Universidade Federal de Uberlândia
Faculdade de Computação
Sistemas Operacionais

Threads

Prof. Dr. Marcelo Zanchetta do Nascimento

Sumário

- Threads: Uma Visão Geral
- Benefícios
- Tipos
- Modelos de multithread
- POSIX Thread
- Exemplos em SO
- Leituras Sugeridas

Threads: Visão Geral

- Um processo é composto por contexto do processo, código, dados, *heap* e pilha.

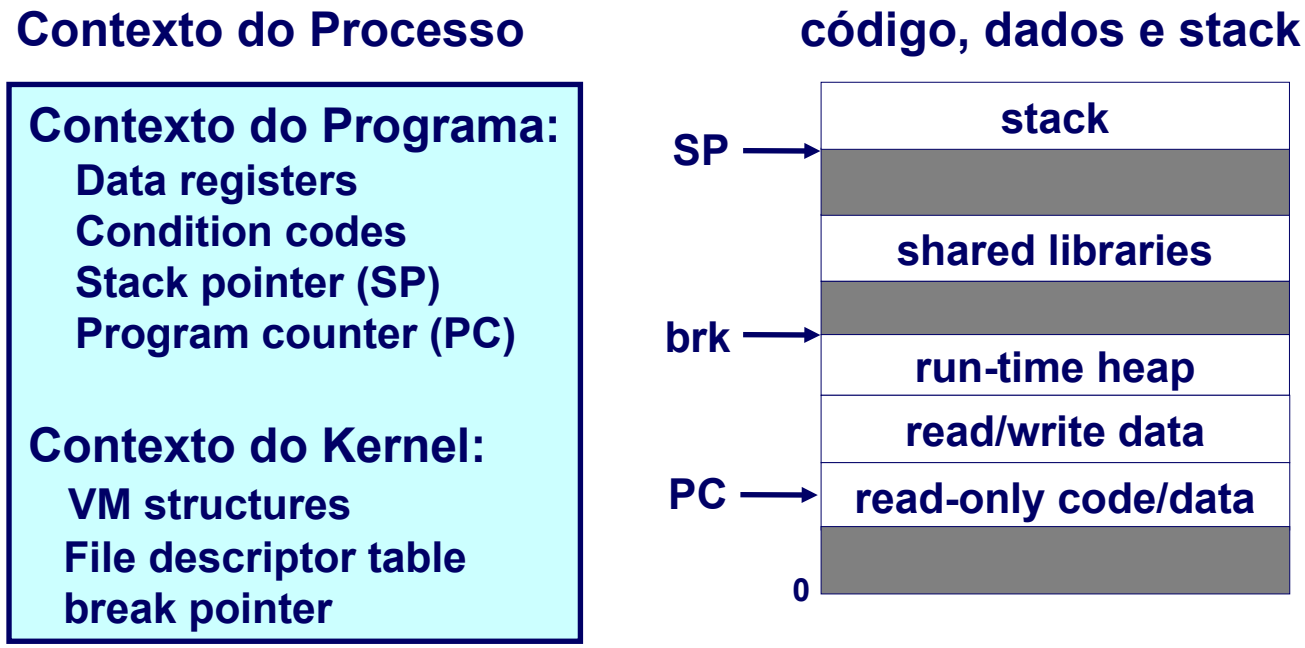


Figura 1 : Processo e a estrutura na memória principal

Processo: Visão Geral

- Estados do Processo

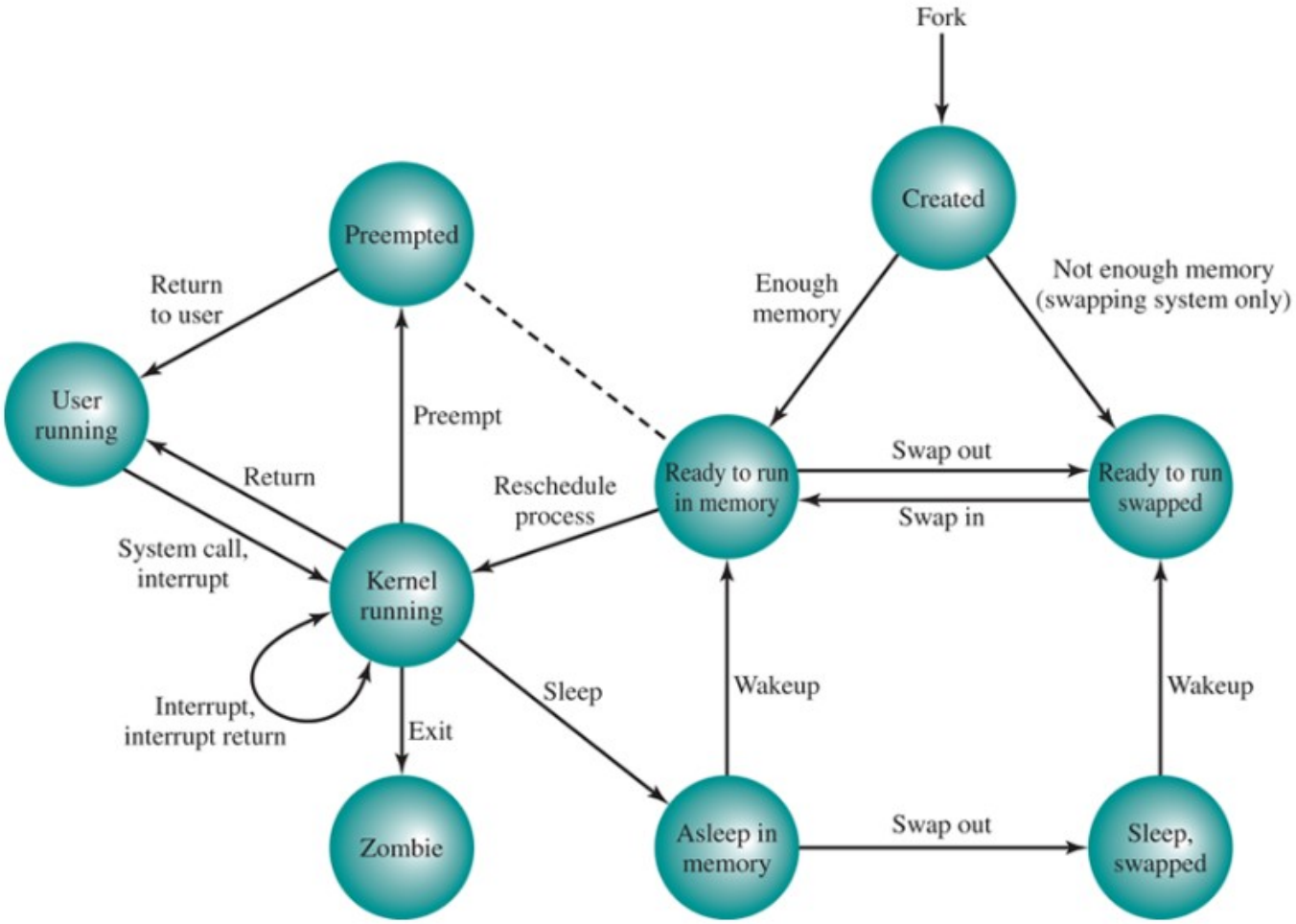


Figura: Diagrama de Transição de Processo UNIX

Threads: Visão Geral

- Processo demanda uma quantidade de sobrecarga para sua execução:

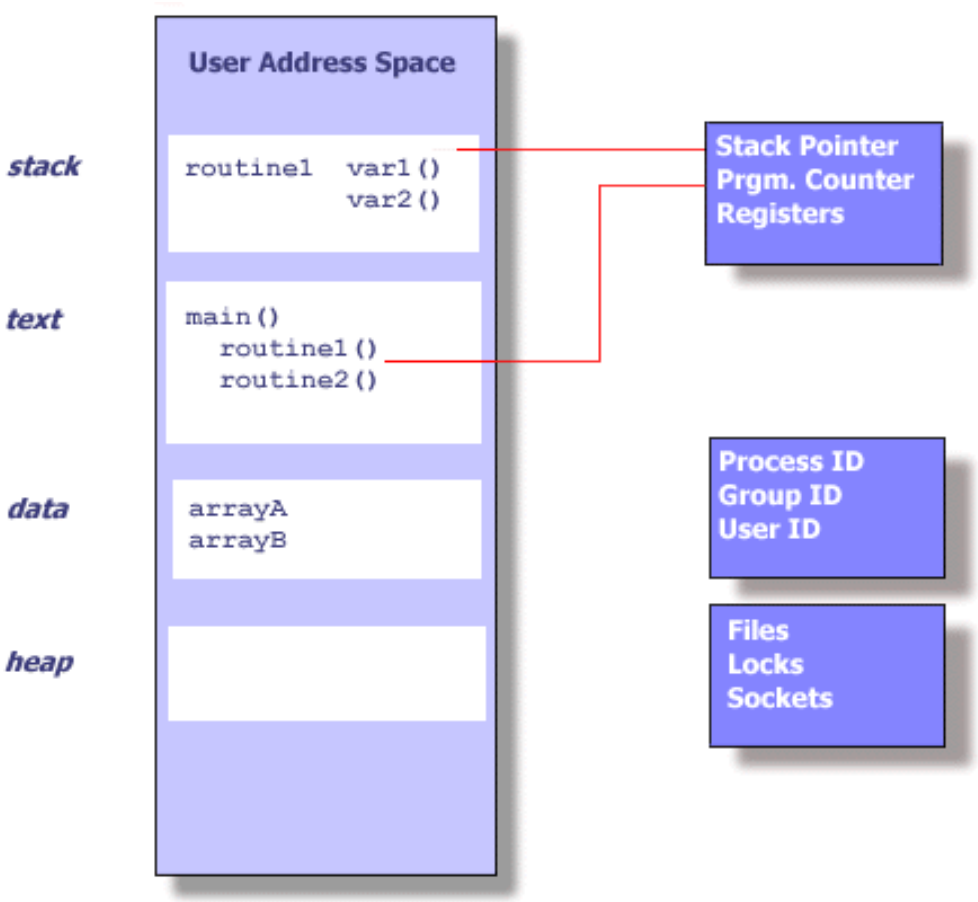


Figura 2 : Essa ação gera uma sobrecarga para execução pelo escalonador

Threads: Visão Geral

- O que ocorre quando há múltiplas ações ao mesmo tempo?
- Se tiver vários processos que estão executando de forma paralela. Como isso é tratado em um sistema com uma única CPU?
- No momento de finalizar um processo quais estruturas serão destruídas?
- Se houver uma estrutura mais leve, isso pode contribuir com o SO?

Threads

Processo leve (Lightweight Process – *Thread*)

- Ano de 1979, introduziram o conceito (o *thread*), onde o espaço de endereçamento era compartilhado;
- **Definição:** é uma unidade básica de utilização da CPU, que compreende um ID, um contador de programa, um conjunto de registradores e uma pilha;
- Um processo pode consistir de vários *threads,* cada uma executada de forma separada.

Visão alternativa de um processo

- Um processo pode conter thread, código, dados e contexto do kernel.

