COMP 346 – Fall 2019

Tutorial # 1

PROCESSES & THREADS

TUT-Introduction

What do we do in our tutoring sessions?

- Quick Review
- Working on Fundamentals
- Solving Sample Questions
- Midterm and Final preparation

How to contact your TA?

- E-mail address available on the course outline :
  - kantesariyashyam@gmail.com
  - m_vadari@encs.concordia.ca
  - arjungk92@gmail.com

Process

- Process is a program in execution
- A process has a self-contained execution environment.
- Each process has its own memory space.
- What the user sees as a single application may in fact be a set of cooperating processes.
- Most implementations of the Java virtual machine run as a single process.

Process

- Program itself is not a process
- Program is a passive entity
- Process is active entity
- A program becomes a process when an executable file is loaded into memory
  - Double-clicking
  - Icon
- Two processes can be associated with the same program

Process

- Example: MS Word, Google Chrome

Process in Memory

- Stack: contains temporary data
- Heap: dynamic memory data
- Data: Global variables
- Text: code

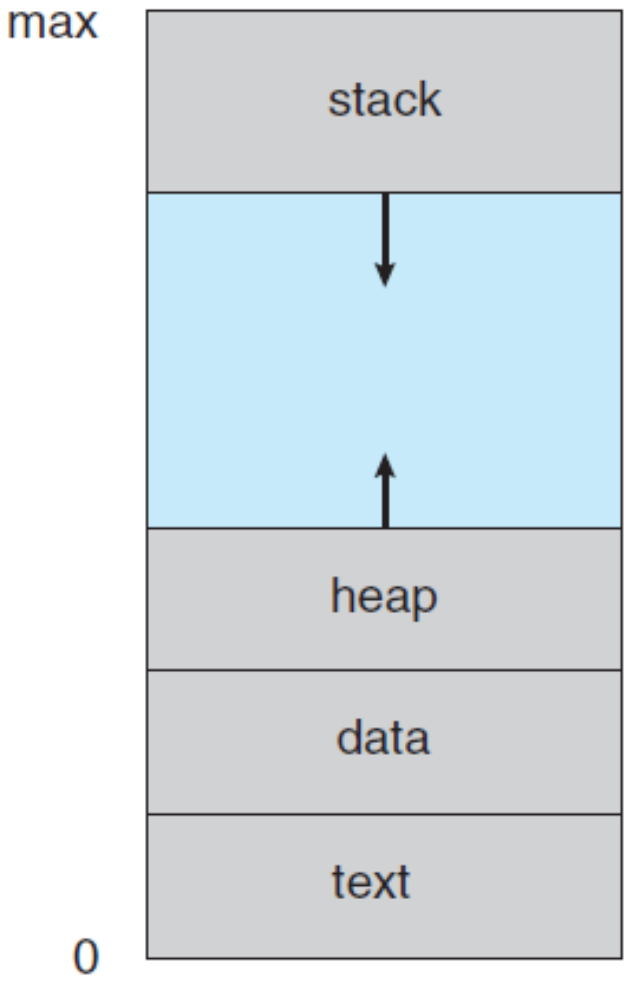
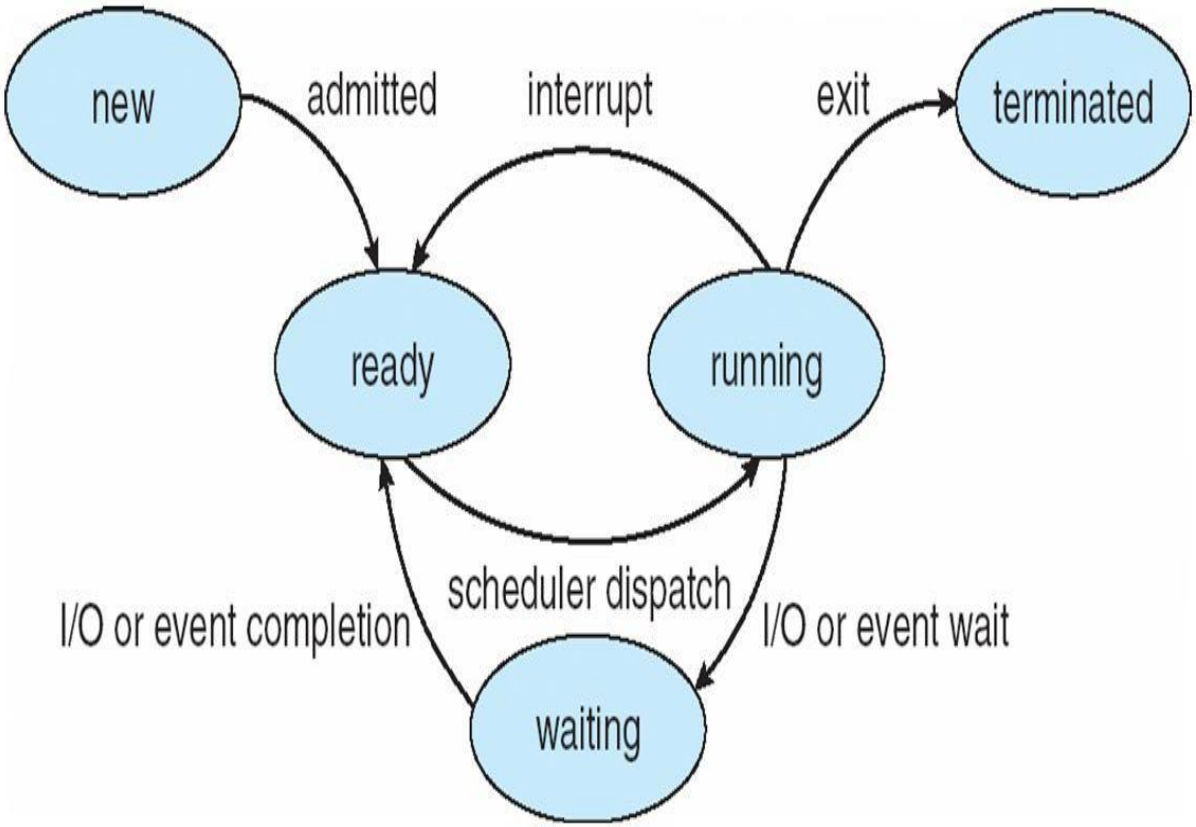
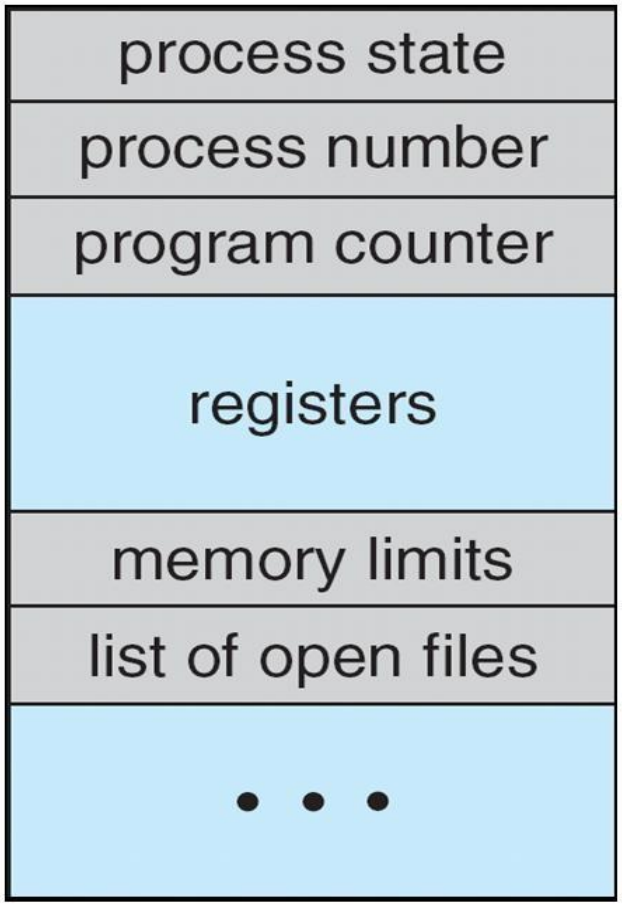


