

**SSC-0143**

**PROGRAMAÇÃO CONCORRENTE**

**Pthreads**

Prof. Jó Ueyama

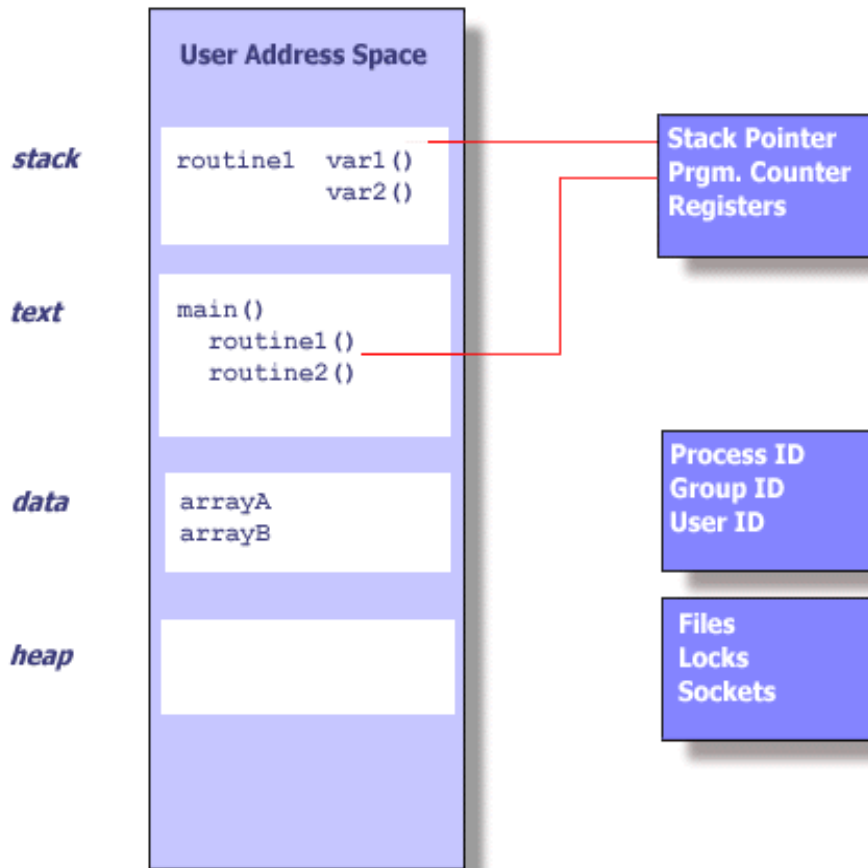
# Créditos

*Os slides integrantes deste material foram construídos a partir dos conteúdos relacionados às referências bibliográficas descritas neste documento*

# Processos

- O que é um processo?
  - Instância de um programa em execução
    - Contadores
    - Registradores
    - Variáveis globais e locais
    - Pilha de execução