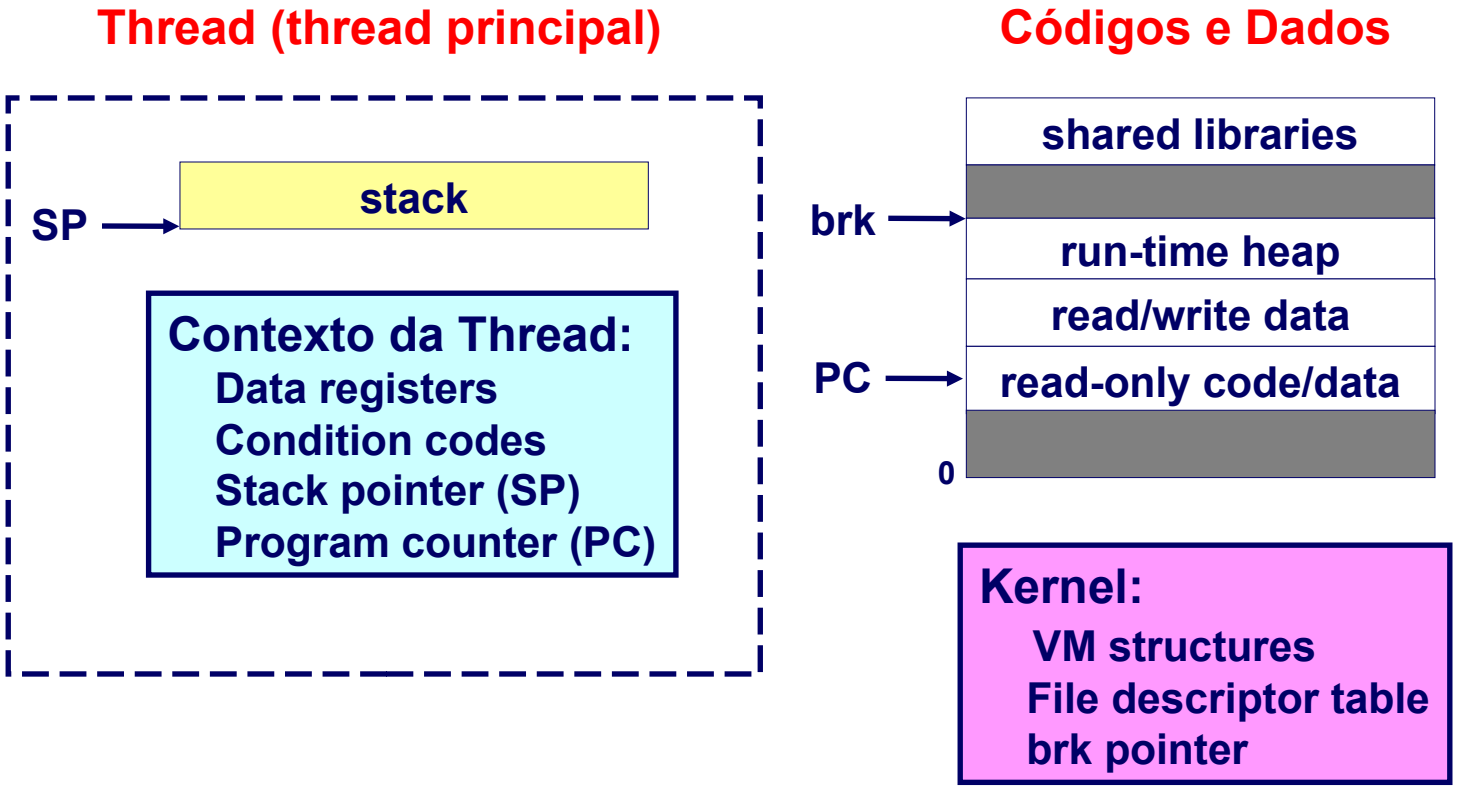


Figura 3 : Um processo com um thread

Um processo com múltiplas threads

Múltiplos threads podem ser associados com um processo

- Cada thread tem seu próprio controle de fluxo lógico
- Cada thread compartilha os dados e contexto do kernel
- Cada thread tem seu próprio ID (TID)

