Threads

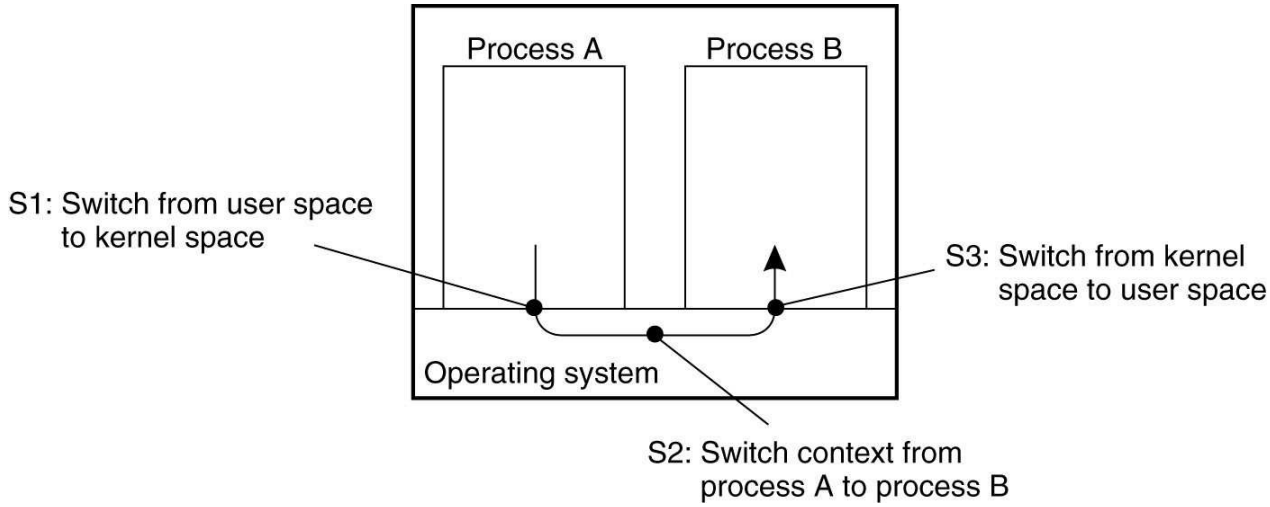
Fontes:

Silberschatz cap 4
Tanenbaum cap 2

- Introdução
- Processos e *Threads*
- Níveis de implementação
- Modelos de *Multithreads*
- Exemplos de uso

Multiprogramação pesada

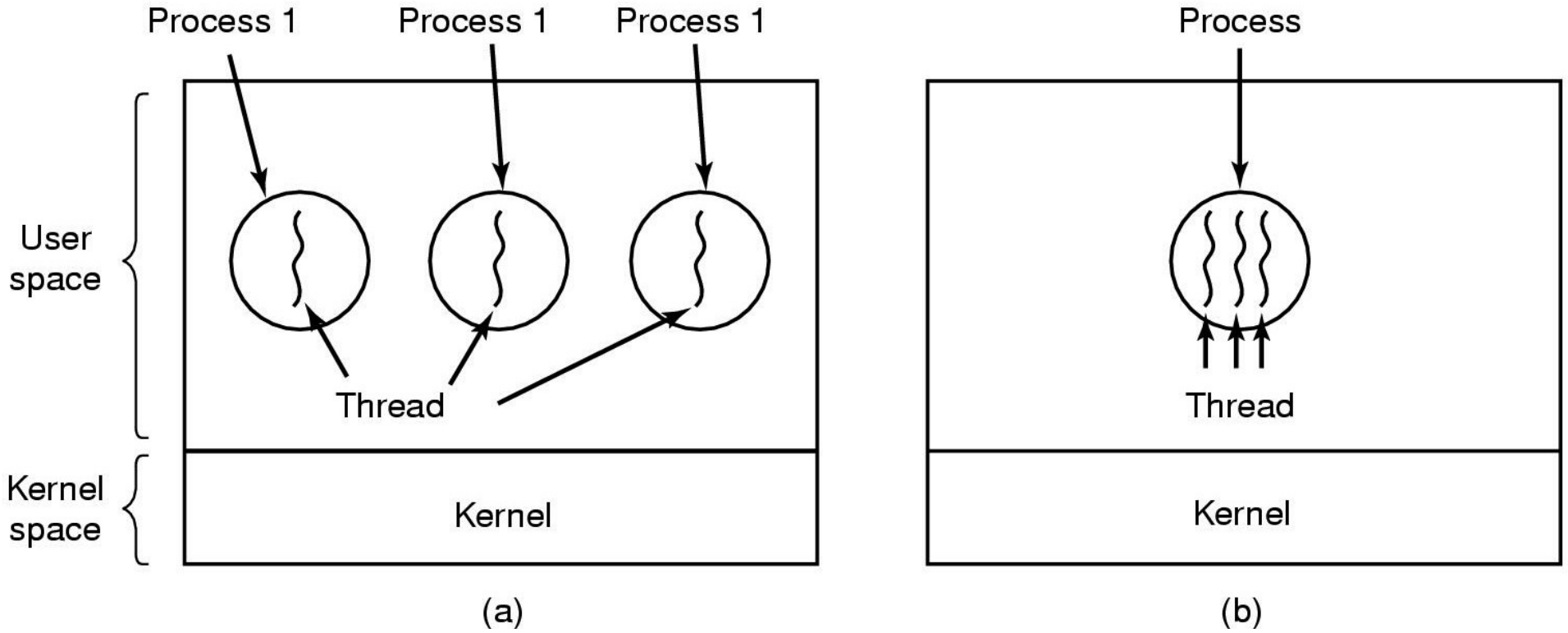
- Custo de gerenciamento de processos – fator limitante
  - Criação do processo
  - Troca de contexto
  - Esquemas de proteção, memória virtual, ...
- Solução:
  - Aliviar os custos
  - Reduzir o overhead envolvido



Multiprogramação leve - Thread

- Fornecido pela abstração de um fluxo de execução → **thread**
- mecanismo que permite a um processo ter mais de um fluxo de controle
- *threads*: compartilham o espaço de endereçamento ("processo leve")
- estados fundamentais: executando, pronta, bloqueada
- unidade de interação passa a ser a função
- contexto: pilha, PC, registradores de uso geral
- comunicação via compartilhamento direto da área de dados

