Process

470410-1
Spring 2016
3/24/2016

Kyoung Shin Park
Multimedia Engineering
Dankook University

Outline

- Kernel
- Process
- Thread
- Process Management
- Multi-Thread Programming

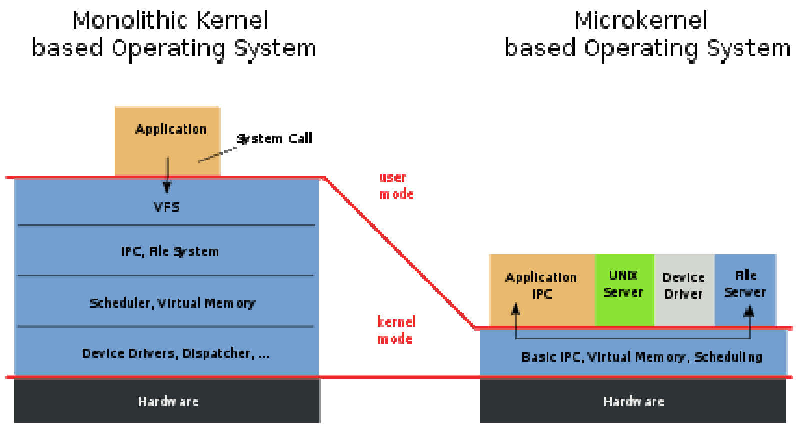
Kernel

- The privileged portion of the OS that has complete access to all resources
- The kernel controls
  - Process management
  - Process migration
  - Process scheduling
  - Address space

Kernel Types

- **Monolithic** kernel
  - Unix, MS-DOS, VMS
  - Every node doesn't need entire kernel in distributed operating system
- **Microkernel**
  - Mach, Chorus (JavaOS)
  - OS services are processes; Microkernel supports messages between such processes

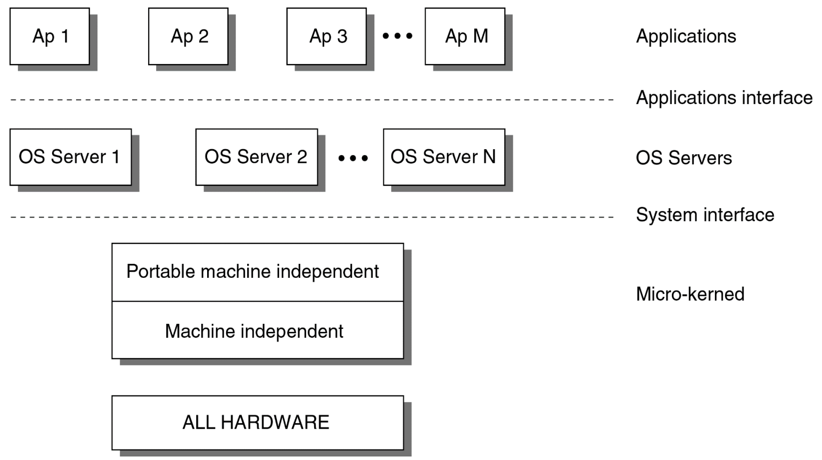
Kernel Types



Microkernel Structure

- The design of a microkernel contains three layers
  - The application layer containing all applications
  - The server layer containing all servers for the OS
  - The microkernel layer
- Between these layers are two interfaces
  - The applications interface
    - Between the application layer and the OS server layer
  - The system interface
    - The OS server layer and the microkernel layer
- The microkernel layer contains (starting from top to bottom)
  - All portable machine independent processes
  - All machine independent processes
  - All hardware