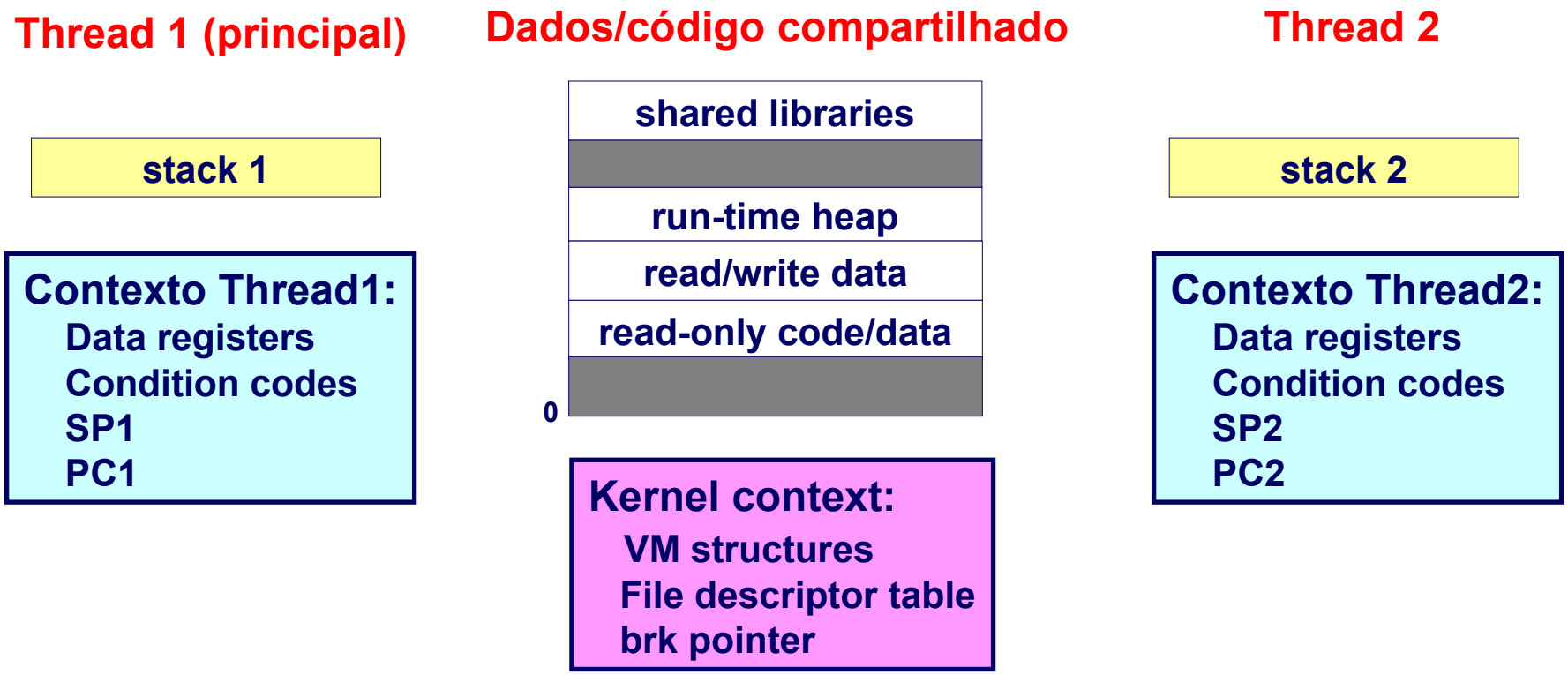


Figura 4 : Um processo com dois threads

Threads

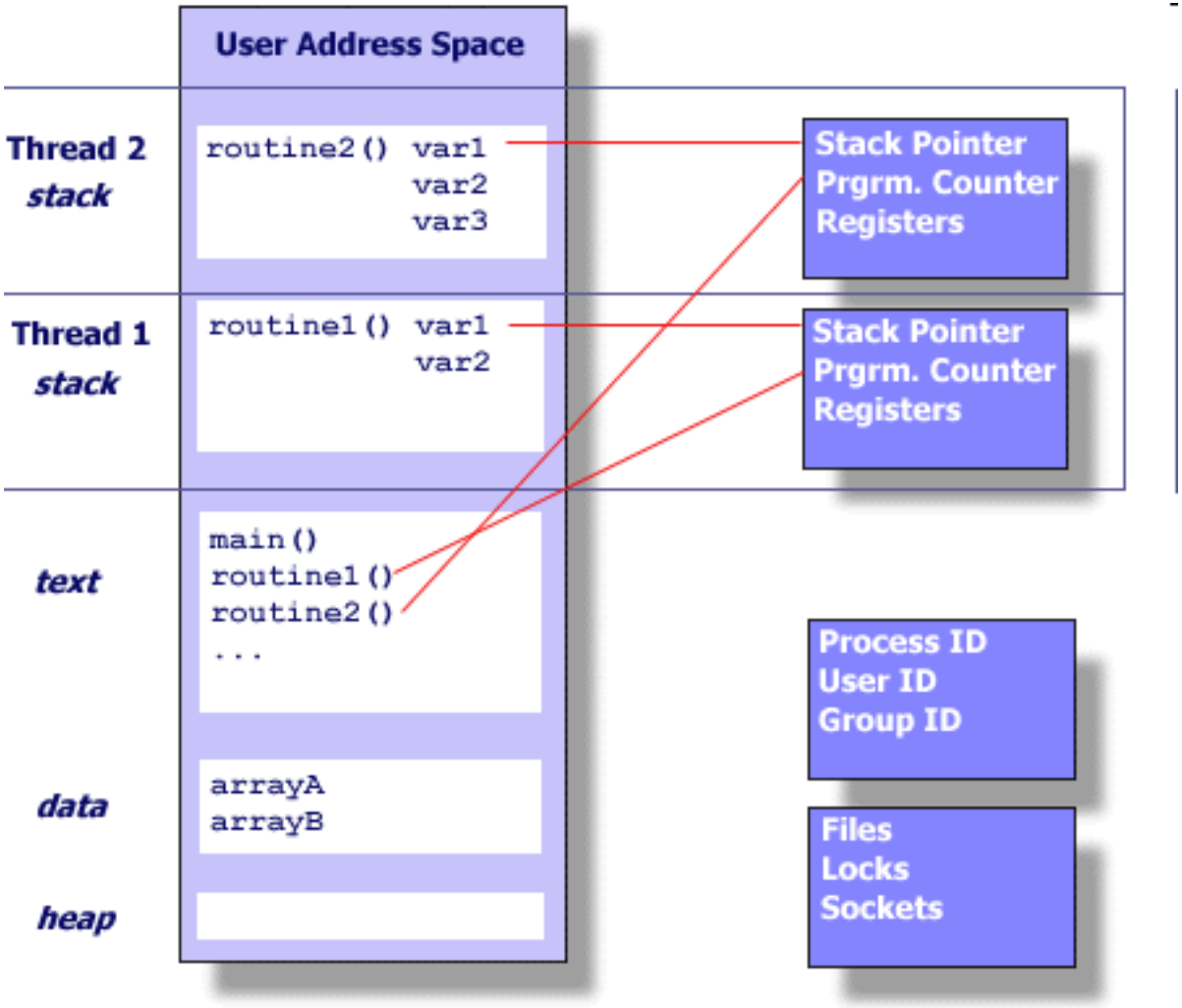


Figura 5 : Threads empregados em funções do programa.

Visão lógica dos threads

Threads associados com um processo forma um ***pool***

- Diferente do processo, o qual forma uma hierarquia

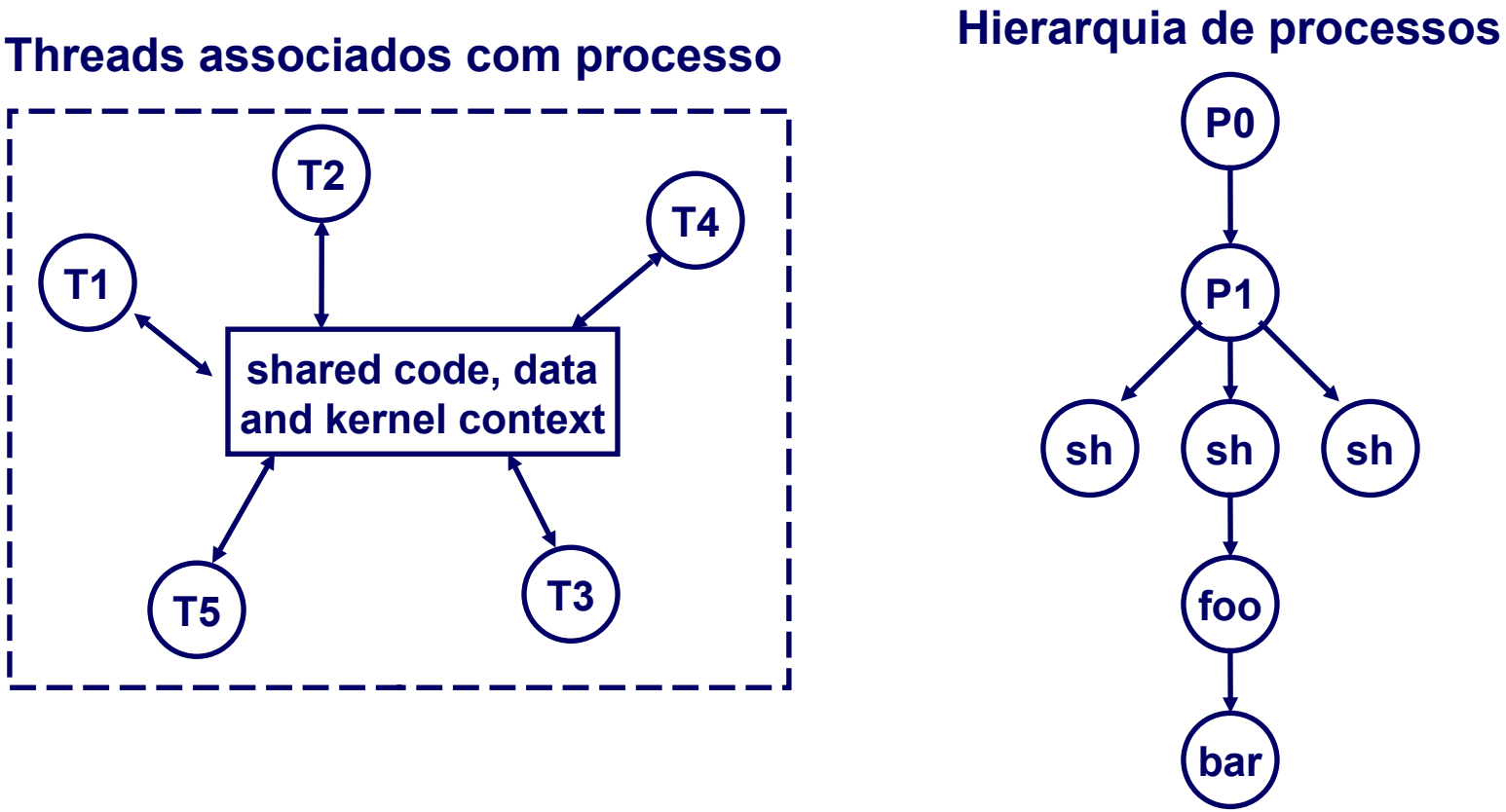


Figura 6 : Representação de Threads e Processo em S.O.