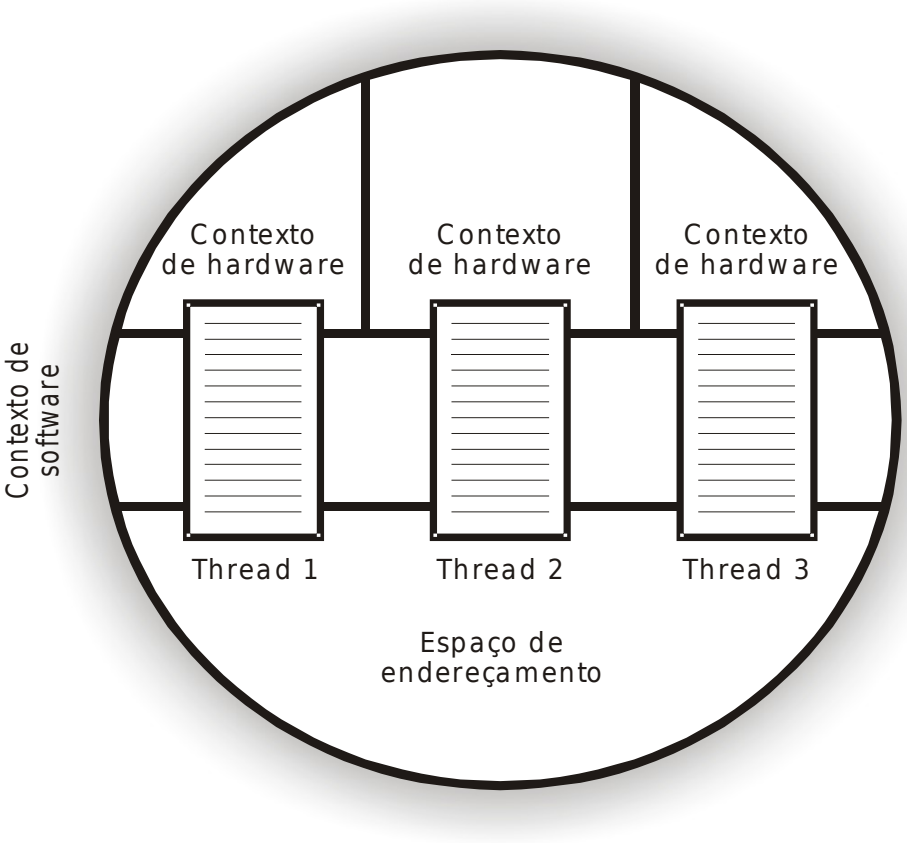
Processos x *Threads* (1)

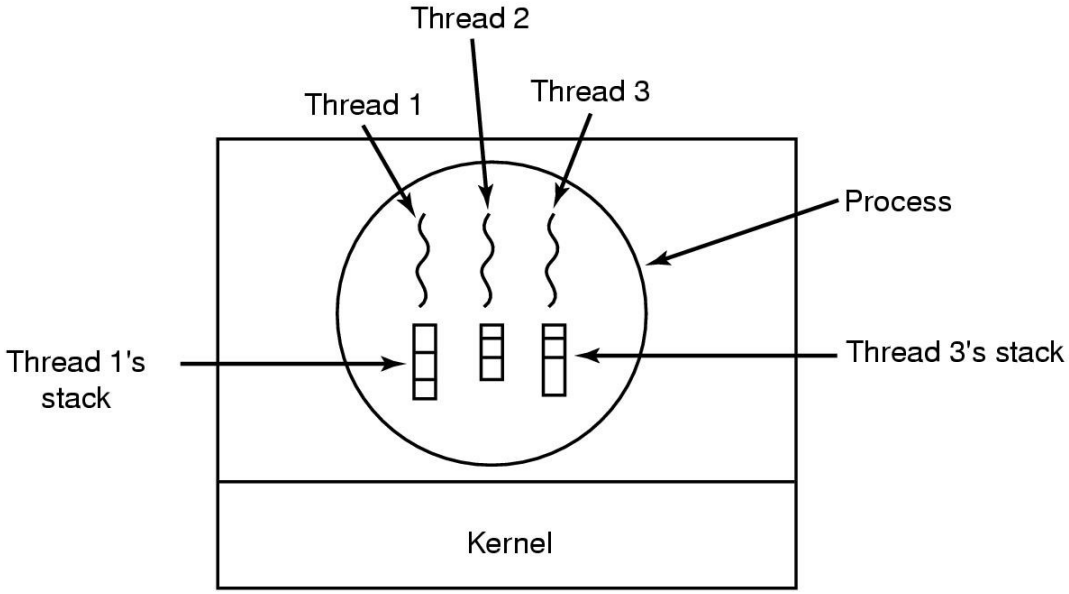


(a) 3 processos, cada um com uma thread
(b) 1 processo com 3 Threads

Processo = fluxo de controle + espaço de endereçamento
Thread = fluxo de controle

Processos x *Threads* (2)





