INE5412 Sistemas Operacionais I

L. F. Friedrich

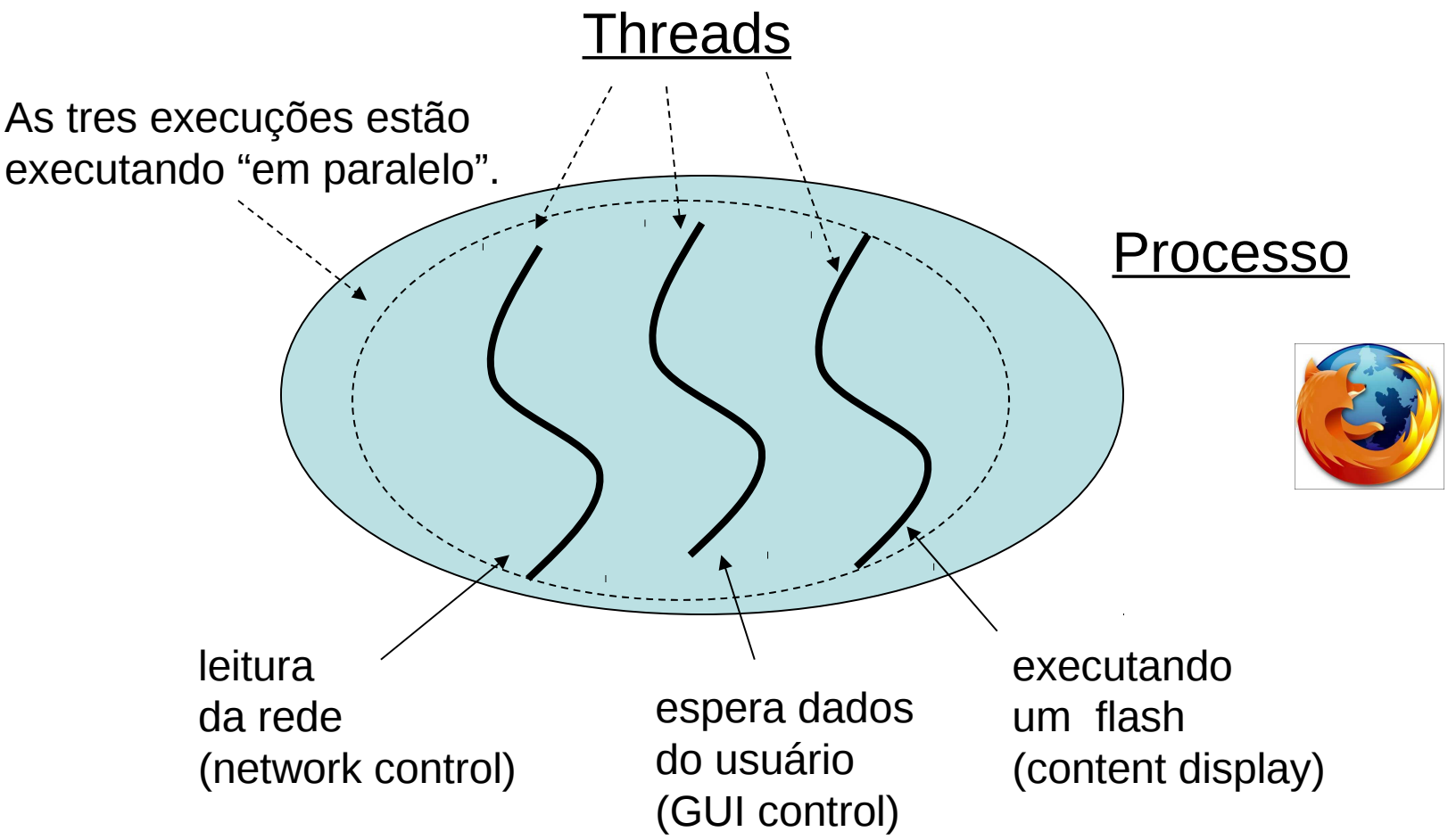
Capítulo 2 – Parte 2

Threads

O que é um(a) thread

- Uma thread é uma entidade de execução dentro de um processo.
- O processo que conhecemos é um processo com uma única thread single-threaded.
  - Existe apenas uma linha de execução no processo.
- Na realidade, é possível ter mais de uma linha de execução em um processo.
- Este tipo de processo é chamado processo com múltiplas threads multi-threaded process.

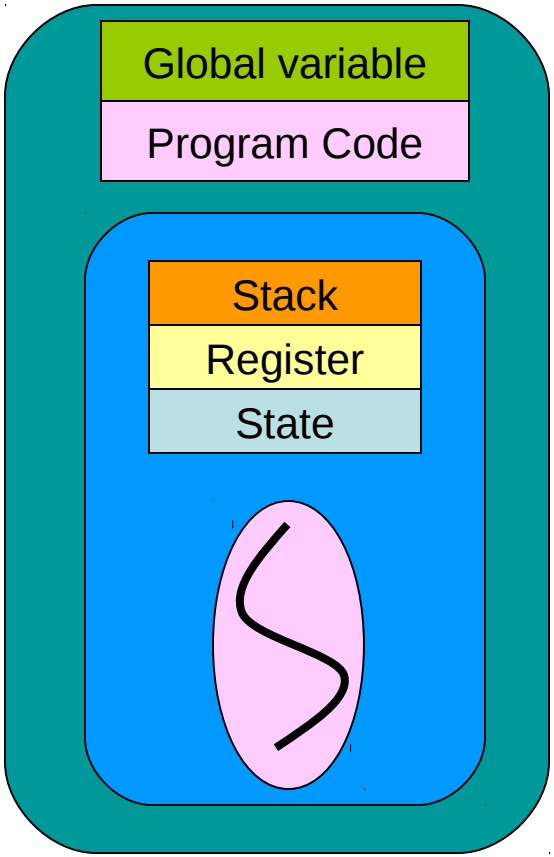
Exemplo de Thread



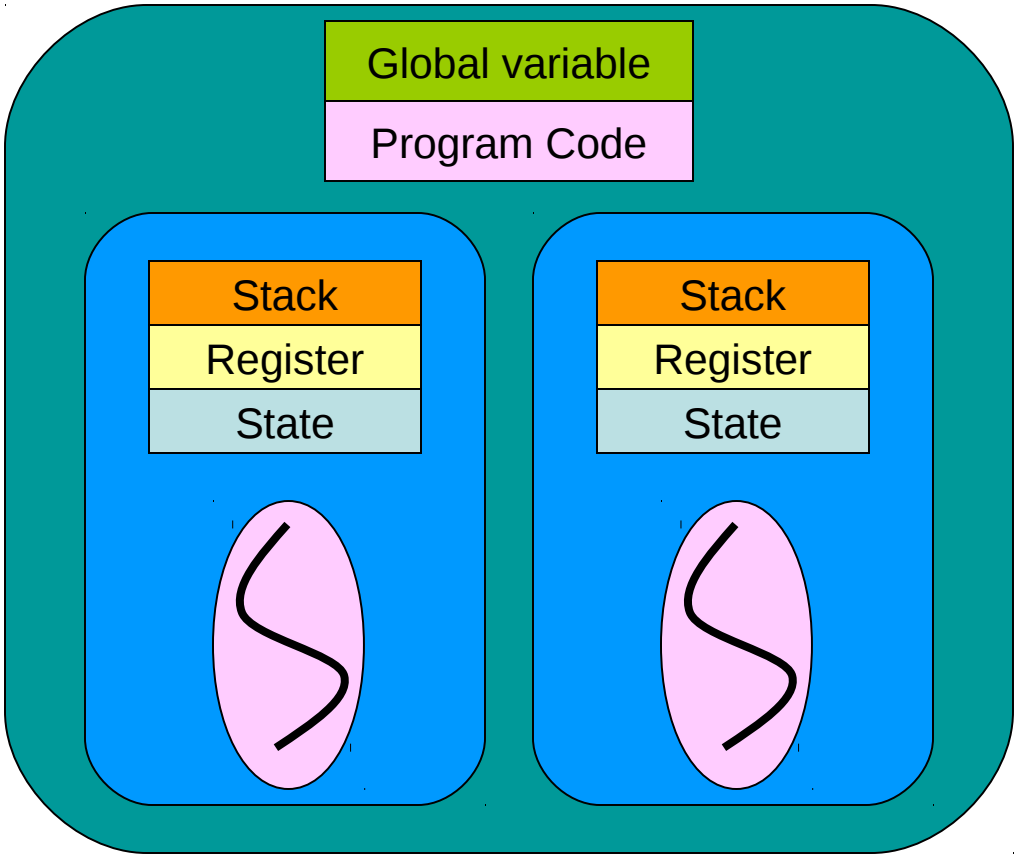
O que é uma thread, tecnicamente?

- Todas as threads de um processo compartilham o mesmo código e o mesmo conjunto de variáveis globais.
  - Elas executam no mesmo código, programa.
  - Elas compartilham o mesmo conjunto de variáveis globais.
    - Problema da exclusão mútua?
    - Problema da sincronização?
- Cada thread tem seu próprio conjunto de registradores e sua própria pilha (stack).
  - Ela tem seu próprio PC (program counter), assim diferentes threads podem estar executando em diferentes segmentos (procedimentos) de código do mesmo programa.
  - Diferentes conjuntos de pilha de memória, significa: mesmo que todas as threads executam nas mesmas funções, elas tem seu próprio conjunto de variáveis locais.
- Além disso, cada thread tem seu próprio estado.
  - Uma thread bloqueada por E/S não afetará outra thread.

O que é uma thread, tecnicamente?

