de endereçamento
  - Único Threads (Fluxo de Controle)
  - Threads quase igual a um processo
- Frequentemente é desejável ter múltiplas Threads
  - No mesmo espaço de endereçamento
  - Executando quase que em paralelo

Uso de Thread

- Por que alguém desejaria ter um tipo de processo dentro de um processo?
  - A principal razão (thread)
    - Muitas aplicações ocorrem múltiplas atividades ao mesmo tempo
- O mesmo argumento para existência de **processos**:
  - No lugar de
    - interrupções, temporizadores e chaveamento de contextos,
- Com threads
  - "Paralelismo"
  - Compartilhamento de endereçamento e dados

Uso de Thread

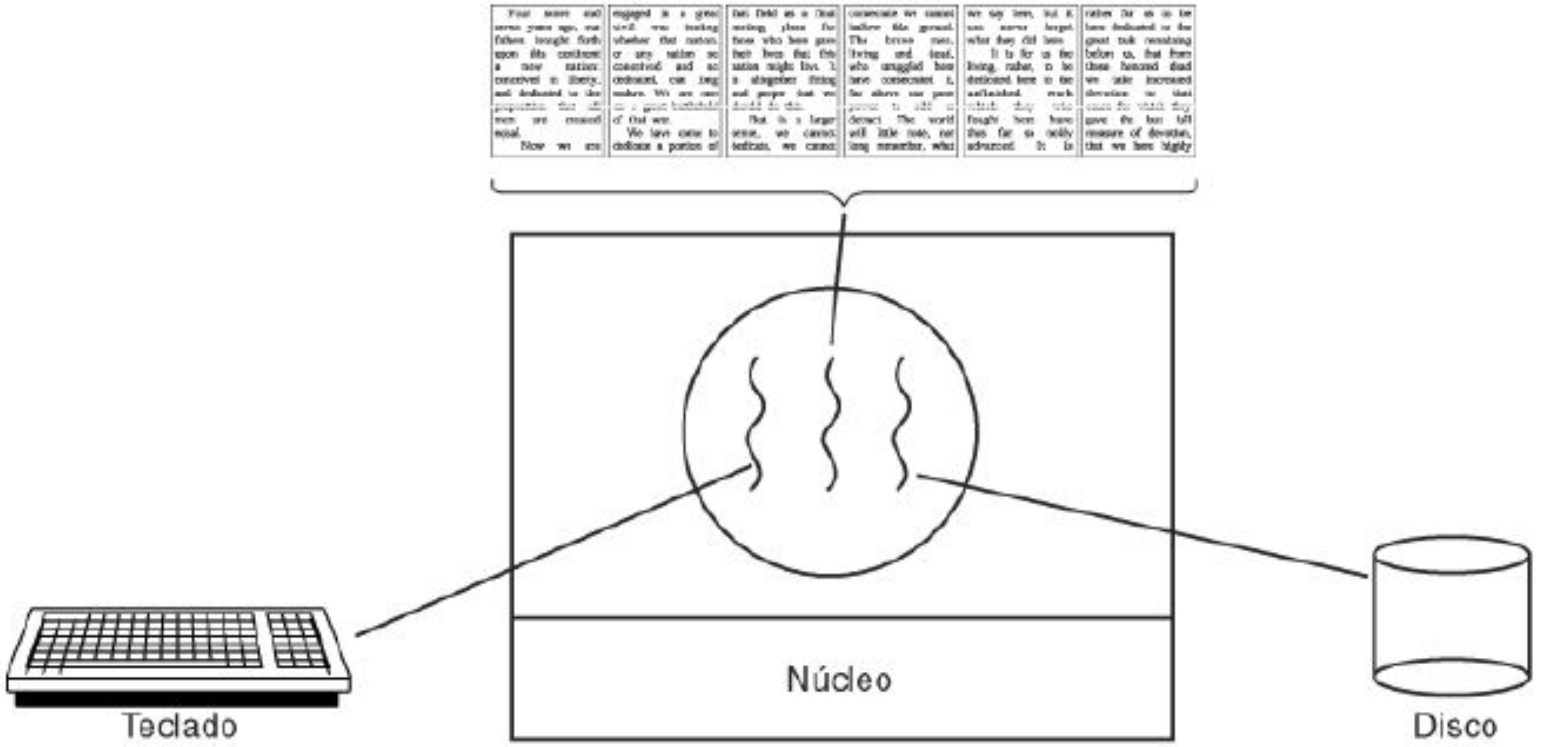
- Argumentos
  1. Compartilhar o espaço de endereçamento e todos os seus dados
     a) Essencial para algumas aplicações
  2. Threads são mais fáceis (rápidos)
     1. Mais leves que processos.
        1. 100 vezes mais rápido criar um thread
  4. Desempenho:
     1. Aplicações que realizam muito E/S
        1. Desempenho melhorado com o uso de threads

