PROCESSOS, *THREADS* E GERENCIAMENTO DE PROCESSOS

DCE131 - Sistemas Operacionais

Atualizado em: 12 de maio de 2023

Iago Carvalho

Departamento de Ciência da Computação

MULTIPROGRAMAÇÃO

Um computador moderno possui alguns núcleos de computação

- Cada um deles executa um único processo por vez

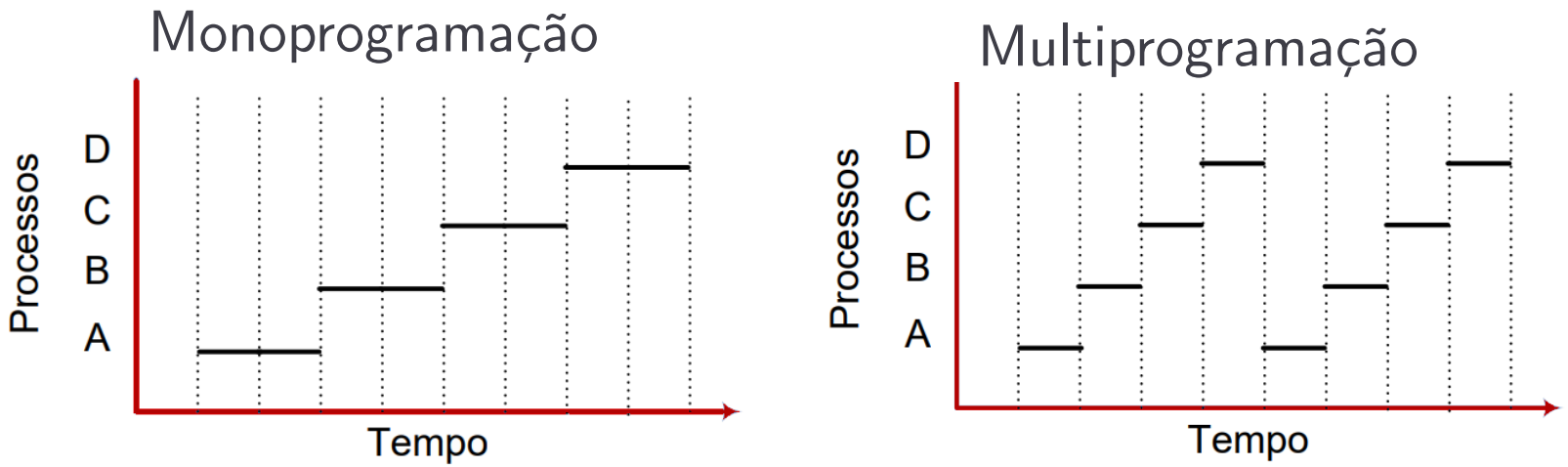
Entretanto, existem mais de uma centena de processos sendo executados a todo momento

- Processos de usuário
- Processos *daemon* (do sistema operacional)

É necessário dar a impressão que tudo está sendo executado ao mesmo tempo

- Para isso, existe a multiprogramação

MULTIPROGRAMAÇÃO



- Na monoprogramação, os processos são executados por completo e nunca deixam a CPU
- Já na multiprogramação, processos podem ser preemptados
  - Isto é, retirados da CPU
  - Cada processo executa por um tempo predeterminado
  - Temos a falsa impressão de paralelismo

MULTIPROGRAMAÇÃO E PARALELISMO

Multiprogramação não é paralelismo!

- Ou é, considerando múltiplos núcleos de processamento

Na multiprogramação, temos somente um processo sendo executado por vez

- Cada processo tem seu contador de programa independente

No paralelismo, temos todos os processos sendo executados ao mesmo tempo

- Também cada um com seu próprio contador de programa

CRIAÇÃO DE PROCESSOS

Cada sistema operacional cria seus processos de forma diferente

- Depende muito da utilidade do sistema

Sistemas embarcados, no geral, criam todos os processos necessários em sua inicialização

Por outro lado, sistemas de uso geral ou multi-usuários tem que criar (e encerrar) processos de forma dinâmica

CRIAÇÃO DE PROCESSOS

Sistemas operacionais podem criar processos em 4 momentos distintos

1. Na inicialização do sistema
2. Durante uma chamada de sistema
   - Entrada e saída
   - Criação de novas linhas de execução de um processo atual
3. A pedido do usuário
   - Quando o usuário quer acessar a um programa
4. Execução de uma tarefa em lote

CRIAÇÃO DE PROCESSOS

UNIX cria processos utilizando o **fork**

- Clona um processo corrente
- Espaço de endereçamento novo
- Logo após, utiliza-se o **execve** para alterar o conteúdo do processo

Windows cria um processo novo utilizando o **CreateProcess**

- Necessário definir os parâmetros do processo no momento de sua criação

FINALIZAÇÃO DE PROCESSOS

De forma voluntária:

- **Saída normal**
- **Saída por erro**

De forma involuntária

- **Erro fatal**
- **Cancelamento por outro processo** O usuário matar processos via terminal.
  - O sistema operacional pode identificar que o processo entrou em *deadlock*.
  - Nem todos os S.O.s implementam esta política;
  - Algoritmos em grafos neste caso são utilizados para a detecção.