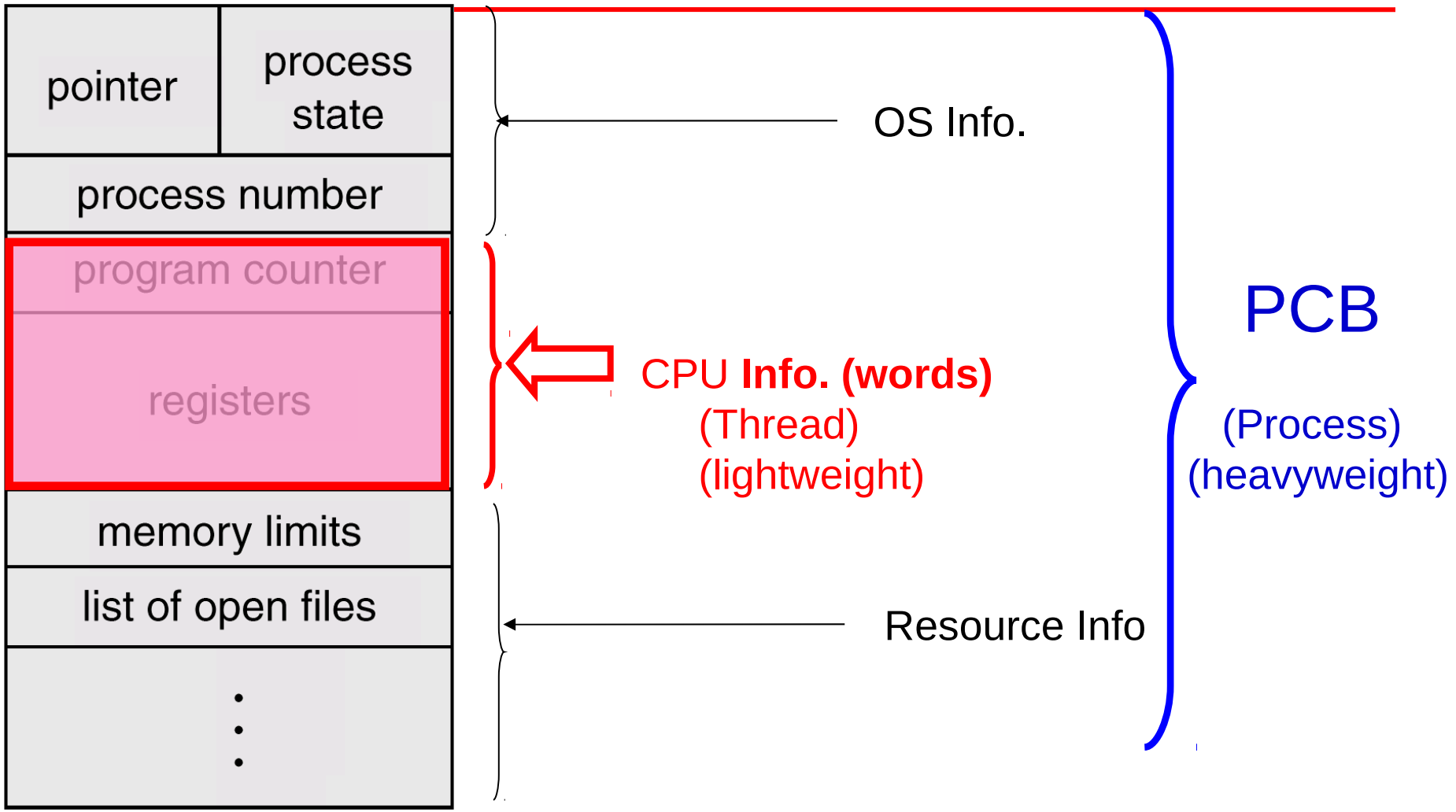


Single-threaded

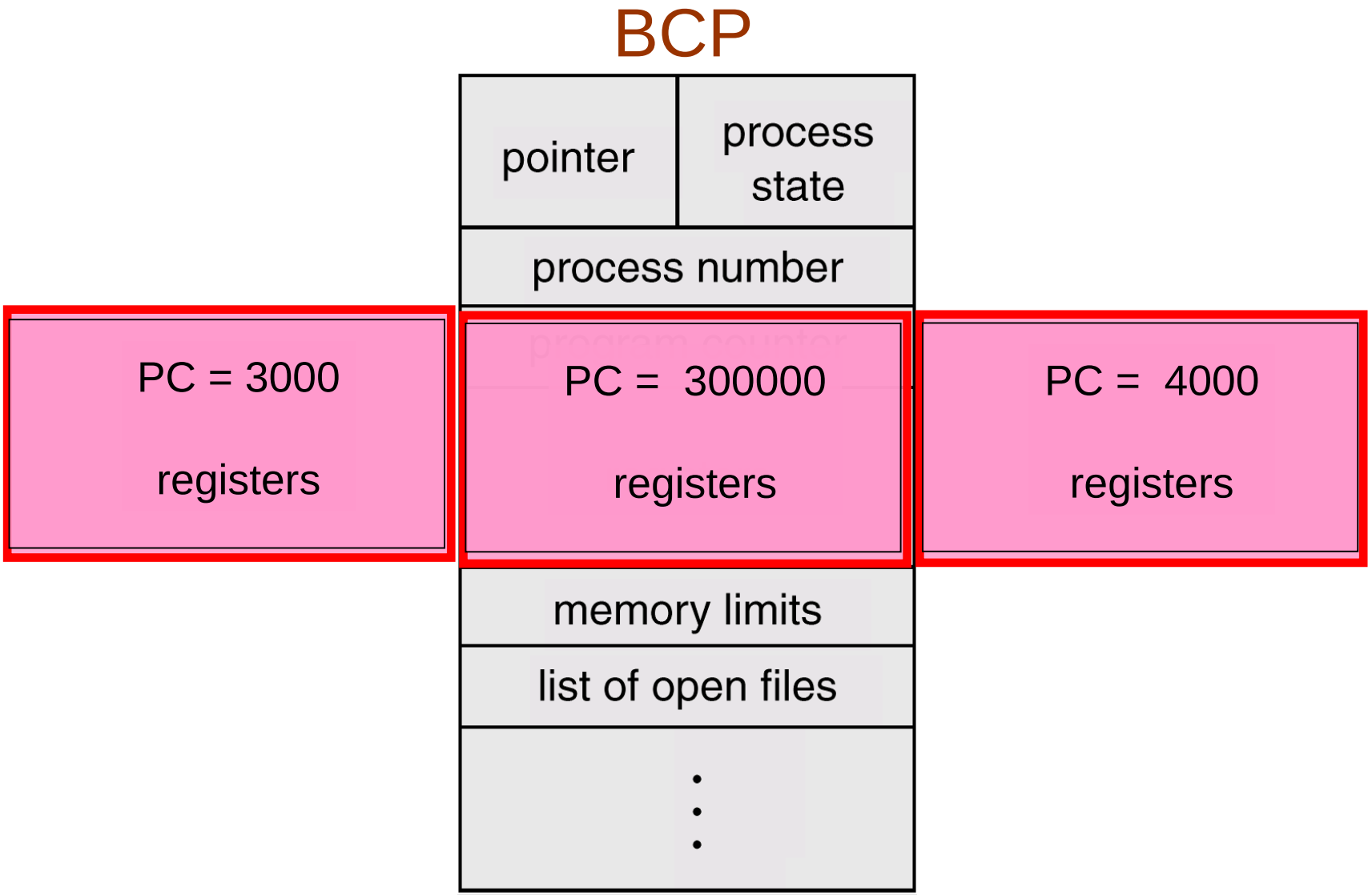


Multiple-threaded

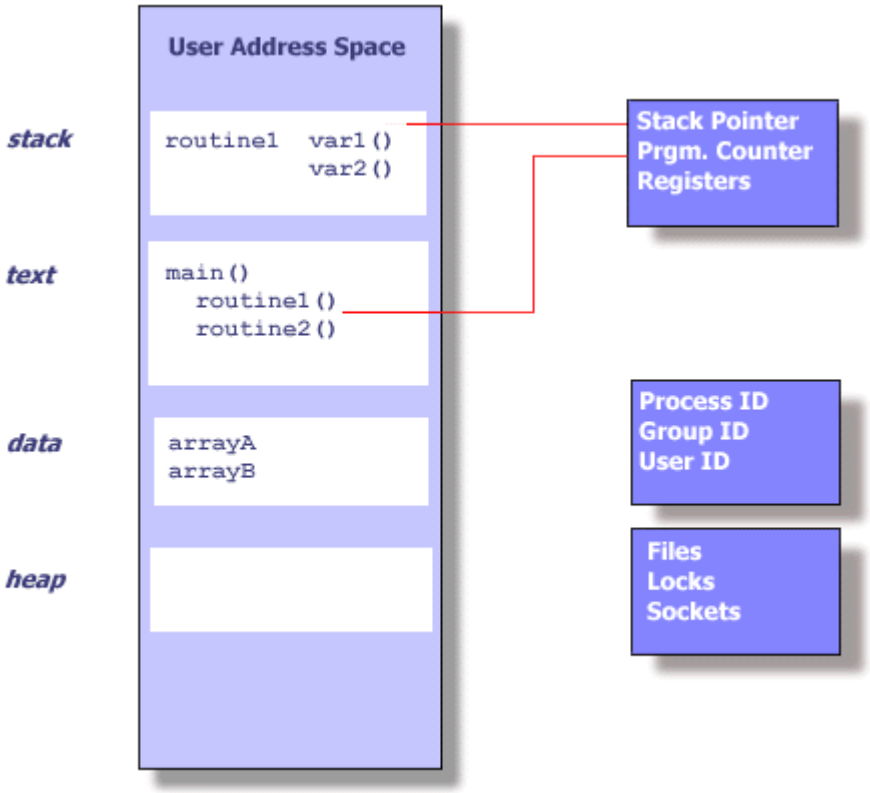
Bloco de Controle de Processo



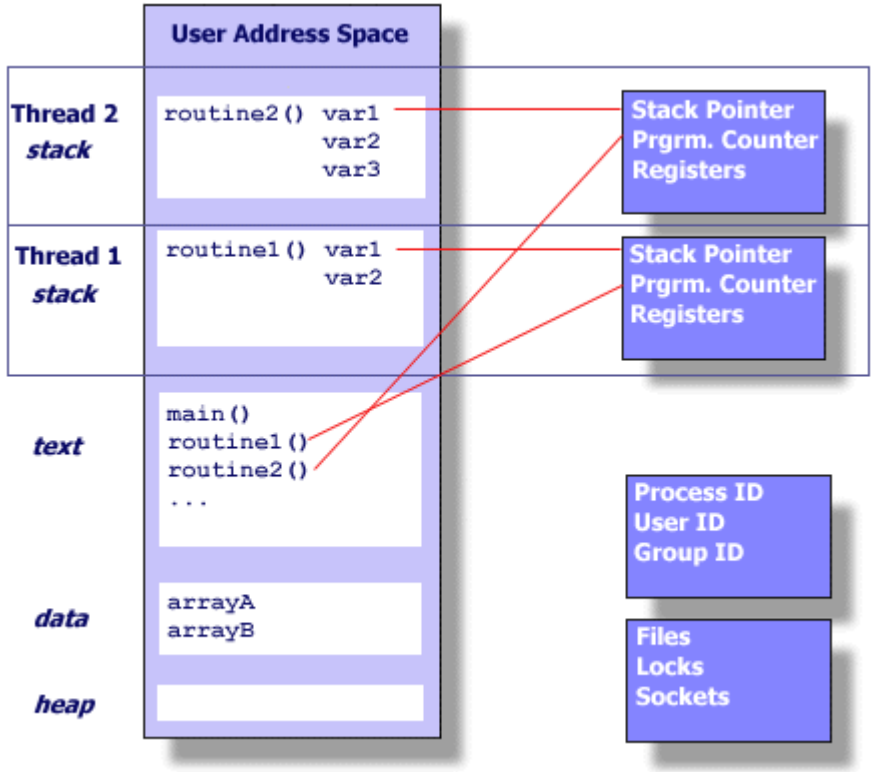
Um Processo – Três Threads



Unix Threads



Unix Process



Threads within a Unix Process

Porque Threads?

- Execução simultânea de funções
  - Permite uma aplicação, por um lado, esperar entradas do usuário, e por outro lado, fornece outras funções.
  - A aplicação não é bloqueada durante espera de E/S; apenas a thread esperando é bloqueada.
  - Permite multi-tarefa no contexto de um processo.
- Compartilhamento de recursos
  - N threads podem compartilhar código, dados, recursos do processo, como arquivos.