Uso de Thread

- **Exemplo prático**
  - Escrita de um livro (800 páginas)
    - Para o autor
      - Livro inteiro (apenas um arquivo)
      - Dividido em Capítulos
    - Mudança das normas da ABNT
      - Apenas um único comando
      - Maior trabalho
  - Remover uma sentença na página 1
    - Depois buscar por palavra específica na página 600

Uso de Thread

- Um processador de texto com 3 threads

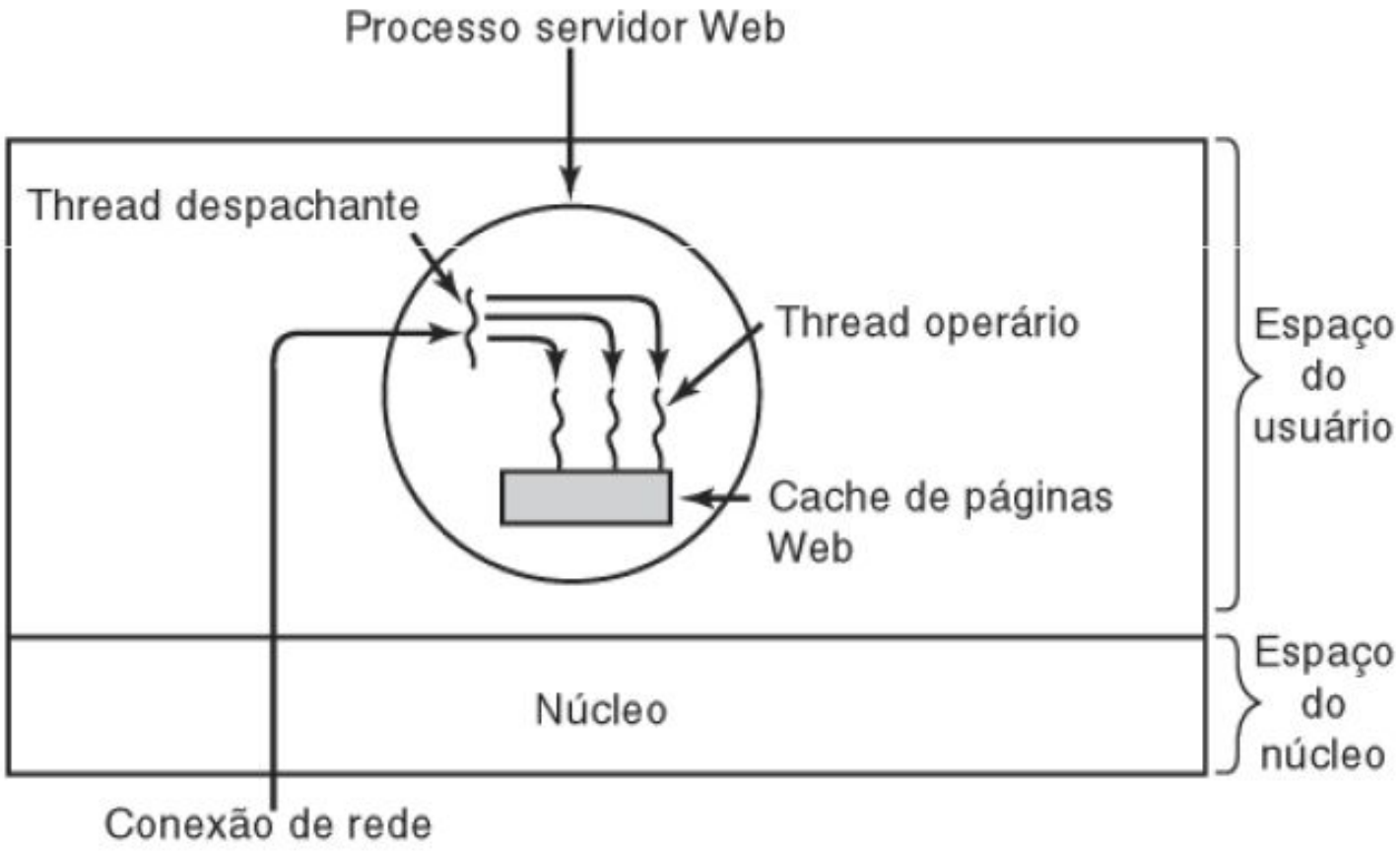


Uso de Thread

- **Esse exemplo prático funcionaria com 3 processos separados?**
  - Sim
    - Por que?
  - Não
    - Por que?