FINALIZAÇÃO DE PROCESSOS

Ambos UNIX e Windows permitem a finalização voluntária e involuntária

UNIX

- Termino voluntário com **exit**
- Término involuntário com **kill**

Windows

- Termino voluntário com **ExitProcess**
- Término involuntário com **TerminateProcess**

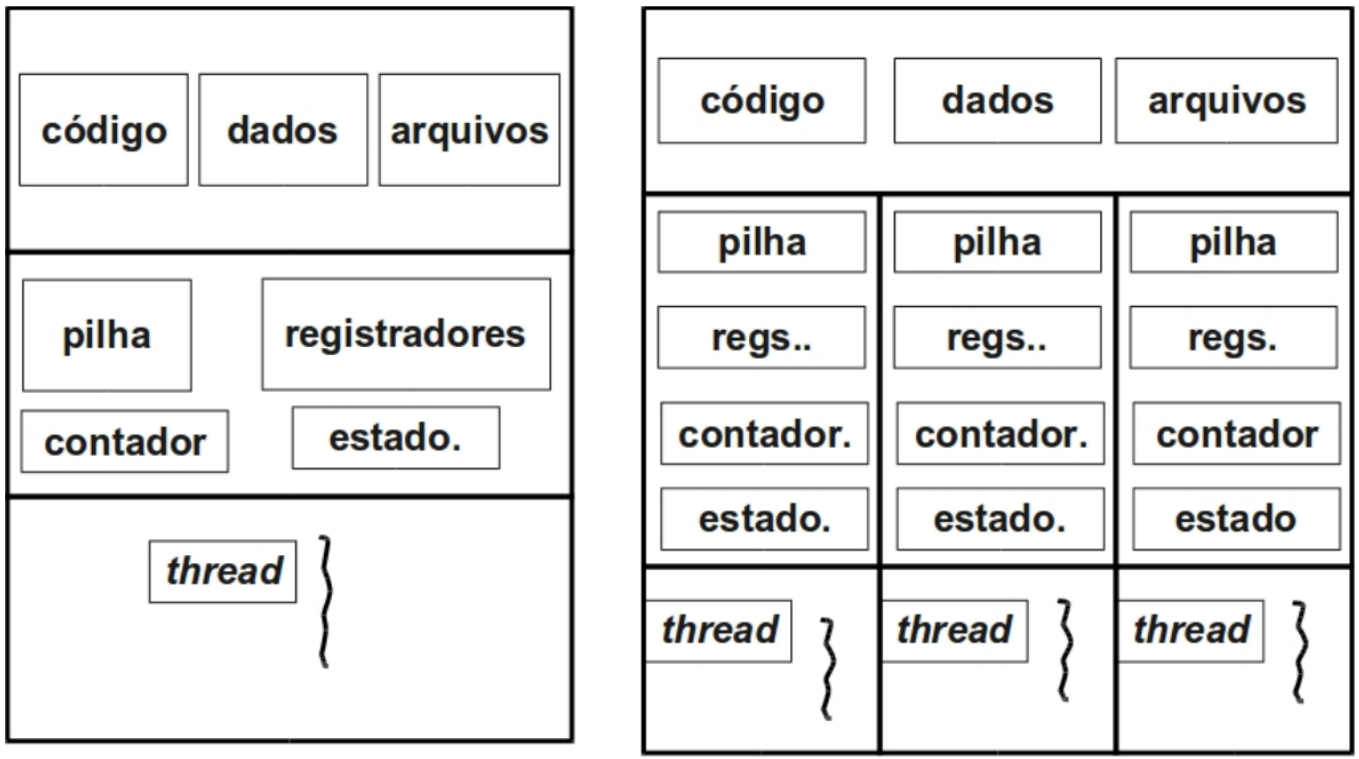
THREADS

THREAD

Uma *thread* é uma linha de execução de um processo

Um processo, inicialmente, tem uma única *thread*

- Entretanto, ele pode criar diversas outras *threads*
- Úteis para lidar com entrada e saída e com processamento paralelo



