# QXD0013 - Sistemas Operacionais

## Threads

Thiago Werlley Bandeira da Silva[1]

[1]Universidade Federal do Ceará, Brazil

27/10/2021

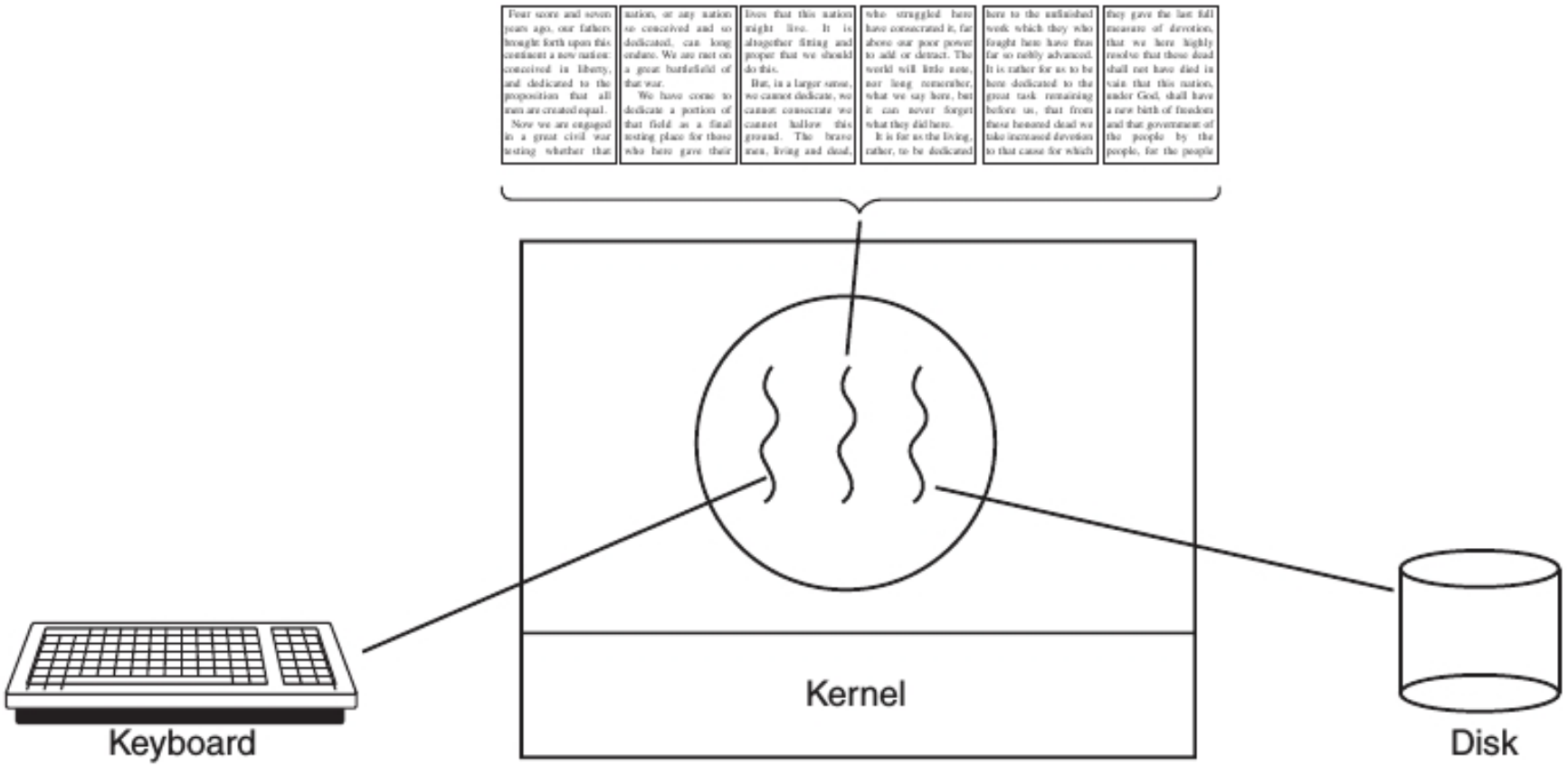
# Introdução

- SOs tradicionais: única thread de controle
- SOs modernos: múltiplas threads
- Também denominadas miniprocessos
- Por que?
  - Múltiplas atividades dentro de uma mesma aplicação
  - Simplificação do modelo de programação
  - Sem chaveamento de contexto
- Atividades paralelas compartilhando endereçamento
- Criação/Destruição mais simples (e rápida)
- Desempenho
  - CPU bound $\rightarrow$ não há ganho
  - CPU + E/S equivalentes $\rightarrow$ aceleração da aplicação

