FATEC

CENTRO PAULA SOUZA
COMPETÊNCIA EM EDUCAÇÃO PÚBLICA PROFISSIONAL
Faculdade de Tecnologia Prof. Jessen Vidal
São José dos Campos

GOVERNO DO ESTADO DE SÃO PAULO

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Disciplina: Sistemas Operacionais I

Aula 11: Thread P2

Prof. Diogo Branquinho Ramos
diogo.branquinho@fatec.sp.gov.br
São José dos Campos - SP

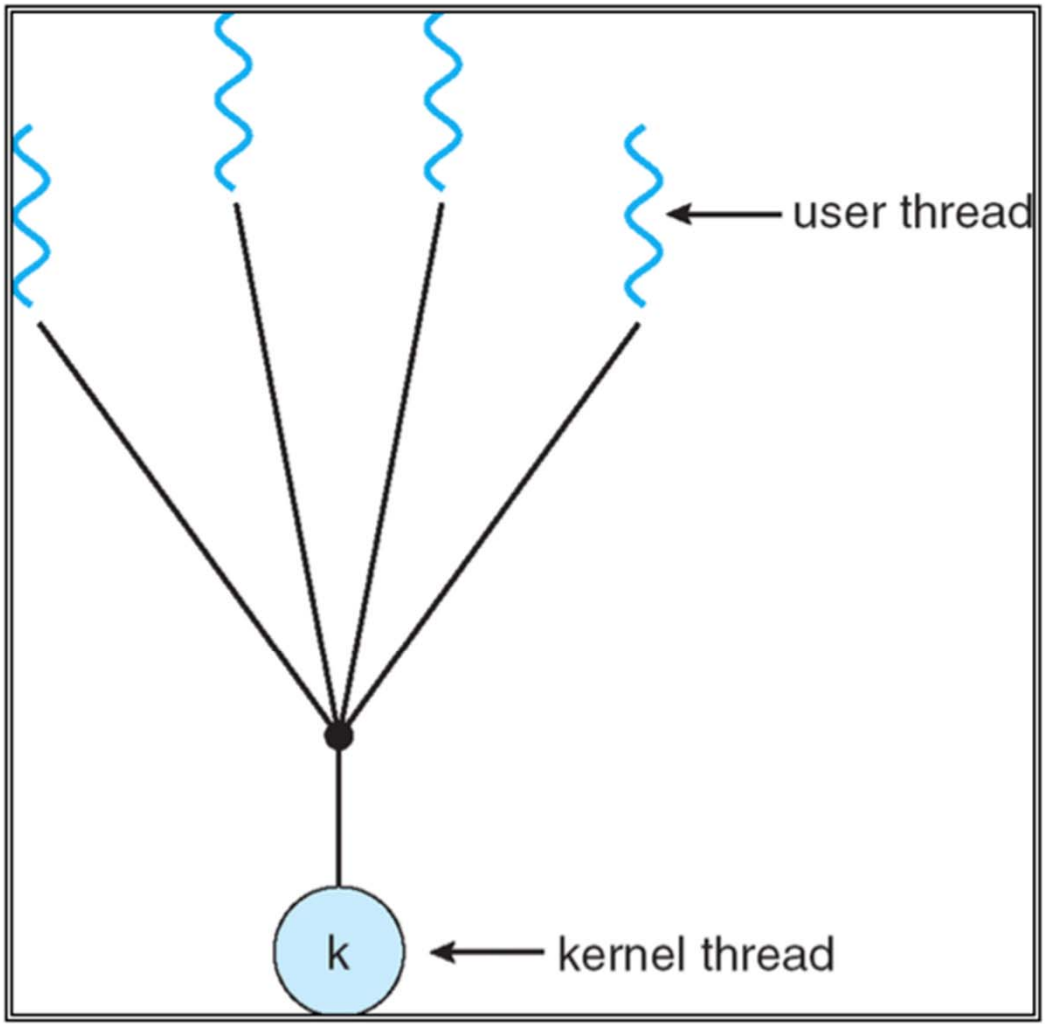
Roteiro

- Modelos de multithreading
- Bibliotecas de threads atuais
- Threads Java
- Algumas questões de threading

Modelo muitos-para-um

- **É preciso que haja um relacionamento entre os threads de cada usuário e os de kernel.**
- **Muitos threads em nível de usuário são mapeados para único thread do kernel.**
  - SO não reconhece threads.
- **Gerenciamento feito pela biblioteca de threads no espaço do usuário.**
  - O processo inteiro será bloqueado se um thread fizer uma syscall bloqueante.
- **Vários threads não podem executar em paralelo.**