Uso de Thread

- Um servidor web com múltiplos threads



Uso de Thread

- Código simplificado
  - (a) Threads despachante
  - (b) Threads operário

```
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)

Modelo de Threads

- O modelo de processo é baseado em dois conceitos independentes
  - Agrupamento de recursos
    - Arquivos abertos, processo filhos, alarmes pendentes etc
  - Execução
- Algumas vezes é útil separá-los
  - Threads de execução

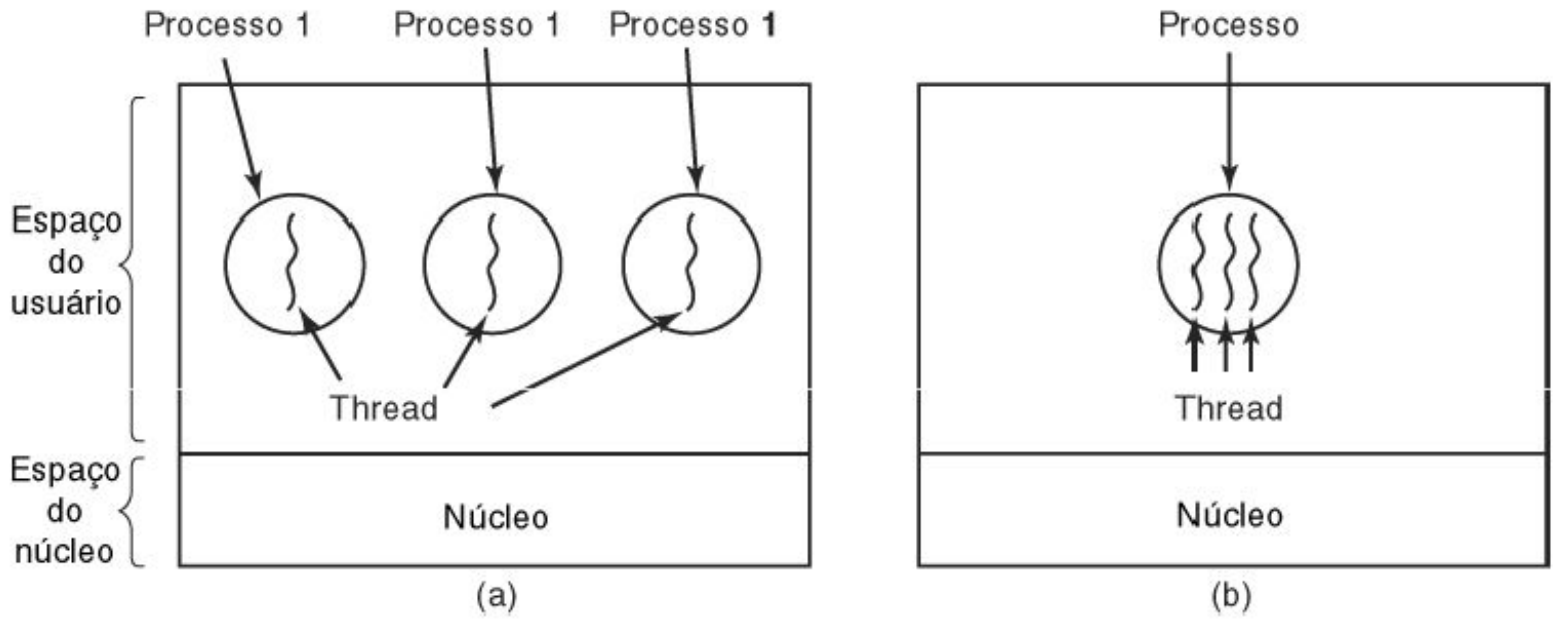
Modelo de Threads

- Threads de execução
  - Contador de programa que mantém o controle de qual a instrução que deve executar em seguida
  - Registradores que contêm suas variáveis atuais
  - Pilha que traz a história da execução, com uma estrutura para cada procedimento chamado mas ainda não retornado
- Threads e processo tem conceitos diferentes
  - **Processos** são usados para agrupar recursos;
  - **Threads** são entidades escalonadas para execução sobre a CPU

Modelo de Threads

- Os threads acrescentam ao modelo de processos
  - Mais de um fluxo de execução de um mesmo processo
    - Independentes um do outro
- Cada thread compartilha grande parte das informações sobre o processo
  - sendo freqüentemente chamados **processos leves**
- Existem informações específicas de cada thread
  - como pilha de execução e variáveis locais

