Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Sistemas de Computação

SSC 140 - SISTEMAS OPERACIONAIS I
Turmas A e B

Aula 10 – Threads

Profa. Sarita Mazzini Bruschi

Slides de autoria de
Luciana A. F. Martimiano baseados no livro
Sistemas Operacionais Modernos de A. Tanenbaum

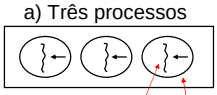
Processos

- Sistemas Operacionais tradicionais:
  - Cada processo tem um único espaço de endereçamento e um único fluxo de controle
- Existem situações onde é desejável ter múltiplos fluxos de controle compartilhando o mesmo espaço de endereçamento:
  - Solução: threads

Threads

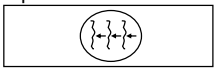
- Um processo tradicional (pesado) possui um contador de programas, um espaço de endereço e apenas uma *thread* de controle (ou fluxo de controle);
- ***Multithreading***: Sistemas atuais suportam múltiplas *threads* de controle, ou seja, pode fazer mais de uma tarefa ao mesmo tempo, servindo ao mesmo propósito;

a) Três processos



Thread Processo

b) Um processo com três *threads*

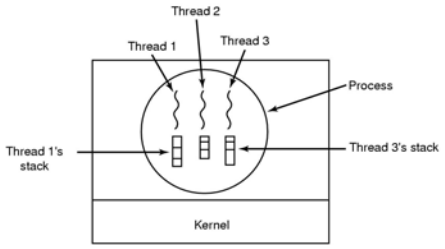


• As três *threads* utilizam o mesmo espaço de endereço

Threads

- *Thread* é uma entidade básica de utilização da CPU.
  - Também conhecidos como processos leves (*lightweight process* ou *LWP*);
- Processos com múltiplas *threads* podem realizar mais de uma tarefa de cada vez;
- Processos são usados para agrupar recursos; *threads* são as entidades escalonadas para execução na CPU
  - A CPU alterna entre as *threads* dando a impressão de que elas estão executando em paralelo;

Threads



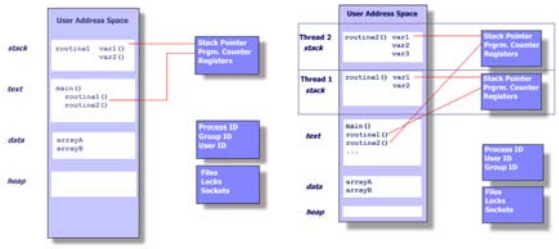
Cada *thread* tem sua pilha de execução

Threads

Itens por Processo	Itens por *Thread*
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores (contexto)
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	

- Compartilhamento de recursos;
- Cooperação para realização de tarefas;

Threads



Processo Unix Threads em um processo Unix

Threads

- Como cada *thread* pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo, uma *thread* pode ler, escrever ou apagar a pilha de outra *thread*;
- Não existe proteção pois:
  - É impossível
  - Não é necessário pois, diferente dos processos que podem pertecem a diferentes usuários, as threads são sempre de um mesmo usuáro

Threads

- Razões para existência de *threads*:
  - Em múltiplas aplicações ocorrem múltiplas atividades "ao mesmo tempo", e algumas dessas atividades podem bloquear de tempos em tempos;
  - As *threads* são mais fáceis de gerenciar do que processos, pois elas não possuem recursos próprios → o processo é que tem!
  - Desempenho: quando há grande quantidade de E/S, as threads permitem que essas atividades se sobreponham, acelerando a aplicação;
  - Paralelismo Real em sistemas com múltiplas CPUs.

Threads

- Considere um servidor de arquivos:
  - Recebe diversas requisições de leitura e escrita em arquivos e envia respostas a essas requisições;
  - Para melhorar o desempenho, o servidor mantém uma *cache* dos arquivos mais recentes, lendo da *cache* e escrevendo na *cache* quando possível;
  - Quando uma requisição é feita, uma *thread* é alocada para seu processamento. Suponha que essa *thread* seja bloqueada esperando uma transferência de arquivos. Nesse caso, outras *threads* podem continuar atendendo a outras requisições;

Threads

- Considere um navegador WEB:
  - Muitas páginas WEB contêm muitas figuras que devem ser mostradas assim que a página é carregada;
  - Para cada figura, o navegador deve estabelecer uma conexão separada com o servidor da página e requisitar a figura → tempo;
  - Com múltiplas *threads*, muitas imagens podem ser requisitadas ao mesmo tempo melhorando o desempenho;

Threads

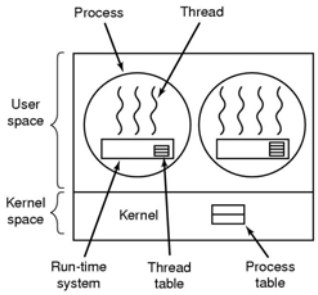
- Benefícios:
  - Capacidade de resposta: aplicações interativas; Ex.: servidor WEB;
  - Compartilhamento de recursos: mesmo endereçamento; memória, recursos;
  - Economia: criar e realizar chaveamento de *threads* é mais barato;
  - Utilização de arquiteturas multiprocessador: processamento paralelo;