Modelo muitos-para-um

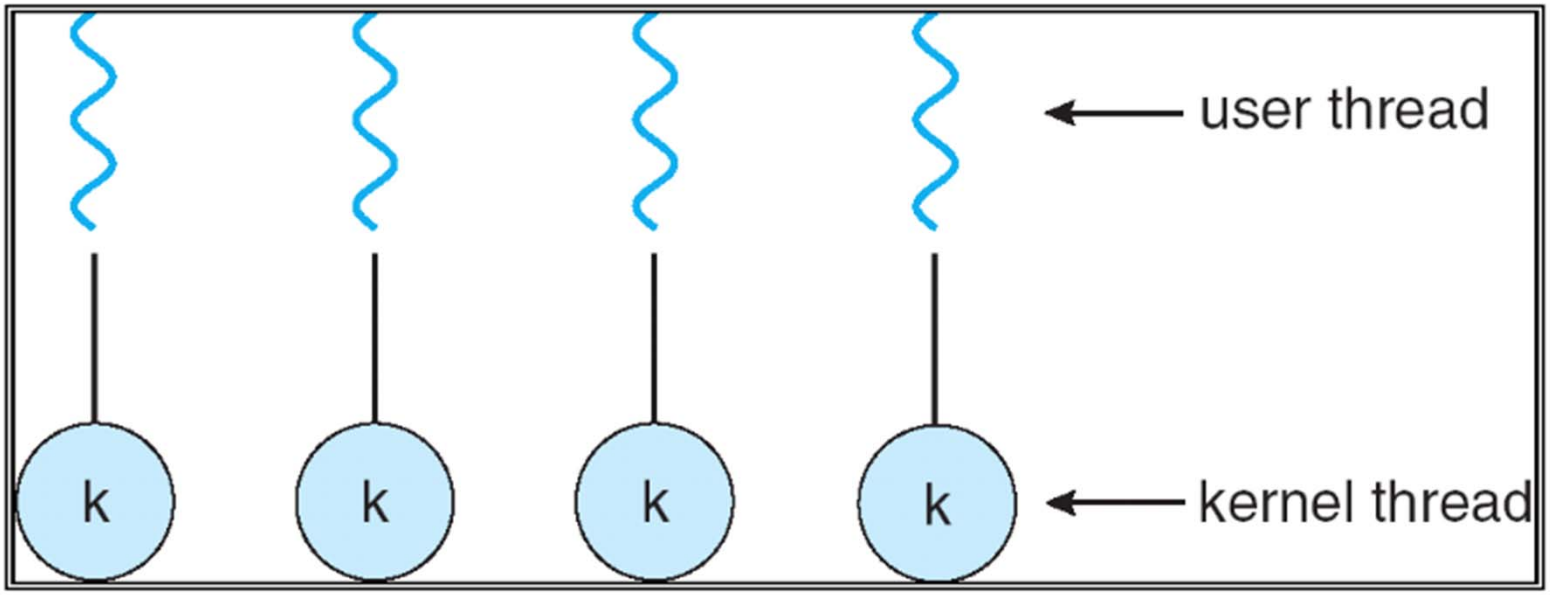


Exemplos:
Solaris Green Threads
GNU Portable Threads

Modelo um-para-um

- Cada thread em nível de usuário é mapeado para thread do kernel.
- Permite que outro thread execute quando um fizer uma syscall bloqueante.
- Provê maior paralelismo.
- Desvantagem: a cada novo thread de usuário, é preciso ter um thread de kernel → custo alto.