# Exemplo: Editor de texto

- Arquivo de 800 páginas
- Ações do usuário
  - Remoção de conteúdo na página 1
  - Em seguida, visualizar página 600
- Ações requeridas para a aplicação
  - Reformatar todo documento até a página 600
  - Apresentar página 600 após formatação
- Atraso na apresentação da página 600
- Modelo com duas threads:
  - Interação com o usuário
  - Formatação em segundo plano
- Possível terceira thread para salvar o arquivo

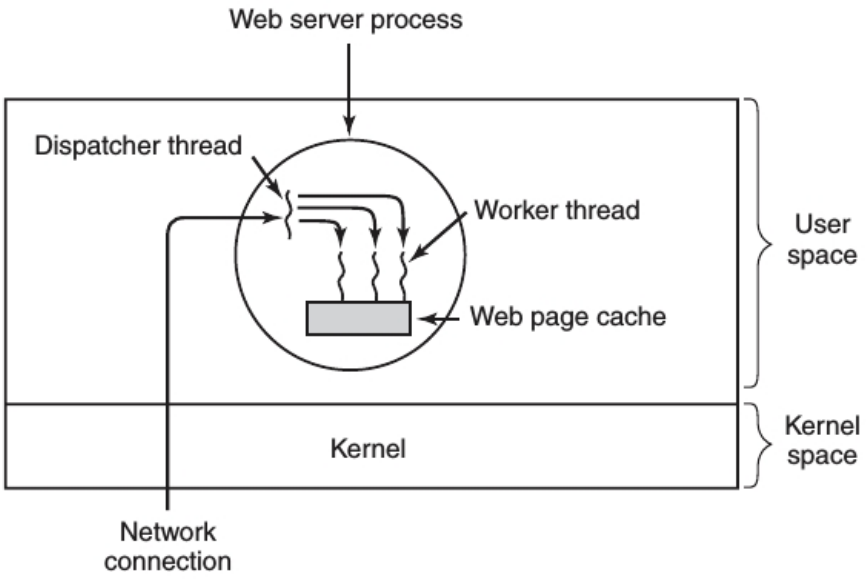
# Exemplo: Editor de texto



# Exemplo: Servidor Web

- Despachante
  - Laço infinito recebendo requisições
  - Ativa operário
- Operário
  - Laço infinito recebendo requisições
  - Se página no chache, entrega ao despachante
  - Caso contrário, acessa disco