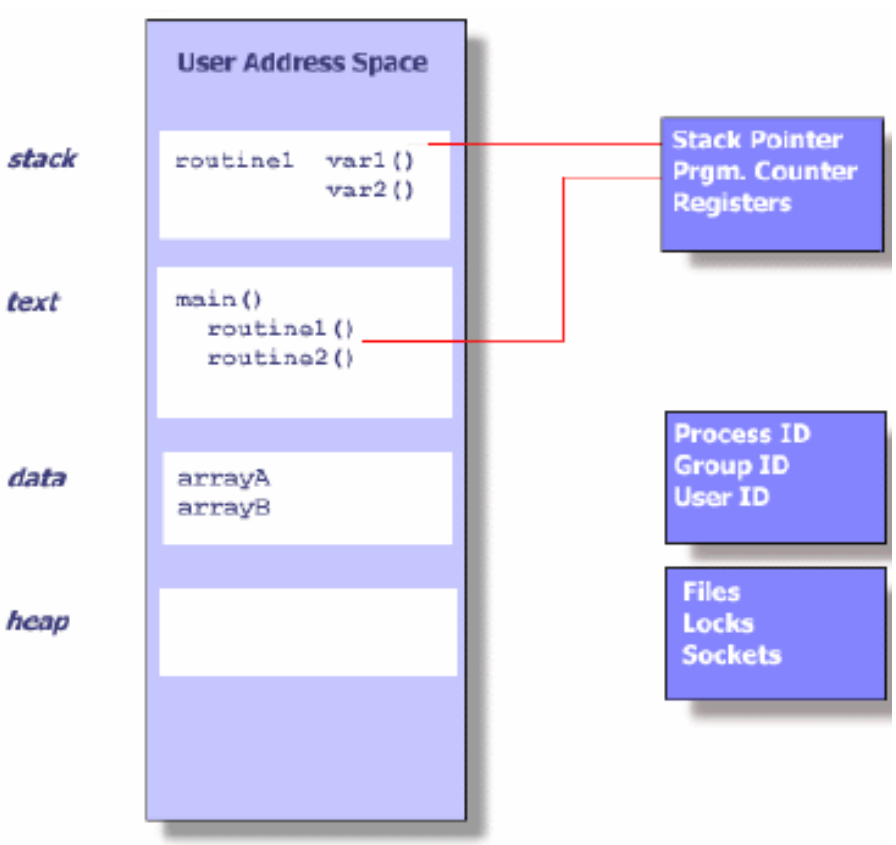
Processos x *Threads* (3)

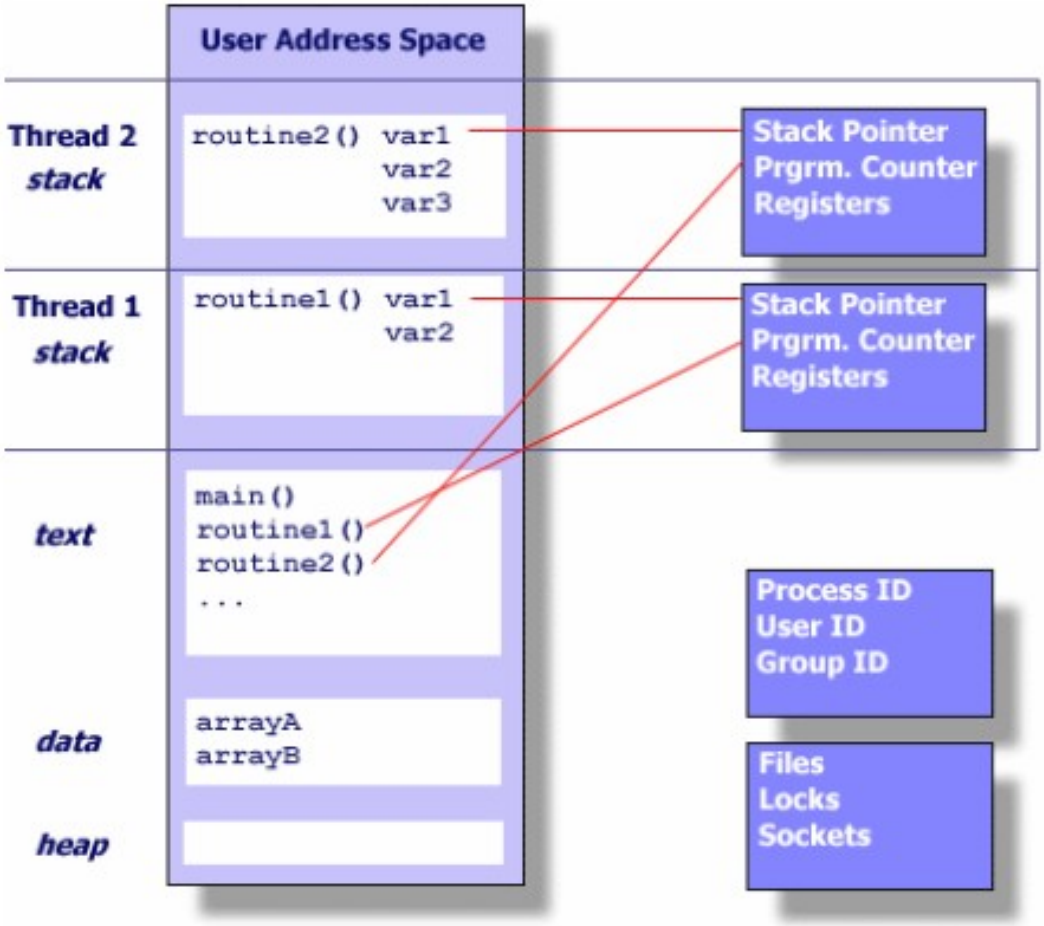
Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e manipuladores de sinais	
Informação de contabilidade	

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.

Fonte: Tanenbaum

Processos x *Threads* (4)