Uso do thread

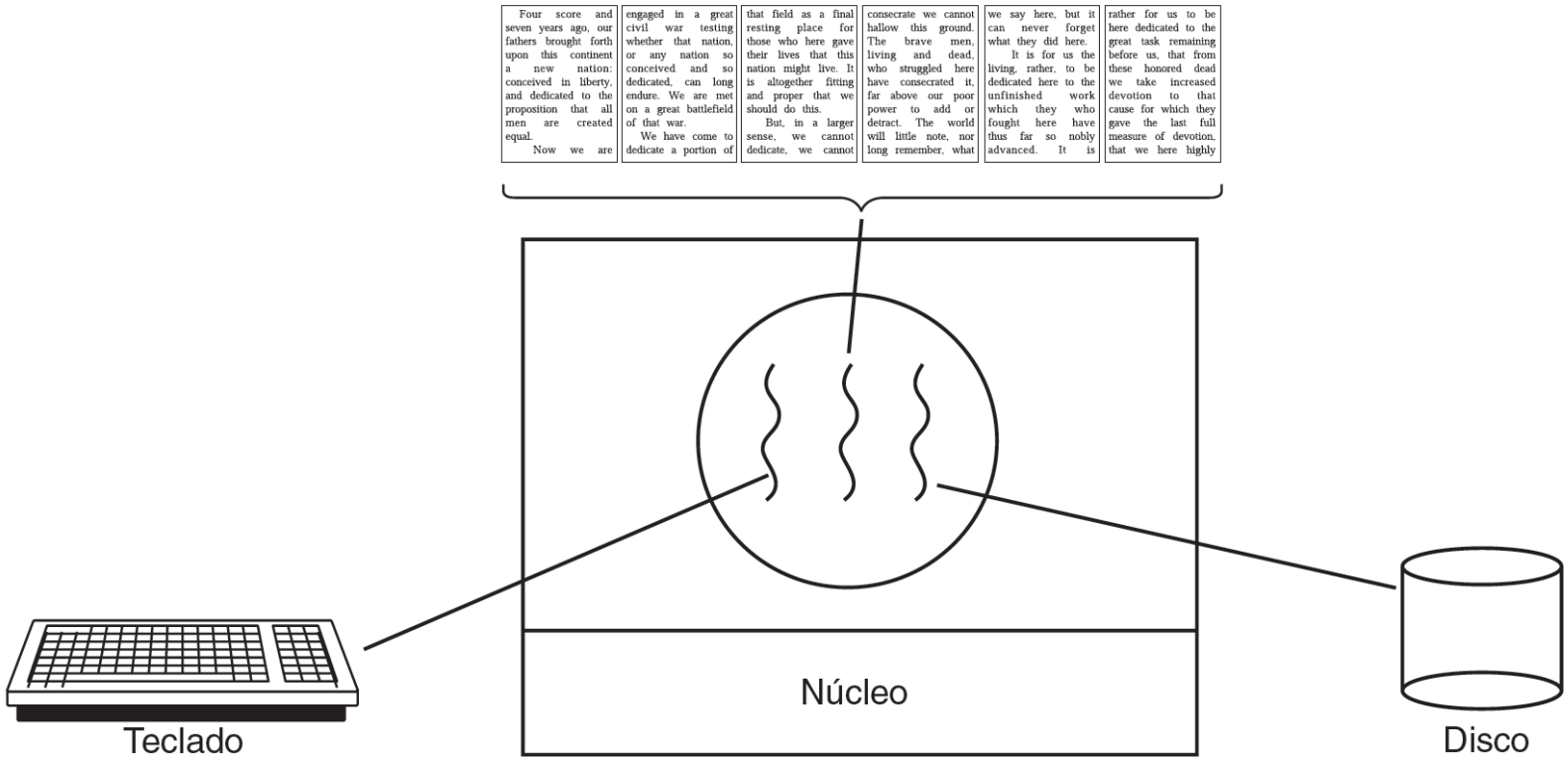


Figura 2.5 Um processador de textos com três threads.

Porque Threads?

- Economia
  - Uma thread é 30 vezes mais rápida de criar e 5 vezes mais rápida de chavear do que um processo.
- Concorrência
  - Processos multi-threaded podem ter o mesmo grau de concorrência que multiplos processos.
  - Utilização de arquiteturas MP

Desempenho (fork()Xpthread)

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
AMD 2.4 GHz Opteron (8cpus/node)	41.07	60.08	9.01	0.66	0.19	0.43
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.24	30.78	27.68	1.75	0.69	1.10
IBM 1.5 GHz POWER4 (8cpus/node)	104.05	48.64	47.21	2.01	1.00	1.52
INTEL 2.4 GHz Xeon (2 cpus/node)	54.95	1.54	20.78	1.64	0.67	0.90
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.54	1.07	22.22	2.03	1.26	0.67

Source

http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.

Real – início ao fim da chamada
User – modo usuário (fora do kernel)
Sys – modo kernel (execução da chamada)

Uso de Threads

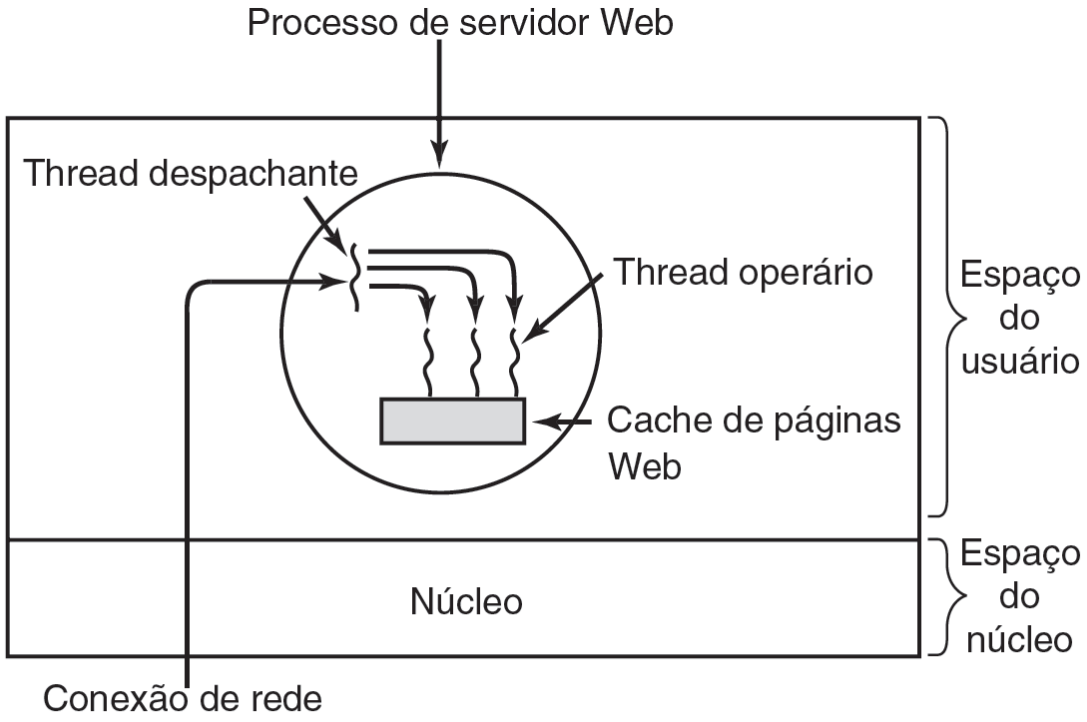


Figura 2.6 Um servidor Web multithread.

```
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)

Figura 2.7 Uma simplificação do código para a Figura 2.6. (a) Thread despachante. (b) Thread operário.

O modelo de thread clássico

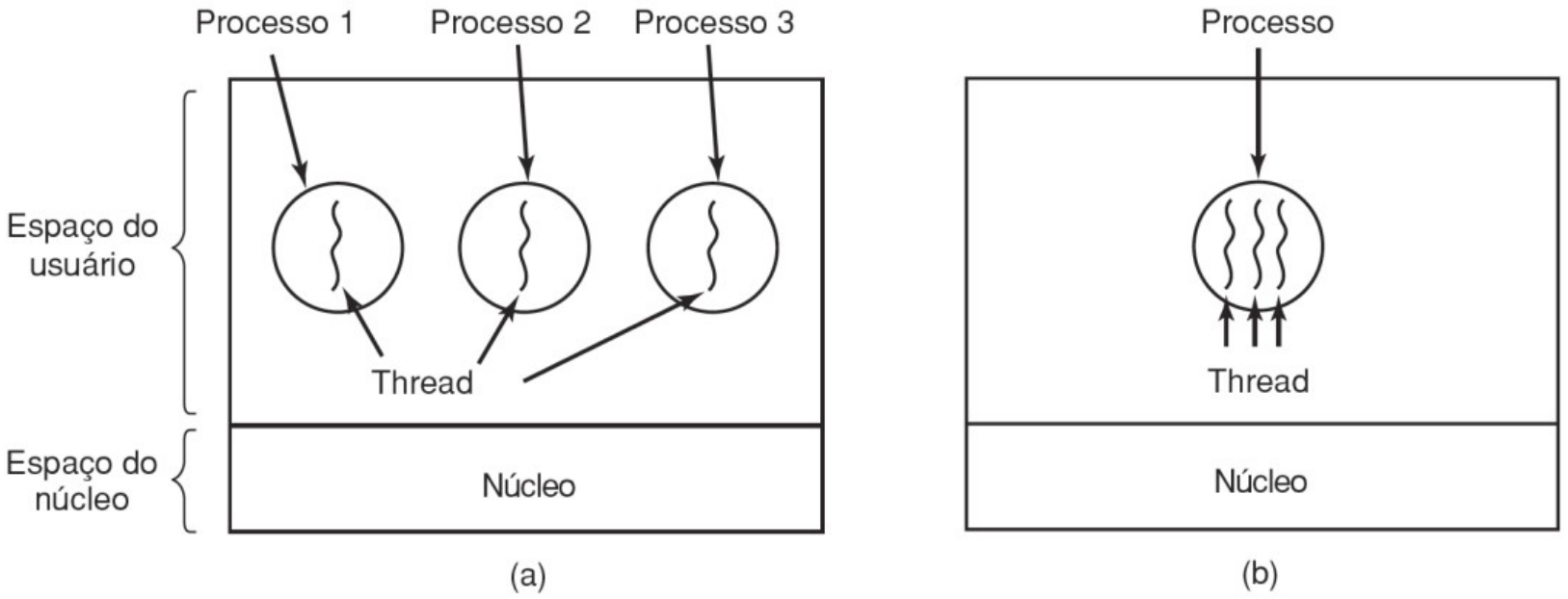


Figura 2.8 (a) Três processos, cada um com um thread. (b) Um processo com três threads.

O modelo de thread clássico

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.

O modelo de thread clássico

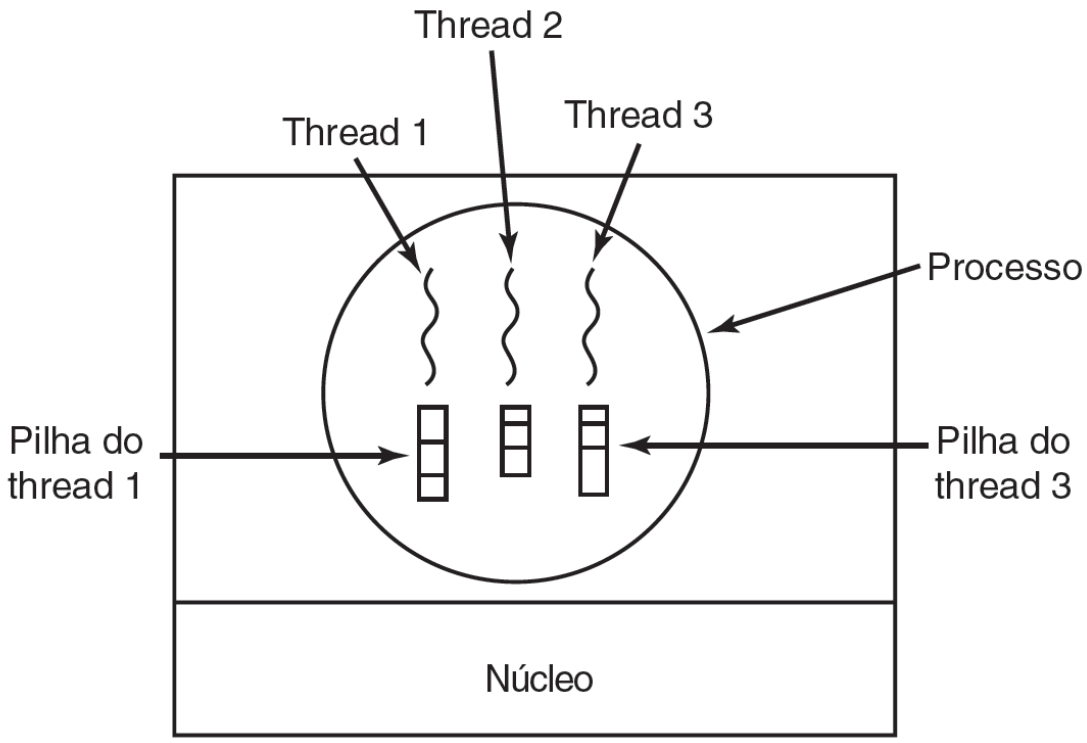


Figura 2.9 Cada thread tem sua própria pilha.

Threads POSIX

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

Tabela 2.5 Algumas das chamadas de função de Pthreads.

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);
}
```

Figura 2.10 Um exemplo de programa usando threads.