Microkernel Design



Ap 1 Ap 2 Ap 3 ••• Ap M

Applications

Applications interface

OS Server 1 OS Server 2 ••• OS Server N

OS Servers

System interface

Portable machine independent

Machine independent

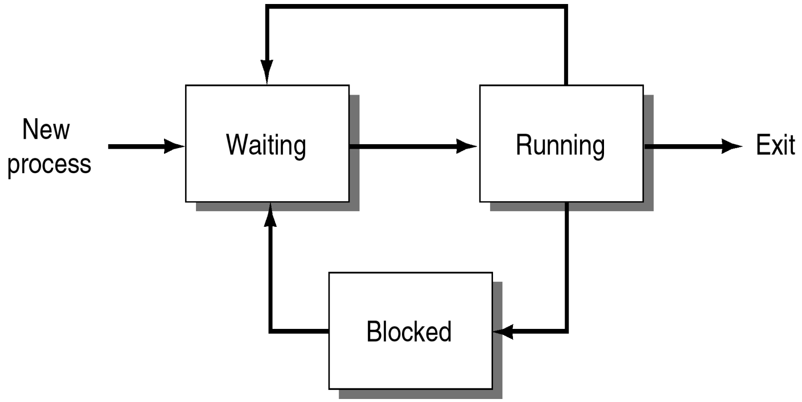
Micro-kerned

ALL HARDWARE

Process

- A program whose execution has started, but not terminated
- Has a current state (ready, running, waiting)
- Has a single address space
- May run serially or concurrently
- May interact with other processes via
  - Shared memory
  - Message passing
- Is a single thread of control

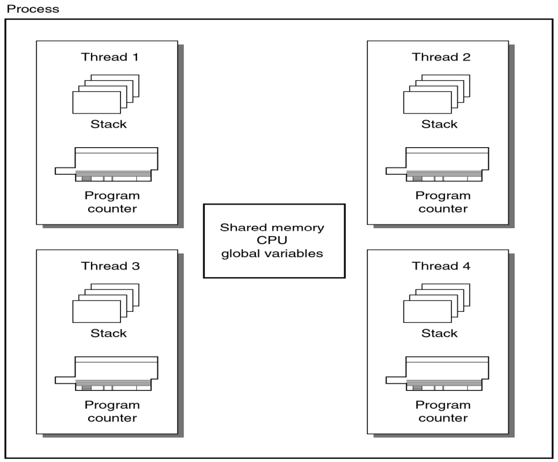
Process States



Thread

- A lightweight process
- Has state
- May share address space with other threads
- May run serially or concurrently
- Interacts with other threads via
  - Shared address space
  - Message passing

A Multithreaded Process



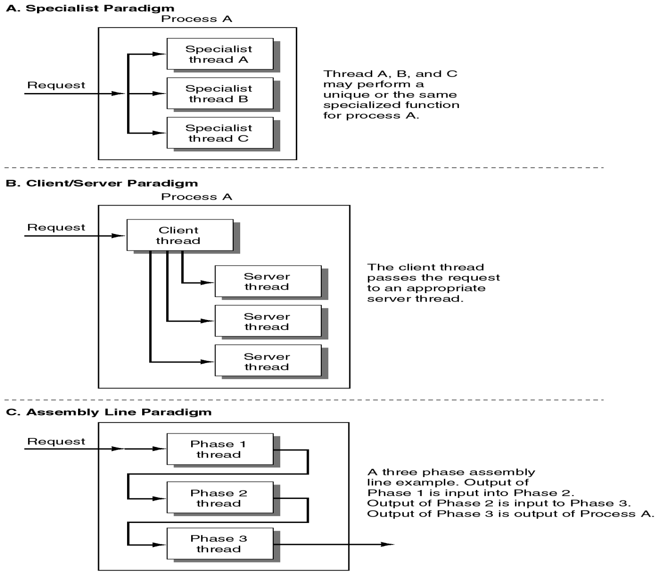
Issues with Processes and Threads

- If there is **shared memory** space
  - There is no protection barrier to other processes and threads
  - There needs to be methods to maintain the integrity of the processes and threads
- Thus **access** to this space must be **synchronized**
  - Processes/Threads must enforce **mutual exclusion**
  - Any code that accesses a shared resource must be a **critical section**

Multi-Threaded Systems

- Three models
  - **Specialist model** which keeps all threads equal
  - **Client/server model** where the server assigns tasks to clients
  - **Assembly line model** which acts like a pipeline
- It is multi-threaded system that support
  - POSIX
  - Java

Multithreaded Process Paradigms


A. Specialist Paradigm
Process A
Request
Specialist thread A
Specialist thread B
Specialist thread C
Thread A, B, and C may perform a unique or the same specialized function for process A.
B. Client/Server Paradigm
Process A
Request
Client thread
Server thread
Server thread
Server thread
The client thread passes the request to an appropriate server thread.
C. Assembly Line Paradigm
Request
Phase 1 thread
Phase 2 thread
Phase 3 thread
A three phase assembly line example. Output of Phase 1 is input into Phase 2. Output of Phase 2 is input to Phase 3. Output of Phase 3 is output of Process A.

Why Threads?