# Exemplo: Servidor Web



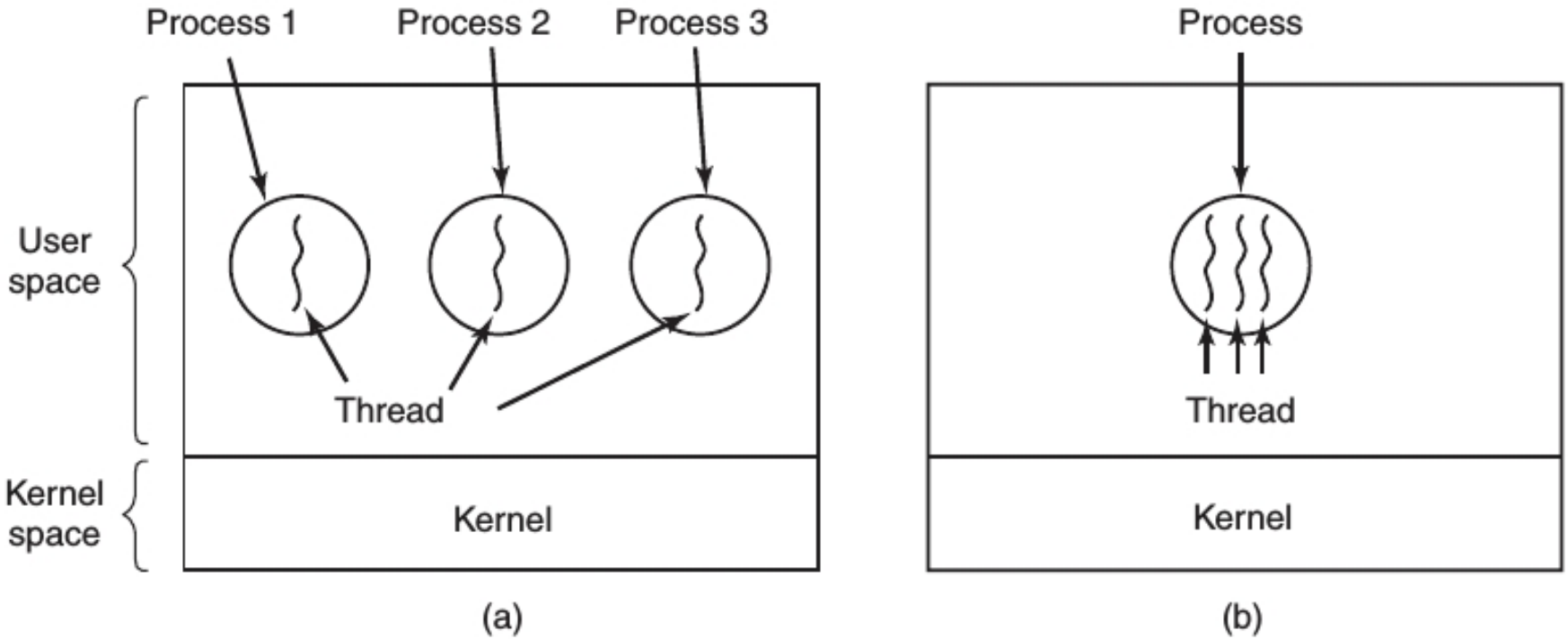

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

```
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);
}
```

# Modelo de Thread Clássico

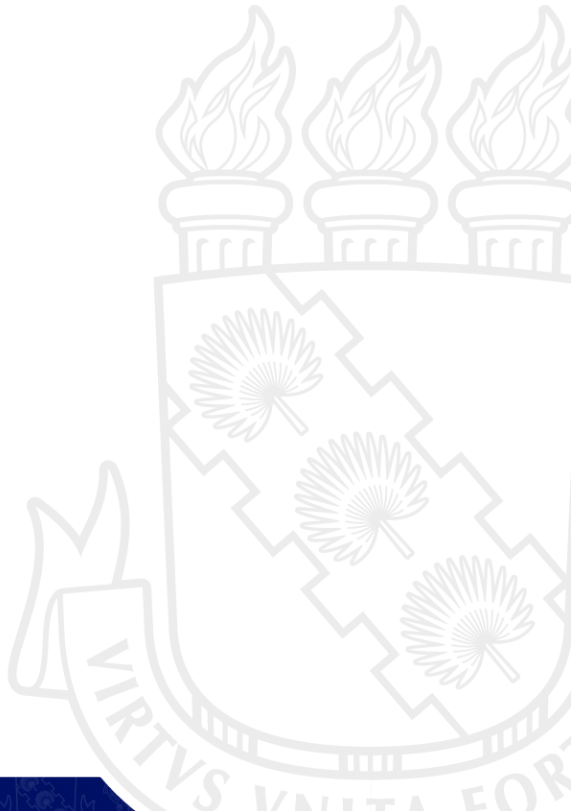
- Thread dependente do processo
- Conceitos distintos
  - Processo: agrupamento de recursos
  - Thread: contador de programa
- Múltiplas execuções dentro do processo
- Threads em um processo $\leftrightarrow$ Processos em um computador
- Threads compartilham endereçamento $\leftrightarrow$ Processos compartilham memória
- Também chamadas de processos leves (*lightweight process*)
- CPUs *multithread* $\rightarrow$ Suporte para chaveamento de *multithreads*