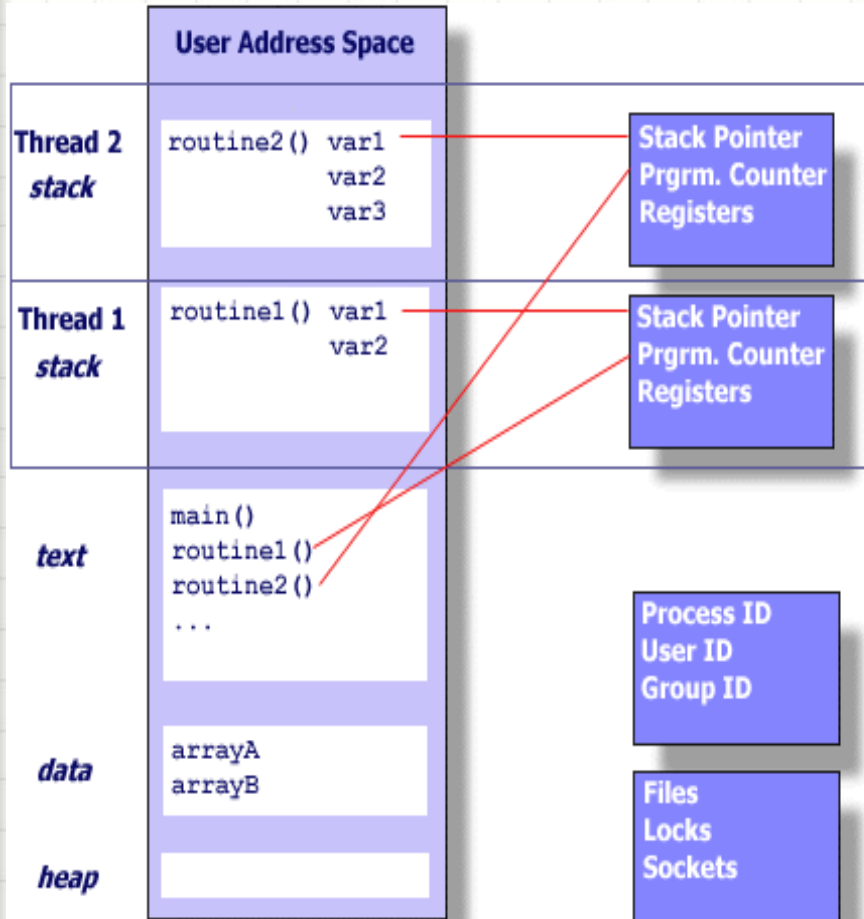
# Regiões de um processo



**Figura:** Regiões de um processo.

- Pilha (*stack*)
  - Região da memória utilizada para guardar dados dinâmicos (parâmetros de funções, variáveis locais, valores de retorno)
- Segmento de Texto (*text*)
  - Código executável (instruções) do programa
- Segmento de Dados (*data*)
  - Variáveis globais inicializadas
- Heap
  - Região utilizada para alocar memória ao processo

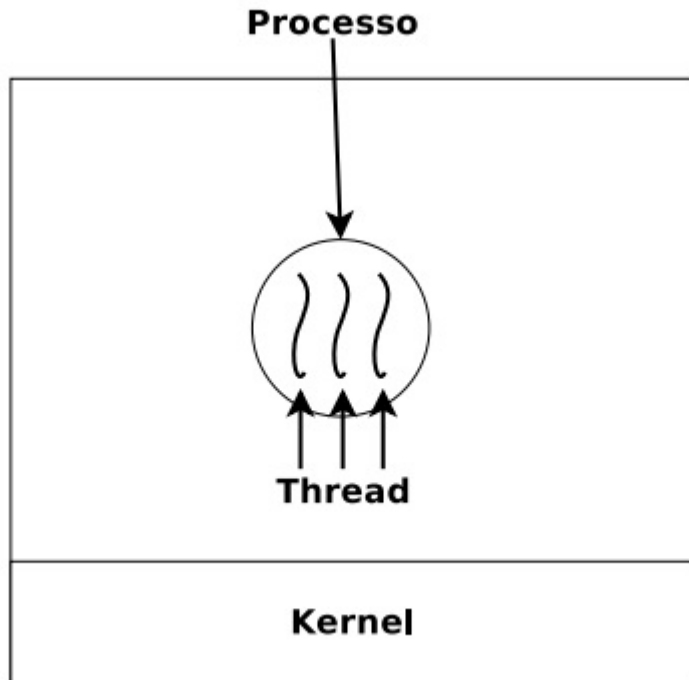
# Threads



**Figura:** Regiões de um processo com *threads*.

- Processos leves
  - Possuem o mesmo endereço na memória
  - Compartilham variáveis globais
  - Cada *thread* possui sua própria pilha de execução
    - Procedimentos e Variáveis locais
- Dois tipos de *threads*
  - Usuário e de Kernel

# Threads de usuário

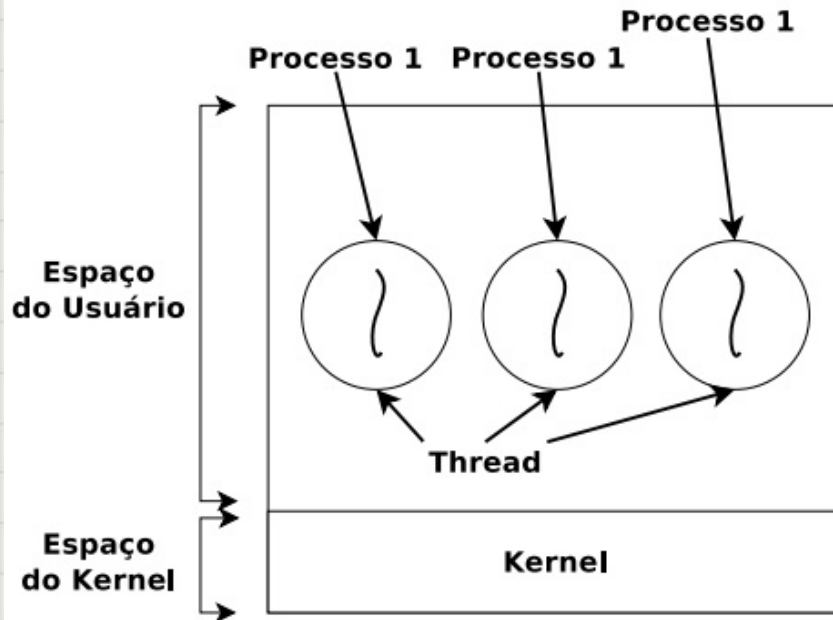


**Figura:** Thread de usuário.

- Podem ser implementadas em SO que não suportam *threads*
- Somente o processo pai é enxergado pelo kernel.
- Cada processo possui sua própria tabela de threads
- O escalonador de threads e os procedimentos que salvam seus estados são locais (mais eficientes que uma chamada ao sistema).