Implementando threads

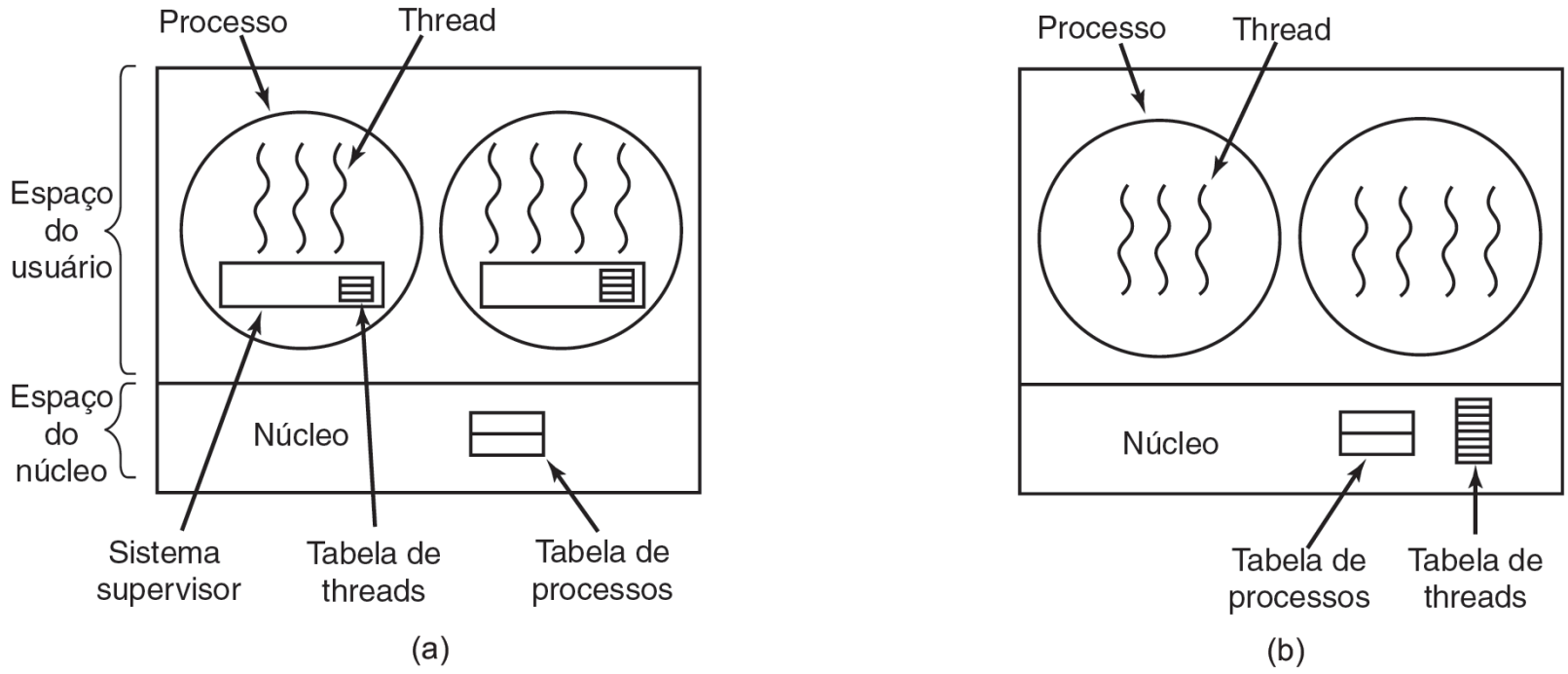


Figura 2.11 (a) Um pacote de threads de usuário. (b) Um pacote de threads administrado pelo núcleo.

Threads de Usuário

- Gerenciamento das Threads feito por biblioteca de threads no nível de usuário
  - Criação, destruição, comunicação, escalonamento, salvamento de contexto, etc.

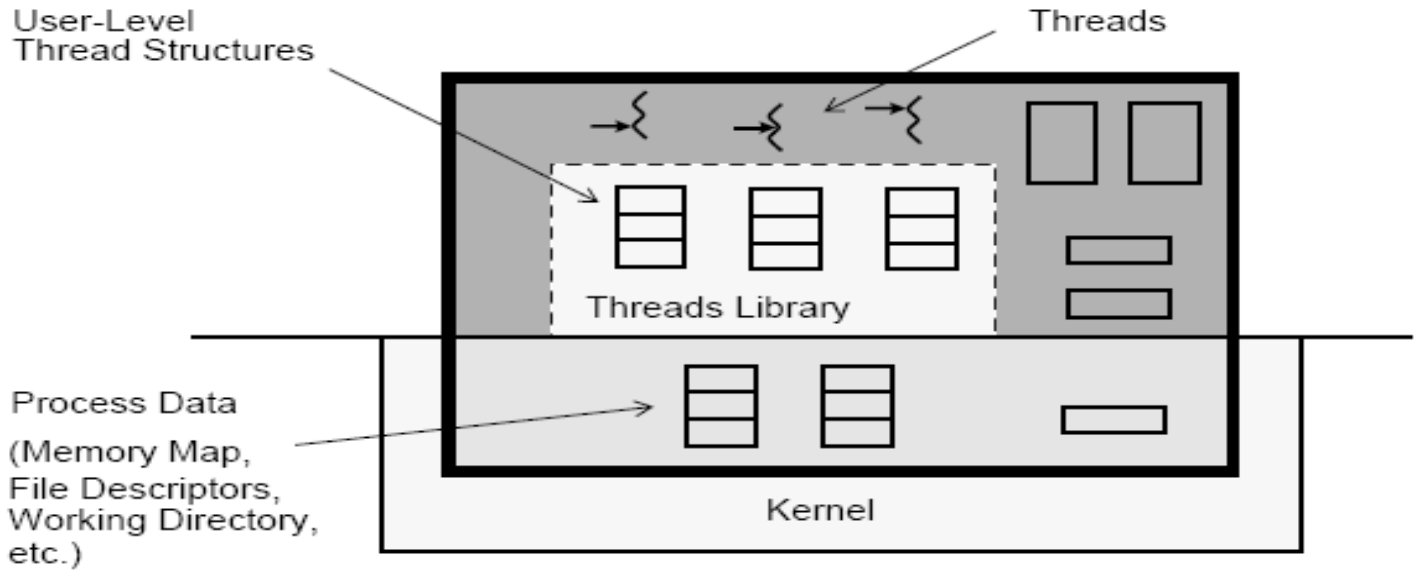
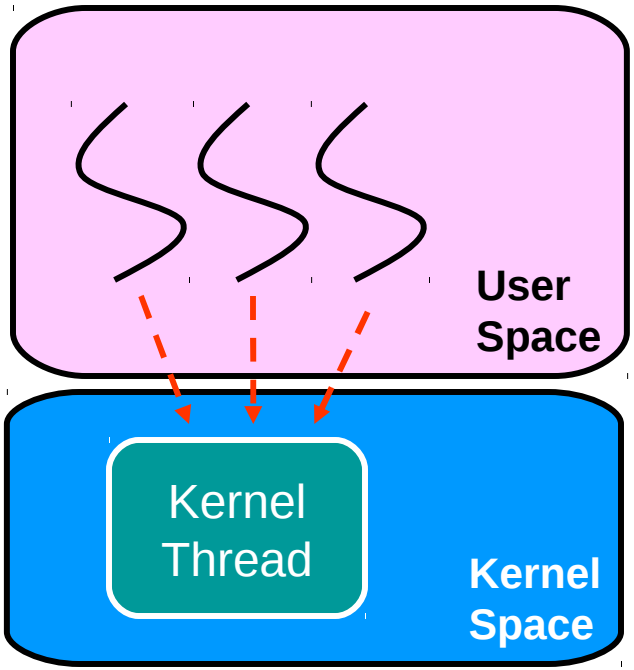


Figure 2-5 The Process Structure and the Thread Structures

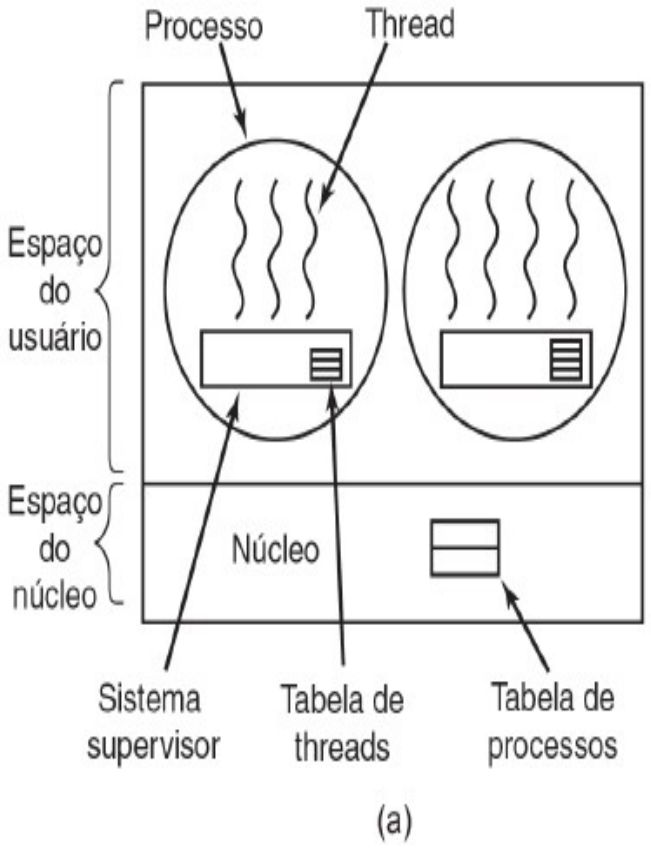
Modelos de Thread - usuário

- Many-to-One Model
  - Todas as threads são mapeadas para uma estrutura thread no kernel.
  - Gerenciamento da thread é feito por uma biblioteca (run-time) no nivel de usuário.
    - Ex., Thread scheduling.
  - Mérito: criar muitas e processamento rápido devido ao modo usuário.
  - Desvantagem: quando uma syscall bloqueante é chamada, todas threads serão bloqueadas.