Diagram of Process State

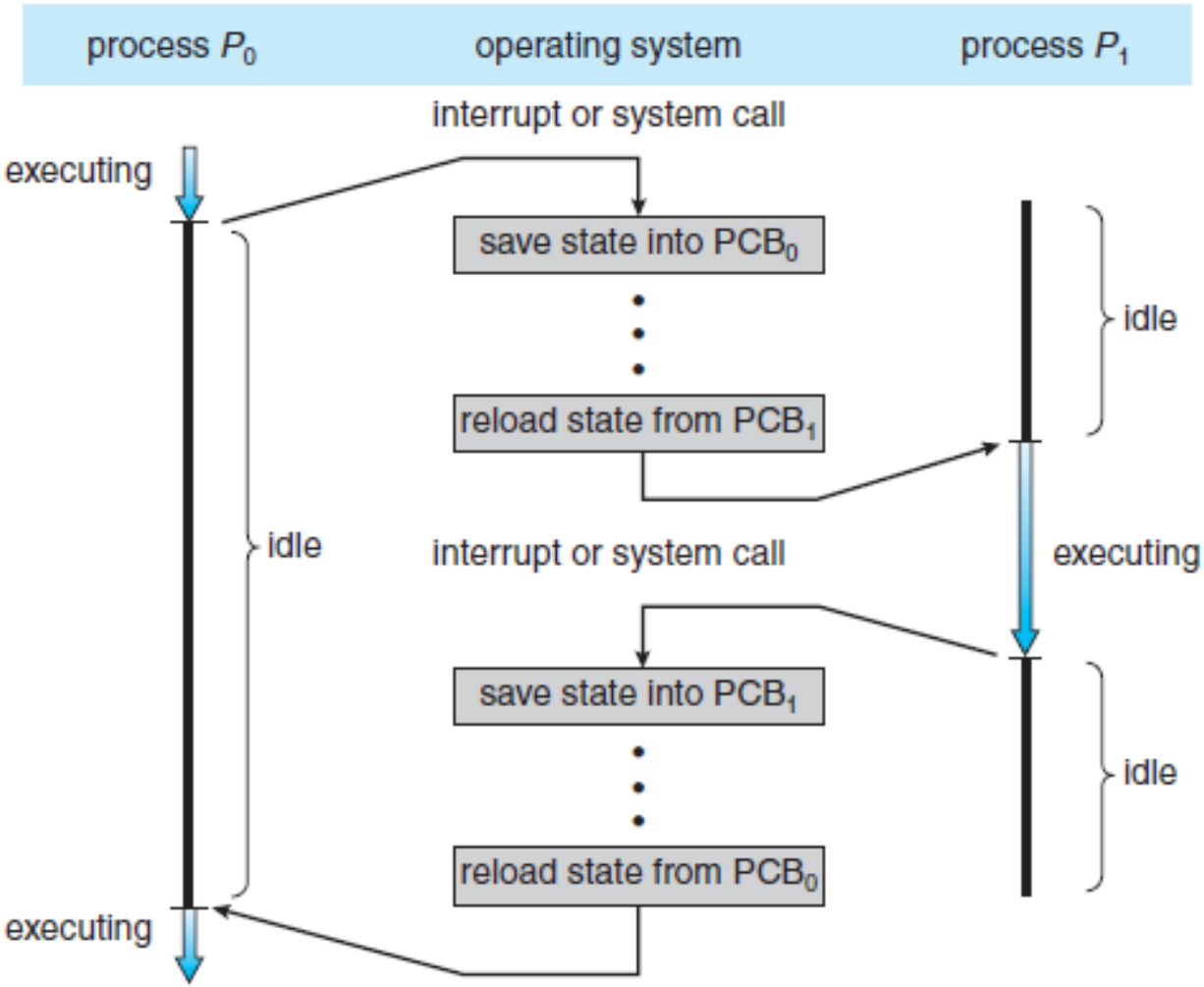


Process Control Block (PCB)

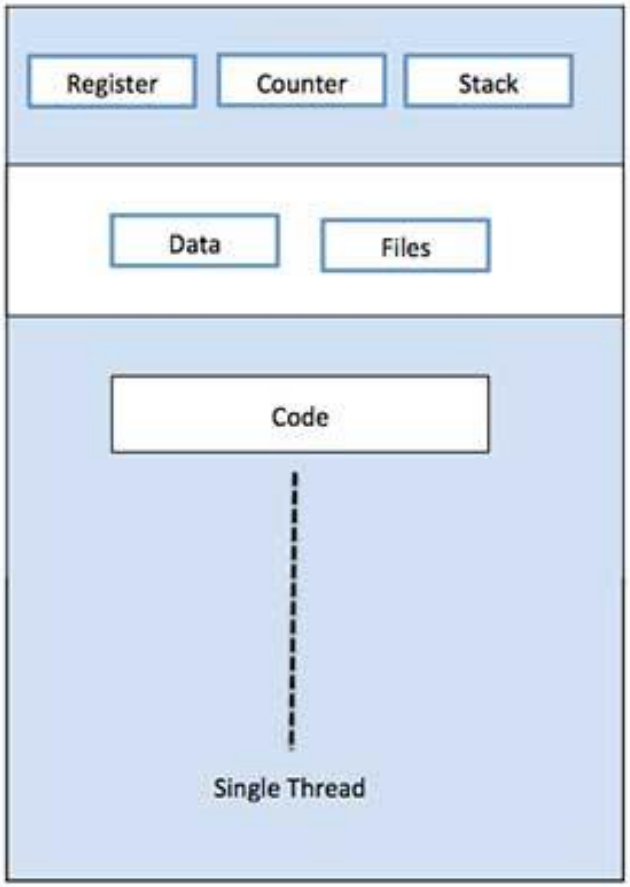
- **Process State**: new, ready, …
- **Process number**: Unique ID
- **Program counter**: the address of the next instruction to be executed
- **Registers**: state information when an interrupt occurs
- **Memory limits**: base and limit of memory address
- **List of open files**:



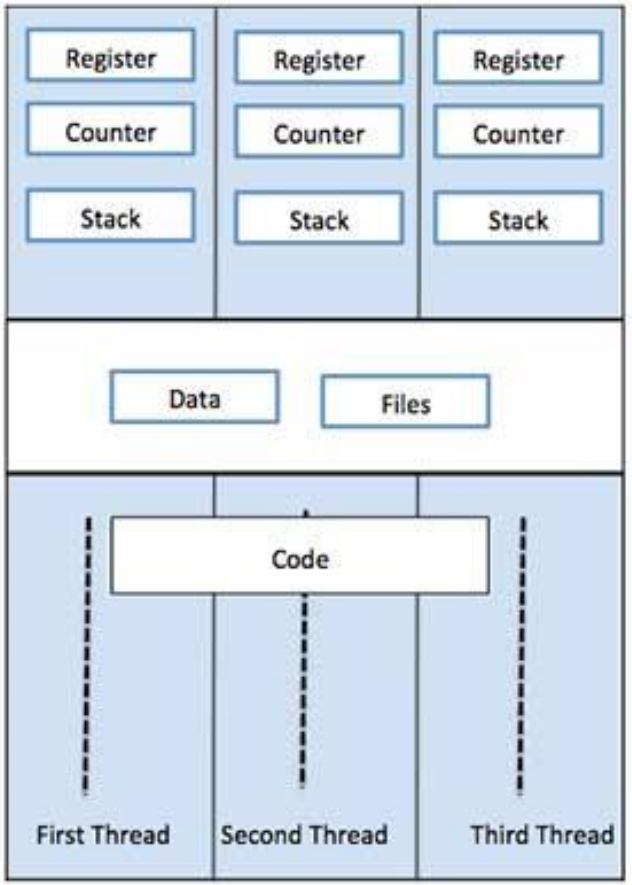
CPU switch from process to process



Single & Multiple Thread of control



Single Process P with single thread

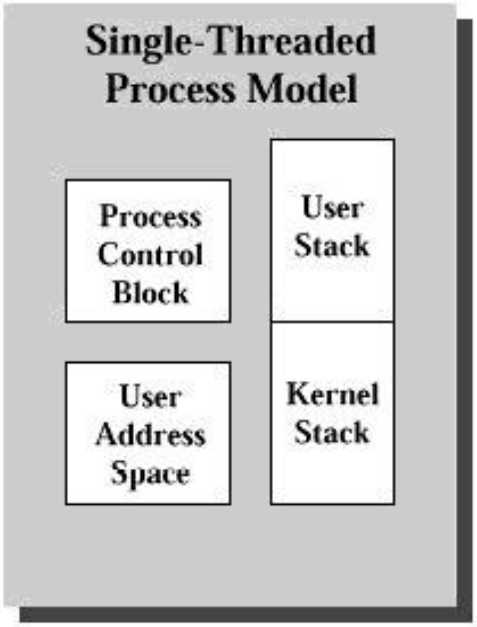


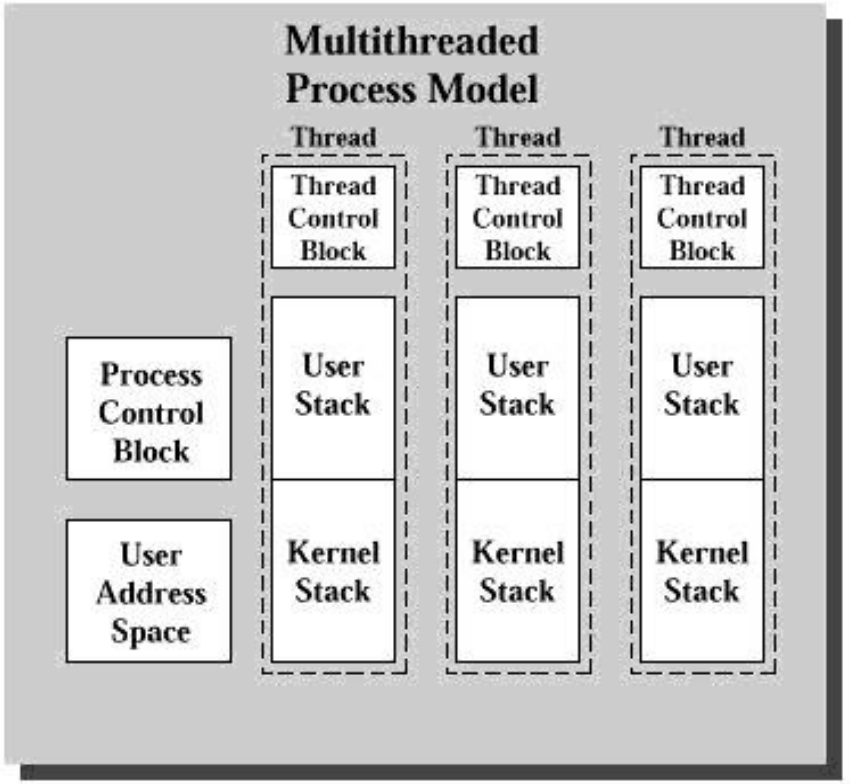
Single Process P with three threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Thread Structure [1]