# Modelo de Thread Clássico



# Modelo de Thread Clássico

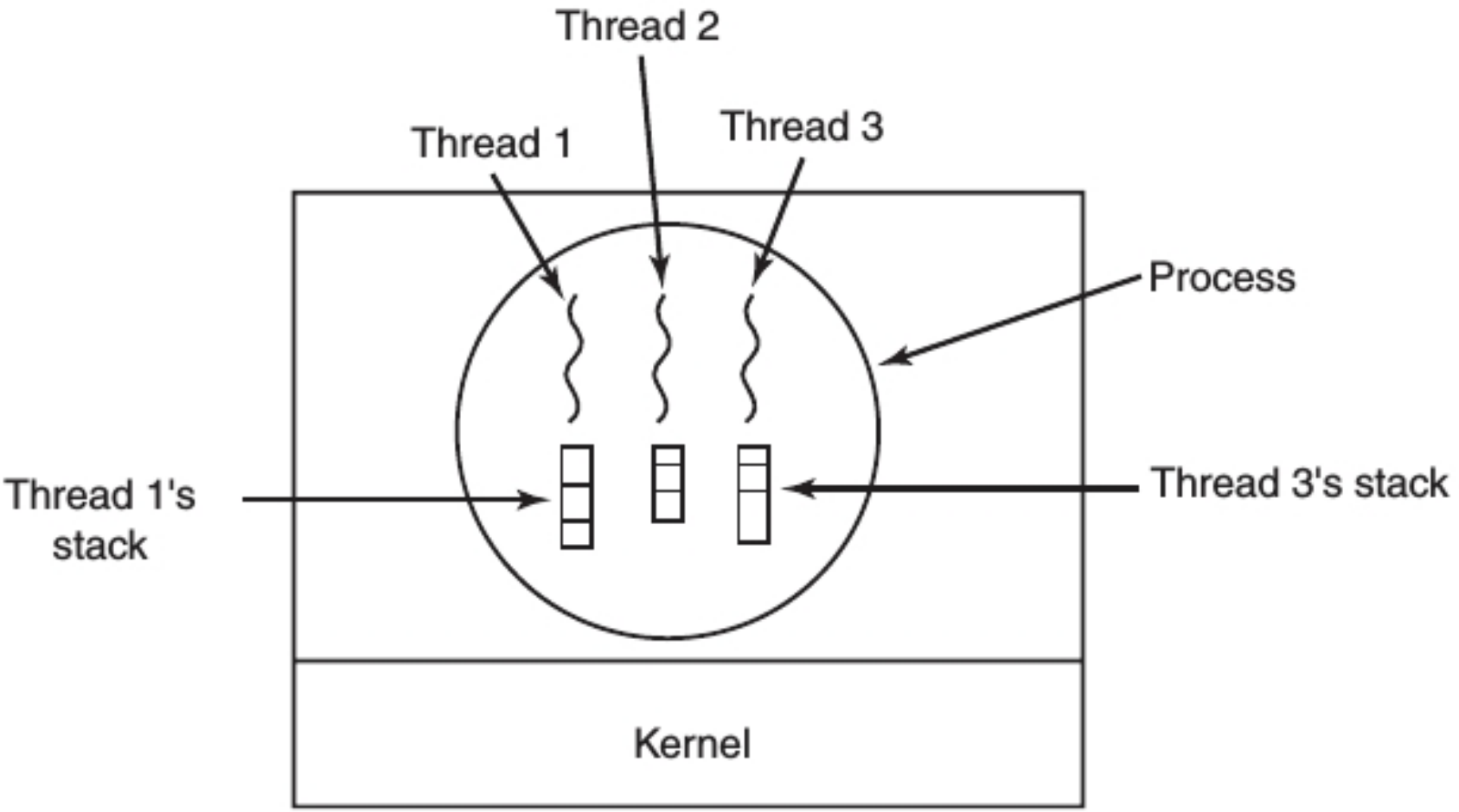
- Dependência entre Threads
  - Não há proteção
  - Cooperação
- Mesma máquina de estados dos processos
- Criação e destruição similares aos processos
- Não existe hierarquia entre threads
- Problemas:
  - Heranças entre processos
  - Compartilhamento de dados
- Cuidados específicos no projeto