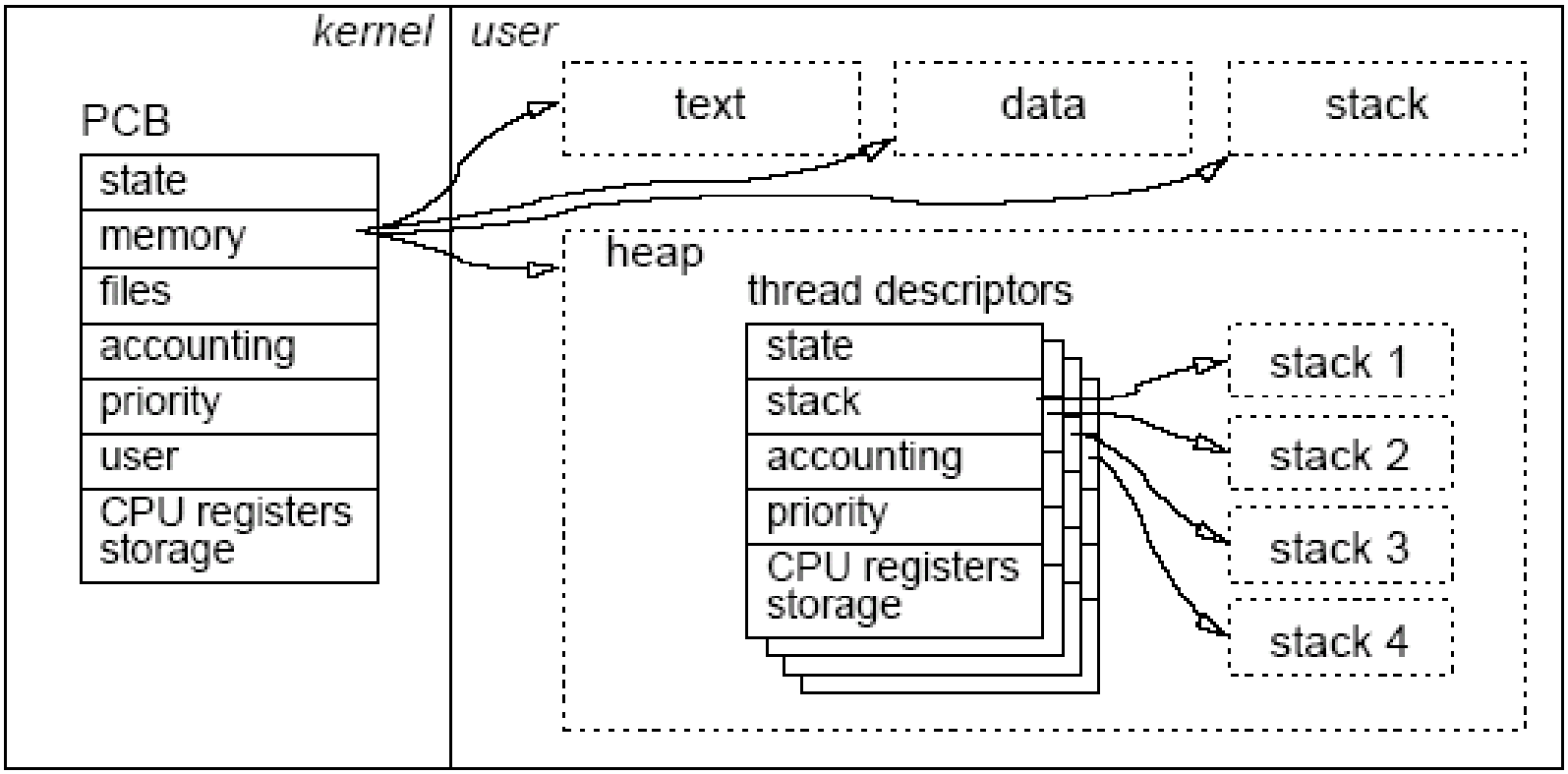


Threads de Usuário

- Supor processo B executando thread 3
  - Chamada de sistema (bloqueia T3)
  - Tempo de B expira
- Vantagens de TU
  - Chaveamento de threads
  - Escalonamento
  - Podem executar em qualquer SO
- Desvantagens
  - Chamadas bloqueantes
  - Sem MP
- Exemplos
  - POSIX *Pthreads,* Solaris *threads, Java threads*



Threads usuário

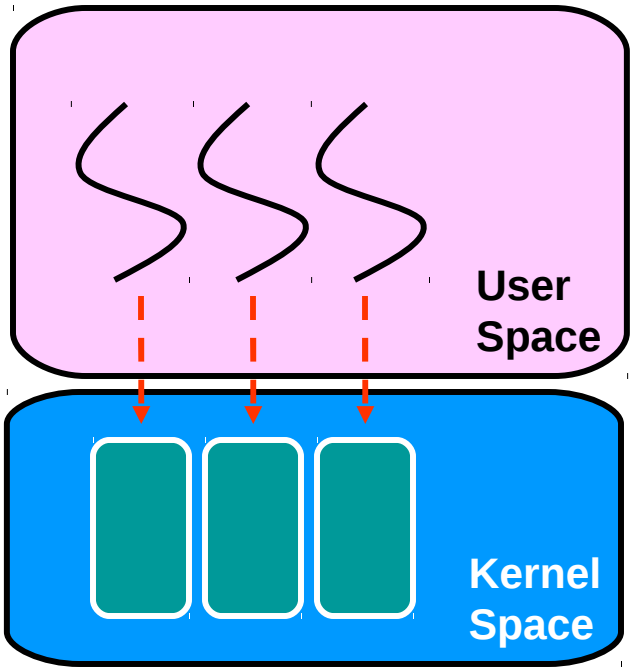


Threads no Kernel

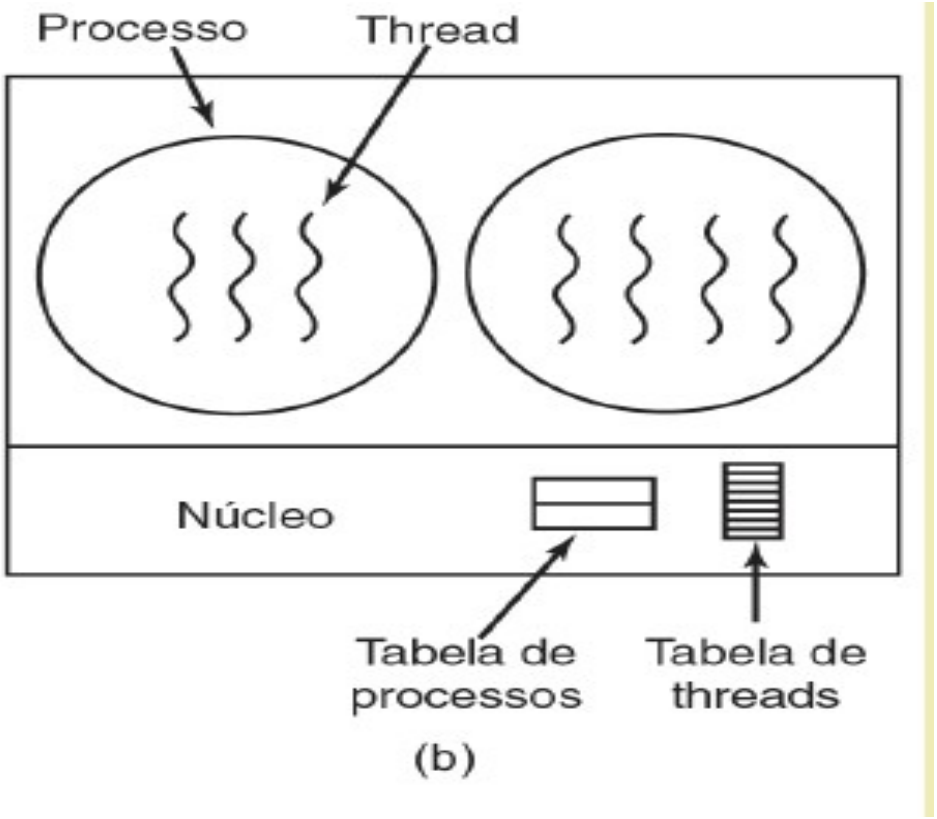
- Suportadas pelo Kernel
  - Kernel mantém informação de contexto
  - Escalonamento é feito com base em thread
- Vantagens
  - Threads do mesmo processo em paralelo
  - Thread bloqueada, escalona outra
- Desvantagem
  - Chaveamento
- Exemplos
  - Windows 95/98/NT/2000
  - Solaris
  - Linux

Modelos de Thread - kernel

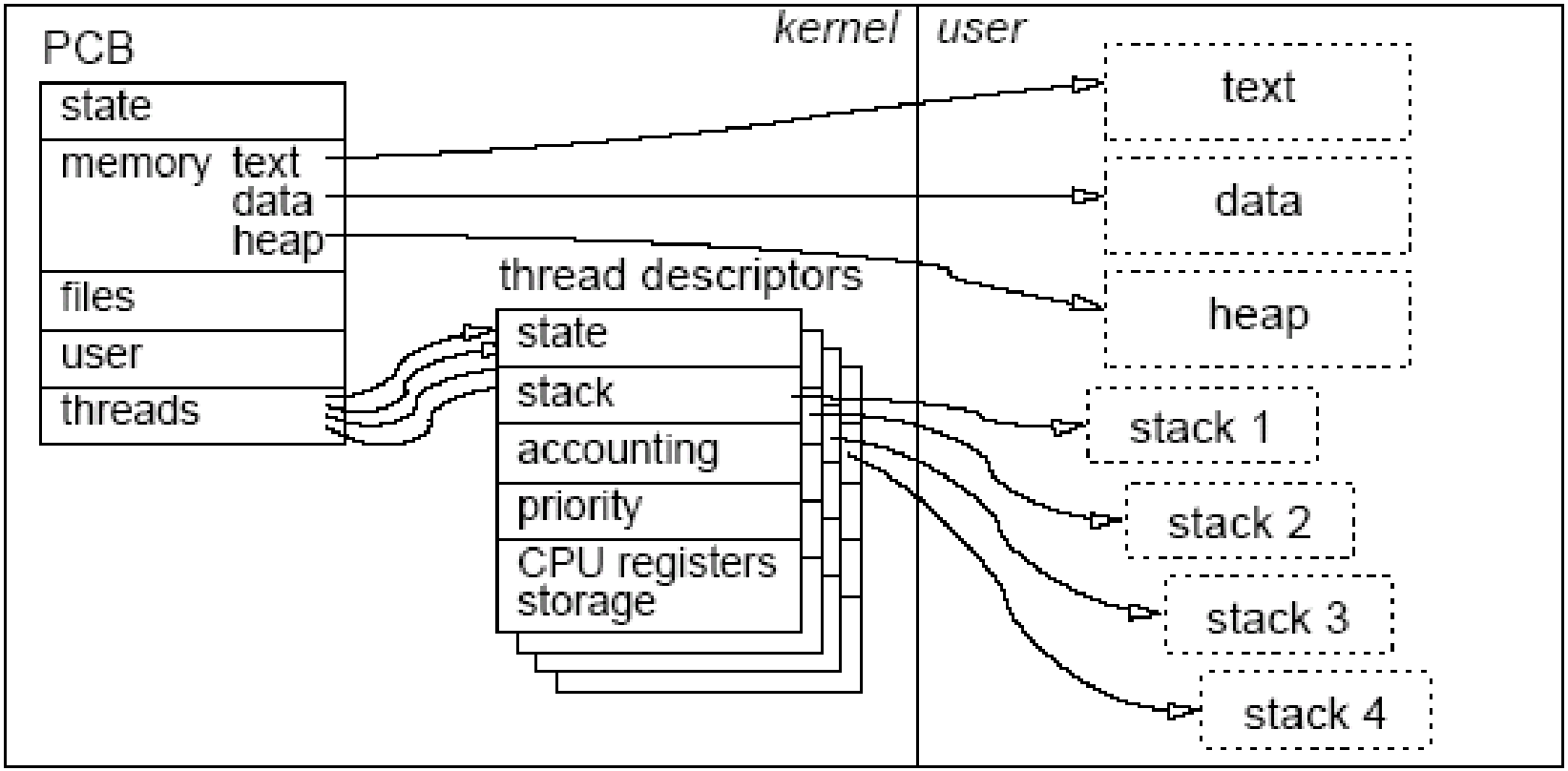
- One-to-One Model
  - Cada thread é mapeada para uma thread no kernel.
  - Mérito: chamadas de syscalls bloqueantes bloqueiam threads chamadoras.
    - Um alto grau de concorrência.
  - Desvantagem: não é possível criar muitas threads devido a restrições de tamanho da memória do kernel. Chaveamento
  - Linux e Windows seguem este model.