- Processes
  - Expensive to create or destroy a process
    - requires more memory space
  - Expensive to restore or swap out a process
    - requires memory map changes
- Threads
  - Can keep a pool of threads and reuse them
  - Memory space is shared and need not always be swapped

Process Management

- Process management controls process (or thread) and its components
- PCB **(Process Control Block)** that holds the current state of the process
  - Process id
  - Process state
  - Process priority
  - Process privileges
  - Virtual memory address
  - Recorded statistics for account

Process Management

- PCB operations include
  - Create
  - Delete
  - Signal
  - Wait
  - Schedule
  - change priority
  - Suspend
  - Resume
- This infers rules are needed for who can do what to whom (a tree hierarchy)
- Synchronization of processes is one of our concerns in distributed systems

Process Types in Distributed Systems

- Indivisible process (entire process must be assign to a single processor)
  - Independent
  - Not divisible into smaller tasks
- Divisible process (a process may be subdivided into smaller sub-processes, tasks, or threads)
  - May be broken up into smaller processes (tasks)
  - Subtasks may run on different nodes
  - Helps to balance load of distributed system
- Task Interaction Graphs (TIG) represents relationships between tasks of a divisible process

Load Distribution

- Goal
  - To utilize resources in an efficient manner
- Load balancing
  - To balance the load equally among resources
- Load sharing
  - To relieve overloaded resources
- Process migration
  - To move a process to another processor

Two Parts of Load Distribution Algorithms

- **Information-gathering**
  - Gather information about loads of other processors and select a suitable migration partner
  - E.g. Identifies idle processors, estimate cost of migration to various sites
  - Status states (site is overloaded/underutilized?)
- **Process selection**
  - Select a process to migrate
  - E.g. What is the communication delay for migrating a process, how to accommodate differences in heterogeneous systems
  - Which process/thread/task to migrate?
  - Expected overhead for migration?
  - Expected execution time?

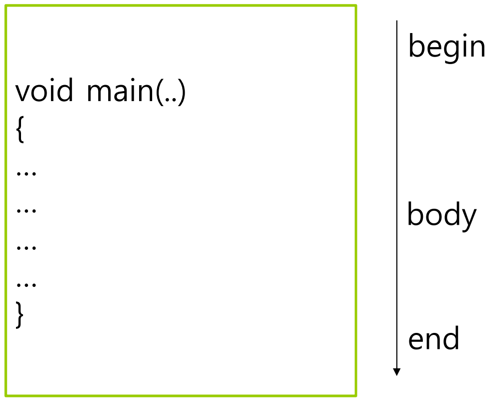
Heterogeneous Environments

- May include different data representations
  - Process migration may require data translation
- External data representation
  - Common data representation between sites used in heterogeneous systems.
  - External data representation greatly reduces the amount of time required to perform cross-platform process migration.

Threaded Applications

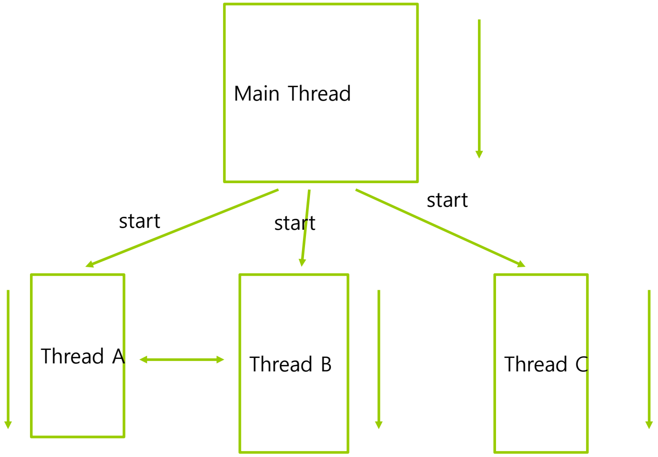
- Modern Applications & Systems
  - Operating System Level
    - Multitasking: Multiple applications running concurrently (i.e., there are multiple processes on your system)
  - Application Level
    - Multithreading: Application performs multiple operations at the same time (i.e., there are multiple threads within a single process)
  - Bottom Line:
    - Illusion of concurrency