Modelo de Threads



(a) Três processos cada um com um thread

(b) Um processo com três threads

Modelo de Threads

- Threads distintos em um processo não são tão independentes quanto processos distintos
- Todos os threads têm exatamente o mesmo espaço de endereçamento, compartilham o mesmo conjunto de arquivos, processos filhos, alarmes, sinais, etc.
- Não há proteção entre threads porque
  - É impossível
  - Não é necessário

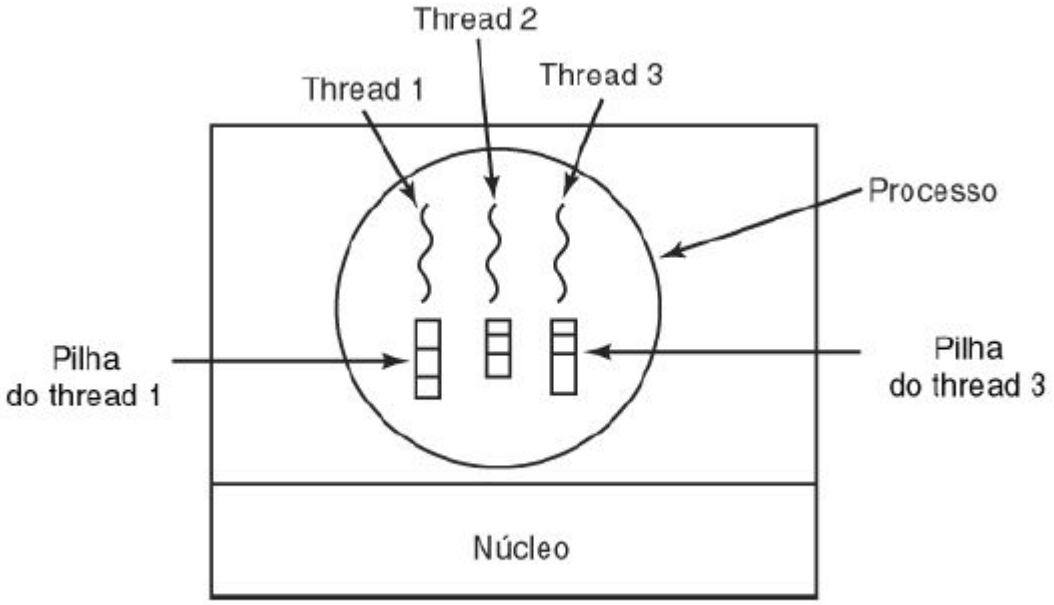
Modelo de Threads

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 tratadores de sinais	
Informação de contabilidade	

- Itens compartilhados por todos os threads em um processo
- Itens privativos de cada thread

Modelo de Threads

- Assim como em processos tradicionais, um thread pode estar em um dos vários estados: em execução, bloqueado, pronto ou finalizado
- Cada thread tem sua própria pilha
  - Estrutura para cada rotina



Modelo de Threads

- Criação e o término do thread são muito parecidos com de processos, inclusive com quase as mesmas opções, mas os threads fazem chamadas a biblioteca
  - *thread_create*
  - *thread_exit*
  - *thread_wait*
  - *thread_yield*

Implementação de Thread

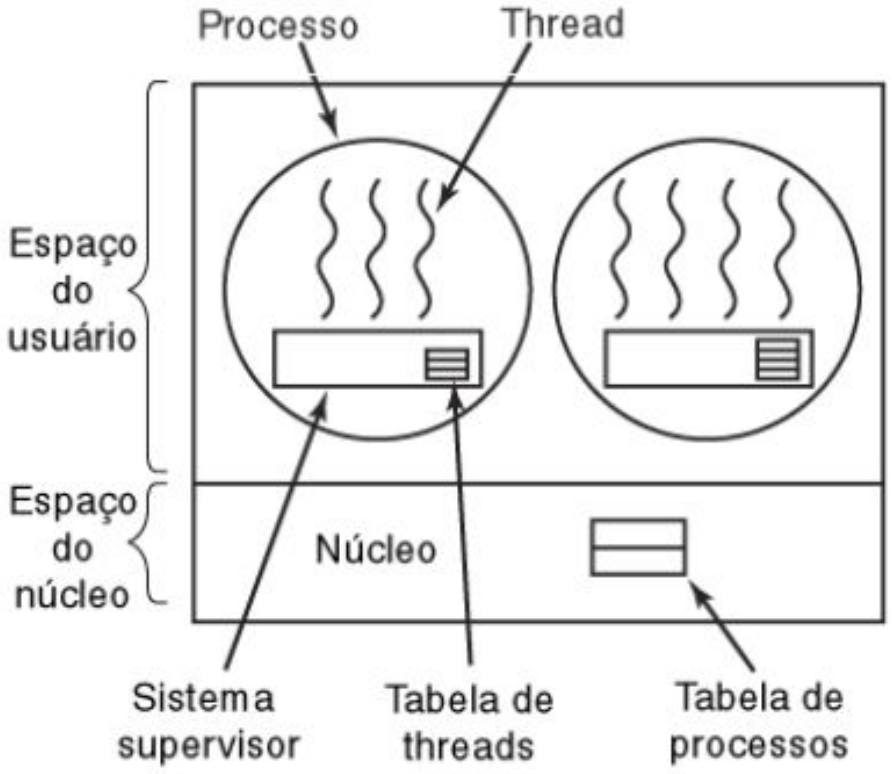
- Podem ser implementadas totalmente no espaço do usuário, pelo SO (no seu núcleo) ou uma implementação híbrida
- Thread de usuário