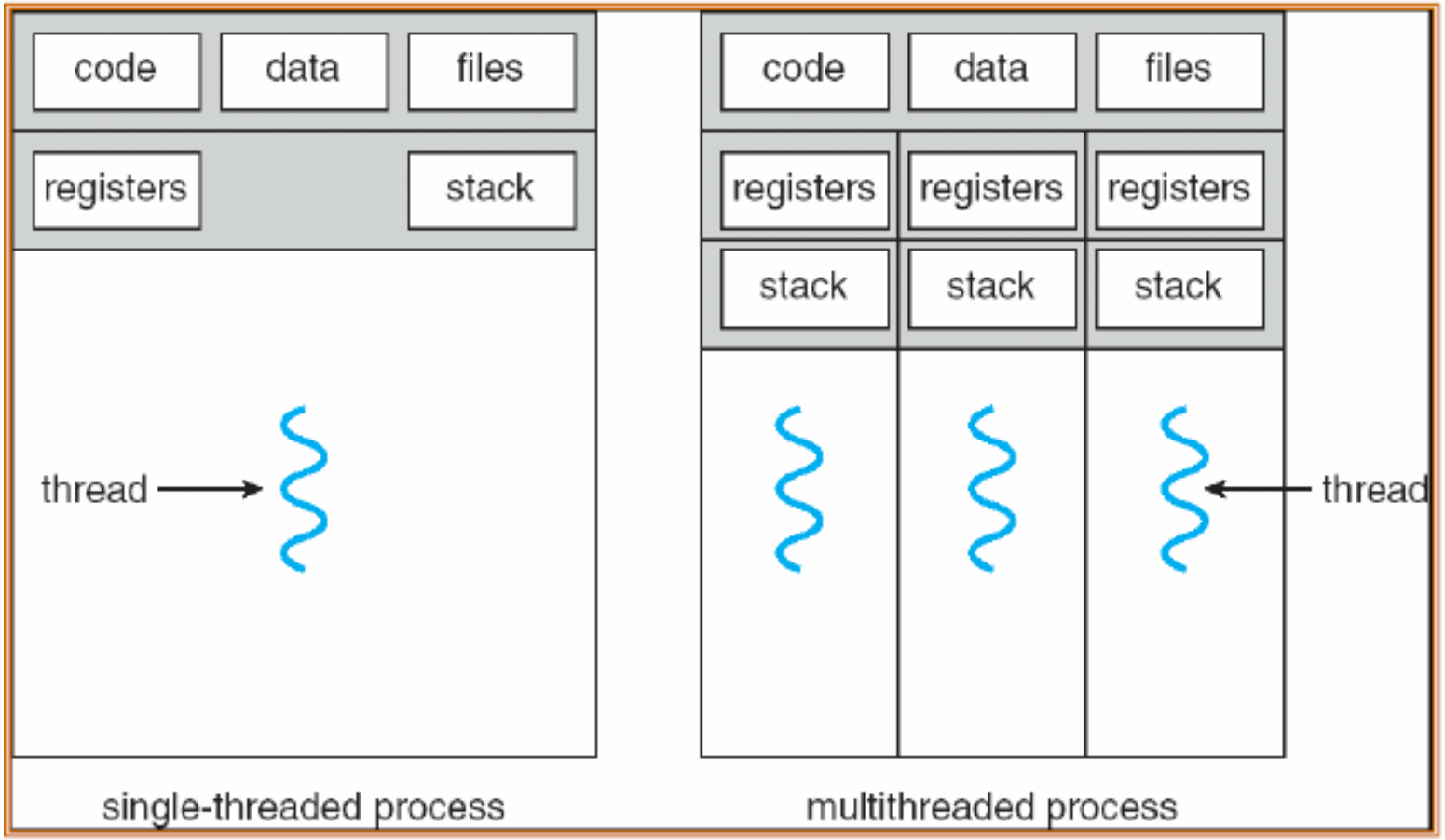
Threads (1)

- é mais rápido e barato criar/terminar um thread do que um processo
- é mais rápido chavear entre threads de um mesmo processo
- threads podem se comunicar sem invocar o kernel, já que compartilham memória e arquivos
  - não há proteção entre threads de um mesmo processo
    - ex.: uma thread pode ler, escrever em variáveis globais de outra thread do mesmo processo

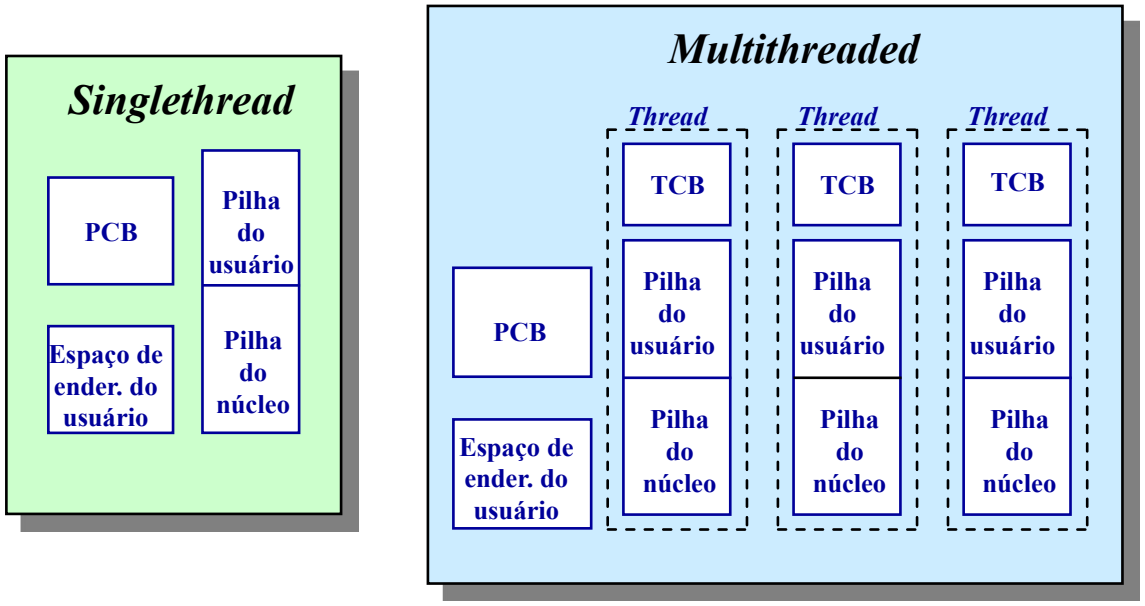
Threads (2)

- simplifica o desenvolvimento de aplicações concorrentes
- em um processo c/ múltiplas threads…
  - enquanto uma thread está bloqueada esperando, outra do mesmo processo pode rodar
    - cooperação entre threads no mesmo processo aumenta o desempenho
    - compartilhamento de recursos (espaço de endereçamento)

Single x *Multithreaded* (1)



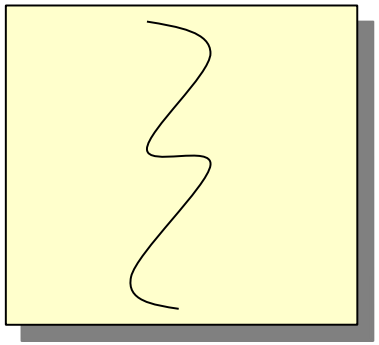
Single x *Multithreaded* (2)