A Single Threaded program



```
void main(..)
{
…
…
…
…
}
```

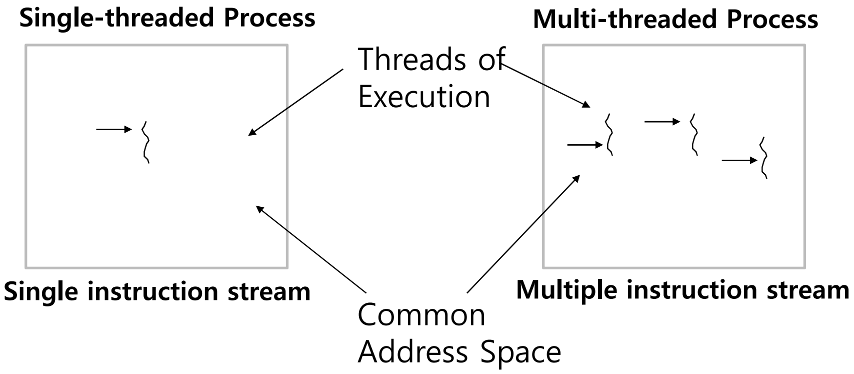
A Multithreaded Program



Threads may switch or exchange data/results

Single vs Multithreaded Processes

Threads are light-weight processes within a process



Single-threaded Process

Multi-threaded Process

Threads of Execution

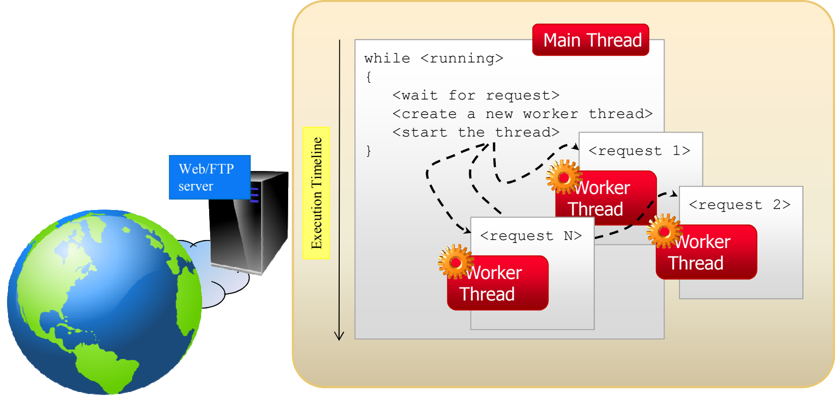
Single instruction stream

Multiple instruction stream

Common Address Space

Threaded Applications

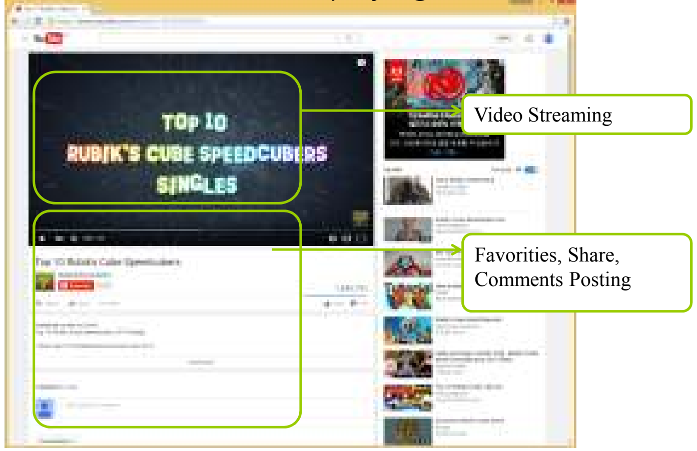
- Multithreaded web/FTP server for serving multiple clients concurrently



```
while <running>
{
   <wait for request>
   <create a new worker thread>
   <start the thread>
}
```

Threaded Applications

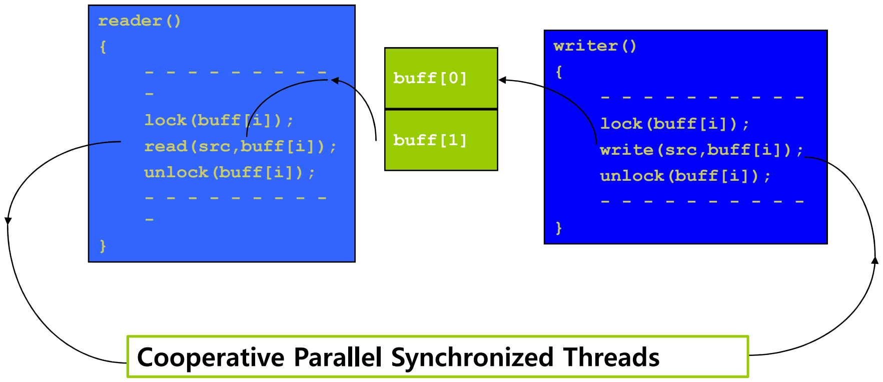
- Web browser for displaying and data retrieval



