Capitulo 4: Threads

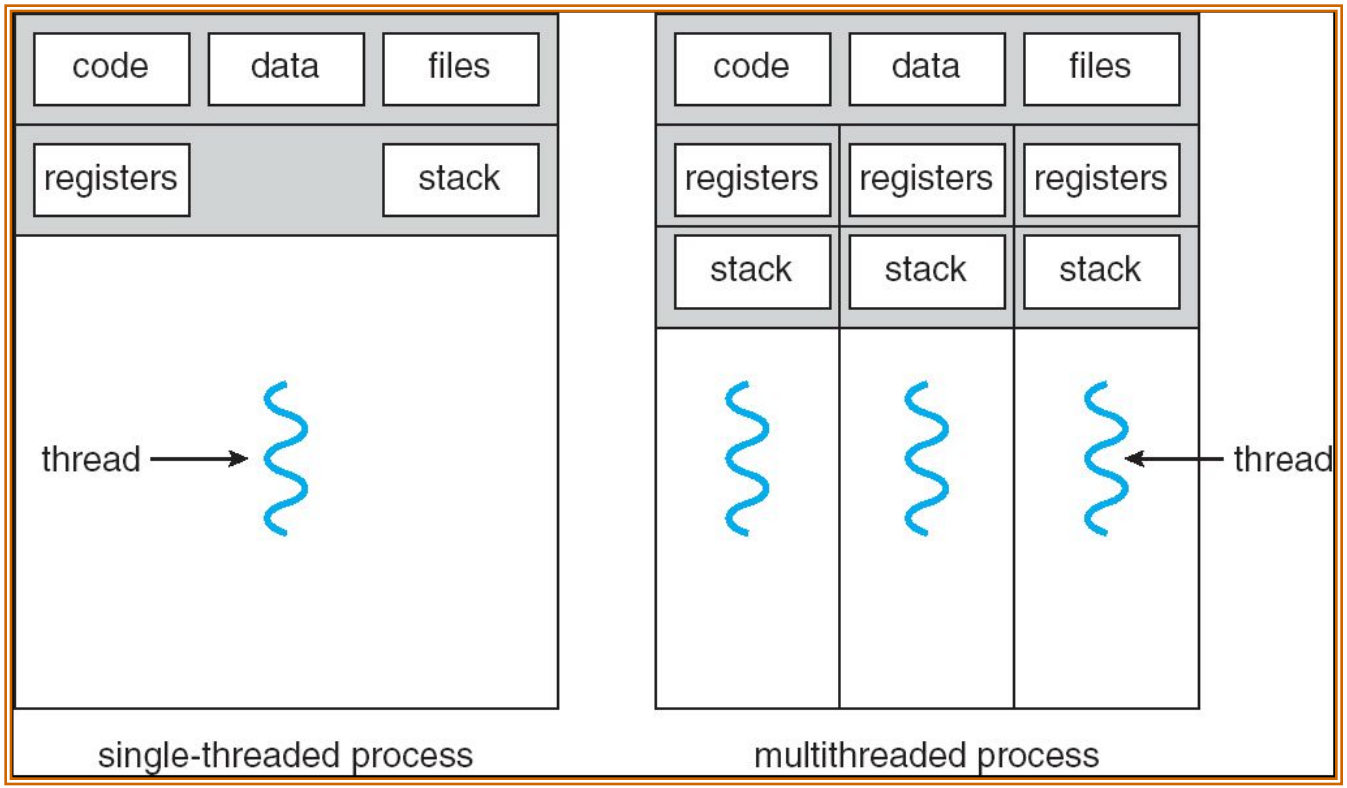
Fique Ligado !