Threads no Kernel



Threads kernel



Threads kernel

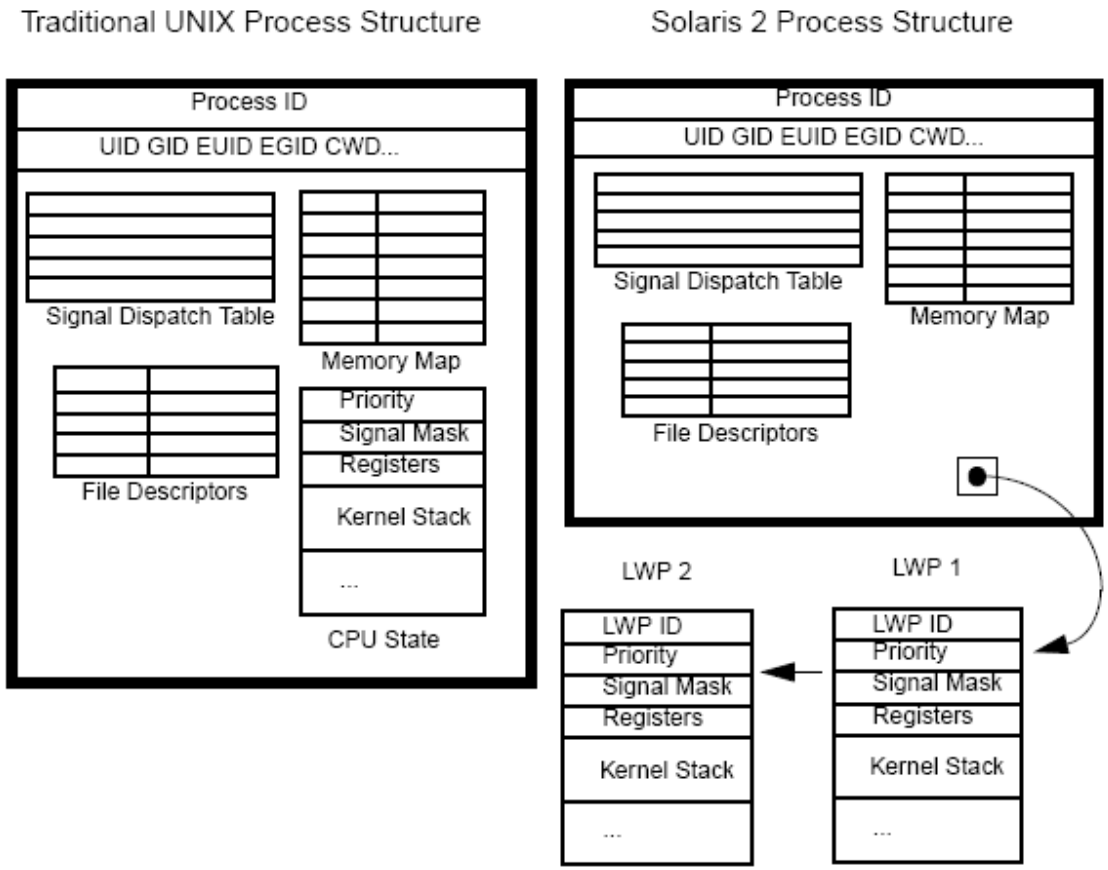
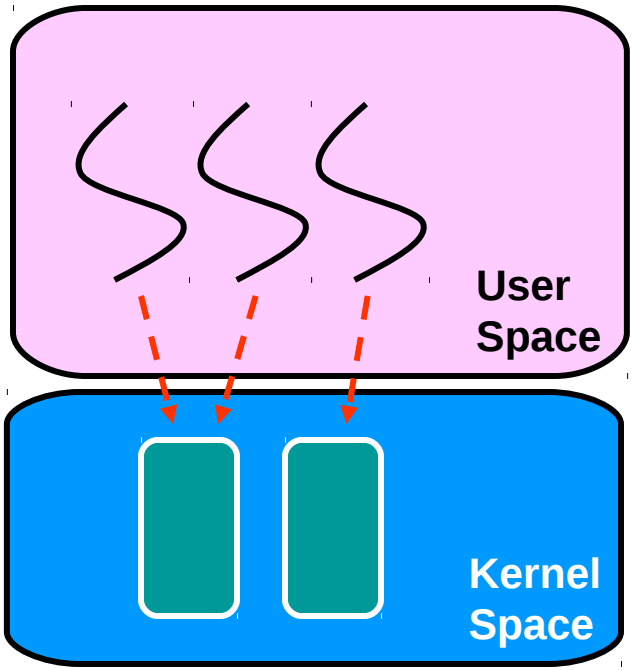


Figure 3-1 The Process Structure in Traditional UNIX and in Solaris 2

Modelos de Thread - híbrida

- Many-to-Many Model
  - Uma abordagem hibrida.
  - Usuário pode criar tantas quanto possível.
  - Usuário pode ter um alto grau de concorrência.
    - Quando uma thread do kernel esta tratando uma syscall bloqueante, o kernel pode escalonar outra thread de usuário dinamicamente.



Implementações híbridas

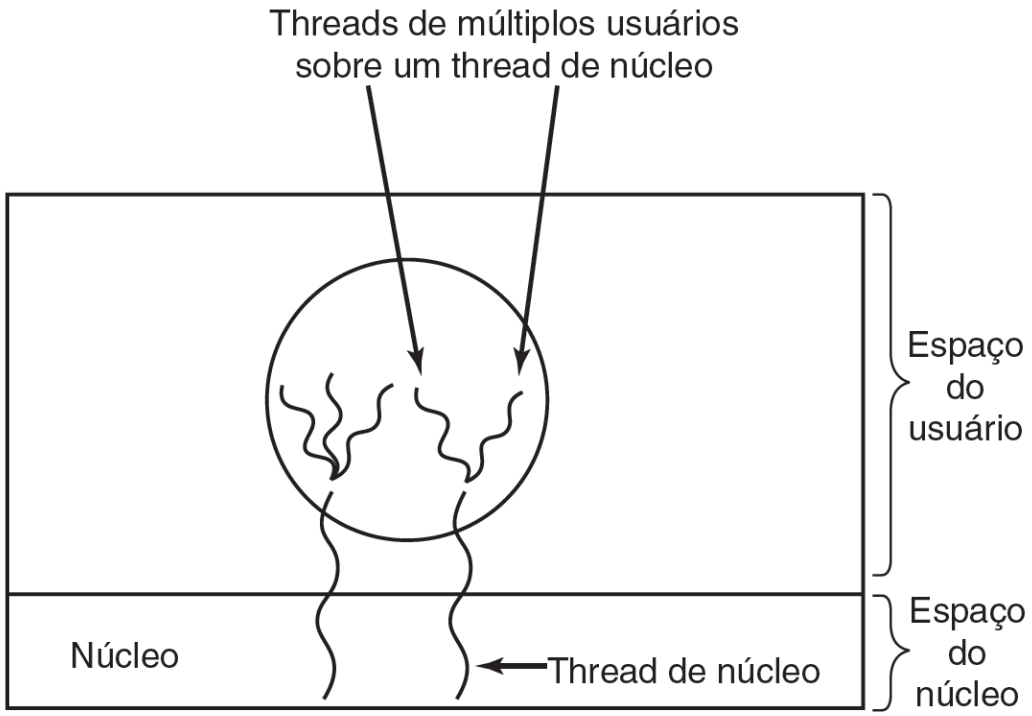
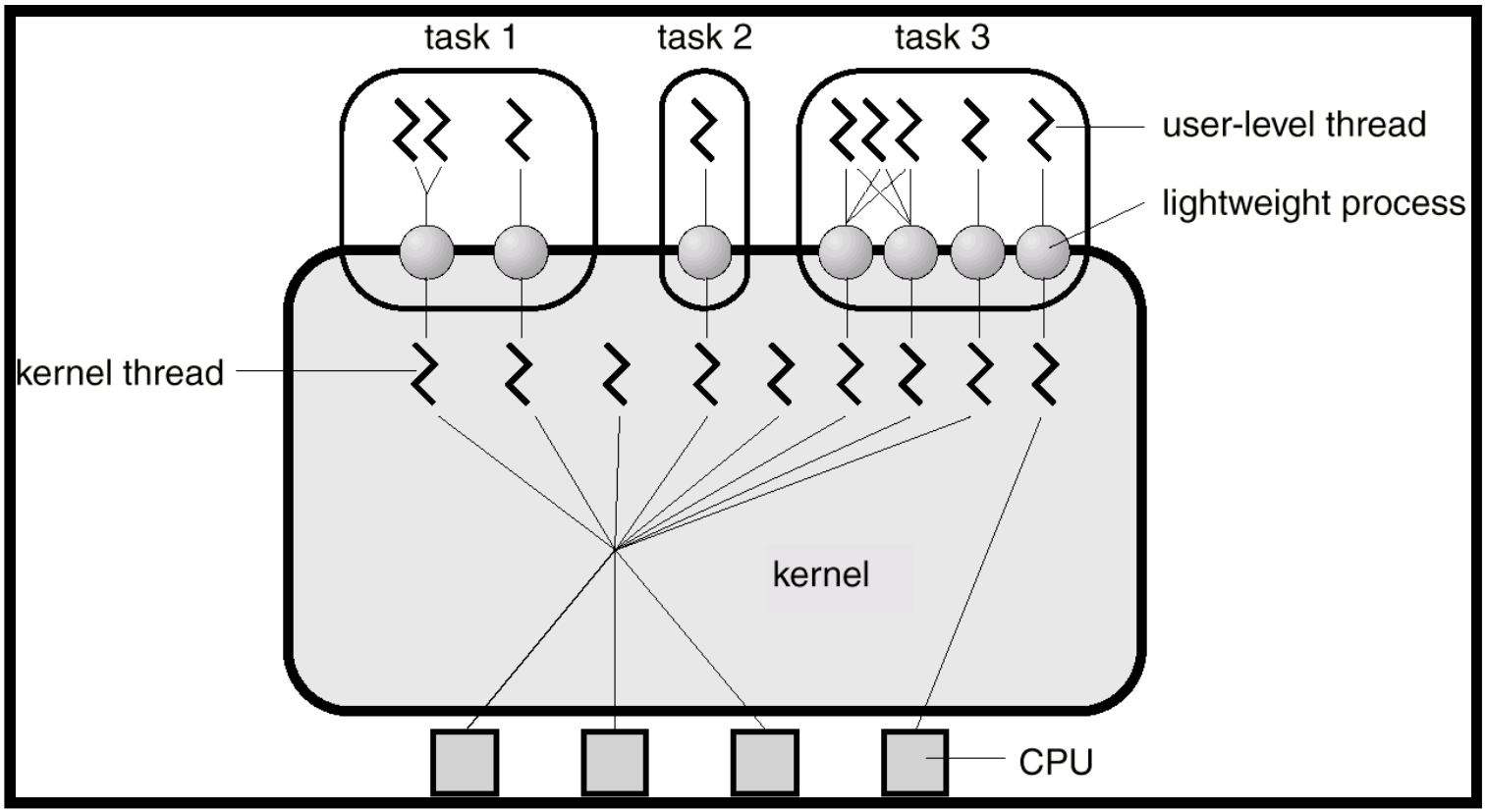


Figura 2.12 Multiplexando threads de usuários sobre threads de núcleo.