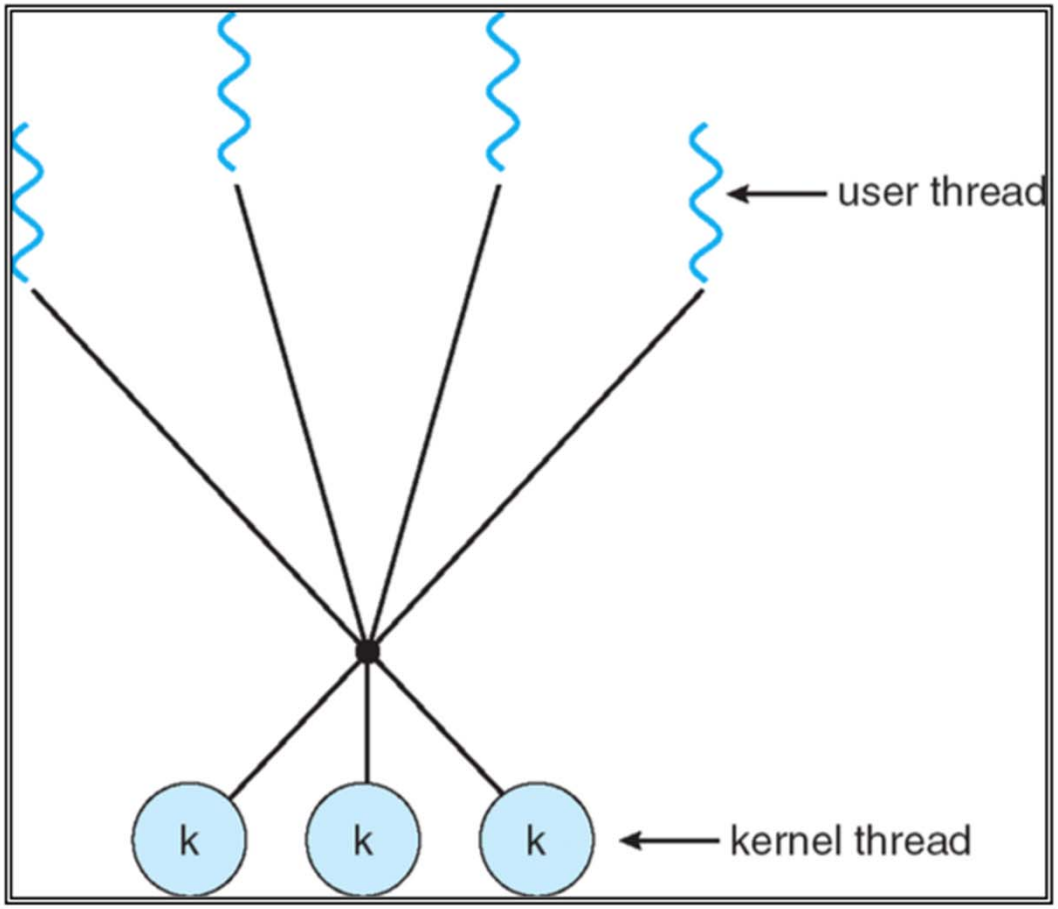
Modelo um-para-um



Modelo muitos-para-muitos

- **Permite que muitos threads em nível de usuário sejam mapeados para muitos threads do kernel (multiplexados).**
- **Permite que o SO crie um número suficiente de threads do kernel: libera o programador do limite de threads!**
- **O número de threads de kernel pode ser específico a determinada aplicação ou máquina.**
  - Monoprocessado x multiprocessado.

Modelo muitos-para-muitos



Exemplos:
Windows NT/2000 com ThreadFiber.
Linux com bibliotecas Pthreads.

Bibliotecas threads

- **Fornecem ao programador uma API para a criação e gerenciamento de threads.**
- **Técnicas**
  - Fornecer uma biblioteca inteiramente no espaço do usuário, sem suporte do kernel.
    - Chamada de função local no espaço do usuário.
  - Implementar uma biblioteca no nível do kernel, com suporte direto do SO.
    - Todas as estruturas estão no espaço do kernel. A chamada de uma função na API para a biblioteca resulta em uma syscall.

Bibliotecas threads atuais

- **Pthreads**
  - Definição
    - API padrão POSIX (IEEE 1003.1c) para thread.
  - Comum em SOs UNIX (Solaris, Linux, Mac OS X).
  - Pode ser fornecida como biblioteca no nível do usuário ou do kernel.
  - Possui cerca de 100 rotinas que manipulam threads.
    - As funções são prefixadas por "pthread_".
- **Pthreads-w32**
  - Biblioteca no nível do kernel disponível para Windows, já que pthreads não são nativos para o Windows.
  - Compatível com Windows 64-bit a partir da versão 2.8.0.

Exemplo pthreads

```
#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);
}
```

Bibliotecas threads atuais

- **Win32 API**
  - Suporta 16, 32 e 64-bit.
  - Provê funcionalidades para o Windows:
    - Serviços Básicos: sistema de arquivo, threads, processos, etc;
    - Serviços Avançados: registry, desligar/reiniciar o sistema, etc;
    - Dispositivo de Interface Gráfica;
    - Interface de Usuário;
    - Biblioteca de Caixa de Diálogo Comum;
    - Biblioteca Comum de Controle;
    - Shell;
    - Serviços de Rede: Winsock, RPC, etc.

Threads Java

- **Java**
  - Threads Java são gerenciados pela JVM.
  - Todo programa Java possui pelo menos um thread de controle.
- **Como criar threads Java**
  - Criar uma nova classe derivada da classe Thread e redefinir o método run(); ou
  - Implementando a interface Runnable (comum da API do Java) e definindo um método run() (código executado na thread).