THREADS

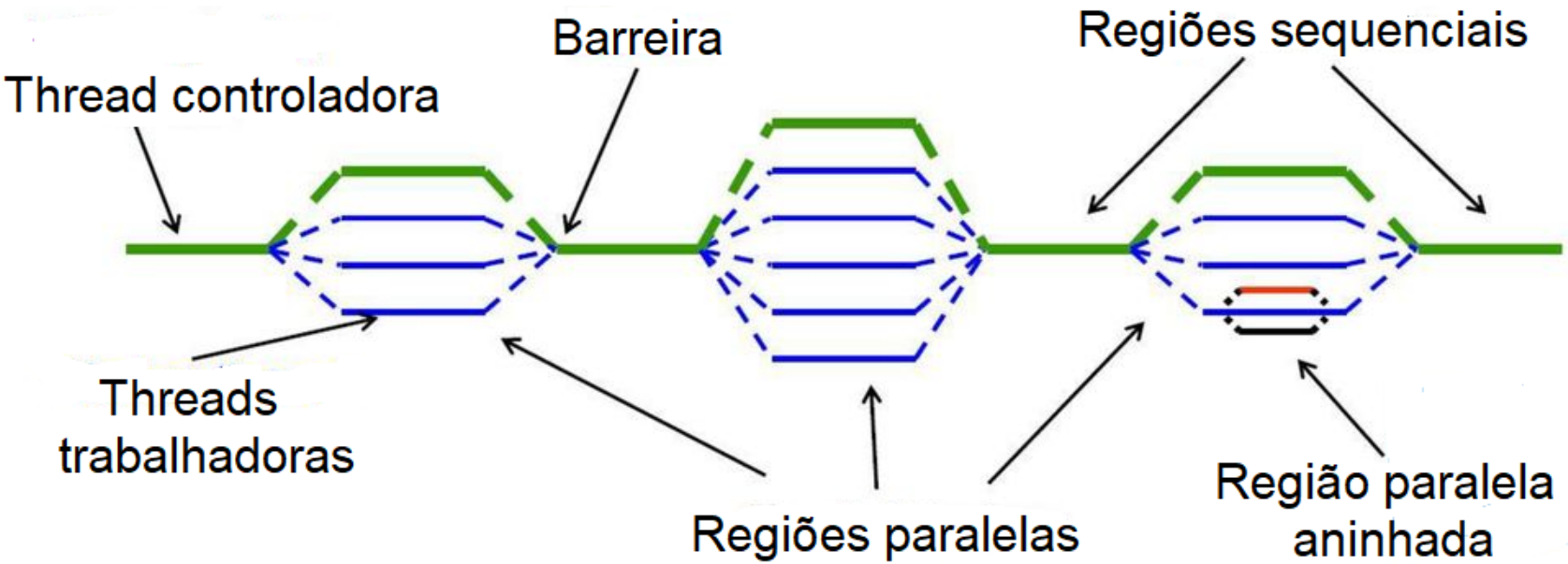
Uma *thread* é extremamente útil em computação

- *Threads* criadas por um mesmo processo compartilham seu código, seus dados e arquivos
- Isto implica que o espaço de endereçamento do processo é compartilhado por todas suas *threads*
- Facilita a comunicação entre as diversas *threads*
- A criação de *threads* é o passo básico para obtermos programação paralela e concorrente

Entretanto, existem alguns perigos inerentes ao uso de *threads*

- O principal deles é a segurança
- Como o espaço de endereçamento é compartilhado, dados podem ser alterados ou roubados
- Na maioria das vezes este risco é mitigado pelo próprio processo pai