Video Streaming

Favorities, Share, Comments Posting

Multithreaded/Parallel File Copy



```
reader()
{
    - - - - - - - - -
    -
    lock(buff[i]);
    read(src,buff[i]);
    unlock(buff[i]);
    - - - - - - - - -
    -
}
```

```
writer()
{
    - - - - - - - - - -
    lock(buff[i]);
    write(src,buff[i]);
    unlock(buff[i]);
    - - - - - - - - - -
}
```

Cooperative Parallel Synchronized Threads

Defining Threads

- Applications – Threads are used to perform:
  - Parallelism and concurrent execution of independent tasks / operations.
  - Implementation of reactive user interfaces.
  - Non blocking I/O operations.
  - Asynchronous behavior.
  - Timer and alarms implementation.

Defining Threads

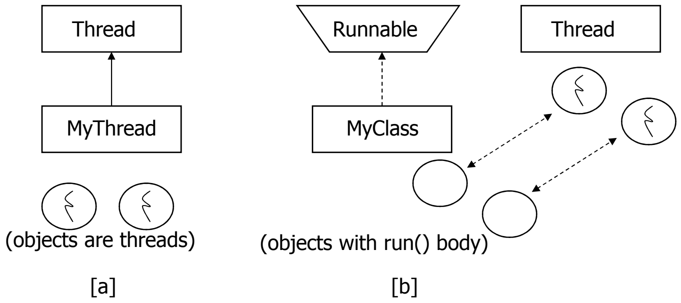
- A Thread is a piece of code that runs in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are used to express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

Java Threads

- Java has built in support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread   start   setPriority
  - yield   run   getPriority
  - sleep   stop   suspend
  - resume
- Java Garbage Collector is a low-priority thread

Java Threads

1. Create a class that extends the Thread class
2. Create a class that implements the Runnable interface



1. Extending the Thread Class

- Create a class by extending Thread class and override run() method:

```
class MyThread extends Thread
{
       public void run()
      {
         // thread body of execution
      }
}
```

- Create a thread:

```
MyThread thr1 = new MyThread();
```

- Start Execution of threads:

```
thr1.start();
```

- Create and Execute:

```
new MyThread().start();
```

1. Extending the Thread Class

```
class MyThread extends Thread {
         public void run() {
                  System.out.println(" this thread is running ... ");
         }
}

class ThreadEx1 {
         public static void main(String [] args  ) {
            MyThread t = new MyThread();
             t.start();
        }
}
```

2. Threads by implementing Runnable interface

- Create a class that implements the interface Runnable and override run() method:

```
class MyThread implements Runnable
{
  .....
  public void run()
  {
     // thread body of execution
  }
}
```

- Creating Object:

```
MyThread myObject = new MyThread();
```

- Creating Thread Object:

```
Thread thr1 = new Thread( myObject );
```

- Start Execution:

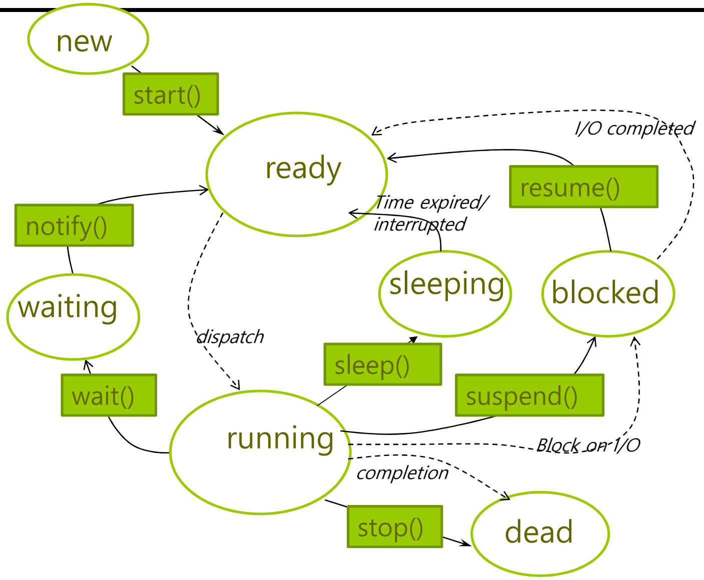
```
thr1.start();
```

2. Threads by implementing Runnable interface

```
class MyThread implements Runnable  {
         public void run() {
                  System.out.println(" this thread is running ... ");
         }
}

class ThreadEx2 {
         public static void main(String [] args  ) {
                  Thread t = new Thread(new MyThread());
                   t.start();
         }
}
```