  - O SO não sabe sobre a existência das threads, na verdade ele nem sequer precisa de suporte a elas
  - Cada processo possui um “supervisor” e uma “tabela de threads”, com as mesmas funções que as tabelas de processos

Implementação de Thread

- Thread de usuário
  - Todas as tarefas relacionadas a criação, execução e término das threads são gerenciadas pelo supervisor. O SO não fica sabendo que elas existem. Esta abordagem é extremamente eficiente, dado que nenhuma interrupção ou alternância entre os modos usuário e núcleo é necessária
  - Cada usuário ou aplicação pode possuir sua própria biblioteca de threads, que atende melhor suas necessidades

Implementação de Thread

- Thread de usuário
  - Pacote de thread de usuário



Implementação de Thread

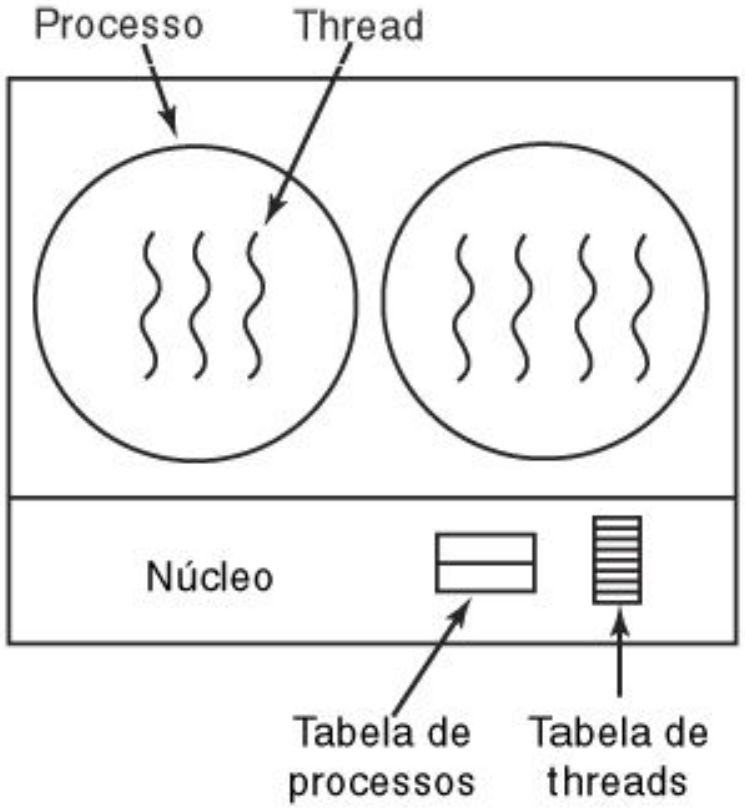
- Thread de usuário
  - Possuem excelente desempenho, mas possui problemas graves
    - Caso o thread realize uma chamada ao sistema como read() e não haja dados disponíveis o processo será bloqueado pelo SO, assim como todas os threads
    - Não há controle do SO sobre os threads, que devem abrir mão da CPU voluntariamente. Caso isso não ocorra ele executará indefinidamente, sempre que o SO escalonar este processo para executar

Implementação de Thread

- Thread de núcleo (**kernel**)