- Os computadores atuais são mais rápidos ?
  - Tempo para ler um disco inteiro
    - 1990: 10 MB disco, 180 kB/s < 1 minuto
    - 2003: 160 GB disco, 40 MB/s ~ 1 hora
    - 2020: 2560 TB disco, 9 GB/s > 3 dias
  - Tempo para escrever toda memoria em disco
    - 1990: 640 KB RAM, 180 KB/s = 3.5 segundos
    - 2003: 2 GB RAM, 40 MB/s = 50 segundos
    - 2020: 6 TB RAM, 9 GB/s ~ 11 minutos
  - Tempo para ler dados para RAM
    - 1990: < 1 ciclo de clock
    - 2003: 200-400 ciclo de clock
    - 2020: > 5000 ciclos

Chapter 4: Threads

- Visao Geral
- Modelos Multithreading
- Questoes associadas a Threading
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads

Processos Single e Multithreaded



Beneficios

- Responsividade
- Compartilhamento de recursos
- Economia de espaco e ciclos de CPU
- Utilizacao de Arquiteturas Multiprocessadas

User e Kernel Threads

- User threads – Gerenciamento realizado pelas bibliotecas no nivel do usuario
- Kernel threads - Threads suportada diretamente pelo kernel

Kernel Threads

Exemplos

- Windows XP/2000
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X

Modelos Multithreading

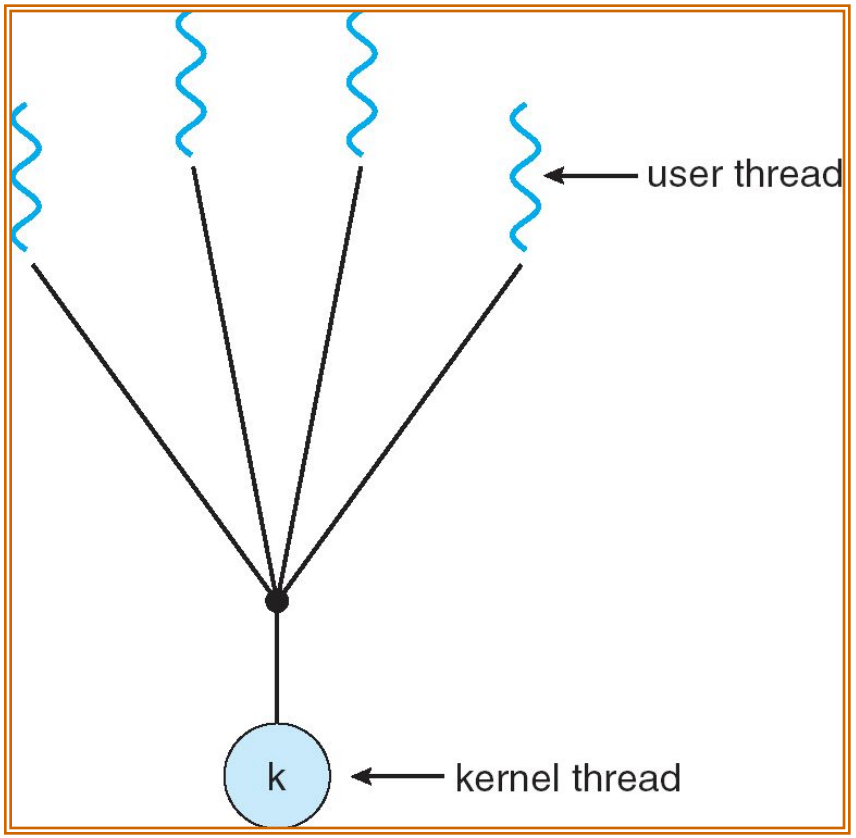
Mapea threads do usuario para threads do kernel:

- Muitos para Um (Many-to-One)
- Um para um (One-to-One)
- Muitos para Muitos (Many-to-Many)

Muitos para Um (Many-to-One)

- Varias threads no nivel do usuario para uma unica thread do kernel
- Exemplos:
  - Solaris Green Threads
  - GNU Portable Threads