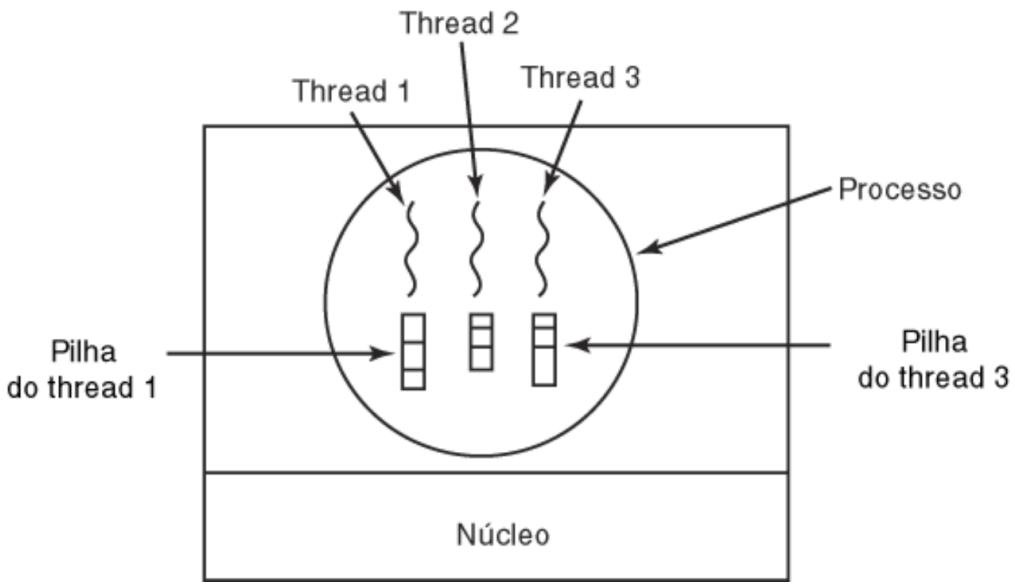
COMPUTAÇÃO PARALELA



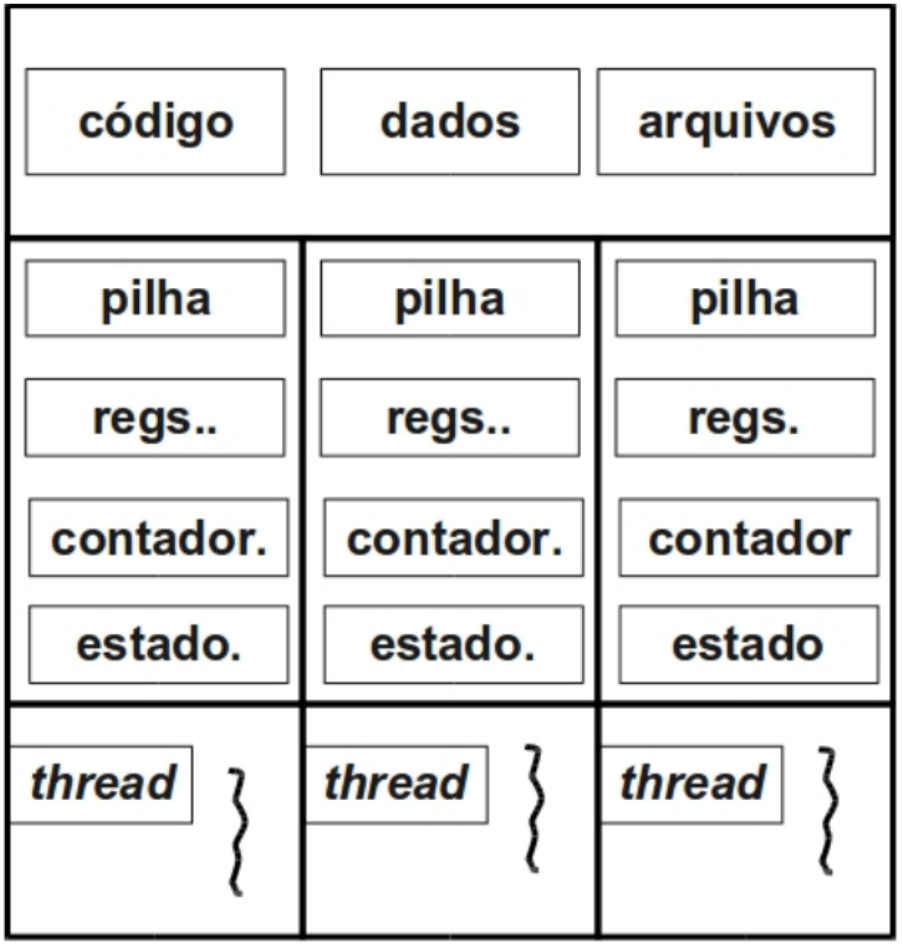
THREADS

Threads são chamadas de processos leves

- É mais barato para o sistema operacional do que um processo
- Tempo de criação e finalização são menores
- Possuem um menor número de estruturas