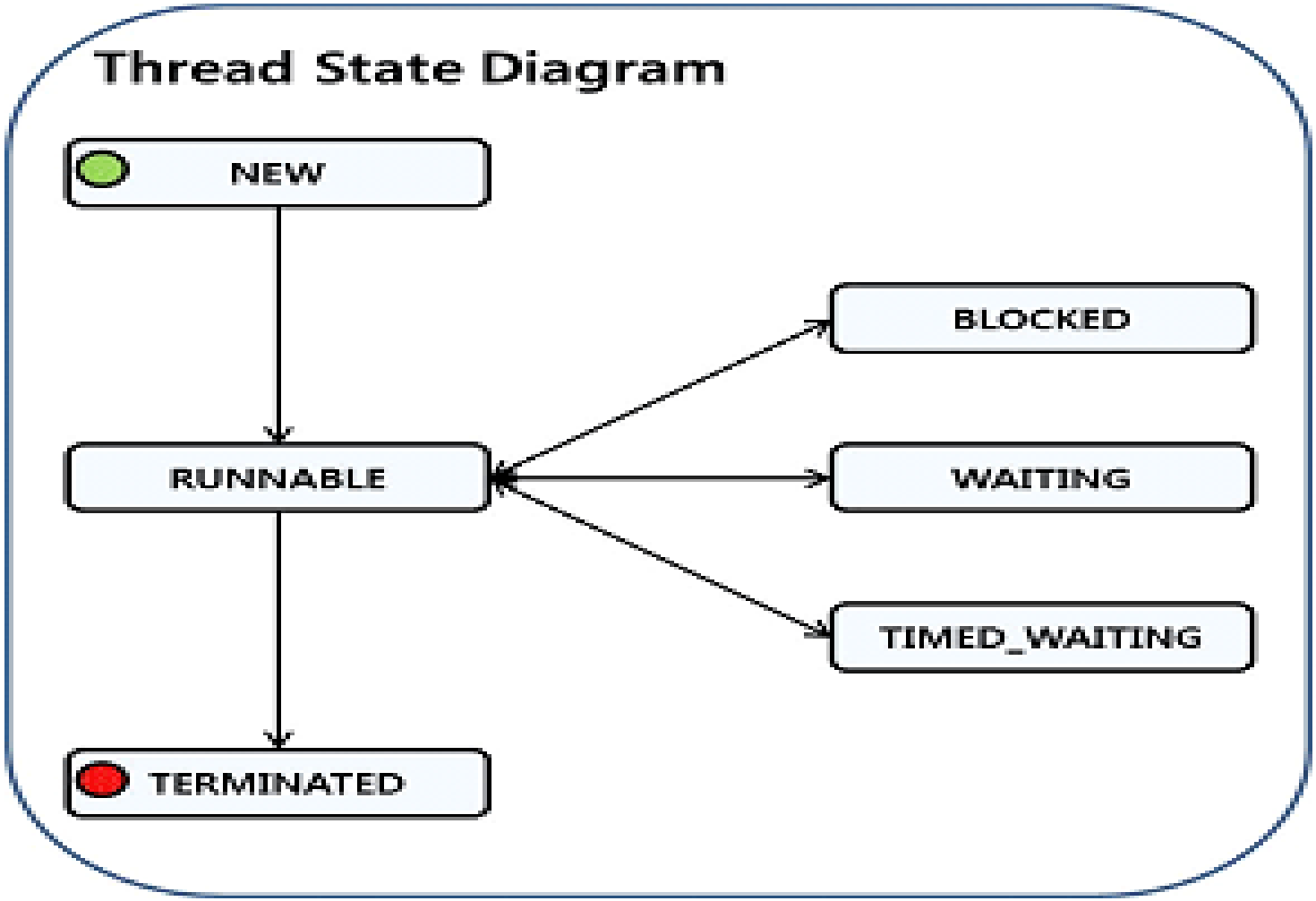


Thread Control Block

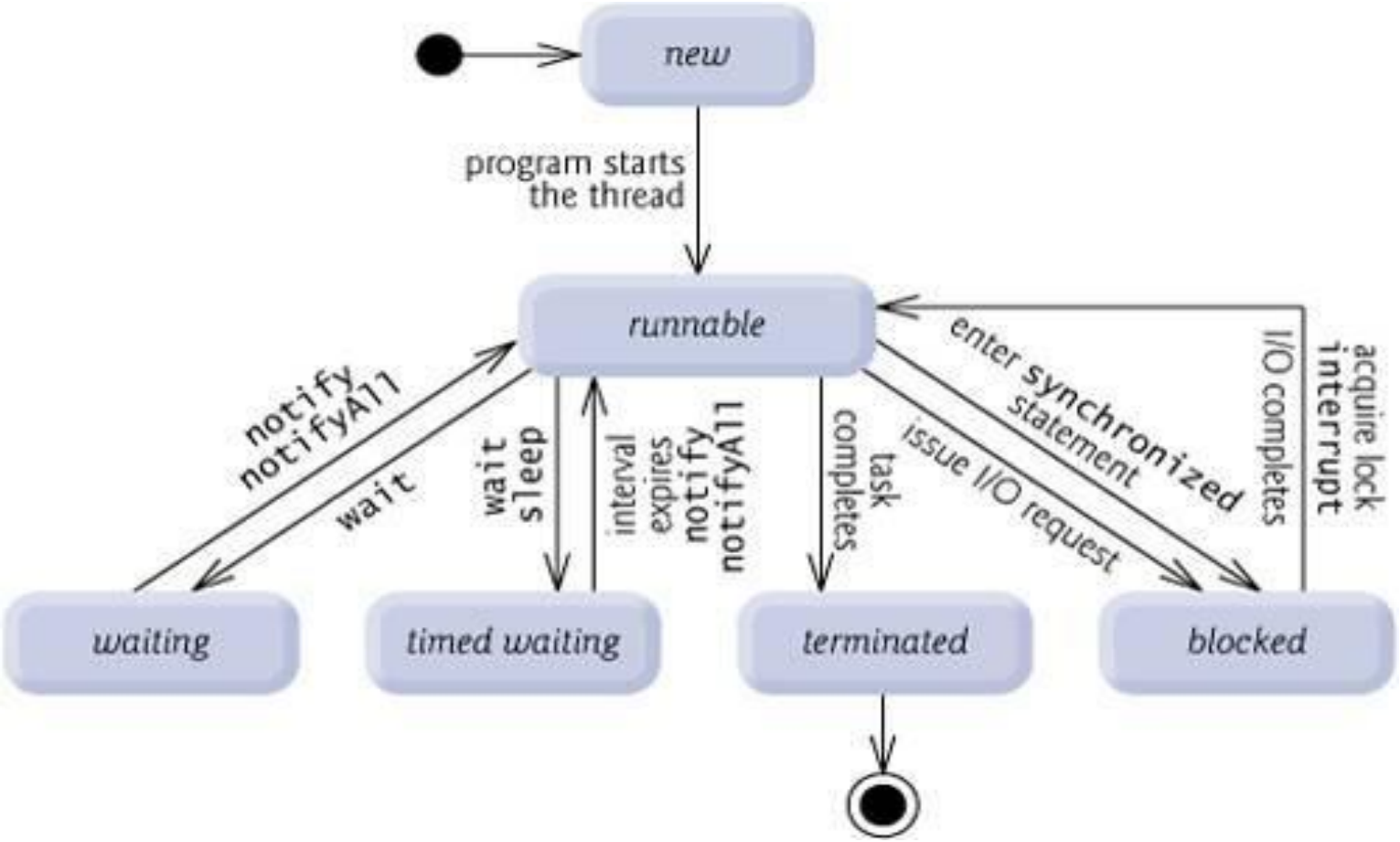
The Thread Control Block (TCB) is a structure used to maintain the state of a thread during run-time.
It Includes:

1 General-Purpose registers
2 Stacks
3 Program Counter
4 Thread ID

Thread States



Thread States



Thread States

- **Figure 1: Thread Status.**
- **NEW**: The thread is created but has not been processed yet.
- **RUNNABLE**: The thread is executing in the java virtual machine, but it may be waiting for other resources from operating system such as processor
- **BLOCKED**: The thread is waiting for a different thread to release its lock in order to get the monitor lock.
- **WAITING**: The thread is waiting by using a wait, join method.
- **TIMED_WAITING**: The thread is waiting by using a sleep, wait, join method. (The difference from WAITING is that the maximum waiting time is specified by the method parameter, and *WAITING* can be relieved by time as well as external changes.)

Advantages of Thread

- **Threads minimize the context switching time.**
- **Use of threads provides concurrency within a process.**
- **Efficient communication.**
- **It is more economical to create and context switch threads.**
- **Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.**

Thread Creation

- Extending the *Thread* class
- Implementing the *Runnable* interface

EXAMPLE 1

```
class ThreadEx1 extends Thread {
   private string name,
   ThreadEx1(String name) {
         this.name = name;
   }
public void run() {
                  for (int i = 0; i < 100; i++) {
    System.out.print("I am " + name);
         }
   }
 }
```

EXAMPLE 1

```
class Test1 {
  public static void main(String args[])
  {
      ThreadEx1 tr1 = new ThreadEx1
  (“alpha");
      ThreadEx1 tr2 = new ThreadEx1
  (“beta");
  tr1.start(); // Start the first thread
  tr2.start(); // Start the second thread
  }
}
```