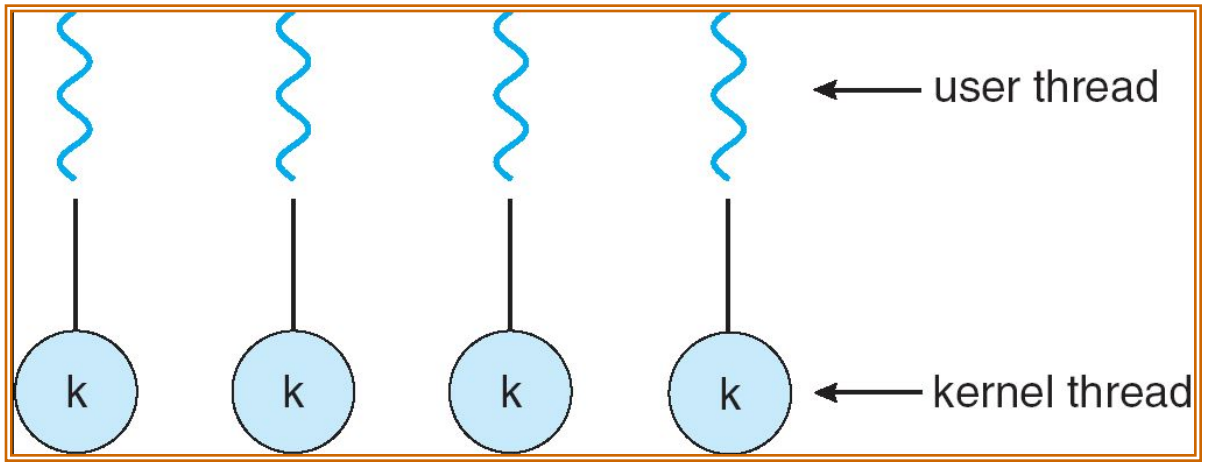
Modelo muitos para um (Many-to-One)



Modelo Um para Um (One-to-One)

- Cada thread no nivel do usuario mapeada para uma thread no nivel do kernel
- Exemplos
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

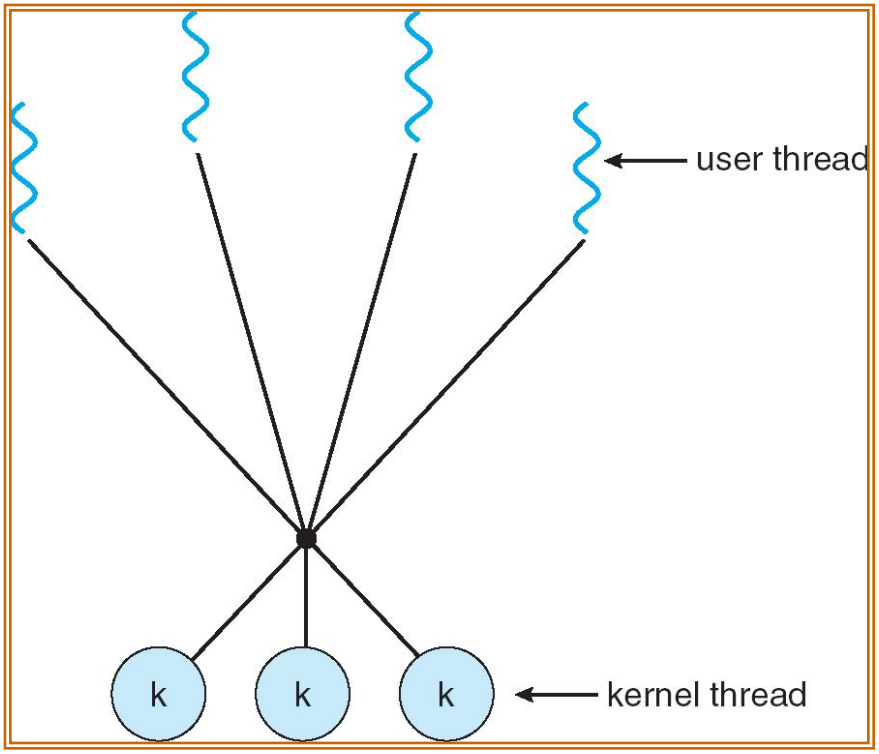
Modelo Um para Um (One-to-One)