Solaris 2 Threads



Solaris

- **Processo**: inclui espaço de endereço do usuário, pilha, e BCP
- **threads nível de usuário**: suporte via biblioteca, SO não vê
- **processos leves**: mapeamento entre TU e TK, suporta 1 ou mais TU -> TK
- **threads do Kernel**: entidades escalonadas e despachadas

Solaris

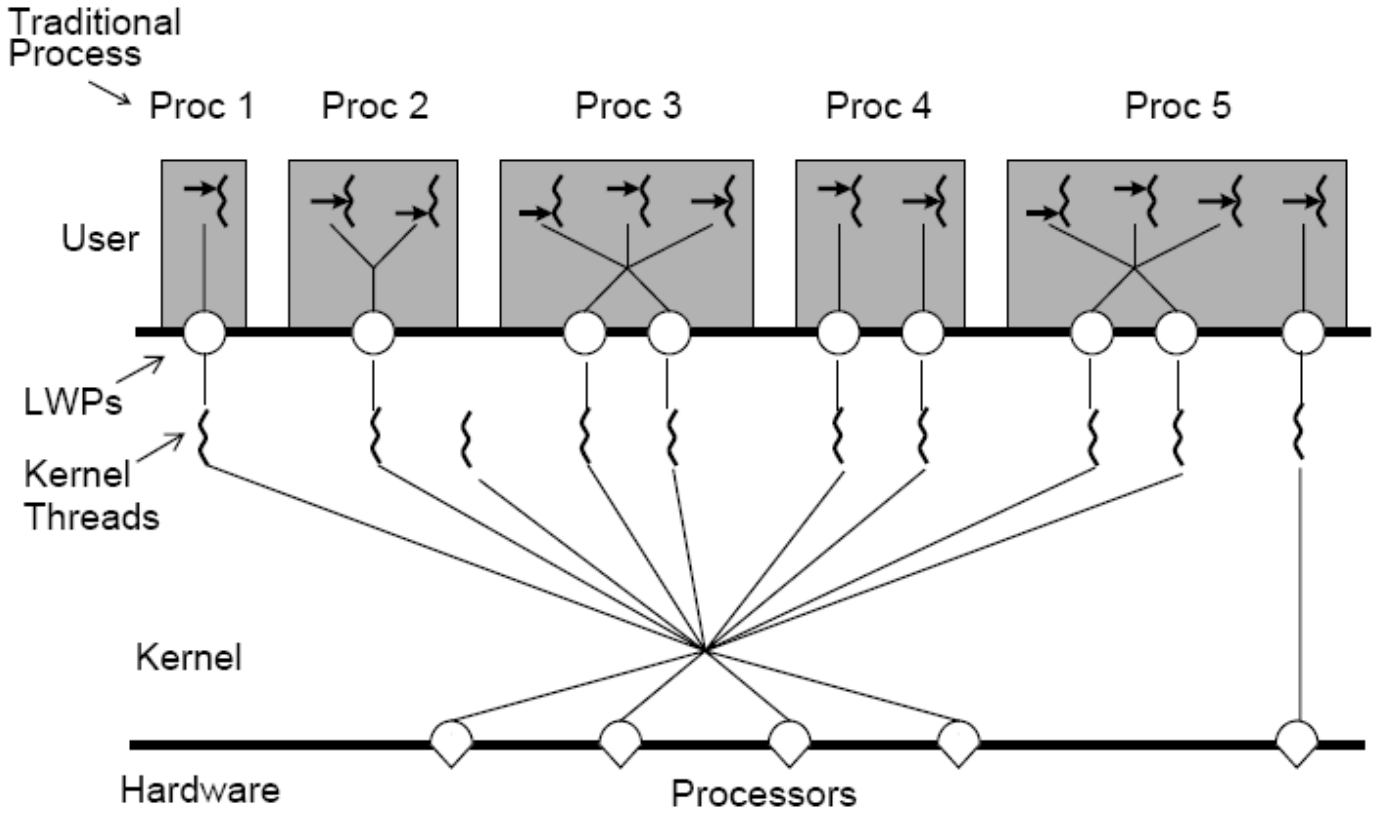


Figure 3-4 The Solaris Multithreaded Architecture

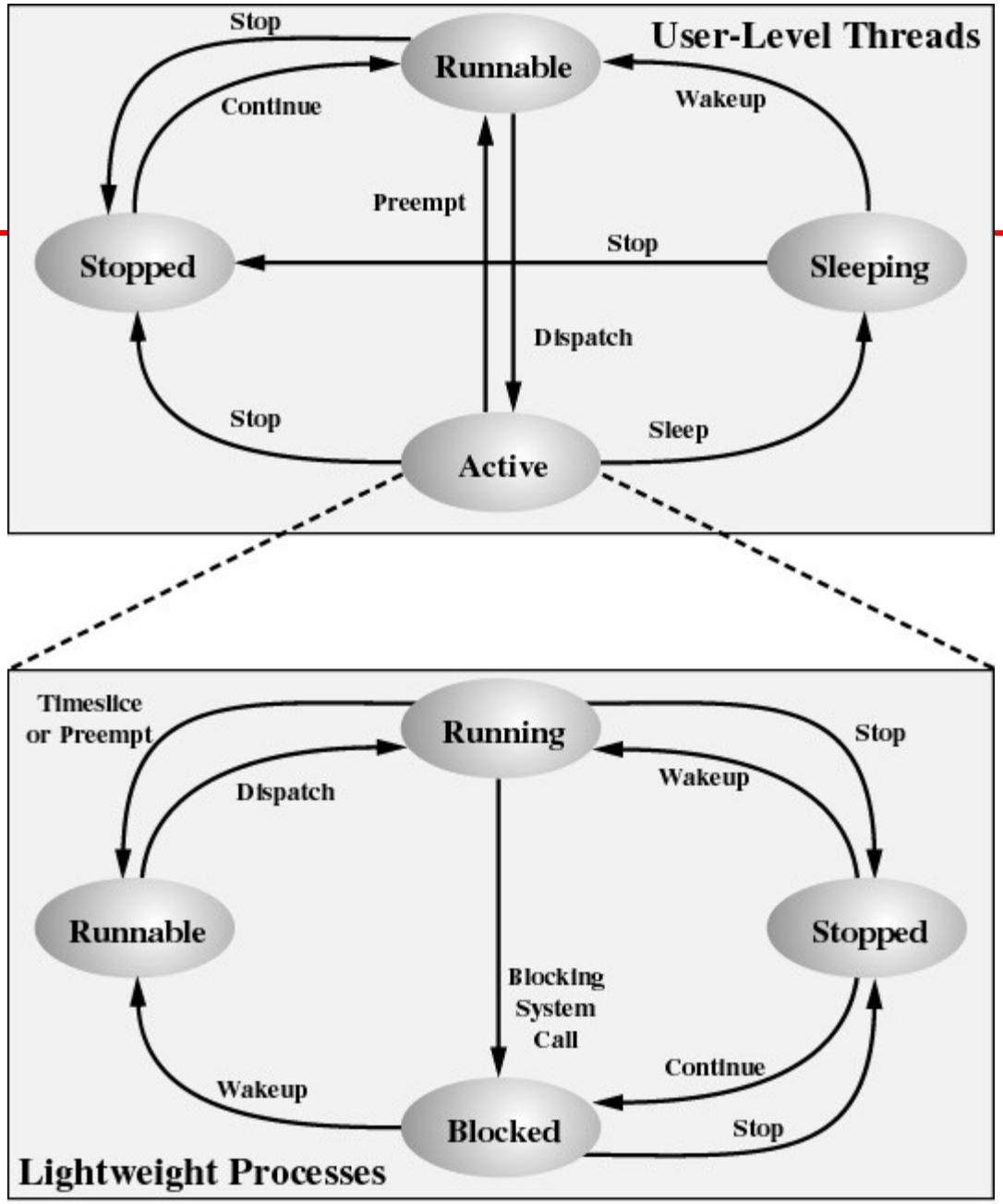


Figure 4.17 Solaris User-Level Thread and LWP States

Threads no Linux

Penguin Specific

Linux tem uma implementação única de threads, no kernel não existe o conceito, que é implementado como processo. (Posix 1003.1c API)

- _ Cada thread é criada por uma chamada de sistema clone()
- _ O kernel não oferece nenhuma estrutura especial para representar threads
- _ Utiliza task_struct

- Então, alguma diferença do fork()?
  - _ Clone() permite o processo/thread compartilhar recursos com outros processos, tal como espaço de endereçamento.
  - _ Usualmente, o que não é compartilhado é a pilha e o conjunto de registradores.
  - _ Diferente do fork(), clone() inicia a execução do processo filho na função fornecida.

```
#include <sched.h>
int clone (int (*fn) (void *), void *child_stack, int flags, void *arg);
```

Ex: Considere um processo com 4 threads em sistemas com suporte explicito (Solaris) comparado com Linux

“LinuxThreads follows the so-called "one-to-one" model: each thread is actually a separate process in the kernel.”

A chamada de sistema clone

Flag	Significado quando marcado	Significado quando limpo
CLONE_VM	Cria um novo thread	Cria um novo processo
CLONE_FS	Compartilha umask e os diretórios-raiz e de trabalho	Não compartilha umask e os diretórios-raiz e de trabalho
CLONE_FILES	Compartilha os descritores de arquivos	Copia os descritores de arquivos
CLONE_SIGHAND	Compartilha a tabela do tratador de sinais	Copia a tabela do tratador de sinais
CLONE_PID	O novo thread obtém o PID antigo	O novo thread obtém seu próprio PID
CLONE_PARENT	O novo thread tem o mesmo par que o chamador	O chamador é o pai do novo thread

Tabela 10.4 Bits no mapa de bits *sharing_ flags*.

Problemas com Threads

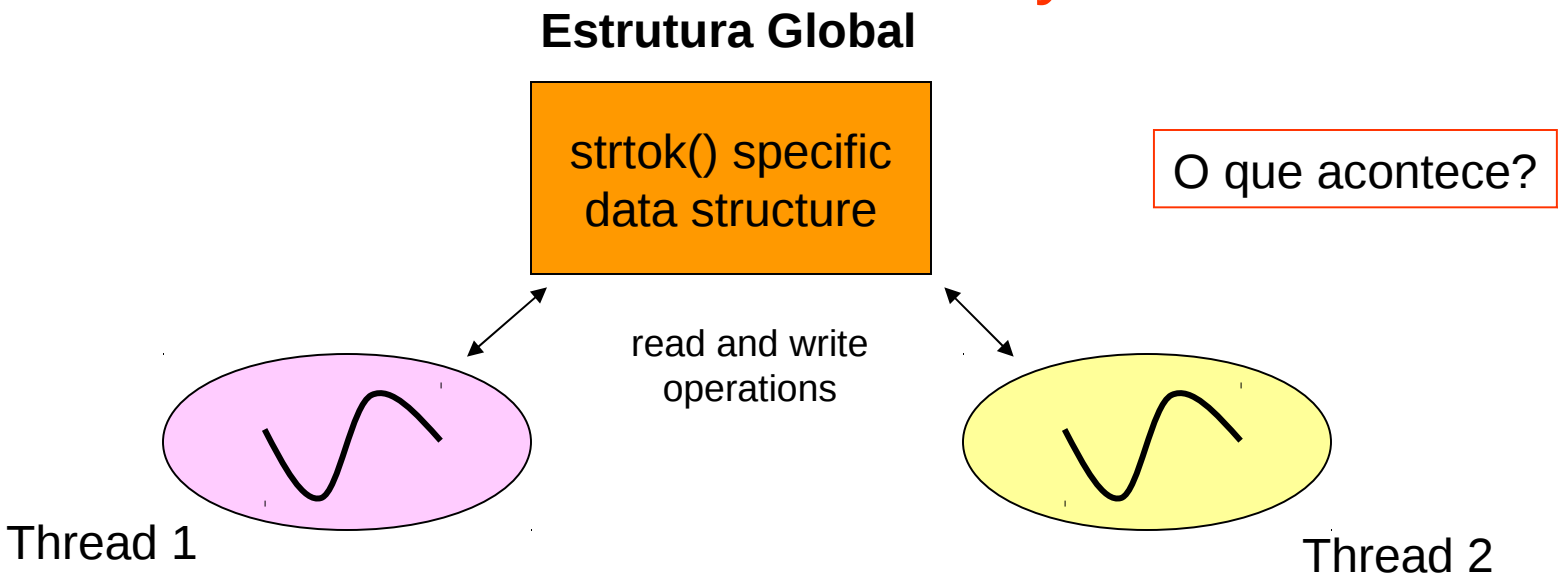
- Multi-threading junto com fork-exec-wait ?
  - Qdo voce usa fork-exec-wait em uma thread, o que irá acontecer?
  - Dependente do sistema.
    - Fork() pode duplicar todas as threads; (calling-1c)
    - Exec() pode cancelar toda a execução das threads;(para todas - 1c)
    - Wait() pode suspender todas as threads;
  - Conclusão: try at your own risk.

Tratamento de Sinais

- Sinais são usados no UNIX para notificar um processo que um evento particular ocorre (interrupção de software)
- Um **signal handler** é usado para tratar sinais
  - Sinal é gerado por um evento particular
  - Sinal é entregue para um processo
  - Sinal é tratado
- Opções:
  - Entregar o sinal para a thread a qual o sinal aplica-se
  - Entregar o sinal para toda thread no processo
  - Entregar o sinal para algumas threads no processo
  - Atribuir uma thread específica para receber todos sinais para o processo

Thread Safety

- Algumas funções, variáveis assumem execução single-threaded. Ex., errno, strtok(), malloc;
- Quando esta função é colocada em execução multi-threaded, nos deparamos com um problema chamado thread safety.