Threads

- Tipos de *threads*:
  - Em modo usuário (espaço do usuário): implementadas por bibliotecas no espaço do usuário;
    - Criação e escalonamento são realizados sem o conhecimento do *kernel*;
      - Sistema Supervisor (*run-time system*): coleção de procedimentos que gerenciam as *threads*;
      - Tabela de *threads* para cada processo;
    - Cada processo possui sua própria tabela de *threads*, que armazena todas a informações referentes à cada *thread* relacionada àquele processo;

13

Threads em modo usuário



14

Threads em modo usuário

- Tipos de *threads*: Em modo usuário
- Vantagens:
  - Alternância de *threads* no nível do usuário é mais rápida do que alternância no *kernel*;
  - Menos chamadas ao *kernel* são realizadas;
  - Permite que cada processo possa ter seu próprio algoritmo de escalonamento;
  - Tem como vantagem poder ser implementado em Sistemas Operacionais que não têm threads
- Principal desvantagem:
  - Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

15

Implementação de threads

- Implementação em espaço de usuário:
  - Problemas:
    - Como permitir chamadas bloqueantes se as chamadas ao sistema são bloqueantes e essa chamada irá bloquear todas as threads?
      - Mudar a chamada ao sistema para não bloqueante, mas isso implica em alterar o SO -> não aconselhável
      - Verificar antes se uma determinada chamada irá bloquear a thread e, se for bloquear, não a executar, simplesmente mudando de thread
    - Page fault
      - Se uma thread causa uma page fault, o kernel, não sabendo da existência da thread, bloqueia o processo todo até que a página que está em falta seja buscada
    - Se uma thread não liberar a CPU voluntariamente, ela executa o quanto quiser
      - Uma thread pode não permitir que o processo escalonador do processo tenha sua vez

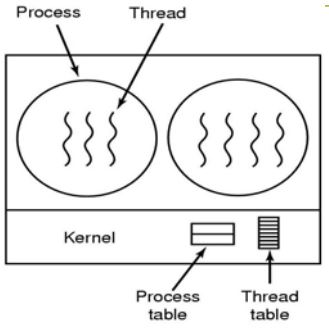
16

Tipos de *Threads*

- Tipos de *threads*:
  - Em modo *kernel*: suportadas diretamente pelo SO;
  - Criação, escalonamento e gerenciamento são feitos pelo *kernel*;
    - Tabela de *threads* e tabela de processos separadas;
      - as tabelas de *threads* possuem as mesmas informações que as tabelas de threads em modo usuário, só que agora estão implementadas no *kernel*;

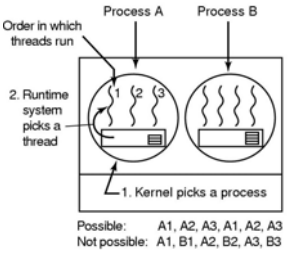
17

Threads em modo *kernel*

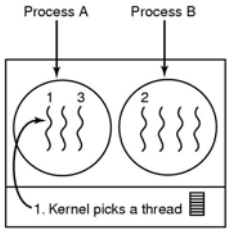


18

Threads em modo Usuário *x* *Threads* em modo *Kernel*



Threads em modo usuário



Threads em modo *kernel*

18

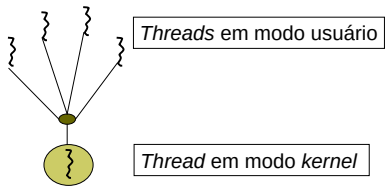
Threads em modo *kernel*

- Tipos de *threads*: em modo *kernel*
- Vantagem:
  - Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;
- Desvantagem:
  - Gerenciar threads em modo *kernel* é mais caro devido às chamadas de sistema durante a alternância entre modo usuário e modo *kernel*;

20

Threads

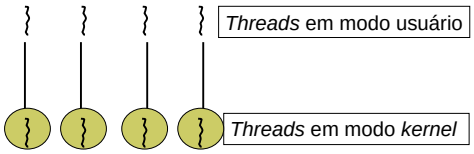
- Modelos *Multithreading*
  - Muitos-para-um:
    - Mapeia muitas *threads* de usuário em apenas uma *thread* de *kernel*;
    - Não permite múltiplas *threads* em paralelo;



21

Threads

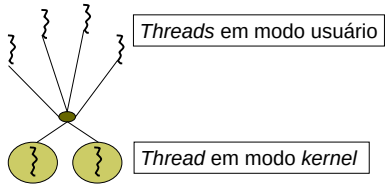
- Modelos *Multithreading*
  - Um-para-um: (Linux, Família Windows, OS/2, Solaris 9)
    - Mapeia para cada *thread* de usuário uma *thread* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



22

Threads

- Modelos *Multithreading*
  - Muitos-para-muitos: (Solaris até versão 8, HP-UX, Tru64 Unix, IRIX)
    - Mapeia para múltiplos *threads* de usuário um número menor ou igual de *threads* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



23

Threads

- Estados: executando, pronta, bloqueada;
- Comandos para manipular threads:
  - *Thread_create;*
  - *Thread_exit;*
  - *Thread_wait;*
  - *Thread_yield* (permite que uma *thread* desista voluntariamente da CPU);