Modelo Muitos para Muitos (Many-to-Many)

- Permite muitas threads do nivel do usuario serem mapeadas para muitas threads no nivel do kernel (quantidade menor ou igual a threads dos usuarios)
- Permite um SO criar um numero suficiente de threads do kernel
- Solaris antes da versao 9
- Windows NT/2000 com o *ThreadFiber* package

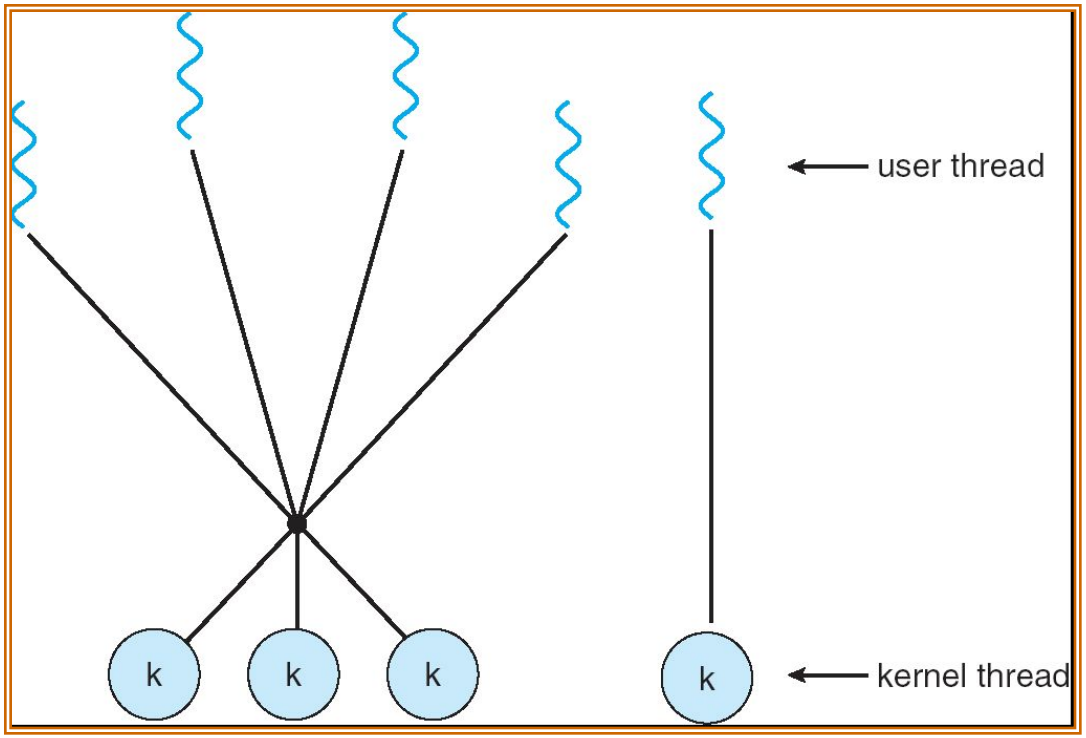
Modelo Muitos para Muitos (Many-to-Many)



Modelo em dois niveis

- Similar ao M:M, exceto que permite limitar uma thread do usuario a uma thread do kernel
- Exemplos
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 e anteriores

Modelo em dois niveis (Two-level)



Bibliotecas de Threads

- Dois tipos: nivel do usuario e nivel do kernel
- Exemplos
  - POSIX Pthreads (especificacao)
  - Win32 Threads (kernel)
  - Java Threads (usa as bibliotecas do sistema hospedeiro)