Multithreading
Suporte a múltiplas threads de execução dentro de um único processo

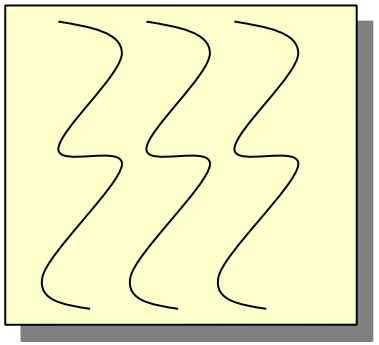
- TCB: Estrutura de dados similar ao descritor de processo (PCB)

Single x *Multithreaded* (3)



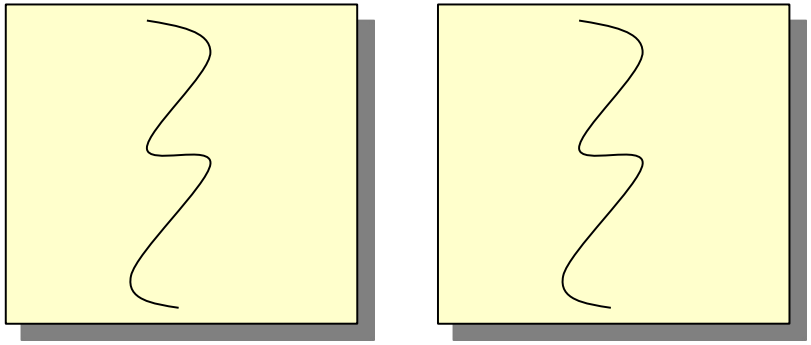
um processo
uma thread

Ex.: MSDOS



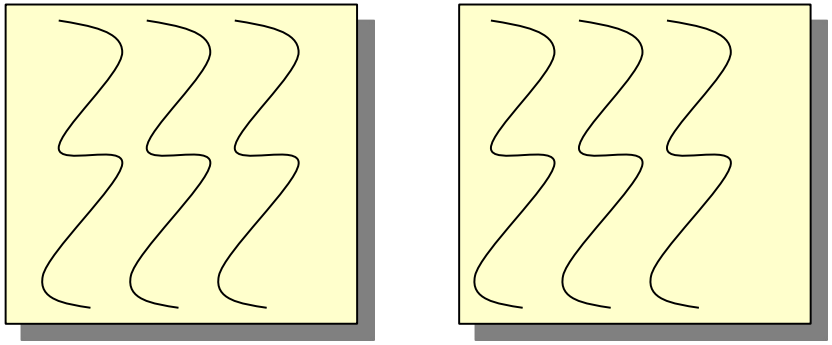
um processo
várias threads

Ex.: JVM



vários processos
uma thread por processo

Ex.: Unix (início)

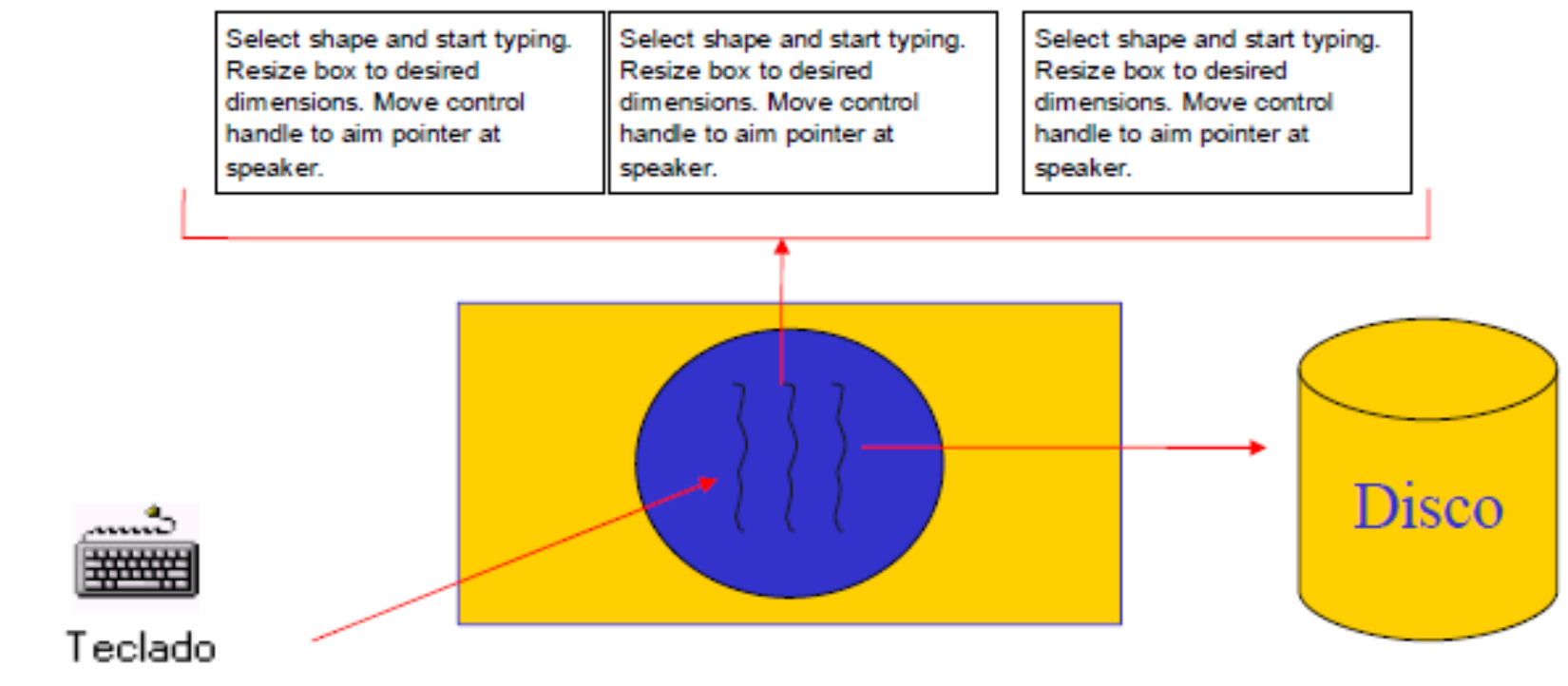


vários processos
várias threads por processo

Ex.: Linux, Win

Exemplo de uso de threads

- Editor de textos com 3 threads (threads para diferentes tarefas)