EXAMPLE 2

```
class ThreadEx2 implements Runnable {
   private String name;
   ThreadEx2(String name) {
         this.name = name;
}

public void run() {
   for (int i = 0; i < 100; i++) {
         System.out.print("I am " + name);
}
}
}
```

EXAMPLE 2

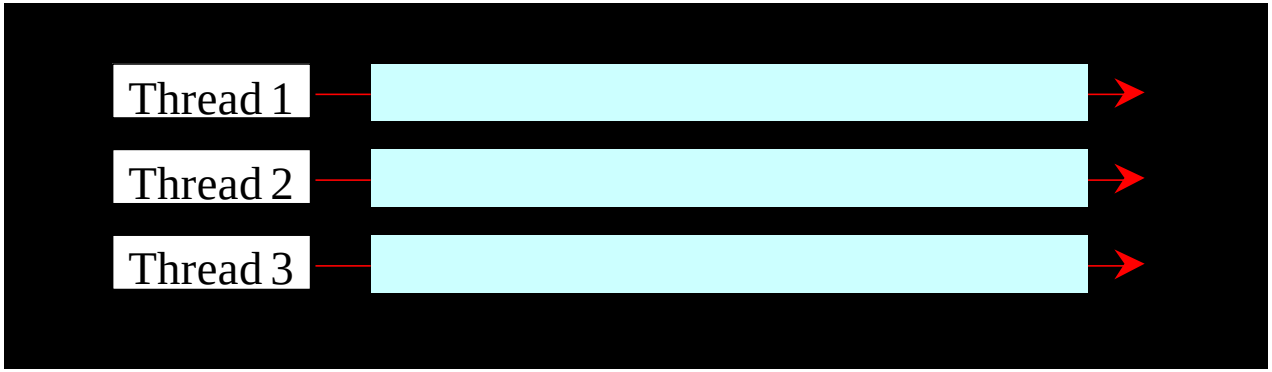
```
class Test2 {
  public static void main(String args[]) {
      Thread tr1 = new Thread(new
  ThreadEx2 (“alpha"));
      Thread tr2 = new Thread(new
  ThreadEx2 (“beta"));
      tr1.start(); // Start the first thread
      tr2.start(); // Start the second thread
  }
}
```

JOIN() METHOD

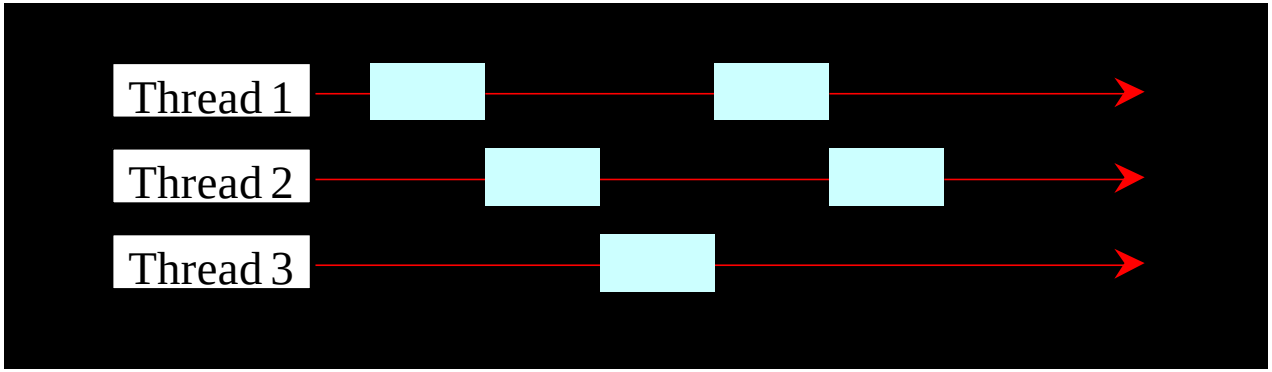
```
class Test2 {
   public static void main(String args[]) {
Thread tr1,tr2
        tr1 = new Thread(new ThreadEx2
   "alpha"));
        tr2 = new Thread( new ThreadEx2
   ("beta"));
        tr1.start(); // Start the first thread
        tr2.start(); // Start the second thread
Try{
        tr1.join();
        tr2.join();
     } catch (InterruptedException e) {}
System.out.println("main finished");
   }
}
```

