Angel Guevara, Jr. 3-31-09

CS340 Assignment 5

* **Execute: date& who; whoami; uname; echo Hello, World!&**
  + **Output:**

[guan7019@venus ~]$ date& who; whoami; uname; echo Hello, World&

[1] 10619

Sun Mar 29 17:58:01 EDT 2009

guan7019 pts/13 Mar 29 17:42 (pool-96-250-18-22.nycmny.fios.verizon.net)

[1]+ Done date

guan7019

Linux

[1] 10623

[guan7019@venus ~]$ Hello, World

* **Give the default value for adjustments.**
  + **10**
* Part 2 – Scheduling Algorithms for XP, Linux, OSX.
  + *http://www2.cs.uregina.ca/~hamilton/courses/330/notes/scheduling/scheduling.html*
  + **Windows XP** – Uses a quantum-based, preemptive priority scheduling algorithm. Instead of scheduling processes, XP schedules threads. This algorithm can also be considered a multiple feedback-queue algorithm, since the preemptive priority algorithm implements multiple queues. Higher-priority threads becoming ready, terminating threads, exhausting the time quantum, and threads performing a blocking system call can result in preemption. A thread with a priority of 0 is at the lowest priority, while a thread with a priority of 31 is at the highest priority. Variable class of priorities range from 1 to 15, while real-time class of priorities range from 16 to 31. Threads in the real-time class have fixed priorities, and the thread that is currently running has the highest priority level. The idle thread is run if no ready thread exists. A thread’s priority is lowered when its time quantum runs out. When a thread’s state changes from waiting to ready, it’s priority is increased. Priority boosts are also given to processes that are currently interacting with the user.
  + **UNIX (Linux)** – Uses a preemptive priority algorithm with 140 possible priority values. A process’ current priority is determined from the process’ nice value set by the user (ranging from -20 to 19, going from highest to lowest). Real-time processes have performance guarantees from the operating system, and have fixed priorities. The scheduler uses a runqueue, which includes 2 groups of queues, one being an active priority array and the other being an expired priority array. At any given time, processes are being scheduled from one priority array while the other is accumulating ready processes. When the active priority array is empty, it is then turned into the expired priority array, while the other becomes active. There is a separate queue for every priority level within a priority array. Bit maps store information about which queues contain processes. Longer quanta are granted to the higher priorities, while shorter quanta are granted to the lower priorities. When the highest priority’s quantum is finished, it is then removed from the CPU and assigned a new priority, then placed into the other priority array. If the running process is preempted, it is put back on the front of the queue, and will run again when the CPU is free. After all the active runqueue’s processes had the chance to execute, the scheduler moves to the other runqueue. When a process changes from the waiting to the ready state, it is given a priority based on how long it was waiting. A longer wait time will yield a higher priority.
  + **Mac OSX** – <http://developer.apple.com/technotes/tn/tn2028.html>#MacOSXThreading  
    Mach threads represent the lowest level threading on the system. POSIX threads (pthreads) are layered on top of Mach thread. Cocoa threads (NSThreads) are layered directly on top of pthreads. Carbon MP tasks are layered on top of pthreads and Carbon Thread Manager cooperative threads are also layered on top of pthreads.

**import** java.util.\*;

**public** **class** HW5 **implements** Runnable

{

**protected** **static** **final** **void** age()

{

//initialize time variables and get the current time

**int** hr=0, min=0, sec=0;

Calendar now = Calendar.*getInstance*();

now.setTimeInMillis(System.*currentTimeMillis*());

//set the current time to variables

hr = now.get(Calendar.*HOUR*); **if**(hr==0) hr = 12;

min = now.get(Calendar.*MINUTE*);

sec = now.get(Calendar.*SECOND*);

//display the current time

**if**(min<10 && sec<10)

System.*out*.print(hr + ":0" + min + ":0" + sec + " - ");//12:01:01

**else** **if**(min<10 && sec>=10)

System.*out*.print(hr + ":0" + min + ":" + sec + " - ");//12:01:10

**else** **if**(min>=10 && sec<10)

System.*out*.print(hr + ":" + min + ":0" + sec + " - ");//12:10:01

**else** **if**(min>=10 && sec>=10)

System.*out*.print(hr + ":" + min + ":" + sec + " - ");//12:10:10

}

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

{

**while**(**true**)

{

*age*();//display the current time

//show which thread is currently running

System.*out*.println(Thread.*currentThread*().getName() + " is now running.");

**try**

{

//make the thread go to sleep for a random amount of time

Thread.*sleep*((**new** Random()).nextInt(10000));

//max = 10 sec, min = 0 sec

}

**catch** (InterruptedException e)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}//run

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

{

//initialize the threads

Thread a = **new** Thread(**new** HW5());

Thread b = **new** Thread(**new** HW5());

Thread c = **new** Thread(**new** HW5());

//start the threads

a.start();

b.start();

c.start();

}//main

}//hw5

Output:

10:33:23 - Thread-1 is now running.

10:33:24 - Thread-0 is now running.

10:33:26 - Thread-2 is now running.

10:33:27 - Thread-2 is now running.

10:33:28 - Thread-0 is now running.

10:33:30 - Thread-1 is now running.

10:33:31 - Thread-0 is now running.

10:33:32 - Thread-1 is now running.

10:33:33 - Thread-0 is now running.

10:33:34 - Thread-2 is now running.