Tipos de threads

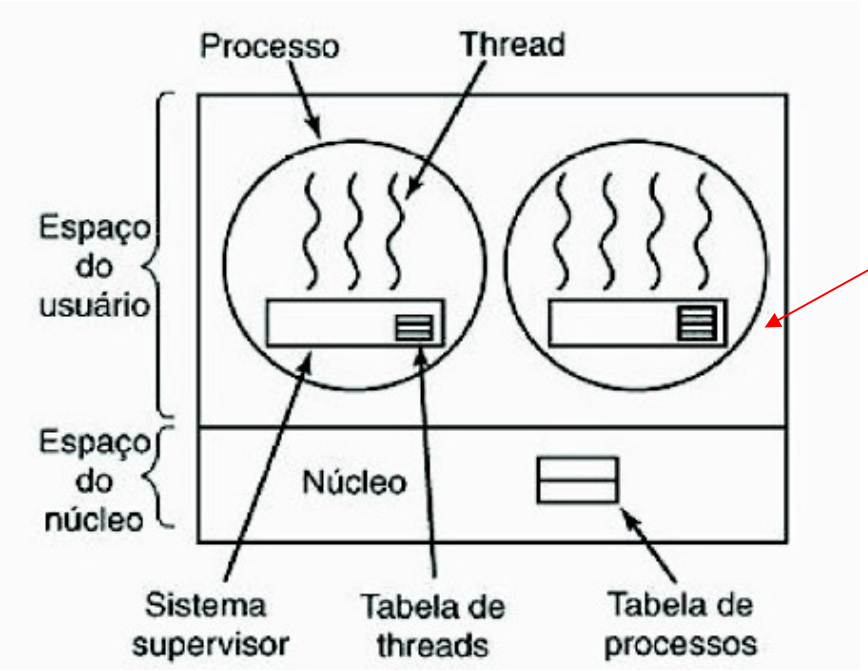
- Threads de usuário
  - fluxos de execução dentro de um processo, associados à execução da aplicação

- Threads de kernel
  - fluxos de execução dentro do kernel
  - representam threads de usuário ou atividades do kernel

- Modelos de implementação
  - N:1, 1:1, N:M

Threads de usuário (User-level threads) – Modelo N:1 (1)

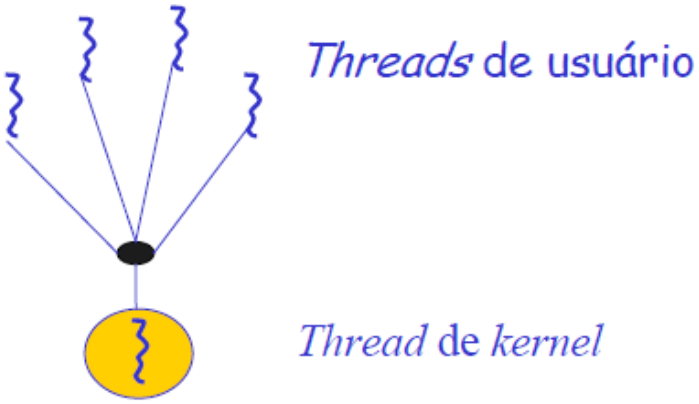
- admitidas no nível do usuário e gerenciadas sem o suporte do kernel
    - thread requisita I/O → bloqueia todo o processo

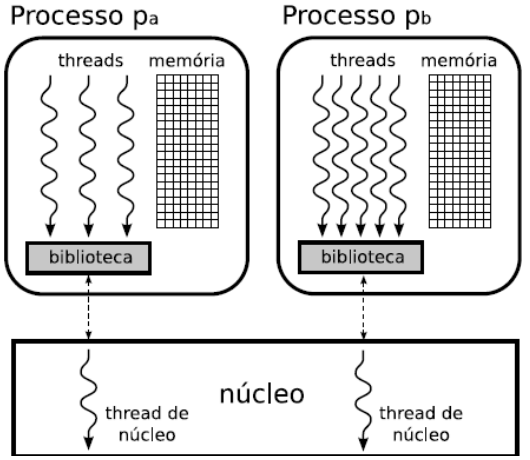


Biblioteca de threads com:
- Tabela de threads
- Escalonador próprio

Threads de usuário (User-level threads) – Modelo N:1 (2)

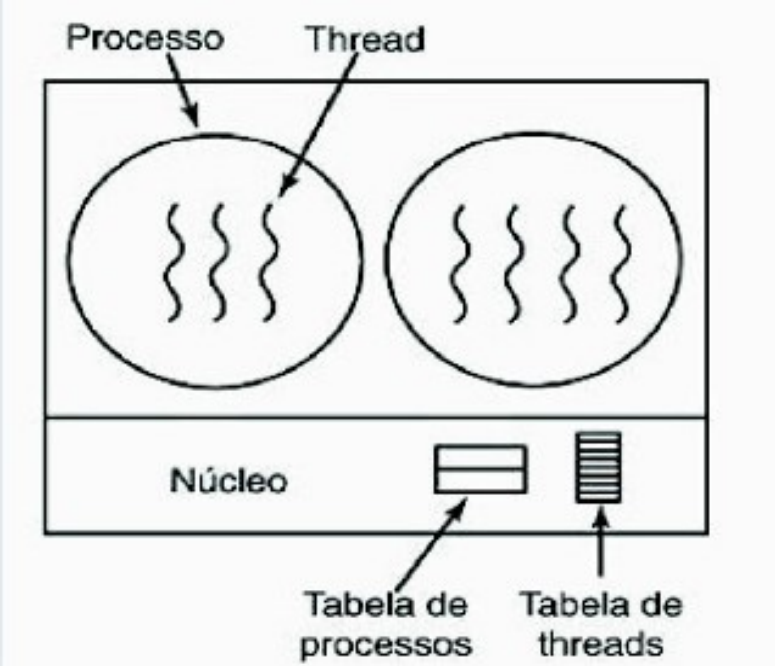
- N threads de usuário mapeadas para 1 thread de kernel
- Gerenciamento feito pela biblioteca de threads no nível de usuário
- Se uma thread faz chamada de sistema bloqueante, todo o processo será bloqueado
- São também chamadas de Green threads
- Ex.: GNU threads, Python