  - Neste caso o SO gerencia os threads. O SO mantém uma tabela de threads além da tabela de processos. Os threads são criados e terminados por chamadas de sistema
  - Uma chamada bloqueante (read()) realizada por um thread irá bloquear especificamente ele
  - Quando um thread bloqueia, o SO decide pela execução de outro thread que pode não pertencer ao mesmo processo
  - A desvantagem dessa abordagem é a perda de desempenho, já que todo o gerenciamento é feito através de chamadas ao SO

Implementação de Thread

- Thread de núcleo (**kernel**)
  - Um pacote de thread de núcleo

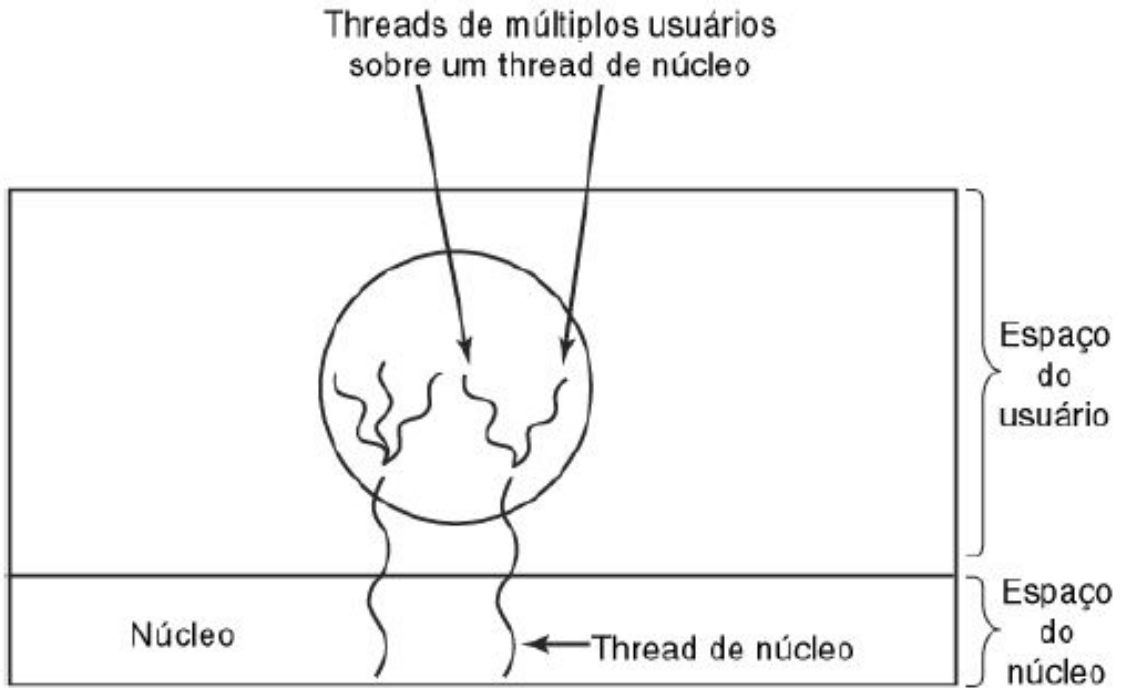


Implementação de Thread

- Implementações híbridas
  - Estratégias que visam combinar as vantagens das duas abordagens
  - A abordagem mais utilizada é possuir suporte aos 2 tipos de threads e agrupar subconjuntos (relacionados) de threads de usuário em uma única thread de núcleo que é tratada diretamente pelo SO

Implementação de Thread

- Implementações híbridas
  - Multiplexação de threads de usuário sobre threads de núcleo



Referências

- Andrew S. Tanenbaum. “Sistemas Operacionais Modernos”. 2ª Edição, Prentice Hall, 2007.

- Francis B. Machado e Luiz P. Maia. “Arquitetura de Sistemas Operacionais”. 3ª. Edição. LTC, 2004.

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

Rus0013 - Sistemas Operacionais

Aula 04: Threads

Professor Pablo Soares

2022.2