CSMC 412

Operating Systems

Prof. Ashok K Agrawala

© 2001 Ashok Agrawala

Set 6

Threads

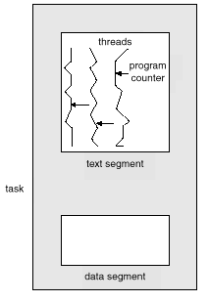
- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:
  - program counter
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating-system resources

  collectively know as a *task*.
- A traditional or *heavyweight* process is equal to a task with one thread

Threads (Cont.)

- In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
  - Cooperation of multiple threads in same job confers higher throughput and improved performance.
  - Applications that require sharing a common buffer (i.e., producer-consumer) benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.
- Kernel-supported threads (Mach and OS/2).
- User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).
- Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).

Multiple Threads within a Task



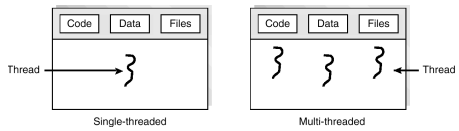
Threads

- Benefits
- User and Kernel Threads
- Multithreading Models
- Solaris 2 Threads
- Java Threads

Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

Single and Multithreaded Processes



User Threads

- **Thread Management Done by User-Level Threads Library**
- **Examples**
  - **POSIX *Pthreads***
  - **Mach *C-threads***
  - **Solaris *threads***

Kernel Threads

- **Supported by the Kernel**
- **Examples**
  - **Windows 95/98/NT**
  - **Solaris**
  - **Digital UNIX**

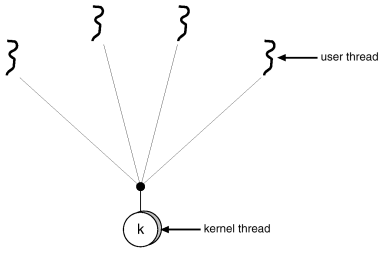
Multithreading Models

- **Many-to-One**
- **One-to-One**
- **Many-to-Many**

Many-to-One

- **Many User-Level Threads Mapped to Single Kernel Thread.**
- **Used on Systems That Do Not Support Kernel Threads.**

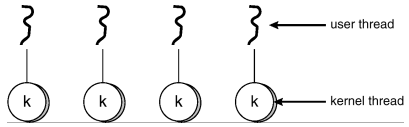
Many-to-one Model



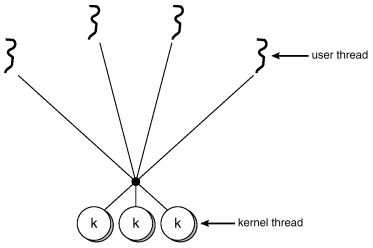
One-to-One

- Each User-Level Thread Maps to Kernel Thread.
- Examples
  - Windows 95/98/NT
  - OS/2

One-to-one Model



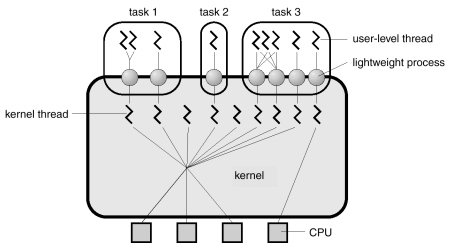
Many-to-many Model



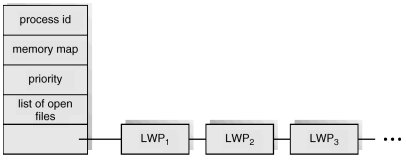
Threads Support in Solaris 2

- Solaris 2 is a version of UNIX with support for threads at the kernel and user levels, symmetric multiprocessing, and real-time scheduling.
- LWP – intermediate level between user-level threads and kernel-level threads.
- Resource needs of thread types:
  - Kernel thread: small data structure and a stack; thread switching does not require changing memory access information – relatively fast.
  - LWP: PCB with register data, accounting and memory information,; switching between LWPs is relatively slow.
  - User-level thread: only ned stack and program counter; no kernel involvement means fast switching. Kernel only sees the LWPs that support user-level threads.