Java Threads

- Java threads sao gerenciadas pela JVM
- Java threads podem ser criadas por:
  - Implementando a Runnable interface

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

Java Threads - Exemplo

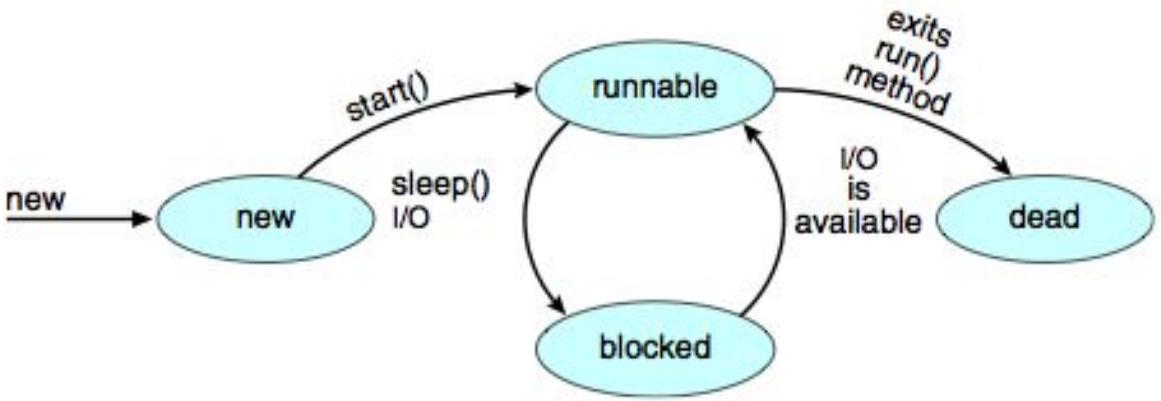
```
class MutableInteger
{
   private int value;
   public int getValue() {
     return value;
   }
   public void setValue(int value) {
     this.value = value;
   }
}

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);
   }
}
```

Java Threads - 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>");
    }
}
```

Java Thread States



Java Threads - Producer-Consumer

```
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();
    }
}
```

Java Threads - Producer-Consumer

```
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);
      }
   }
}
```

Java Threads - Producer-Consumer

```
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);
      }
   }
}
```

Questões

- Semantica do **fork()** e **exec()** system calls
- Cancelando Threads
- Tratamento de sinal (Signal handling)
- Thread pools
- Dados Especificos para Thread
- Ativacoes de Escalonador

Semantica do fork() e exec()

- Um **fork()** duplica somente a thread em questao ou todas as threads do processo?

Cancelamento de Thread

- Término de uma thread antes desta terminar

- Duas abordagens:
  - **Cancelamento assincrono:** termina a thread imediatamente
  - **Cancelamento deferido:** permite que a thread verifique periodicamente se pode ser cancelada

Cancelamento de Thread

Cancelamento deferido de thread em Java

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

Cancelamento de Thread

Cancelamento deferido de thread em Java
Checando estado da interrupcao

```
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
   }
}
```

Manipulacao de Sinal

- Sinal sao usados pelo sistema UNIX para notificar um processo quando um evento ocorre
- Um **signal handler** é usado para processar sinais
  1. Sinal é gerado por um evento particular
  2. Sinal é entregue a um processo
  3. Sinal é manipulado
- Opcoes:
  - Entregar o sinal a uma thread no qual o sinal se aplica
  - Entregar o sinal para todas as threads em um processo
  - Entregar o sinal para certas threads de um processo
  - Associar uma thread especifica para receber todos os sinais relativo ao processo

Thread Pools

- Criar um número de threads em um pool onde essas estao prontas para execucao

- Vantagens:
  - Mais rapido do que criar thread sob demanda
  - Limita o numero de threads da aplicacao de acordo com o tamanho do pool

Thread Pools

- Java prove 3 tipos de arquiteturas:

1. **Single thread executor** – tamanho do pool = 1.
   - `static ExecutorService newSingleThreadExecutor()`
2. **Fixed thread executor** – pool de tamanho fixo.
   - `static ExecutorService newFixedThreadPool(int nThreads)`
3. **Cached thread pool** –tamanho ilimitado
   - `static ExecutorService newCachedThreadPool()`

Thread Pools

Uma tarefa a ser servida em um pool