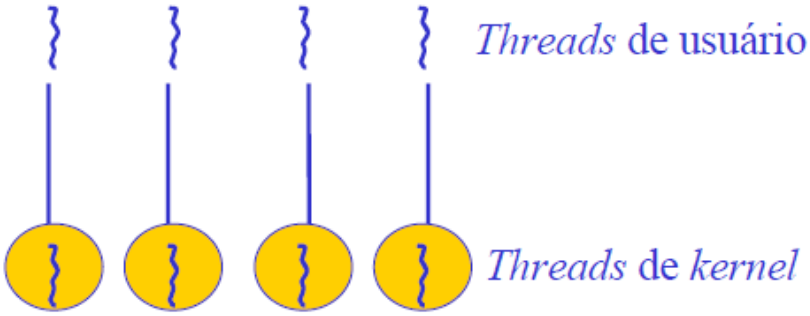
Threads de kernel (Kernel-level threads) – Modelo 1:1 (1)

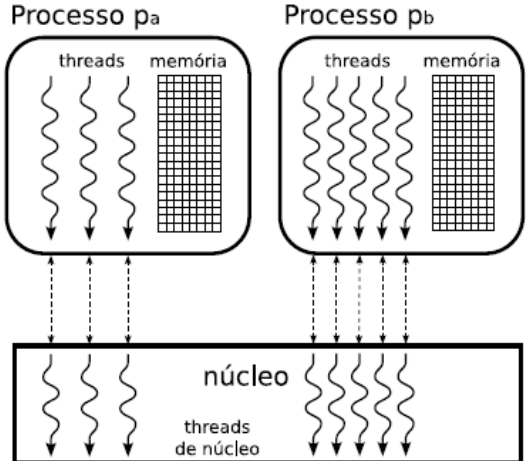
- admitidas e gerenciadas diretamente pelo SO
  - kernel chaveia entre threads, independente do processo a que pertencem



Threads de kernel (Kernel-level threads) – Modelo 1:1 (2)

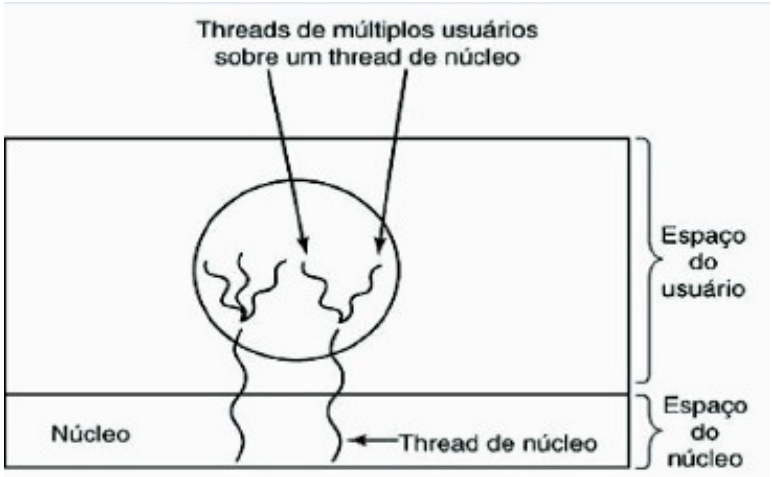
- Mapeia cada thread de usuário para 1 thread de kernel
- Thread é a unidade de escalonamento do núcleo
- Biblioteca de chamadas de sistema inclui operações para criar/controlar threads
- É a implementação mais frequente atualmente
- Ex.: Linux, Windows





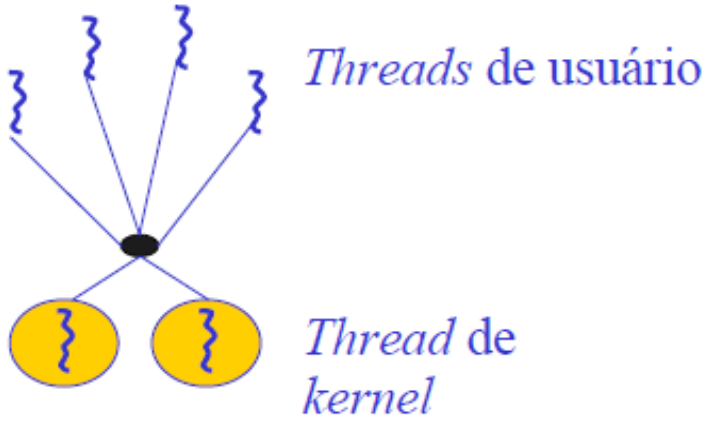
Threads: implementação híbrida – Modelo N:M (1)

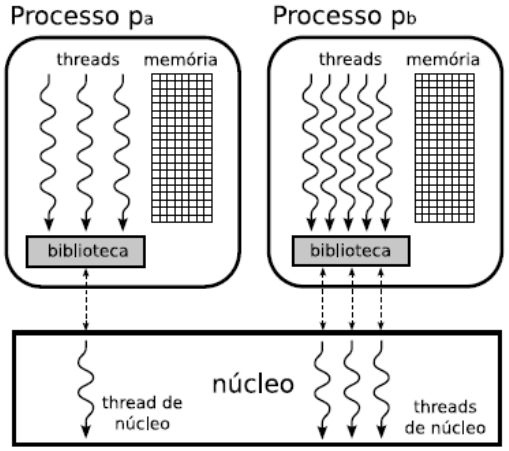
- Tenta combinar as vantagens dos 2 modelos anteriores
  - Usuário: rápida criação e chaveamento entre threads
  - Kernel: o processo todo não é bloqueado pelo bloqueio de uma thread
- A ideia é utilizar algumas threads de kernel e multiplexar threads de usuário sobre elas



Threads: implementação híbrida – Modelo N:M (2)