Threads Concept

Multiple threads on multiple CPUs



Multiple threads sharing a single CPU



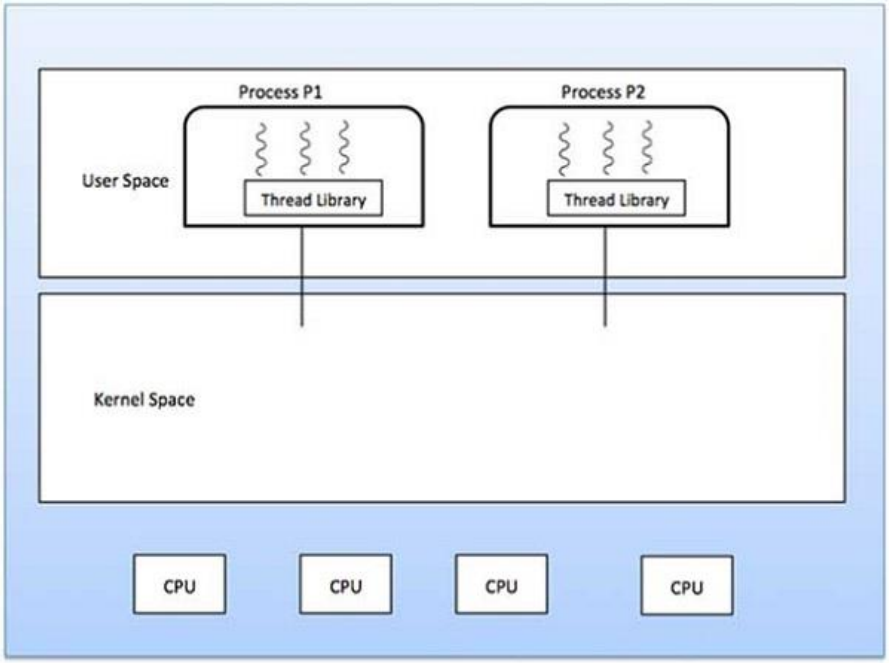
Types of Thread

Threads are implemented in following two ways:

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core

User Level Threads

- **In this case, the thread management kernel is not aware of the existence of threads.**
- **The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.**
- **The application starts with a single thread.**



User Level Threads – Cont…

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

- **In this case, thread management is done by the Kernel.**
- **There is no thread management code in the application area.**
- **Kernel threads are supported directly by the operating system.**
- **Any application can be programmed to be multithreaded.**
- **All of the threads within an application are supported within a single process.**
- **The Kernel maintains context information for the process as a whole and for individuals threads within the process.**
- **Scheduling by the Kernel is done on a thread basis.**
- **The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.**

Kernel Level Threads – Cont…

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Context Switch

- A *context switch* (also sometimes referred to as a *process switch* or a *task switch*) is the switching of the CPU from one process or thread to another.
- A *context* is the contents of a CPU's registers and program counter at any point in time

Steps

1suspending the progression of one process and storing the CPU's *state* (i.e., the context) for that process somewhere in memory

2retrieving the context of the next process from memory and restoring it in the CPU's registers

3returning to the location indicated by the program counter (i.e., returning to the line of code at which the process was interrupted) in order to resume the process.

Thread Scheduling

- **Two different thread implementations**
  - **"Native thread" implementation (e.g. Windows):**
    Performs time-slicing. Interrupts the running thread periodically to give other threads a chance to run.
  - **"Green thread" implementation (e.g. Solaris)**
    Does not perform time-slicing. It keeps a running thread active until a higher-priority thread awakes and takes control.

Sleep() vs. yield()

There is a big difference

- Calling sleep put the current running thread into the blocked state
- Calling yield does not put the calling thread, t1 into the blocked state
  - It merely let the scheduler kick in and pick another thread to run.
  - It might happen that the t1 is select to run again. This happens when t1 has a higher priority than all other runnable threads.

Problem-Solving1

In a non-preemptive system, can a process still go into the waiting state?

Problem-Solving2

Could we run multiple threads on a single core machine without pre-emption?

Problem-Solving3

Explain the difference between the following pairs, or justify why they are identical:

Threads and Processes
Interrupts and System calls
Single Core and Single Program System
fork() and start()

What controls Java Threads?

Problem-Solving4

Thread Concurrency –pseudo code

Deposit(amount)	Withdraw(amount)
Balance += amount;	Balance -= amount;

- Translate the deposit() and withdraw() code into pseudo-assembly
- When could a context switch / interrupt occur?
- Show how a deposit method can fail with two parallel threads
- Same for withdraw

References

- http//:courses.washington.edu/css430/syllabi/w05.html
- http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh's%20files/User%20and%20Kernel%20Level%20Threads.html
- http://www.pcguide.com/ref/hdd/if/ide/modesDMA-c.html
- www.cse.ust.hk/~liao/comp201/slides/24slide.ppt
- http://architects.dzone.com/articles/how-analyze-java-thread-dumps
- http://sivaeluri.blogspot.ca/2012/08/java-programming-interview-questions-2.html
- Source of sample questions: Prepared by: François Gingras Myriam Kharma
- https://www.tutorialspoint.com/operating_system
- http://www.geeksforgeeks.org/