Life Cycle of Thread



new

start()

ready

I/O completed

resume()

Time expired/ interrupted

notify()

sleeping

blocked

waiting

dispatch

sleep()

suspend()

wait()

running

Block on I/O

completion

stop()

dead

Three threads example

```
class A extends Thread
{
        public void run()
          {
                for(int i=1;i<=5;i++)
                  {
                         System.out.println("₩t From ThreadA: i= "+i);
                  }
                   System.out.println("Exit from A");
          }
}
class B extends Thread
{
        public void run()
          {
                for(int j=1;j<=5;j++)
                  {
                         System.out.println("₩t From ThreadB: j= "+j);
                  }
                   System.out.println("Exit from B");
          }
}
```

Three threads example

```
class C extends Thread
{
        public void run()
          {
                for(int k=1;k<=5;k++)
                  {
                         System.out.println("₩t From ThreadC: k= "+k);
                  }

                   System.out.println("Exit from C");
          }
}
class ThreadTest
{
           public static void main(String args[])
            {
                      new A().start();
                      new B().start();
                      new C().start();
            }
}
```

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (NORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

Thread Priority Example

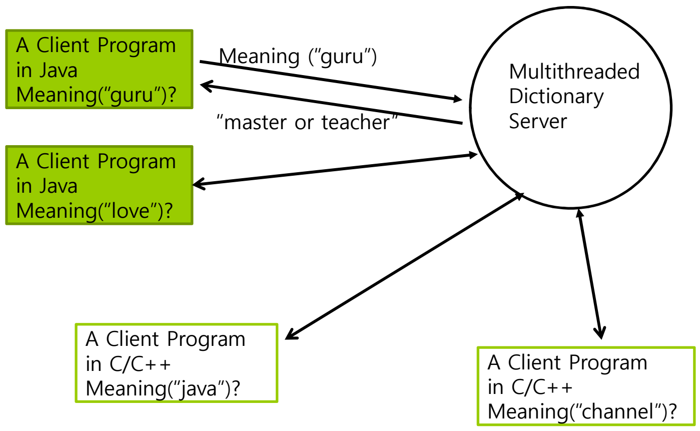
```
class A extends Thread {
        public void run()
        {
                System.out.println("Thread A started");
                for(int i=1;i<=4;i++) {
                        System.out.println("₩t From ThreadA: i= "+i);
                }
                System.out.println("Exit from A");
        }
}
class B extends Thread {
        public void run()
        {
                System.out.println("Thread B started");
                for(int j=1;j<=4;j++) {
                        System.out.println("₩t From ThreadB: j= "+j);
                }
                System.out.println("Exit from B");
        }
}
```

Thread Priority Example

```
class C extends Thread {
       public void run()
       {
                System.out.println("Thread C started");
                for(int k=1;k<=4;k++) {
                        System.out.println("₩t From ThreadC: k= "+k);
                }
                System.out.println("Exit from C");
        }
}
class ThreadPriority {
       public static void main(String args[])
       {
                  A threadA=new A(); B threadB=new B(); C threadC=new C();
                  threadC.setPriority(Thread.MAX_PRIORITY);
                  threadB.setPriority(threadA.getPriority()+1);
                  threadA.setPriority(Thread.MIN_PRIORITY);
                  System.out.println("Started Thread A"); threadA.start();
                  System.out.println("Started Thread B"); threadB.start();
                  System.out.println("Started Thread C"); threadC.start();
                  System.out.println("End of main thread");
       }
}
```

Multithreaded Server

- Multithreaded Dictionary Server – Demonstrates the use of Sockets and Threads



A Client Program in Java Meaning("guru")?

Meaning ("guru")

"master or teacher"

Multithreaded Dictionary Server

A Client Program in Java Meaning("love")?

A Client Program in C/C++ Meaning("java")?

A Client Program in C/C++ Meaning("channel")?

Reference

- http://www.cs.colostate.edu/~cs551/CourseNotes/Processes/ProcessTOC.html
- http://www.cs.colostate.edu/~cs551/CourseNotes/Processes/LMNotes.html
- http://www.cloudbus.org/652/L3-Threads.ppt