- Se uma thread é bloqueada por uma chamada ao sistema todas as outras threads em execução também são bloqueadas.
- Threads só podem escalonar entre si.(Não pode escalonar threads entre processos.)

# Threads de Kernel



**Figura:** *Thread* de Kernel.

- Uma única tabela de threads localizada no kernel gerencia todas as threads do sistema
- Criar e destruir threads – chamadas ao sistema
- O bloqueio de uma thread não atrapalha as outras threads em execução

- O escalonador de processos pode escalonar threads entre processos diferentes
- Chamadas ao sistema são custosas – overhead.

# *Pthreads*

- ***POSIX Threads: Padrão IEEE POSIX 1003.1c – 1995***
- Define uma API para a criação e manipulação de *threads*
- Alternativa para as soluções proprietárias de *threads* desenvolvidas pelos fabricantes de hardware



# Pthreads

Estrutura e funções básicas:

- pthread\_t (struct)
- pthread\_create
- pthread\_join
- pthread\_exit

# Pthreads

- **pthread\_create**: Cria uma nova thread e a torna executável

```
int pthread_create(pthread_t *thread, const
pthread_attr_t *attr, void *
(*start_routine) (void *), void *arg);
```

- **thread**: Estrutura para thread criada;
  - **attr**: Atributos para nova thread ou NULL para valores padrão;
  - **start\_routine**: rotina que será executada quando a thread for iniciada
  - **arg**: argumento a ser passado para a rotina de início
- Retorna 0 se houve sucesso, ou o código do erro, caso contrário**

# Pthreads

- **pthread\_exit:** Finaliza a thread;  
void pthread\_exit(void \*retval);
- **retval:** Código de retorno da Thread.

# Pthreads

## Exemplo 1

# Pthreads

- **pthread\_create:** permite apenas que um argumento seja passado;
- Para casos em que vários argumentos tenham que ser passados utiliza-se structs;
- Todos argumentos desta função devem ser passados por referência e um cast para (void \*) deve ser feito

## Exemplo 2

# Pthreads

- **pthread\_join:** Aguarda o término de alguma Thread  
`int pthread_join(pthread_t thread, void **retval);`
- Esta função aguarda pelo término da Thread especificada no argumento `thread`. Se a mesma já terminou sua execução quando a função foi chamada, então a função retorna imediatamente.
- **retval:** Se não for NULL, este argumento conterá o código de retorno da Thread que estava sendo aguardada.
- Retorna 0 se houve sucesso, ou o código do erro, caso contrário

# Pthreads

## Exemplo 3

# Mutexes

- Mecanismos de exclusão mútua entre threads
- Garantem que regiões críticas do código não sejam executadas simultaneamente
- protegem estruturas de dados compartilhadas de modificações simultâneas
- Dois estados: lock e unlock



# Pthreads

Estrutura e funções básicas:

- `pthread_mutex_t`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`

# Pthreads

- **pthread\_mutex\_lock**: muda o estado do mutex para *locked*(bloqueado).

```
pthread_mutex_lock(pthread_mutex_t *mutex);
```

- **mutex**: Estrutura para o mutex criado

# Pthreads

- **pthread\_mutex\_unlock:** muda o estado do mutex para *unlocked* (desbloqueado).

```
pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- **mutex:** Estrutura para o mutex criado

# Exemplo 4

```
void* doSomething(void *arg)
{
    pthread_mutex_lock(&lock);

    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d started\n", counter);

    for(i=0; i<(0xFFFFFFFF);i++);

    printf("\n Job %d finished\n", counter);

    pthread_mutex_unlock(&lock);

    return NULL;
}
```

# Dúvidas



# Bibliografia

Mutex, disponível em:

[http://www.lsd.ic.unicamp.br/mo806wiki/index.php/Estudo\\_Mutex](http://www.lsd.ic.unicamp.br/mo806wiki/index.php/Estudo_Mutex)

Pthreads, disponível em:

<https://computing.llnl.gov/tutorials/pthreads/#Pthread>

TANENBAUM, Andrew. Sistemas operacionais modernos. Rio de Janeiro: LTC. 1999.

Material de Aula, Renê Souza Pinto