```
public class Task implements Runnable
{
   public void run() {
    System.out.println("I am working on a task.");
    . . .
   }
}
```

Thread Pools

Criando um thread pool em Java

```
import java.util.concurrent.*;

public class TPExample
{
   public static void main(String[] args) {
      int numTasks = Integer.parseInt(args[0].trim());

      // create the thread pool
      ExecutorService pool = Executors.newCachedThreadPool();

      // run each task using a thread in the pool
      for (int i = 0; i < numTasks; i++)
         pool.execute(new Task());

      // Shut down the pool. This shuts down the pool only
      // after all threads have completed.
      pool.shutdown();
   }
}
```

Dados Especificos de Thread

- Permite cada thread ter sua propria copia de dados
- Util quando voce nao tem controle sobre a criacao da thread (como no caso de uso de thread pool)

Dados Especificos de Thread

Dados especificos para thread em Java.

```
class Service
{
   private static ThreadLocal errorCode =
     new ThreadLocal();

   public static void transaction() {
      try {
         /**
          * some operation where an error may occur
          . . .
          */
      }
      catch (Exception e) {
          errorCode.set(e);
      }
   }

   /**
    * get the error code for this transaction
    */
   public static Object getErrorCode() {
      return errorCode.get();
   }
}
```

Exemplo de Implementacao de ThreadLocal

```
public class ThreadLocal {
  private Map values = Collections.synchronizedMap(new HashMap());
  public Object get() {
    Thread curThread = Thread.currentThread();
    Object o = values.get(curThread);
    if (o == null && !values.containsKey(curThread)) {
      o = initialValue();
      values.put(curThread, o);
    }
    return o;
  }
  public void set(Object newValue) {
    values.put(Thread.currentThread(), newValue);
  }
  public Object initialValue() {
    return null;
  }
}
```

Exemplo de Uso

```
public class ConnectionDispenser {
  private static class ThreadLocalConnection extends ThreadLocal {
    public Object initialValue() {
      return DriverManager.getConnection(ConfigurationSingleton.getDbUrl());
    }
  }

  private static ThreadLocalConnection conn = new ThreadLocalConnection();

  public static Connection getConnection() {
    return (Connection) conn.get();
  }
}
```

Outro Exemplo de Uso

```
public class DebugLogger {
  private static class ThreadLocalList extends ThreadLocal {
    public Object initialValue() {
      return new ArrayList();
    }
    public List getList() {
      return (List) super.get();
    }
  }
  private ThreadLocalList list = new ThreadLocalList();
  private static String[] stringArray = new String[0];
  public void clear() {
    list.getList().clear();
  }
```

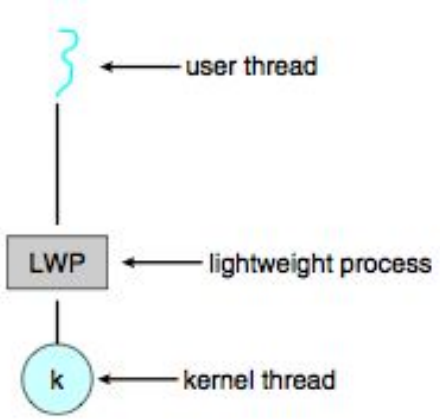
Outro Exemplo de Uso (cont)