# Modelo de Thread Clássico

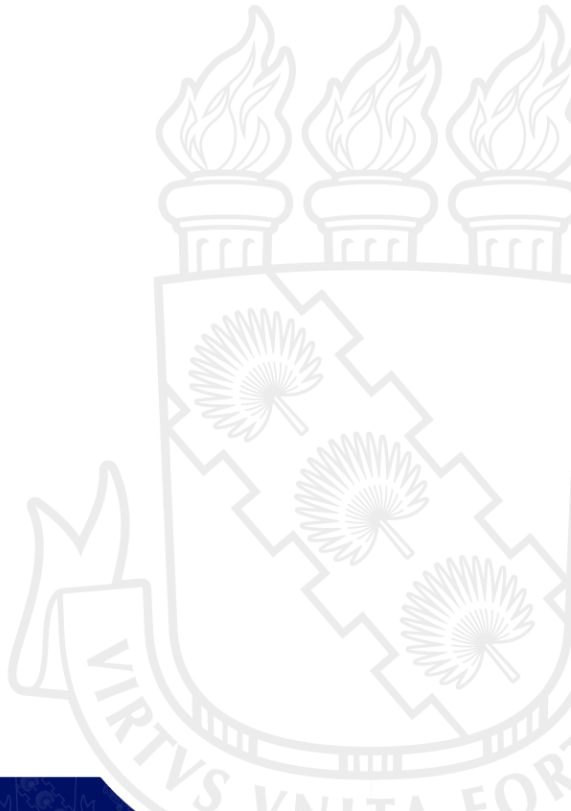
| Per-process items | Per-thread items |
|---|---|
| Address space | Program counter |
| Global variables | Registers |
| Open files | Stack |
| Child processes | State |
| Pending alarms | |
| Signals and signal handlers | |
| Accounting information | |

# Modelo de Thread Clássico



# Threads POSIX

- Padrão IEEE 1003.1c
- Portabilidade
- Pacote *Pthreads* (mais de 60 funções)
- Propriedades de uma thread *Pthreads*:
  - Identificador
  - Registros
  - Atributos



# Threads POSIX

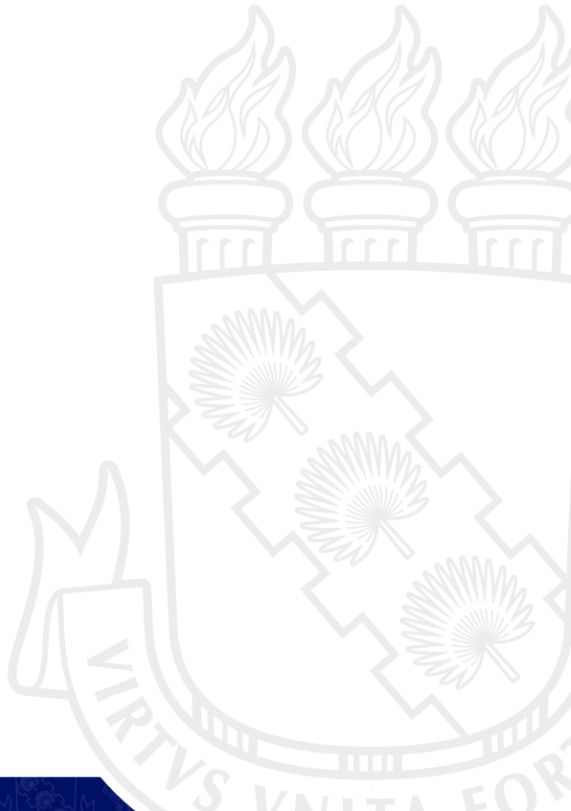
| Thread call | Description |
|---|---|
| Pthread_create | Create a new thread |
| Pthread_exit | Terminate the calling thread |
| Pthread_join | Wait for a specific thread to exit |
| Pthread_yield | Release the CPU to let another thread run |
| Pthread_attr_init | Create and initialize a thread's attribute structure |
| Pthread_attr_destroy | Remove a thread's attribute structure |

# Threads POSIX

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS    10

void *print_hello_world(void *tid)
{
      /* Esta função imprime o identificador do thread e sai. */
      printf("Hello World. Greetings from thread %d\n", tid);
      pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
      /* O programa principal cria 10 threads e sai. */
      pthread_t threads[NUMBER_OF_THREADS];
      int status, i;

      for(i=0; i < NUMBER_OF_THREADS; i++) {
            printf("Main here. Creating thread %d\n", i);
            status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