Convertendo o código monothread em código multithread

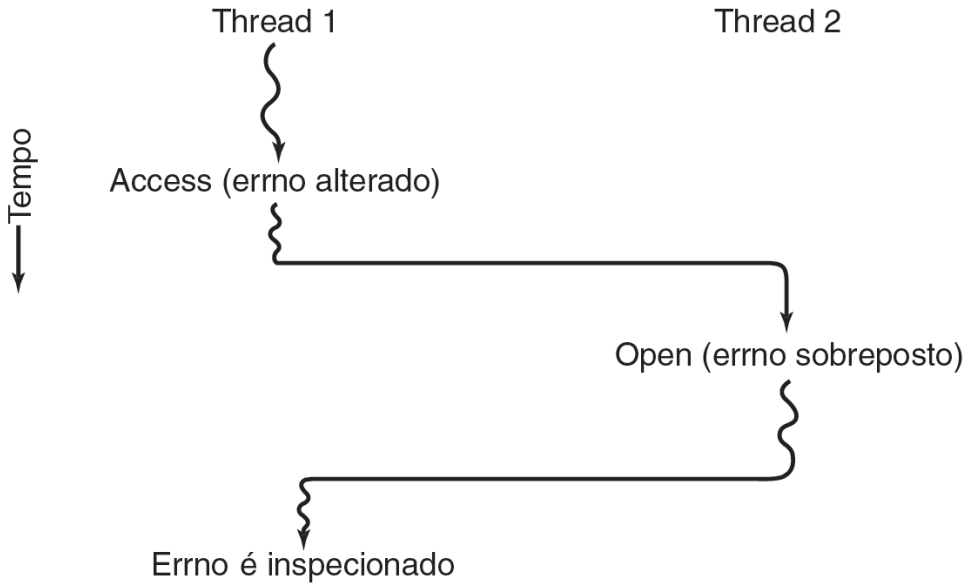


Figura 2.14 Conflitos entre threads sobre o uso de uma variável global.

Alternativas

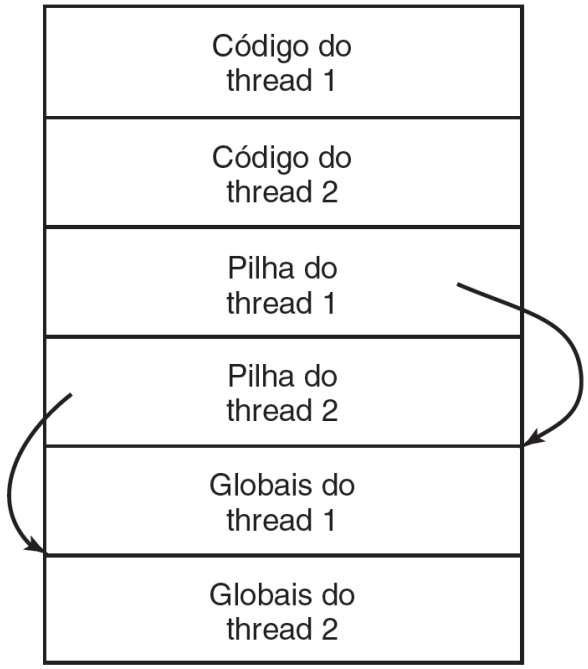


Figura 2.15 Threads podem ter variáveis globais individuais.

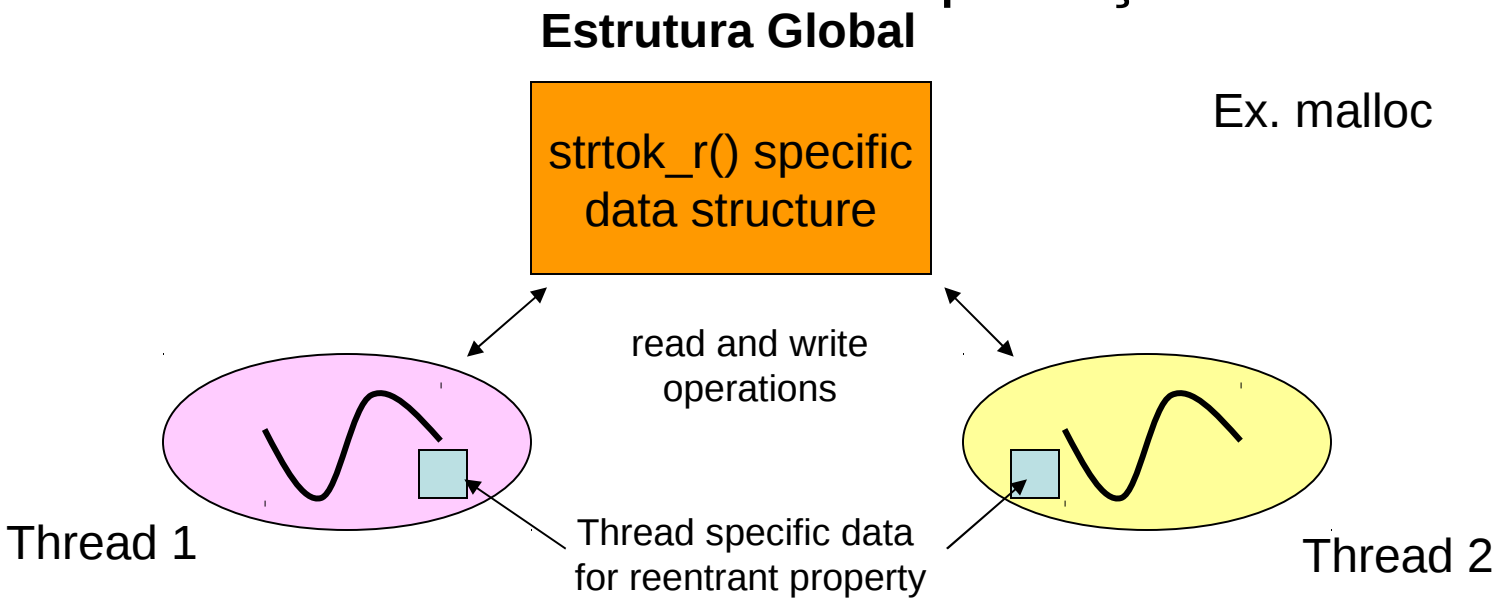
- Proibir variáveis Globais.
- Atribuir a cada thread suas próprias variáveis globais privadas.
- Não é trivial, linguagens conhecem variáveis

Thread Safety

Uma função *thread-safe* tem que implementar a propriedade reentrante .

A parte *thread-safe* strtok() : strtok_r().

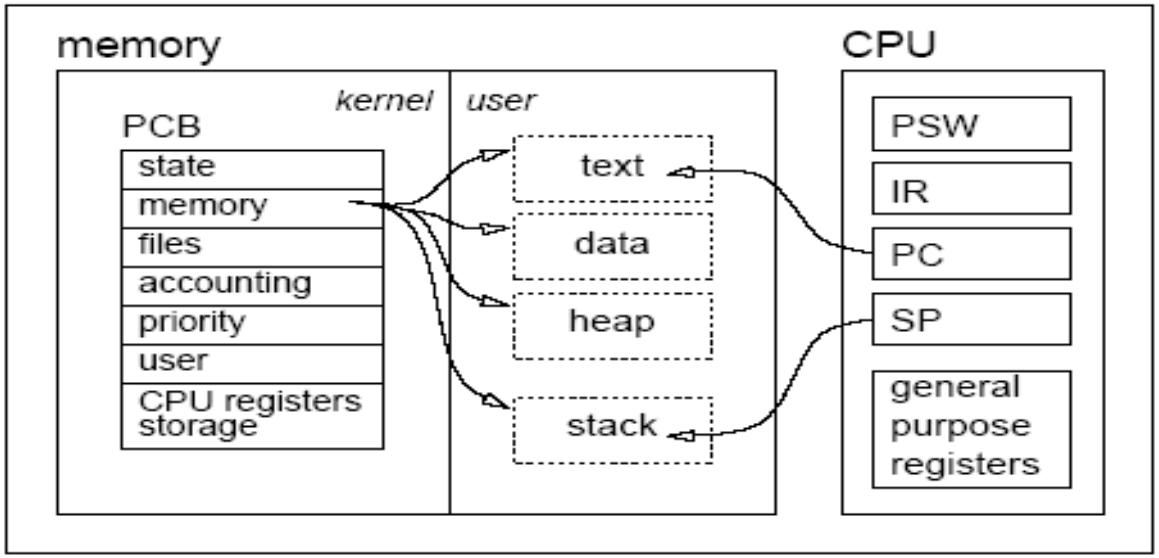
Alternativa: cada rotina tem sua proteção.



Gerenciamento da pilha

O que acontece quando ocorre transbordo da pilha de um processo?

Quando multithread, como o processo deve se comportar?



Programas Multi-Threaded

- POSIX thread – pthread.
- Básico :
  - Usar biblioteca: `#include <pthread.h>;`
  - Criar uma thread: `pthread_create();`
  - Terminar uma thread: `pthread_exit();`
  - Esperar o término de uma thread: `pthread_join();`

Penguin Specific

Reference: http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

Threads POSIX

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable

```
err = pthread_create( &tid, attr, function, arg );
```

Pthreads

- Um padrão POSIX (IEEE 1003.1c) API para criação de thread e sincronização.
  - http://www.humanfactor.com/pthreads/
  - http://www.cs.nmsu.edu/~jcook/Tools/pthreads/library.html
  - http://www.cs.wm.edu/wmpthreads.html
  - https://computing.llnl.gov/tutorials/pthreads/

Exemplo Pthread

Penguin Specific

```
void * hello(void *input) {
    printf("%s\n", (char *) input);
    pthread_exit(0);
}
```

Passing parameters

```
int main(void) {
    pthread_t tid;  // Thread type
    pthread_create(&tid, NULL, hello, "hello world");
    pthread_join(tid, NULL);    // just like wait();
    return 0;
}
```

Exemplo: programa C usando Pthread API

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

int sum;        /* Shared data */
void *runner(void *param);/* The Thread */

int main(int argc, char *argv[])
{
       pthread_t tid;   /* the thread identified */
       pthread_attr_t attr;     /* set of thread attributes */
   . . . . . .
       /* get the default attributes */
       pthread__attribute(&attr);
       /* create the thread */
       pthread_create(&tid, &attr, runner, argv[1]);
       /* wait for the thread to exit */
       pthread_join(tid, NULL);
 . . . . . .
}

void *runner(void *param)
{
       . . . .
}
```

Exclusão mútua em Pthread

- Tipo de dados:
  - pthread_mutex_t - semáforo binário.
- Operações:
  - pthread_mutex_lock() – uma operação atomica, similar a operação “down” (P).
  - pthread_mutex_unlock () – uma operação atomica, similar a operação “up” (V).

Sincronização em Pthread

- Tipo de dados
  - pthread_cond_t – a variável condição no monitor .
- Operações
  - pthread_cond_wait() – uma operação que suspende a thread chamadora na váriável condição especificada.
  - pthread_cond_signal() – uma operação que acorda a thread que foi suspensa em função da variável de condição especificada, escolhida randomicamente.
  - pthread_cond_broadcast() – uma operação que acorda todas threads que foram suspensas em função da variável de condição especificada.

Conclusão

- Thread é uma ferramenta conveniente para conseguir:
  - Multi-tasking no contexto de um processo;
  - Um ambiente de memoria compartilhada.
- Entretanto, é preciso cuidado com:
  - Exclusão Mútua;
  - Sincronização;
  - Thread safety.
- Conclusão: preciso muito cuidado na escrita programas multi-threaded e multi-processo.