24

Implementação

- Java
  - Classe *Threads*
  - A própria linguagem fornece suporte para a criação e o gerenciamento das *threads*, as quais são gerenciadas pela JVM e não por uma biblioteca do usuário ou do kernel.
- C
  - Biblioteca *Ptreads*
  - Padrão POSIX (IEEE 1003.1c) que define uma API para a criação e sincronismo de *threads;* não é uma implementação

25

Threads em Java
Primeira maneira de criação

- Criação da thread:
  - Definir uma nova classe derivada da classe Thread
  - Redefinir o método run()
- A definição dessa nova classe não cria a nova thread
  - A criação é feita através do método start()
    - Aloca memória e inicializa uma nova thread na JVM
    - Chama o método run(), tornando a thread elegível para ser executada pela JVM

26

Threads em Java
Exemplo

```
class Worker1 extends Thread
{
   public void run() {
         System.out.println("Eu sou uma thread criada");
   }
}

public class First
{
   public static void main(String args[]) {
         Thread runner = new Worker1();
         runner.start();
         System.out.println("Eu sou a thread principal");
   }
}
```

27

Threads em Java
Segunda maneira de criação

- Criação da thread
  - Definição de uma classe que implemente a interface Runnable.

```
public interface Runnable
{
   public abstract void run();
}
```

  - Definição de um método run()
- A implementação da classe Runnable é semelhante à extensão, exceto que “extends Thread” é substituído por “implements Runnable”

28

Threads em Java
Segunda maneira de criação

- Como a nova classe não estende Threads, ela não tem acesso aos métodos estáticos ou de instância, como por exemplo, o método start(), da classe Thread
- Porém, um método start() ainda é necessário, pois é ele que cria uma nova thread de controle
- Um novo objeto Thread deve ser criado, recebendo um objeto Runnable em seu construtor
- Quando a thread é criada com o método start(), a nova thread inicia a execução no método run() do objeto Runnable.

29

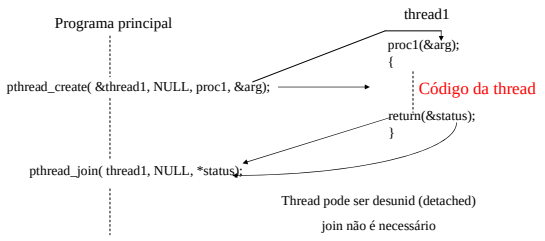
Threads em Java
Exemplo

```
class Worker2 implements Runnable
{
   public void run() {
         System.out.println("Eu sou uma thread criada");
   }
}

public class Second
{
   public static void main(String args[]) {
         Thread thrd = new Thread(new Worker2());
         thrd.start();
         System.out.println("Eu sou a thread principal");
   }
}
```

30

Threads em C
PThreads



Threads em C
PThreads

- pthread_create (thread,attr,start_routine,arg)
  - **thread**: identificador único para a nova *thread* retornada pela função.
  - **attr**: Um objeto que pode ser usado para definir os atributos (como por exemplo, prioridade de escalonamento) da *thread*. Quando não há atributos, define-se como NULL.
  - **start_routine**: A rotina em C que a *thread* irá executar quando for criada.
  - arg: Um argumento que pode ser passado para a *start_routine*. Deve ser passado por referência com um *casting* para um ponteiro do tipo void. Pode ser usado NULL se nenhum argumento for passado.

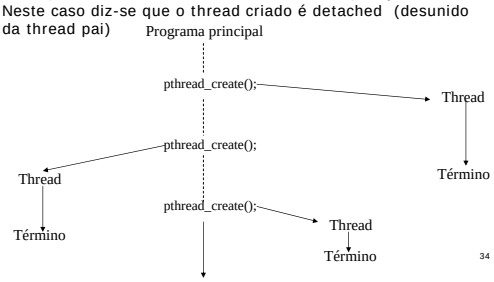
Threads em C
PThreads

- PThread Join
  - A rotina *pthread_join*() espera pelo término de uma thread específica

```
for (i = 0; i < n; i++)
    pthread_create(&thread[i], NULL, (void *) slave, (void *) &arg);
// código thread mestre
// código thread mestre
for (i = 0; i < n; i++)
    pthread_join(thread[i], NULL);
```


Threads em C
PThreads

- Detached Threads (desunidas)
  - Pode ser que uma thread não precisa saber do término de uma outra por ela criada, então não executará a operação de união. Neste caso diz-se que o thread criado é detached (desunido da thread pai)



Threads em C
PThreads

```
/******************************************************************************
* FILE: hello.c
* DESCRIPTION:
*   A "hello world" Pthreads program.  Demonstrates thread creation and
*   termination.
* AUTHOR: Blaise Barney
* LAST REVISED: 01/29/09
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}
```


Threads em C
PThreads

```
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc;
long t;
for(t=0;t<NUM_THREADS;t++){
  printf("In main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
pthread_exit(NULL);
}
```

Referências

- PThreads
  - https://computing.llnl.gov/tutorials/pthreads/

37