Threads

- O **thread** mantém as estruturas:
  - Pilha;
  - Registradores;
  - Propriedades de escalonamento (política ou prioridade);
  - Conjunto de sinais de bloqueio;
  - Dados específicos do thread.

- Threads pertencentes ao mesmo processo compartilham:
  - Seção de código;
  - Seção de dados;
  - Outros recursos do S.O., como por exemplo, os arquivos abertos.

Threads: Estados

- **Nova:** quando um novo processo é gerado, um thread é criada. Uma thread dentro de um processo pode gerar outra thread dentro do mesmo processo

- **Bloqueada:** uma thread precisa esperar por um evento, ela será bloqueada (salvando seus registros de usuário, contador de programa e ponteiros de pilha).

- **Desbloqueada:** Quando ocorre o evento para o qual um thread está bloqueado, o thread é movido para a fila Pronto.

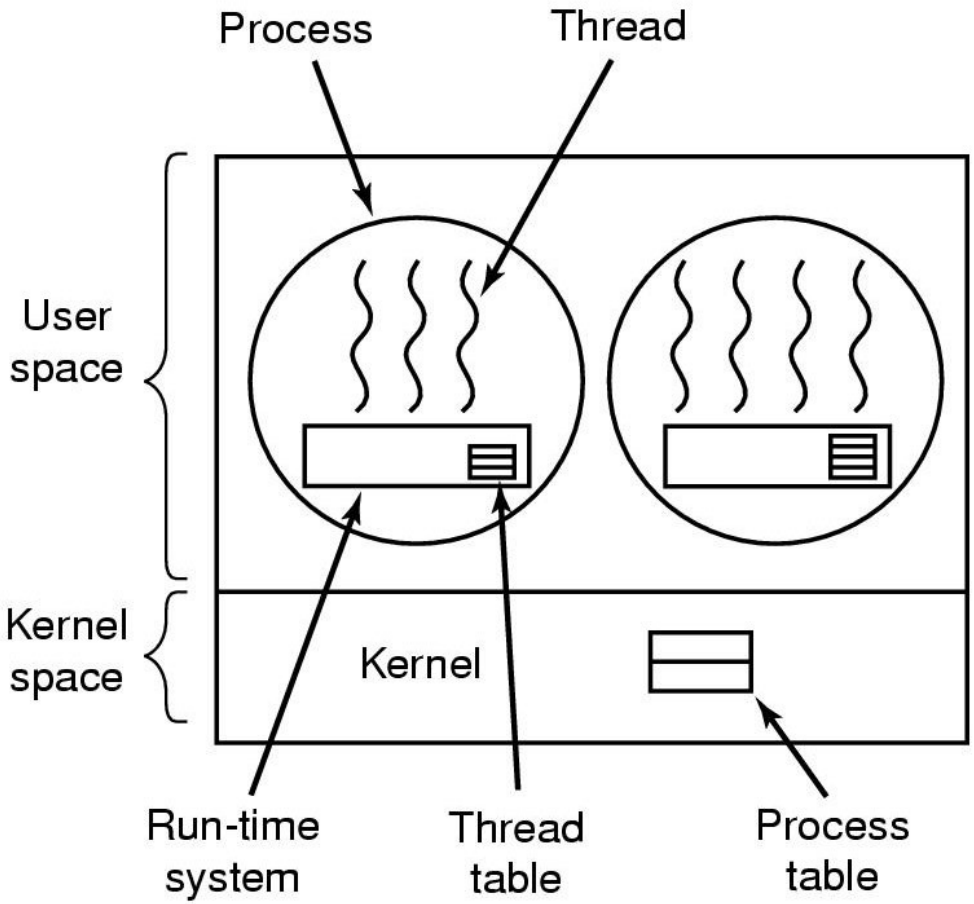
- **Finalizada:** quando um thread é concluído, seu contexto de registro e pilhas são desalocados.

Modelos de múltiplos threads

Threads de Usuário:

- Suportada acima do kernel e implementada por bibliotecas no **nível do usuário**;
- Criação e escalonamento são realizados sem o **conhecimento do *kernel***;
  - Tabela de threads para cada processo;
- Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;
  - Exemplo: ***Pthreads do POSIX (IEEE 1003.1c)***.

Modelos de múltiplos threads



Mapeia todos os threads de um processo multi-thread para um único contexto de execução

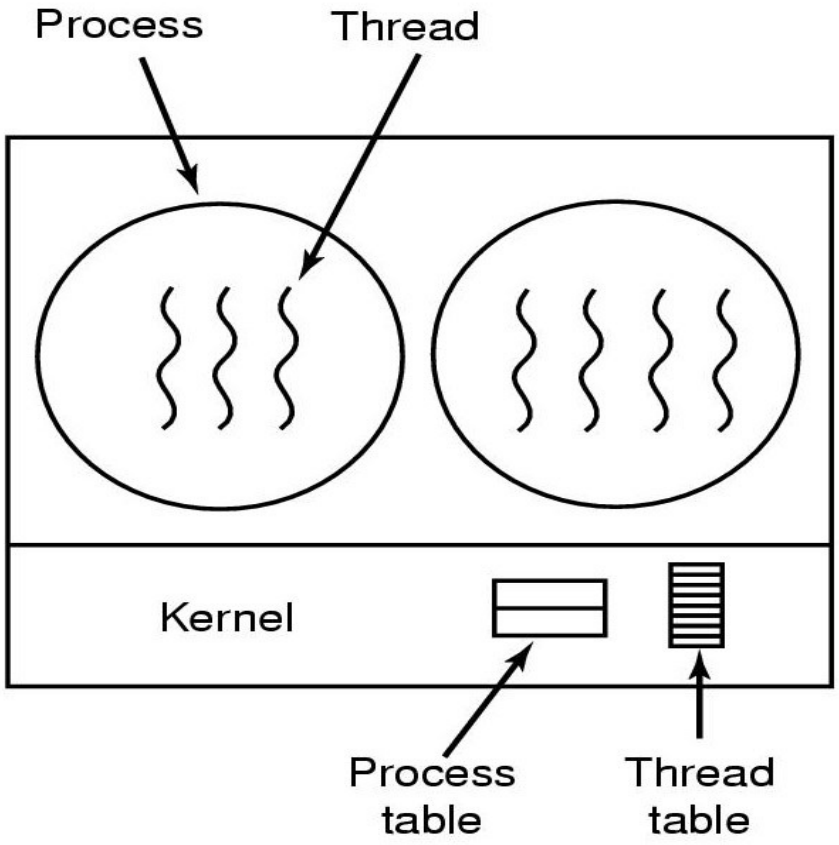
Figura 7 : Threads de usuário

Modelos de múltiplos threads

Threads de Kernel:

- Suportado diretamente pelo SO;
- Criação, escalonamento e gerenciamento feitos pelo *kernel*;
- **Trata de forma separada:**
  - Tabela de threads – informações dos threads de usuário;
  - Tabela de processos – informações de processos monothreads;
  - Processo não é bloqueado se um *thread* realizar uma chamada bloqueante ao sistema (E/S);
- Gerenciamento de threads de nível kernel é mais lento que de nível usuário:
  - Sinais enviados para os processos;

Modelos de múltiplos threads



Tenta resolver as limitações dos threads de usuário mapeando cada thread para seu próprio contexto de execução

Figura 8 : Threads administrados pelo Kernel

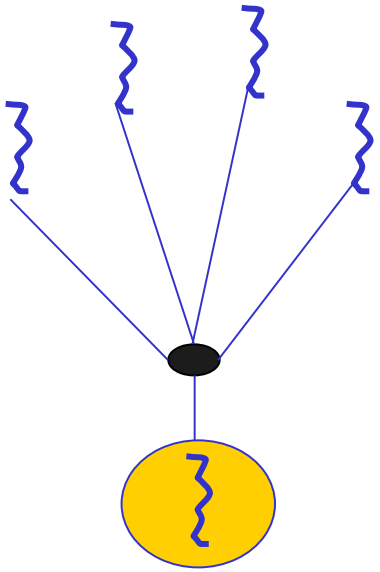
Threads - Kernel

- Quase todos os S.O.s modernos implementam esse modelo:
  - *Windows XP;*
  - *Solaris*
  - *Linux;*
  - *Tru64 UNIX;*
  - *Mac OS X.*

Modelo de multithreading

Muitos-para-um:

- Mapeia muitos threads de usuário em apenas uma thread de kernel (a gerência de threads é feita em nível usuário);
- Não permite múltiplos threads em paralelo;



Threads de usuário

Um thread de *kernel*

Exemplo: *Threads* escalonados em ambiente virtual em S.O. nativo.

Figura 9 : Muitos threads de usuário para um thread de nível kernel

Modelo de multithreading

Um-para-um:

- Mapeia cada *thread* de usuário em um *thread* de *kernel*;
- Permite múltiplos *threads* em paralelo;
- Desvantagem: exige que um thread de usuário crie um thread de kernel.

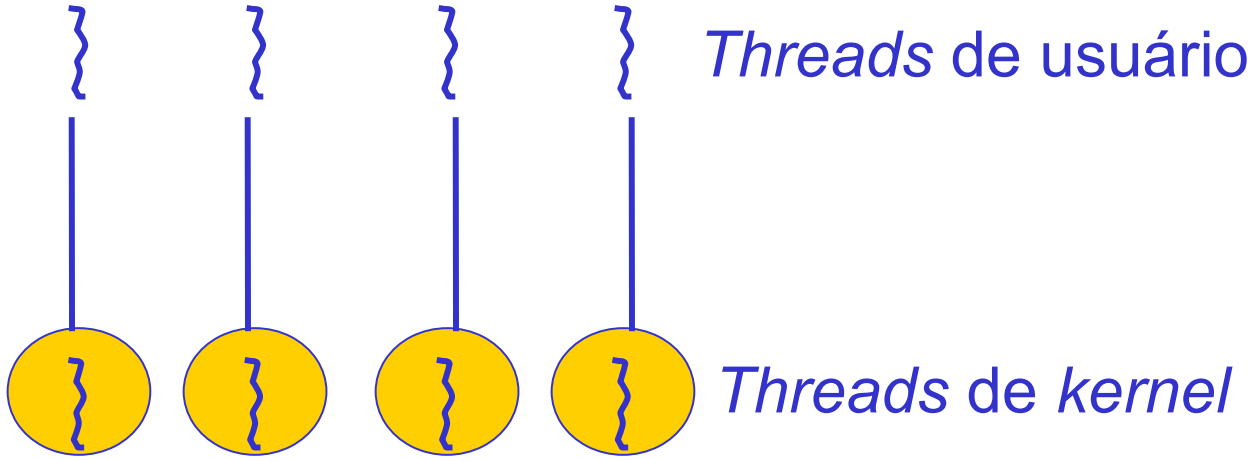
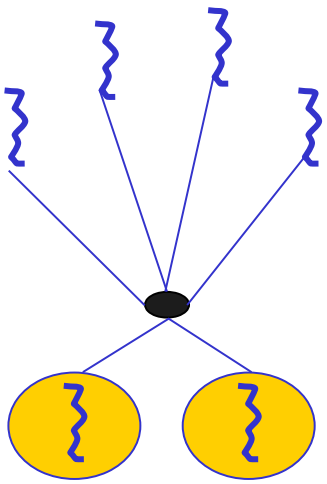


Figura 10 : Cada thread de usuário para cada thread de nível kernel

Modelo de multithreading

Muitos-para-muitos:

- Mapeia múltiplos *threads* de usuário em um número menor ou igual de *threads* de *kernel*;
- Permite múltiplos *threads* em paralelo;
- Modelo Hibrido: Muitos para muitos e thread usuário ligada diretamente a nível kernel.



Threads de usuário

Thread de *kernel*

Exemplo:

Solaris 2

Figura 11 : Muitos threads de usuário para muitos threads de nível kernel

Bibliotecas Threads

- **POSIX Threads (Biblioteca Pthreads)**
  - Define uma APIe implementada sobre o SO;
  - Utilizado em sistema UNIX: Linux, Mac OS X;
  - Padrão IEEE POSIX 1003.1c ;
- **Win 32 Threads**
  - Implementação do modelo um para um no kernel;
- **Java**
  - threads são gerenciadas pela JVM, a qual é executada sobre um SO (nativo);
  - JVM especifica a interface com SO;
  - Utiliza uma biblioteca de thread do S.O. hospedeiro.