Solaris 2 Threads



Solaris Process



Solaris process

Java Threads

- Java Threads May be Created by:
  - Extending Thread class
  - Implementing the Runnable interface

Extending the Thread Class

```
class  Worker1 extends Thread
{
   public void run() {
         System.out.println("I am a Worker Thread");
   }
}
```


Creating the Thread

```
public class First
{
   public static void main(String args[]) {
         Worker runner = new Worker1();

         runner.start();

         System.out.println("I am the main thread");
   }
}
```


The Runnable Interface

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


Implementing the Runnable Interface

```
class  Worker2 implements Runnable
{
   public void run() {
         System.out.println("I am a Worker Thread");
   }
}
```


Creating the Thread

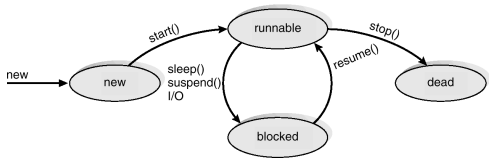
```
public class Second
{
   public static void main(String args[]) {
         Runnable runner = new Worker2();
         Thread thrd = new Thread(runner);
         thrd.start();

         System.out.println("I am the main thread");
   }
}
```

Java Thread Management

- suspend() – **suspends execution of the currently running thread.**
- sleep() – **puts the currently running thread to sleep for a specified amount of time.**
- resume() – **resumes execution of a suspended thread.**
- stop() – **stops execution of a thread.**

Java Thread States



Producer Consumer Problem

```
public class Server {
  public Server() {
        MessageQueue mailBox = new MessageQueue();

        Producer producerThread = new Producer(mailBox);
        Consumer consumerThread = new
  Consumer(mailBox);

        producerThread.start();
        consumerThread.start();
  }
  public static void main(String args[])  {
        Server server = new Server();
  }
}
```


Producer Thread

```
class Producer extends Thread {
  public Producer(MessageQueue m) {
        mbox = m;
  }

  public void run() {
   while (true) {
         // produce an item & enter it into the buffer
        Date message = new Date();
        mbox.send(message);
    }
  }
  private  MessageQueue mbox;
}
```


Consumer Thread

```
class Consumer extends Thread {
  public Consumer(MessageQueue m) {
        mbox = m;
  }

  public void run() {
   while (true) {
        Date message = (Date)mbox.receive();
        if (message != null)
              // consume the message
  }
  }
  private  MessageQueue mbox;
}
```


Interprocess Communication (IPC)

- **Mechanism for processes to communicate and to synchronize their actions.**
- **Message system – processes communicate with each other without resorting to shared variables.**
- **IPC facility provides two operations:**
  - send(***message***) – **message size fixed or variable**
  - receive(***message***)
- **If *P* and *Q* wish to communicate, they need to:**
  - **establish a *communication link* between them**
  - **exchange messages via send/receive**
- **Implementation of communication link**
  - **physical (e.g., shared memory, hardware bus)**
  - **logical (e.g., logical properties)**

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Direct Communication

- Processes must name each other explicitly:
  - send (*P, message*) – send a message to process P
  - receive(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Lilnks are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - Between each pair there exists exactly one link.
  - The link may be unidirectional, but is usually bi-directional.

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes can communicate only if they share a mailbox.
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes.
  - Each pair of processes may share several communication links.
  - Link may be unidirectional or bi-directional.
- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

Indirect Communication (Continued)

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A.
  - P_1, sends; P_2 and P_3 receive.
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes.
  - Allow only one process at a time to execute a receive operation.
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Buffering

- Queue of messages attached to the link; implemented in one of three ways.
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous).
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full.
  3. Unbounded capacity – infinite length
     Sender never waits.

Exception Conditions – Error Recovery

- Process terminates
- Lost messages
- Scrambled Messages