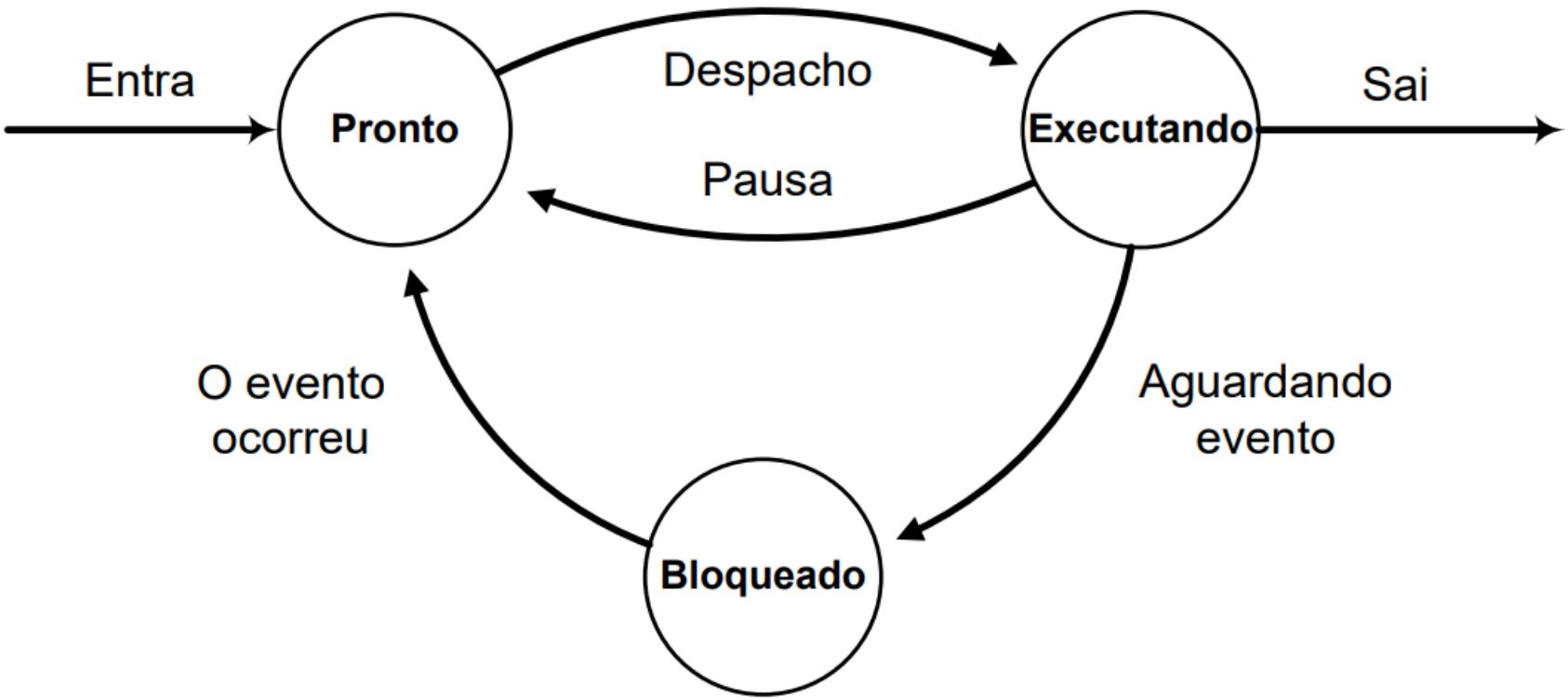


PROCESSOS LEVES



ESTADOS DE THREADS

Threads assumem os mesmos estados que processos normais

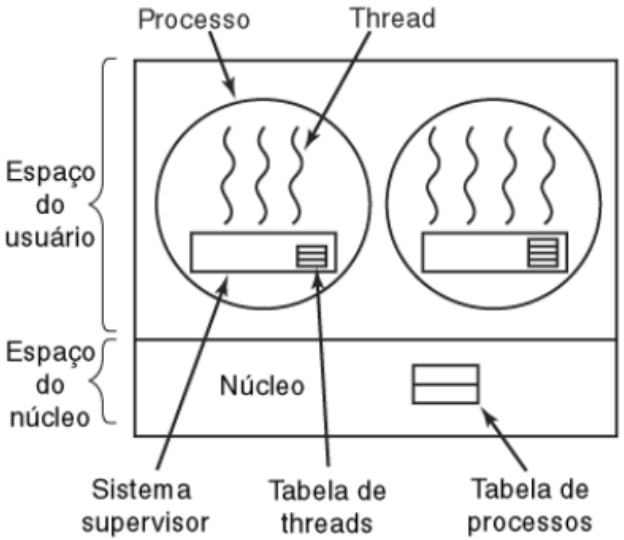


NÍVEIS DE THREAD

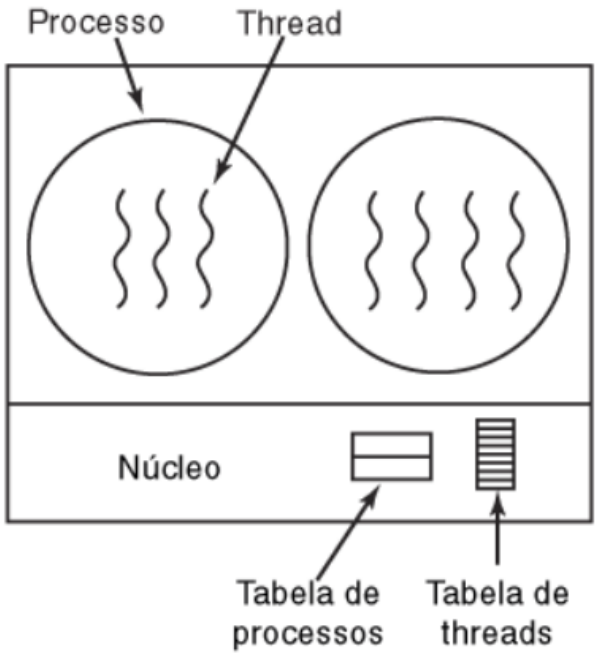
Existem dois níveis (ou tipos) de *threads*

1. De usuário
2. De núcleo (*kernel*)

Implementação a **nível de usuário**



Implementação a **nível de kernel**



THREADS DE USUÁRIO

Nesta estrutura, cada processo toma conta de suas próprias *threads*