POSIX Threads

Criação de threads:

```
int pthread_create(thread, attr, função, argumento)
```

- Esta função cria um novo thread e torna-o executável.
- A função retorna a identificação da thread através do parâmetro thread;
- attr: é usado para definir as propriedades do thread.
  - Pode-se especificar os atributos ou NULL para atributos padrões, o qual é criado pela função pthread_attr_init();
- função: é o nome da função que vai ser executada pelo thread;
- Pode-se passar um parâmetro para a função através de argumento.
- Se retornar 0 significa que foi criada com sucesso.

POSIX:

Junção de threads:

```
int pthread_join(pthread_t thread, void **status);
```

- A função pthread_join() espera pelo thread identificado;
- Se aquele thread já terminou, pthread_join() retorna imediatamente;
- Essa operação é denominada junção.
- O status contém um ponteiro para o argumento (status) passado pelo thread finalizado como parte da função pthread_exit().
  - Se o thread terminar com um retorno, o status contém o ponteiro para o valor retornado.

Terminar a execução de um thread.

```
pthread_exit(estado)
```

- Existem várias maneiras de um thread terminar:
  - Thread retorna da sua função inicial (a função main para o thread inicial).
  - Faz o status para qualquer thread que chama pthread_join() com o ID do thread finalizado.
  - Thread executa a função pthread_exit.
  - Thread foi cancelado por outro thread através da função pthread_cancel (rc = pthread_cancel(thread)).
  - Todo o processo foi terminado devido à utilização da função exec (que substitui a imagem do processo) ou através da função exit (que termina um processo).

POSIX:

Algumas das chamadas de sistema para threads

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

Acesse o link para mais detalhes:

https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ceeam00/cluice7.htm

- **POSIX - Exemplo:**

gcc -pthread -o term term.c

```
#include <pthread.h>   → Biblioteca para trabalhar com thread
#include <stdio.h>
int sum; /* compartilhado entre as threads*/
void *runner(void *param); /* a thread */

int main(int argc, char *argv[]){
  pthread_t tid; /* identificador da thread*/
  pthread_attr_t attr; /* atributos para a thread */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
```

- **POSIX Threads – Exemplo (1):**

```
 if (atoi(argv[1]) < 0) {
  fprintf(stderr,"Argumento %d deve ser não    negativo\
n",atoi(argv[1]));
  return -1;
  }
 /* recebe os atributos - ex. escalonamento */
 pthread_attr_init(&attr);
 /* cria a thread */
 pthread_create(&tid,&attr, runner, argv[1]);
 /* espera a thread parar finalizar */
 pthread_join(tid,NULL);        Sem estado atual->exit

 printf("soma= %d\n",sum);
}
```

- POSIX Threads – Exemplo (1):

```
/** A thread começa controlar essa função */
void *runner(void *param) {
 int i, upper = atoi(param);
 sum = 0;

 if (upper > 0) {
  for (i = 1; i <= upper; i++)
    sum += i;
 }

  pthread_exit(0);
}
```

- POSIX Threads – Exemplo (2):

```
void* SayHello(void *foo) {
  printf( "Hello, world!\n" );
  return NULL;
}

int main() {
  pthread_t threads[16];
  int tn;
  for(tn=0; tn<16; tn++) {
    pthread_create(&threads[tn], NULL, SayHello, NULL);
  }
  for(tn=0; tn<16 ; tn++) {
    pthread_join(threads[tn], NULL);
  }
  return 0;
}
```

Espera o thread terminar

Aspectos do uso de threads

Semântica do fork() e exec()

- fork( ) cria novo processo: sistemas UNIX podem ter duas versões de fork( ):
  - Uma que duplica todas as threads
  - Outra que duplica apenas a thread que invocou a chamada de sistema fork()
- fork( ) seguida de exec( ):
  - é feita apenas a duplicação da thread de chamada;
- fork( ) não seguido de exec( ):
  - é feita a duplicação de todas as threads;
- Chamada de sistema exec( ) isoladas:
  - em geral substituem o processo inteiro, incluindo todas as suas threads.

Tratamento de Sinais

- Sinais são usados nos sistemas UNIX para notificar um processo de que um evento específico ocorreu;
- Um sinal pode ser:
  - Síncrono: se forem liberados para o mesmo processo que provocou o sinal;
    - Exemplo: processo executa divisão por 0 e recebe sinal de notificação.
  - Assíncrono: se forem gerados por um evento externo (ou outro processo) e entregue a um processo;

Tratamento de Sinais

- Sinais para processos comuns
  - São liberados apenas para o processo específico (PID);

- Sinais para processos multithread: várias opções
  - Liberar o sinal para a thread conveniente (ex.: a que executou divisão por zero);
  - Liberar o sinal para todas as threads do processo (ex.: sinal para término do processo);
  - Liberar o sinal para determinadas threads;
  - Designar uma thread específica para receber todos os sinais.

Threads em Windows

- O Windows implementa a Windows API;
- Uma aplicação no Windows executa como um processo e cada processo pode conter um ou mais threads
- Geralmente, o thread tem
  - Um ID thread para identificar o thread
  - *Um conjunto registrados com status do processador*
  - *Um pilha usuário empregada em modo usuário e uma pilha kernel para modo kernel;*
  - *Uma area de armazenado privada (DLLs);*
    - Essas informações são conhecidas como **contexto** de uma thread.

