Lonique Alexander  
 Home Work #5

Part 1: Processes and Job Control

Execute:

date& who; whoami; uname; echo Hello, World!&

Show the output of your command.

Part 2: Give an outline of the scheduling algorithms currently used in Windows XP, UNIX (Linux) and

Mac-OS.

Over view of the type of scheduling algorithms:

FCFS, SJF, Priority based scheduling, Round-robin scheduling, and multilevel queue scheduling, Multilevel Feedback-Queue Scheduling.

**UNIX**:

UNIX uses “multi-level feedback queue implementation”. Therefore, processes with higher priority have first access to CPU time. Priority of a process is based on the type of process, where the swapper has highest priority and user processes have lowest priority.

In addition, UNIX can use round robin; a process can be assigned one of three priority real-time (the highest priority and may pre-empt processes at any other level), kernel, and time shared.

**Mac OS**:

The Mac OS X uses a “multi-level feedback queue” as well UNIX. However, they have four priority bands for threads: normal, system high priority, kernel mode only, and real-time. Threads are scheduled preemptively.

**Windows XP**: “<http://www2.cs.uregina.ca/~hamilton/courses/330/notes/scheduling/scheduling.html>”

Windows XP uses a “quantum-based, preemptive priority scheduling algorithm”. Threads are scheduled rather than processes. So, each class of thread is normally restricted to a small band of 5 priority levels.

Preemption can occur for any of 4 reasons: higher-priority thread becomes ready, thread terminates, time quantum exhausted, thread performs a blocking system call, such as for I/O.

There are 32 priority levels ( 31 is the highest priority and priority 0 is the lowest priority). Threads in the real-time class have fixed priorities. The running thread is always one with the highest priority level.

Part 3 Java Threads:

In Java, write a program that creates three concurrent threads.

Each thread will keep printing a statement and sleep for a random time.

**public** **interface** Runnable {

**void** run();

}

/---------------------------------------------------------------------------/

**import** java.lang.Runnable;

**public** **class** RunnableThread **implements** Runnable{

**public** **static** **long** *startTime*=0;

Thread runner;

**public** RunnableThread () {

}

**public** RunnableThread (String threadName) {

runner = **new** Thread(**this**, threadName); // (1) Create a new thread.

System.*out*.println(runner.getName());

*startTime*=System.*currentTimeMillis*(); // get the start time of the thread

runner.start(); // (2) Start the thread.

}

**public** **void** run() {

//Display info about this particular thread

//while(true){

System.*out*.println(Thread.*currentThread*().getName() + " the time of thread "+ *age*()+"ms");

//System.out.println(Thread.currentThread());

//System.out.println(age());

//}

}

**protected** **static** **final** **long** age() {

**return** (System.*currentTimeMillis*() - *startTime*);

}

}

/---------------------------------------------------------------------------/

**import** java.util.Random;

**public** **class** ThreadsDemo {

**public** **static** **void** main(String [] args){

Thread thread1 = **new** Thread(**new** RunnableThread(), "thread1");

Thread thread2 = **new** Thread(**new** RunnableThread(), "thread2");

Thread thread3 = **new** Thread(**new** RunnableThread(), "thread3");

//Start the threads

thread1.start();

thread2.start();

thread3.start();

//

Random rand = **new** Random(16543);

**try** {

//delay for one second

**int** waitTime = rand.nextInt(1500);

**if**(Thread.*currentThread*().equals(thread1.*currentThread*())){

Thread.*currentThread*().*sleep*(waitTime);

System.*out*.println("hello"+thread1.getState()+", "+thread1);

}

**else** **if** (Thread.*currentThread*().equals(thread2.*currentThread*())){

System.*out*.println(thread1.getState()+", "+thread2);

}

**else**{

System.*out*.println(thread1.getState()+", "+thread3);

}

} **catch** (InterruptedException e) {System.*out*.println("Interrupted!");}

}

}

Sample output:

thread2 the time of thread 1302544671531ms

thread3 the time of thread 1302544671532ms

thread1 the time of thread 1302544671532ms

TERMINATED, Thread[thread1,5,]