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

Threads Java: exemplo

```
class MutableInteger
{
  private int value;
  public int getValue() {
   return value;
  }
  public void setValue(int value) {
   this.value = value;
  }
}

class Summation implements Runnable
{
  private int upper;
  private MutableInteger sumValue;
  public Summation(int upper, MutableInteger sumValue) {
   this.upper = upper;
   this.sumValue = sumValue;
  }
  public void run() {
   int sum = 0;
   for (int i = 0; i <= upper; i++)
      sum += i;
   sumValue.setValue(sum);
  }
}
```

Thread separada

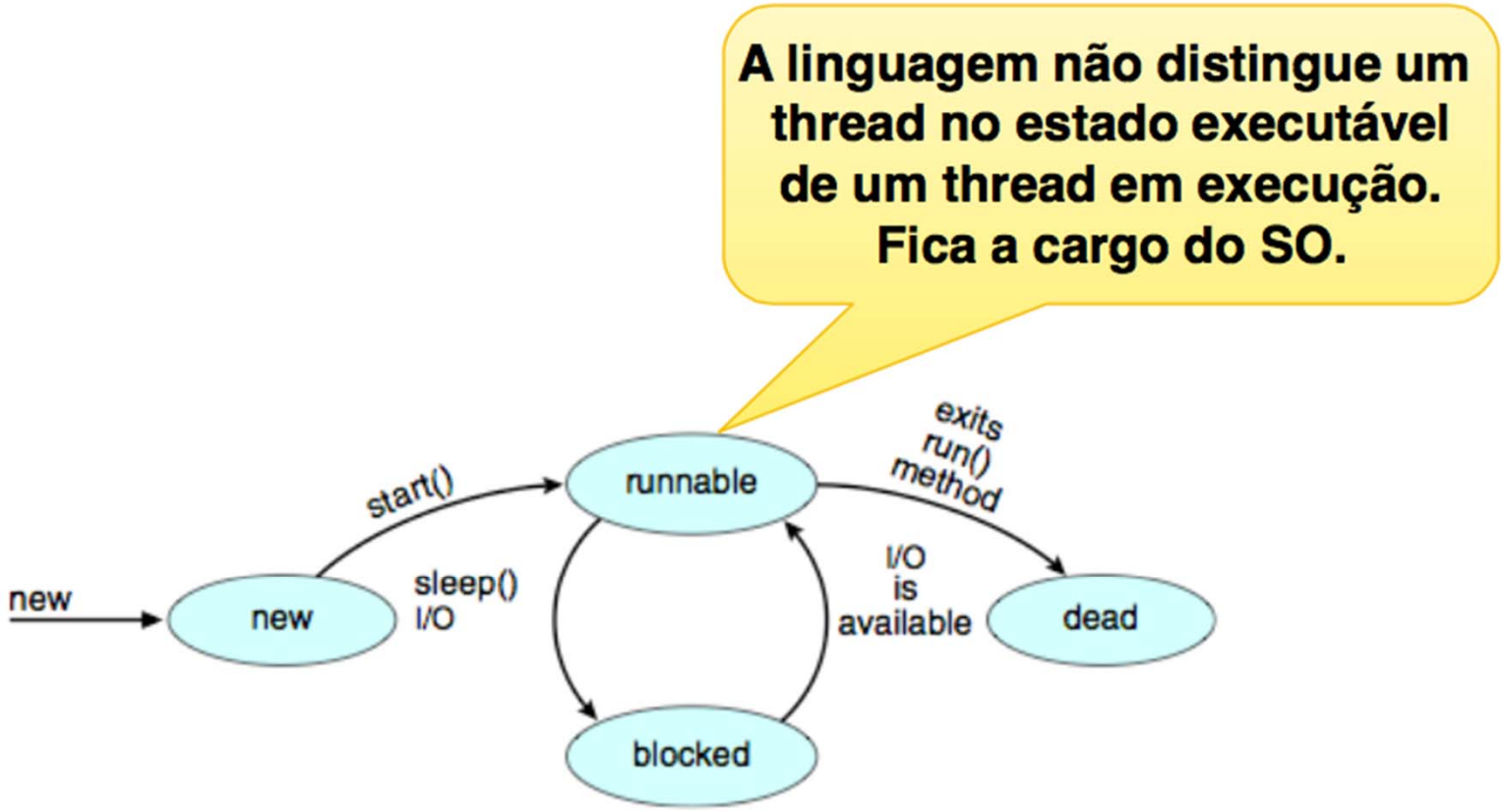
Threads Java: exemplo

```
public class Driver
{
  public static void main(String[] args) {
    if (args.length > 0) {
      if (Integer.parseInt(args[0]) < 0)
        System.err.println(args[0] + " must be >= 0.");
      else {
        // create the object to be shared
        MutableInteger sum = new MutableInteger();
        int upper = Integer.parseInt(args[0]);
        Thread thrd = new Thread(new Summation(upper, sum));
        thrd.start();
        try {
          thrd.join();
          System.out.println
                  ("The sum of "+upper+" is "+sum.getValue());
        } catch (InterruptedException ie) { }
      }
    }
    else
      System.err.println("Usage: Summation <integer value>");
    }
}
```