Threads em Windows

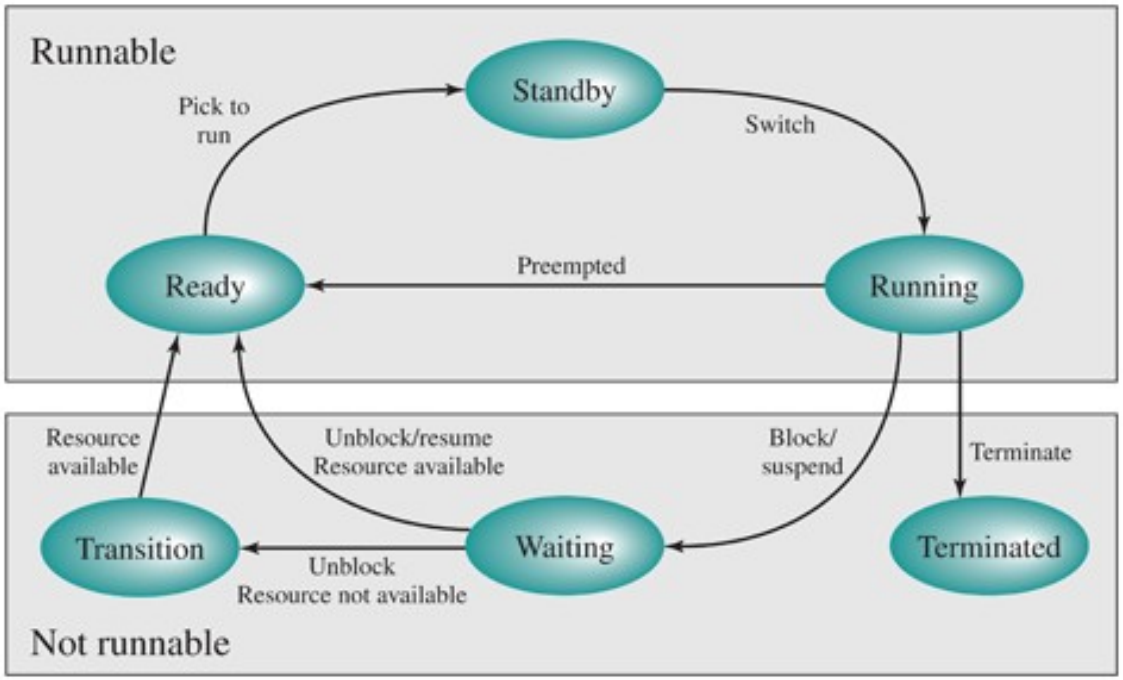


Figura: Os estados de uma thread no Windows

Threads em Linux

- O Linux refere-se aos threads como tarefas, em vez de threads;
- A criação de thread é feita através da chamada de sistema clone( );

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Threads / Processo Linux

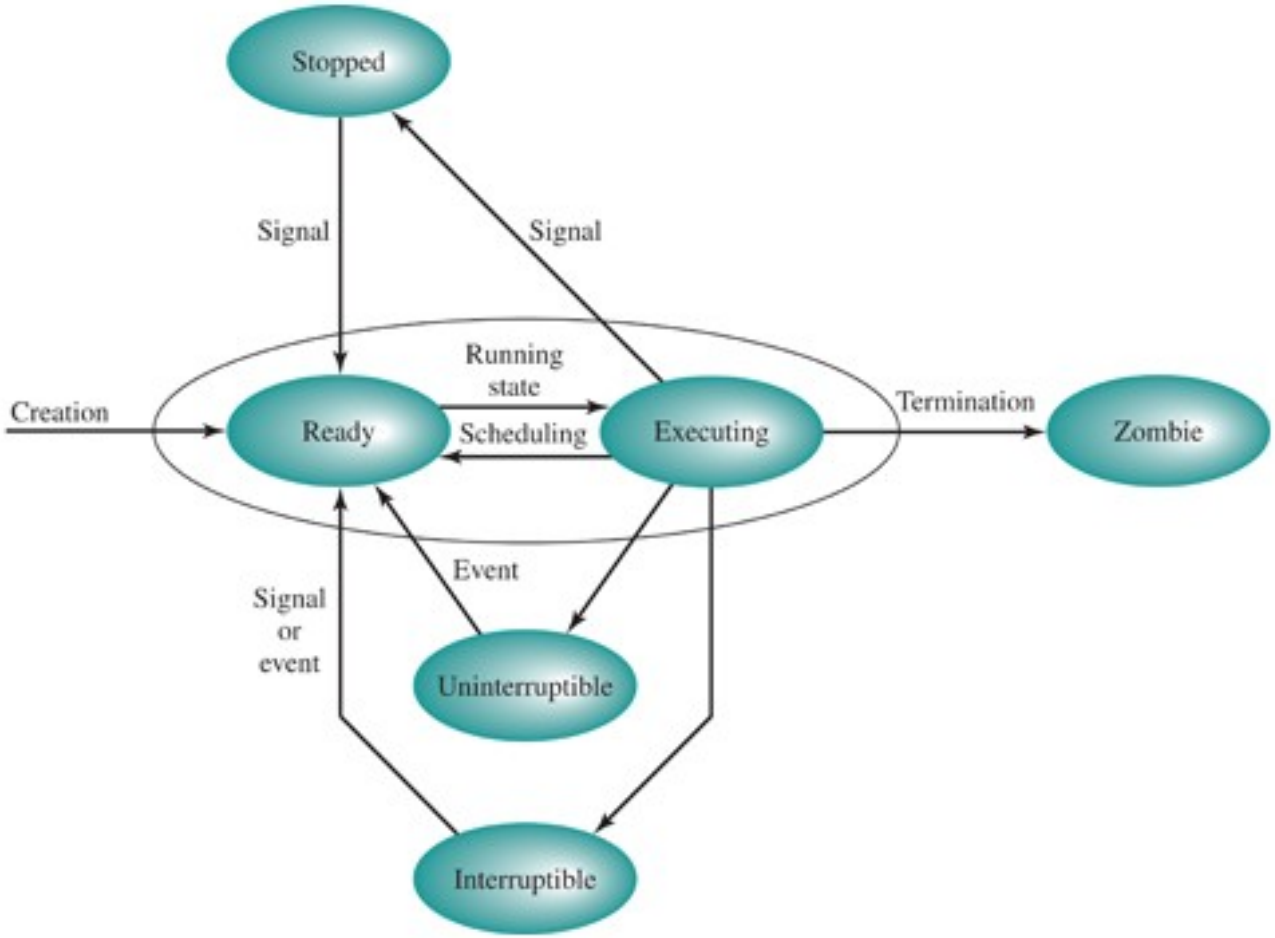
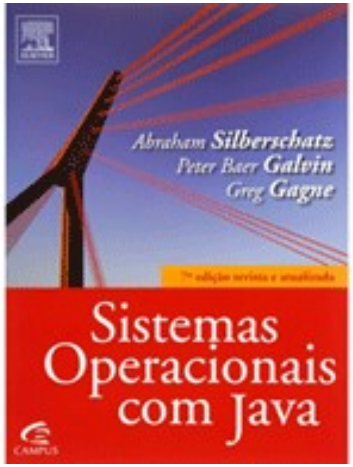


Figura: Os estados de uma tarefa no Linux

Leitura Sugerida

- Silberschatz, A., Galvin, P. B. Gagne, G. Sistemas Operacionais com Java. 7º , edição. Editora, Campus, 2008.
- Capítulo 4



- Andrew S. Tanembaum. **Sistemas Operacionais. Modernos**. 2ª Ed. Editora Pearson, 2003.
- Capítulo 2