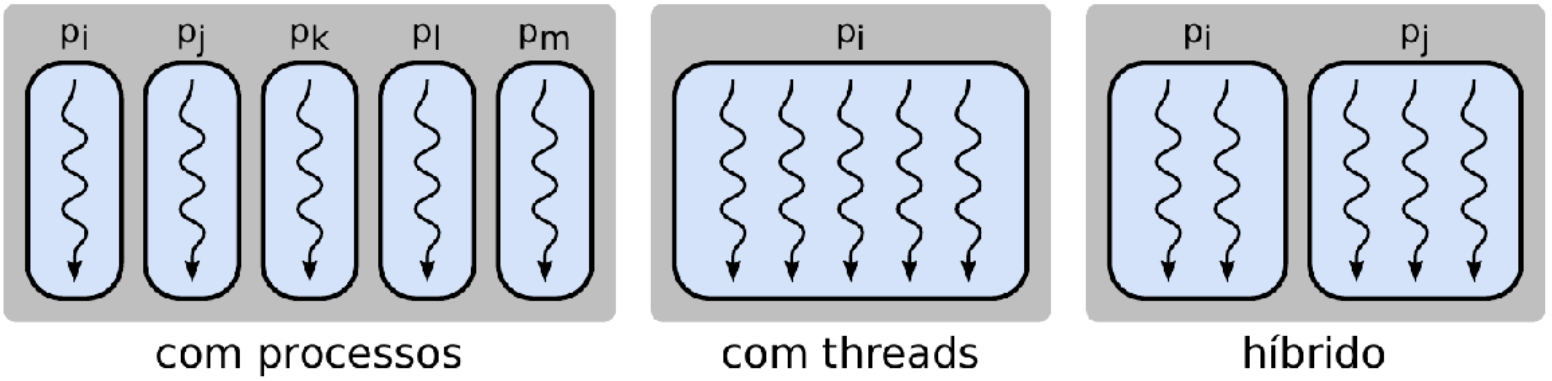
- Mapeia N threads de usuário para M threads de kernel (M<N)
- Uma biblioteca no processo gerencia *user threads*. Essas threads são mapeadas em *threads do kernel*
- Quando uma thread faz chamada de sistema bloqueante, SO pode escalonar outra thread do mesmo processo
- Ex.: Solaris , FreeBSD





Usar threads ou processos? (1)

- Implementar um programa com várias tarefas:
  - Colocar cada tarefa em um processo
  - Colocar tarefas como threads no mesmo processo
  - Usar vários processos com várias threads (abordagem híbrida)



créditos: prof. Maziero

Usar threads ou processos? (2)

Característica	Com processos	Com *threads* (1:1)	Híbrido
Custo de criação de tarefas	alto	baixo	médio
Troca de contexto	lenta	rápida	variável
Uso de memória	alto	baixo	médio
Compartilhamento de dados entre tarefas	canais de comunicação e áreas de memoria compartilhada.	variáveis globais e dinâmicas.	ambos.
Robustez	um erro fica contido no processo.	um erro pode afetar todas as *threads*.	um erro pode afetar as *threads* no mesmo processo.
Segurança	cada processo pode executar com usuários e permissões distintas.	todas as *threads* herdam as permissões do processo onde executam.	*threads* com as mesmas permissões podem ser agrupadas em um mesmo processo.
Exemplos	Apache 1.*, PostGres	Apache 2.*, MySQL	Chrome, Firefox, Oracle

créditos: prof. Maziero

Bibliotecas de Threads

- Oferecem uma API para a criação e gerenciamento de threads
- 2 formas de implementar bibliotecas:
  - Biblioteca no espaço do usuário sem suporte do kernel
  - Biblioteca no espaço do núcleo com suporte direto do SO
- Bibliotecas mais comuns
  - POSIX Pthreads (nível de usuário ou de kernel)
  - Win32 (nível de kernel)
  - Java (nível de usuário, mas usa a biblioteca do SO hospedeiro)

Ex.: biblioteca Pthreads (threads.c)

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>

void * Thread0(){
  int i;

  for(i=0;i<10;i++){
    printf(" Thread0 [%d] - a\n", i);
    sleep(1+(rand()%10));
  }
}

void * Thread1(){
  int i;

  for(i=0;i<10;i++){
    printf(" Thread1 [%d] - b\n", i);
    sleep(1+(rand()%10));
  }
}

int main(){
  pthread_t t0, t1;

  srand((unsigned)time(NULL));

  pthread_create(&t0, NULL, Thread0,NULL);
  pthread_create(&t1, NULL, Thread1,NULL);
  pthread_join(t0,NULL);
  pthread_join(t1,NULL);
  printf("Fim do main ....\n");
  return 0;
```

```
pitthan@pitthan: ~/Área de Trabalho/2020-1/elc1080/aulas/p1/a7threads
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
pitthan@pitthan:~/Área de Trabalho/2020-1/elc1080/aulas/p1/a7thread
gcc -o threads threads.c -pthread
pitthan@pitthan:~/Área de Trabalho/2020-1/elc1080/aulas/p1/a7thread
./threads
 Thread1 [0] - b
 Thread0 [0] - a
 Thread1 [1] - b
 Thread1 [2] - b
 Thread0 [1] - a
 Thread1 [3] - b
 Thread1 [4] - b
 Thread1 [5] - b
 Thread0 [2] - a
 Thread1 [6] - b
 Thread0 [3] - a
 Thread0 [4] - a
 Thread1 [7] - b
 Thread0 [5] - a
 Thread1 [8] - b
 Thread0 [6] - a
 Thread1 [9] - b
 Thread0 [7] - a
 Thread0 [8] - a
 Thread0 [9] - a
Fim do main ....
pitthan@pitthan:~/Área de Trabalho/2020-1/elc1080/aulas/p1/a7thread
```

Mais exemplos... (biblioteca Pthreads)

- Sem e com argumentos

  hello.c, hello_arg.c, hello_args.c, hello_join.c

- Comparação entre processos e threads

  fork-compara.c, thread-compara.c

Aplicações de threads:

- cálculo de soma de matrizes, cálculo de números primos, cálculo de divisores, …

→ restritivas quanto à sincronização