CS 4/53201 # Homework #2 **Operating Systems**

Due in class on Monday 19 October 1998

1. **Consider the following code, with the assumption that each executes in a separate thread:**

```
i=0;                      i=0;
while (i < 10)            while (i > –10)
     i++;                      i—;
printf(“a wins”);         printf(“b wins”);
```

   a. **Is it <u>guaranteed</u> that someone wins? Explain.**

   No. If the left thread increments, then the right thread decrements before the left does a test, then the left thread increments before the right does a test, etc. they could keep going forever. Also, if the instructions that make up the increment and decrement interleave, the same thing could happen.

   b. **Is it <u>possible</u> for someone to win? Explain.**

   Yes. If one has a chance to get through the loop enough times while the other isn’t running (e.g., it’s on a faster processor in a multiprocessor system, or it finishes during one time slice while the other is waiting on the ready queue), it could win.

2. **Implementing semaphores by disabling interrupts has several problems, one of which is that it doesn’t work with on multiprocessors. Explain why this statement is true.**

When the semaphore code running on one CPU disables the interrupts for that CPU, it does not also disable the interrupts for other CPUs. Therefore the thread on that first CPU enters the critical section, but there’s nothing to prevent threads running on other CPUs from also running the semaphore code, disabling their own interrupts, and also entering the critical section.

3. **In several situations, such as when a thread tries to release a lock that it hasn’t acquired, it’s recommended that the Nachos ASSERT function be used to crash the operating system. Is this a good idea or a bad idea? Explain your answer.**

For Nachos, it may good idea. Otherwise, what should the OS do when a thread tries to release a lock it hasn’t acquired? Clearly there’s something wrong with the code that’s running, and since any code in Nachos is part of the OS, there’s a problem with what the OS is doing, so crashing it and making someone fix the problem may be reasonable.

However, in a real OS, this would be a very bad idea. Not only should user processes not have some way to crash the OS, but even if a part of the OS has problems, it should crash the entire OS. Instead, only the offending thread / process should be terminated.

2. (5.3 from OSC, parts (a) through (c) only, modified as follows) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P5, P4, P3, P2, P1, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum=1) scheduling.

P5	P4	P3	P2	P1
5	1	2	1	10

P4	P2	P3	P5	P1
1	1	2	5	10

P2	P5	P3	P1	P4
1	5	2	10	1

P5	P4	P3	P2	P1	P5	P3	P1	P5	P1	P5	P1	P5	P1	P1	P1	P1	P1	P1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

FCFS: P5 = 5, P4 = 6, P3 =8, P2=9, P1=19

SJF: P4 = 1, P2 = 2, P3 =4, P5=9, P1=19 (in order P5–P1: 9, 1, 4, 2, 19)

NP Priority: P2 = 1, P5 = 6, P3 =8, P1=18, P4=19 (in order P5–P1: 6, 19, 8, 1, 18)

RR: P5 = 13, P4 = 2, P3 =7, P2=4, P1=19

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

FCFS: P5 = 0, P4 = 5, P3 =6, P2=8, P5=9

SJF: P4 = 0, P2 = 1, P3 =2, P5=4, P1=9 (in order P5–P1: 4, 0, 2, 1, 9)

NP Priority: P2 = 0, P5 = 1, P3 =6, P1=8, P4=18 (in order P5–P1: 1, 18, 6, 0, 8)

RR: P5 = 8, P4 = 1, P3 =5, P2=3, P1=9

5. **(Exercise 5.6 from OSC) What advantage is there in having different time-quantum sizes on different levels of a multilevel queuing system?**

This allows better "tuning" of the scheduler. For example, short time slices could be given to high-priority interactive jobs, so that the user will see quick response, but long time slices could be given to low-priority CPU-bound jobs, so that once they start they will have a higher chance of finishing and exiting the system.