            if (status != 0) {
                  printf("Oops. pthread_create returned error code %d\n", status);
                  exit(-1);
            }
      }
      exit(NULL);
}
```

# Implementação de Threads

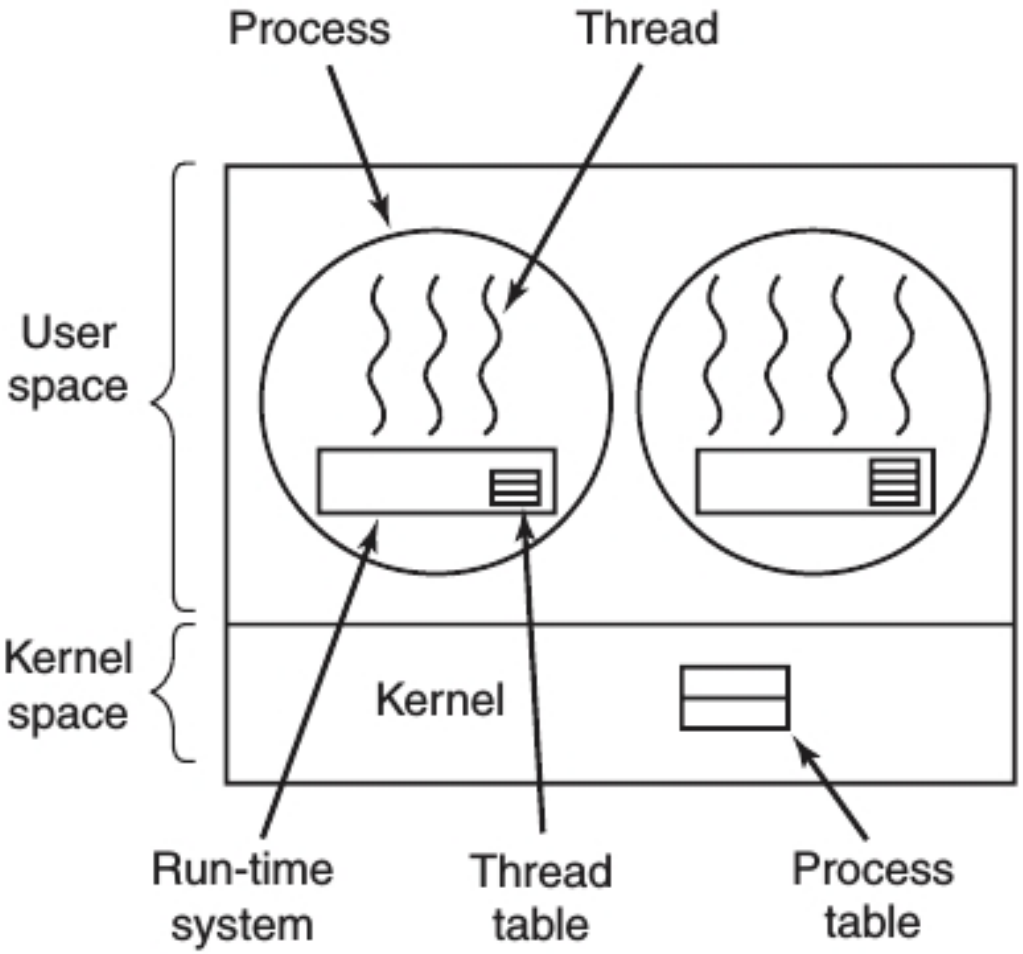
- Modo usuário
- Modo núcleo (kernel)
- Modo híbrido



# Threads em Modo Usuário

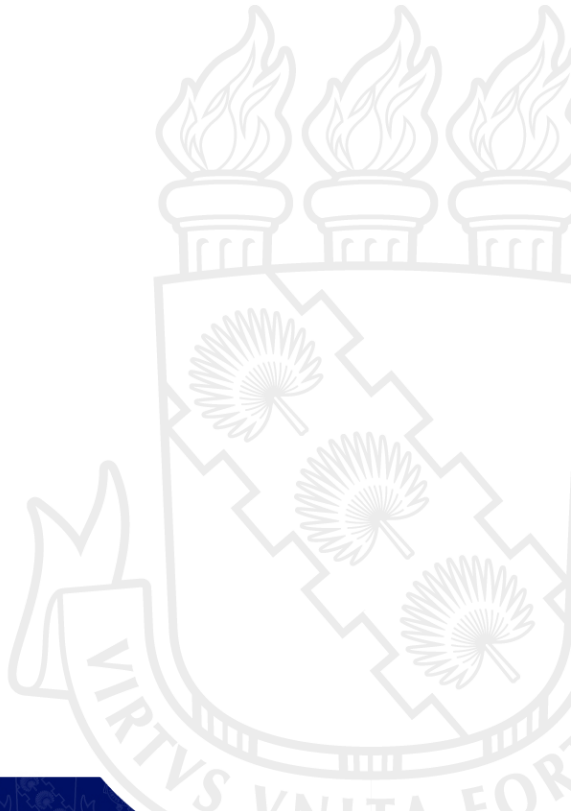
- Núcleo não é informado sobre sua existência
- SO que não suporta threads
- Implementação através de uma biblioteca
- Processo $\rightarrow$ Tabela de Threads
- Chaveamento com poucas instruções