O sistema operacional não tem conhecimento das diversas *threads* e trata tudo como um único processo

- Isso implica em um tempo de processamento compartilhado entre todas as *threads*
- Necessário desenvolver um escalonador próprio de execução de *threads*

As *threads* de usuário são mais rápidas que as de núcleo, pois a troca de contexto é mais eficiente

THREADS DE NÚCLEO

O sistema operacional conhece todas as *threads*

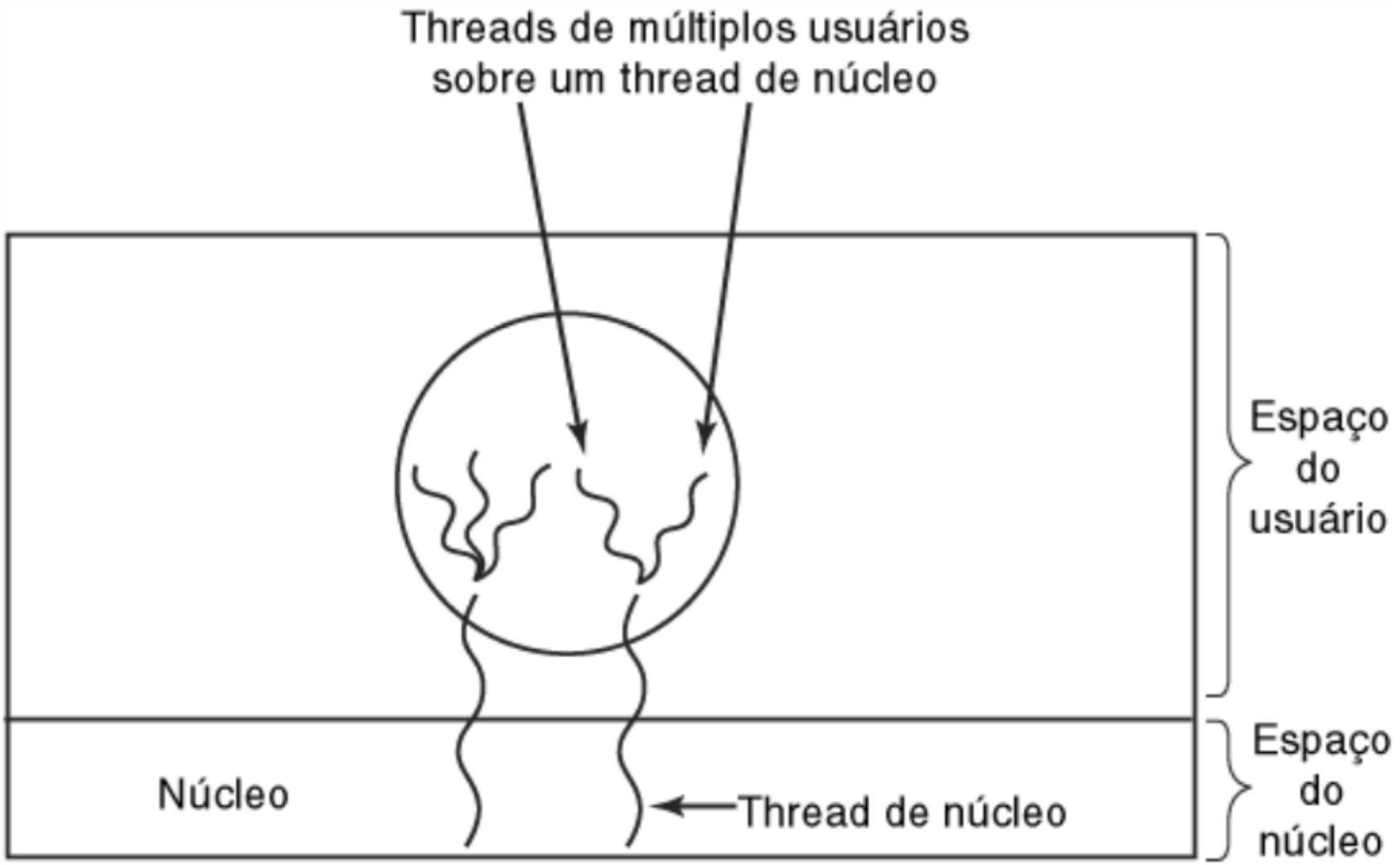
- Possui uma tabela de *threads* separado da tabela de processos

