Austin Dailey
Prof. Nico
CSC 453
Lab 03

1. In Minix (or any other Unix), if user 2 links to a file owned by user 1, then user 1 removes the file, what happens when user 2 tries to read the file? (Tanenbaum and Woodhull, Ch. 1, Ex. 15)

When files are linked using a link in a Unix system, the new directory creates a new entry using the same i-number as the the file had previously, although it could have a new name. However, this is still a reference to the same file so changes from either user will be reflected on both directories. So if user 1 were to delete the file and user 2 tried to access it again, they would get an error because the file no longer exists and their reference to the file is invalid.

2. Under what circumstances is multiprogramming likely to increase CPU utilization? Why?

Multiprogramming can increase CPU utilization when processes are just waiting. For example, if a process is waiting to read data from the disk or somewhere else it can do nothing but wait. Without multiprogramming, the CPU would just have to sit idle during this time, but with multiprogramming, another process could be loaded in and ran during this time instead of sitting idle.

3. Suppose a computer can execute 1 billion instructions/sec and that a system call takes 1000 instructions, including the trap and all the context switching. How many system calls can the computer execute per second and still have half the CPU capacity for running application code? (T&W 1-21)

$$\text{Usable capacity} = \frac{1{,}000{,}000{,}000}{2} = 500,000,000\ instructions$$

$$\frac{500{,}000{,}000}{1000} = 500,000\ System\ calls\ per\ second$$

The system can make 500,000 system calls per second and still have half the CPU capacity

4. What is a race condition? What are the symptoms of a race condition?(T&W 2-9)

A race condition occurs when using multiprogramming and coming across some bad luck. Specifically when multiple processes are sharing some sort of data like a variable, chunk of memory, or a file. If a process A reads a variable as 1 and plans to change it after it reads, but gets interrupted before it can go through with the change, a second process may also read it as 1 even though A meant it to be read as 0. Symptoms of a race condition include errors that occur rarely because they are dependent on the exact timing certain processes take to run.

5. Does the busy waiting solution using the turn variable (Fig. 2-10 in T&W) work when the two processes are running on a shared-memory multiprocessor, that is, two CPUs, sharing a common memory? (T&W, 2-13)

No this solution does not work, if process A was much faster than process B, process A would eventually get blocked by the non-critical section of process B which is not idea and against OS principles. Depending on the processor and cache it may even fail to take turns if the cache does not update fast enough or correctly they could get stuck since this solution doesn't use atomic operations.

6. Describe how an operating system that can disable interrupts could implement semaphores. That is, what steps would have to happen in which order to implement the semaphore operations safely. (T&W, 2-10)

Being able to disable interrupts is key to a semaphore performance since semaphores rely on atomic actions meaning the read, write and sleep can be done in one action and are guaranteed to happen directly after one another. To implement this every time a semaphore function up or down is called, before a read or write the interrupts should be disabled. Once interrupts are disabled, the semaphore can then read the value, write to it if necessary, then put a process to sleep, or wake a process up if allowed based on the semaphore value. Once those actions have been completed then interrupts can be reenabled and the system can continue on normally.

7. Round robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this? (T&W, 2-25) (And what is the reason. "Yes" or "no" would not be considered a sufficient answer.)

If a process occurred twice in the list, it would have increased chances of being able to run again, essentially giving it a higher priority. Some reasons this could be useful is for tasks that desire low latency like IO tasks or just generally important tasks that the system has decided need to be completed faster than all other tasks.

8. Five batch jobs, A through E, arrive at a computer center, in alphabetical order, at almost the same time. They have estimated running times of 10, 3, 4, 7, and 6 seconds respectively. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the time at which each job is completed and the mean process turnaround time. Assume a 1-second quantum and ignore process switching overhead. (Modified from T&W, 2-28) (a) Round robin. (b) Priority scheduling. (c) First-come, first-served (given that they arrive in alphabetical order). (d) Shortest job first.

A: 10s, P: 3 B: 3s, P:5, C: 4s, P: 2, D: 7s P: 1, E: 6s P: 4

a.

Process	Running Order	Completion Time(seconds)
A	1	30
B	2	12
C	3	17

D	4	27
E	5	25

Mean Process Turnaround: 22.5s

b.

Process	Running Order	Completion Time(seconds)
A	3	19
B	1	3
C	4	23
D	5	30
E	2	9

Mean Process Turnaround: 16.8s

c.

Process	Running Order	Completion Time(seconds)
A	1	10
B	2	13
C	3	17
D	4	24
E	5	30

Mean Process Turnaround: 18.8s

d.

Process	Running Order	Completion Time(seconds)
A	5	30
B	1	3
C	2	7
D	4	20
E	3	13

Mean Process Turnaround: 14.6s