- Escalonamento personalizado
- Problemas:
  - Sistemas bloqueantes
  - Thread deve ceder CPU voluntariamente

# Threads em Modo Usuário

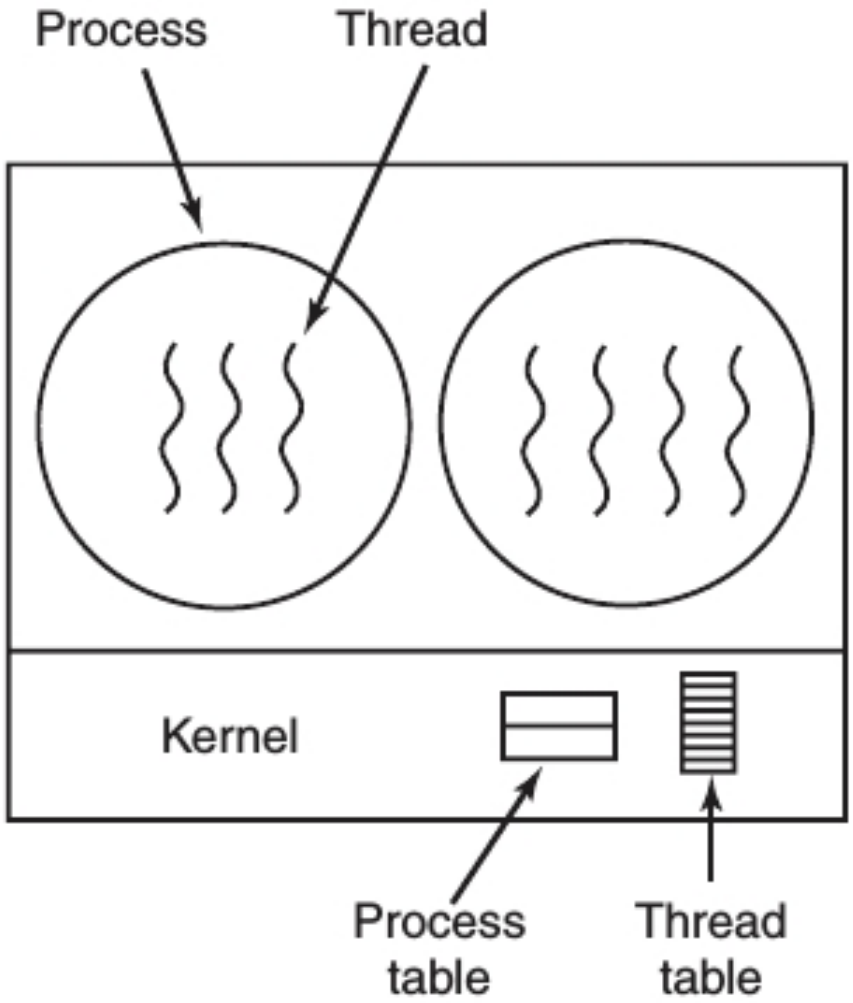


# Threads em Modo Núcleo

- Núcleo gerencia as threads
- Tabela de threads do sistema
- Chamadas ao núcleo para criação/destruição
- Thread bloqueado
  - Mesmo processo
  - Mudar processo
- Custo maior $\rightarrow$ reciclagem



# Threads em Modo Núcleo



# Threads em Modo Híbrido

- Combina duas abordagens
- Threads de núcleo $\leftrightarrow$ Threads de usuário
  - Multiplexação

# Threads em Modo Híbrido