Não é a criação do objeto que cria o thread! Aqui é indicada a criação, na obtenção de um objeto da classe Thread com um objeto Runnable como parâmetro no construtor.

O método start() aloca memória, inicializa um novo thread na JVM e invoca o método run(), tornando o thread elegível para ser executado pela JVM.

Estados thread - Java



A JVM e o SO hospedeiro

- **A JVM abstrai a aplicação dos detalhes da implementação do SO subjacente.**
  - A JVM implementa especificamente as associações de threads de usuário e de kernel.
- **Mapeamentos em bibliotecas tradicionais**
  - Windows 2000 (Win32): um-para-um;
  - Tru64 UNIX (Pthreads): muitos-para-muitos.
- **Relacionamento entre bibliotecas threads Java e as do SO.**
  - A JVM sobre um Windows: usa API Win32.
  - A JVM sobre um Linux: usa API Pthreads.

Threads Java: Produtor-Consumidor

```
class Producer implements Runnable
{
   private Channel mbox;

   public Producer(Channel mbox) {
      this.mbox = mbox;
   }

   public void run() {
      Date message;

      while (true) {
         // nap for awhile
         SleepUtilities.nap();

         // produce an item and enter it into the buffer
         message = new Date();

         System.out.println("Producer produced " + message);
         mbox.send(message);
      }
   }
}
```

Threads Java: Produtor-Consumidor

```
class Consumer implements Runnable
{
   private Channel mbox;

   public Consumer(Channel mbox) {
      this.mbox = mbox;
   }

   public void run() {
      Date message;

      while (true) {
         // nap for awhile
         SleepUtilities.nap();

         // consume an item from the buffer
         message = (Date)mbox.receive();

         if (message != null)
            System.out.println("Consumer consumed " + message);
      }
   }
}
```

Threads Java: Produtor-Consumidor

```
public class Factory
{
    public Factory() {
       // First create the message buffer.
       Channel mailBox = new MessageQueue();

       // Create the producer and consumer threads and pass
       // each thread a reference to the mailBox object.
       Thread producerThread = new Thread(
         new Producer(mailBox));
       Thread consumerThread = new Thread(
         new Consumer(mailBox));

       // Start the threads.
       producerThread.start();
       consumerThread.start();
    }

    public static void main(String args[]) {
       Factory server = new Factory();
    }
}
```

Algumas questões de threading

Semântica das syscalls fork() e exec()

- **fork() duplica apenas o thread que chama ou todos os threads?**
  - Se exec() for chamada imediatamente em seguida, então duplicar todos os threads é desnecessário.
    - Alguns sistemas fornecem duas versões de fork(): fork puro e fork-com-exec.
    - Qual versão será utilizada depende da aplicação.

Algumas questões de threading

Cancelamento de thread

- **Terminando um thread antes que ele tenha sido concluído: exemplo de threads consultando um BD.**
- **Duas técnicas gerais:**
  - O cancelamento assíncrono termina o thread de destino imediatamente.
    - Pode gerar problemas: cancelamento do thread enquanto atualiza dados compartilhados com outros threads.

```
Thread thrd = new Thread(new InterruptibleThread());
thrd.start();
. . .
thrd.interrupt();
```

Algumas questões de threading

Cancelamento de thread

- **Duas técnicas gerais (cont.):**
  - O cancelamento adiado permite que o thread de destino verifique periodicamente se ele deve ser cancelado: término controlado.
  - Para isso, são necessários pontos de cancelamento
    - Sinalizador verificado pelo thread periodicamente (em um ponto seguro de execução!) que indique se o thread deve ser cancelado: **isInterrupted()**.

Algumas questões de threading

Cancelamento de thread: adiado

```
class InterruptibleThread implements Runnable
{
   /**
    * This thread will continue to run as long
    * as it is not interrupted.
    */
   public void run() {
      while (true) {
         /**
          * do some work for awhile
          * . . .
          */

         if (Thread.currentThread().isInterrupted()) {
            System.out.println("I'm interrupted!");
            break;
         }
      }
      // clean up and terminate
   }
}
```