```
public void put(String text) {
    list.getList().add(text);
  }

  public String[] get() {
    return list.getList().toArray(stringArray);
  }
}
```

Scheduler Activations

- Modelos M:M e Two-level requerem que uma comunicacao seja mantida para manter numero apropriado de threads do kernel alocadas a aplicacao
- Esquema de comunicacao “Scheduler activation” prove **upcalls** – o que permite a comunicacao entre o kernel e a biblioteca de thread



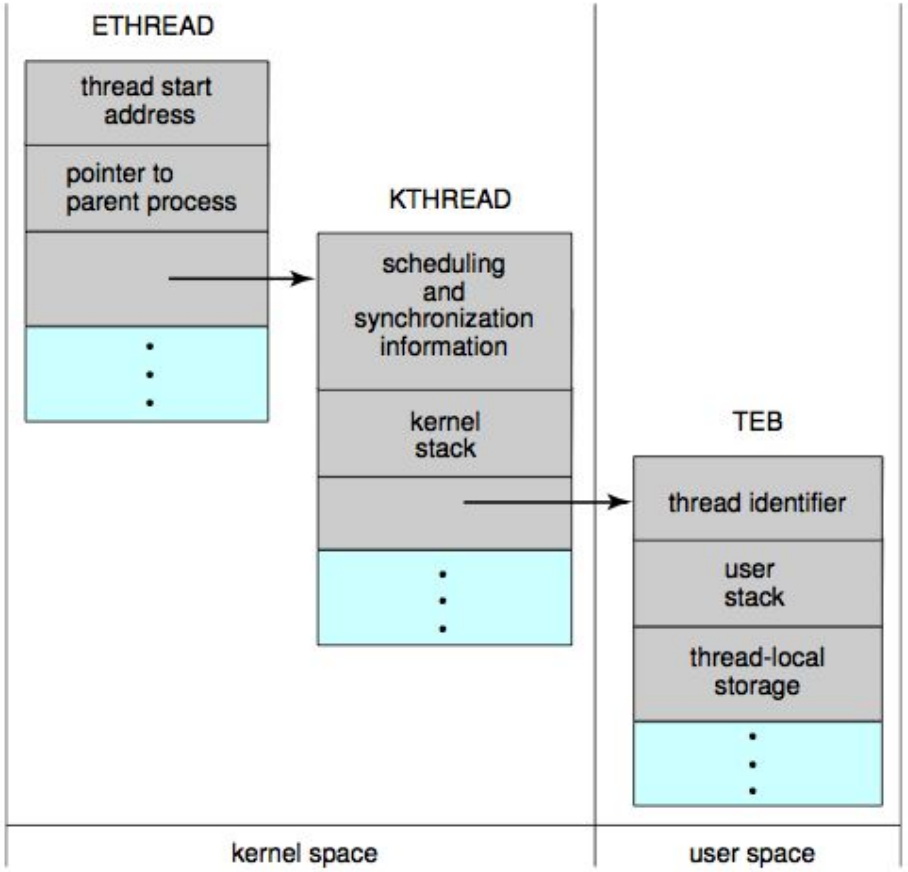
Pthreads

- Segue o padrao da API POSIX (IEEE 1003.1c) para criacao e sincronizacao de threads
- API especifica comportamento da biblioteca de thread, a implementacao depende do desenvolvimento da biblioteca
- Comum em sistemas operacionais UNIX (Solaris, Linux, Mac OS X)

Windows XP Threads

- Implementa mapeamento 1:1 (suporta mapeamento n:n)
- Cada thread contem
  - Um thread id
  - Um Register set
  - Stacks separadas para modo user e kernel
  - Area de armazenamento de dados privada
- register set, stacks, e area de armazenamento privado sao conhecidos como contexto das threads

Windows XP Threads



Linux Threads

- Linux se refere a threads/processos como tarefas
- Criacao de thread é feita atraves da chamada ao sistema **clone()**
- **clone()** permite uma tarefa filha compartilhar o espaco de endereco da tarefa pai (processo)

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.

Fim do Capitulo 4