O escalonador trata as *threads* separadamente

- Cada uma tem seu próprio tempo de execução

Mais custosas, pois a troca de contexto é mais demorada

MODELO HÍBRIDO



PORQUE USAR THREADS

Processamento paralelo

- Realizar, de forma paralela, tarefas independentes
- Exemplo: fazer o *download* de arquivos de um site

Eficiência

- Criação de *threads* é mais rápida que a de processos

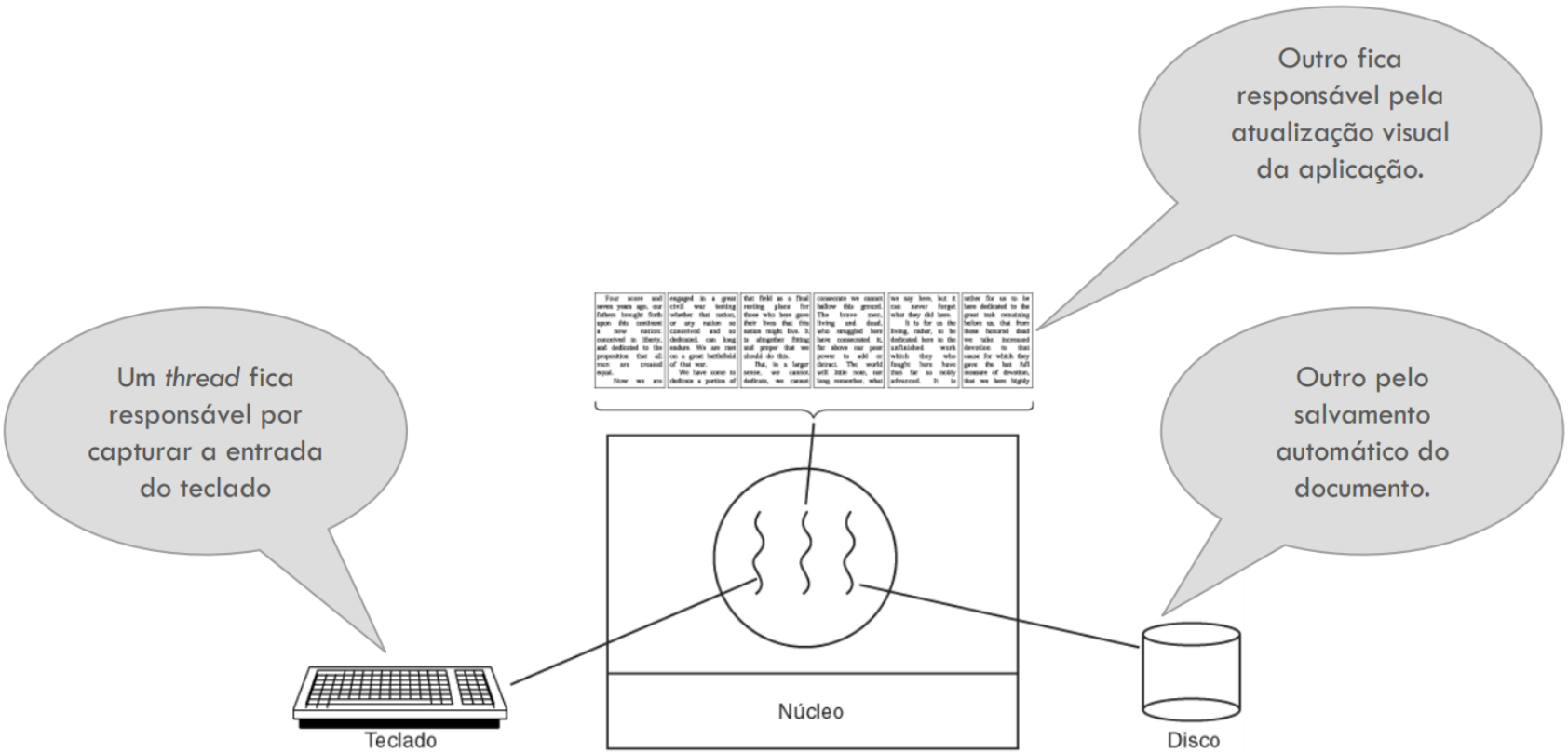
Decomposição em tarefas

- Cada tarefa é realizada por uma *thread* separada
- Encapsulamento de tarefas

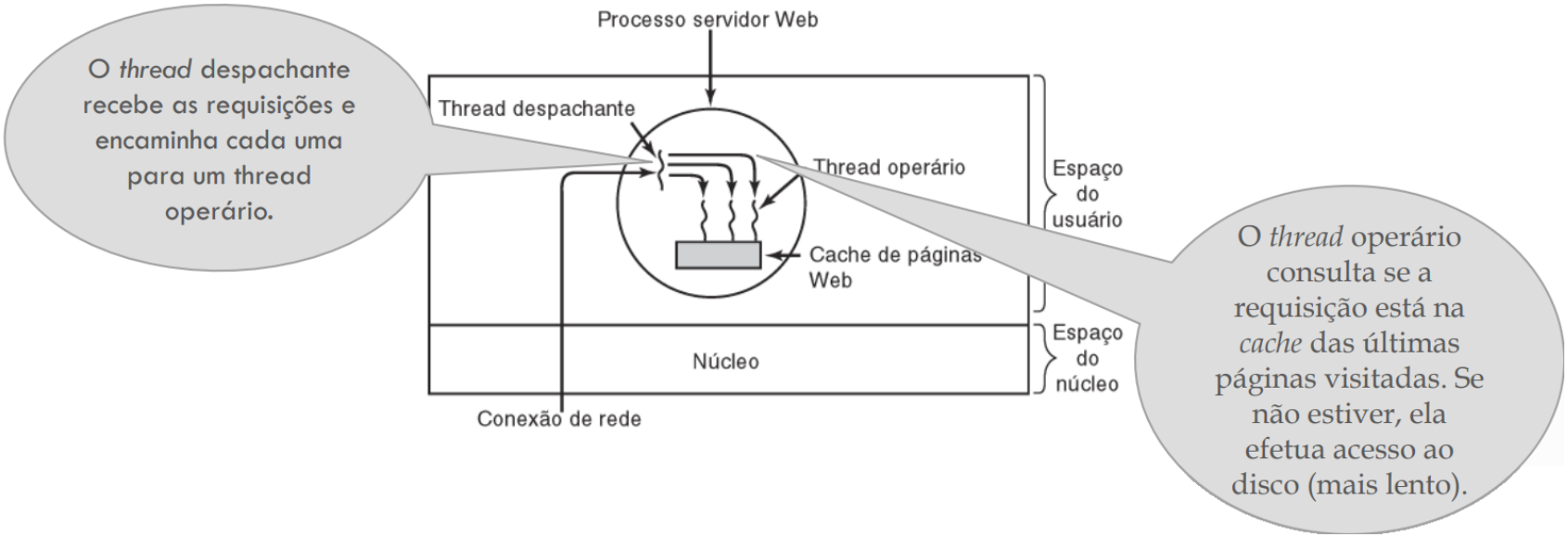
Lidar com bloqueio

- Uma *thread* é criada para lidar com entrada e saída
- *Thread* fica bloqueada
- Processo continua sua execução normal

USOS DE THREADS - EDITOR DE TEXTO



USOS DE THREADS - SERVIDOR WEB



THREADS POSIX - PTHREADS

Padronização IEEE 1003.1c-1995

Modelo de interface para utilização de *threads*

- Não é uma implementação de *threads*
- Fornece a descrição de como *threads* devem ser implementadas em diferentes sistemas

Cada sistema operacional e linguagem de programação tem sua própria implementação de pthreads

PTHREADS

Existem mais de 100 métodos descritos pela padronização pthreads

Chamada de thread	Descrição
Pthread_create	Cria um novo thread
Pthread_exit	Conclui a chamada de thread
Pthread_join	Espera que um thread específico seja abandonado
Pthread_yield	Libera a CPU para que outro thread seja executado
Pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
Pthread_attr_destroy	Remove uma estrutura de atributos do thread