CS 252 00 : Spring 2012 : Quiz 1

Name:

Date:

1) **What is a process?**

For this question, many answers are valid. I am looking for an understanding that a process is a single instance of an executing program. A process has its own virtual address space, program counter, etc.

(For 2 and 3) Suppose you have three processes that are currently "ready", and will run for 10, 14, and 3ms, respectively, before blocking. Assume a time quantum of 10ms, and that context switches are effectively cost-free.

2) **What is the average service time using first come-first served scheduling?**

Average service time: average of the service times. Service time of a process: completion time minus arrival time. (Note this definition is somewhat vague – a service time sometimes refers to when the wait time completes. Asking for clarification is completely okay).

Number processes P1-P3 that take 10, 14, and 3ms, respectively. In FCFS, processes are serviced in turn. P1 completes in 10ms. P2 then gets to run, for 14ms, and completes after a total of 24ms. P3 finally gets to run, for 3ms, completing after a total of 27ms. The average service or completion time is therefore (10 + 24 + 27) / 3 = 61/3 (= 20.3333 ms). Note: writing out the underlined answer is sufficient for credit. Working it out may be harmful if you make a mistake and skip the expression. Partial credit is easier to give if I see your thinking.

3) **What is the average service time using round robin scheduling?**

In RR scheduling, P1 runs from time 0-10, completing (service time 10). P2 runs from 10-20, with 4 remaining. Then P3 runs from 20-23, completing (service time 23). P1 is now skipped, since it is completed, and P2 runs from 23-27, completing (service time 27). So the average service time is (10 + 27 + 23) / 3 = 60 / 3 (= 20 ms). In this case, RR mainly helps because shorter processes do not get blocked excessively behind the longer earlier processes. It is left as an exercise to the student to imagine other scenarios where RR makes a bigger difference. For testing purposes, students may also wish to consider the potentials of context switching overhead, impact of selected time

quantum, and other definitions of "fairness" or "ideal scheduling". Try re-doing the problem if the time quantum is 5ms or the context switching overhead is 1ms. Under what circumstances does RR actually do worse?

4) **In the following code, how many processes will print hello world?**

```
1: int ret = fork();
2: ret = ret && fork();
3: if (ret == 0) {
4:   fork();
5: }
6: printf("Hello World!\n");
```

In line 1, process 1 creates process 2. (process ids are for illustration only). In the parent process (process 1), ret is 2, while in the child process (process 2), ret is 0. So in line 2, the parent process creates process 3, while process 2 does not call fork() as ret is 0 and && is a Boolean operator that is short circuited. Ret is '0' then, in process 2 and 3 (both have been the child of a fork(), that returns 0 in the child). In process 1, ret is 1 (2 && 3 = 1). In line 3, processes 2 and 3 continue to line 4, while process 1 skips it. Processes 2 and 3 then create processes 4 and 5, for a total of 5 processes. All processes then print "Hello World!\n".

Note: if you, on your solution, would have not realized that ret && fork() would prevent executing fork() when ret was false, you would have come up with 7 processes instead, as in line 2 another process would have been created, which would have ret, which would in turn lead to another process with ret of 0, creating another process to execute line 4. I would have accepted 7, provided you had an explanation of how you reached that answer.

5) **In which segment of a binary do you usually find the instructions of a program?**

The text segment.

6) **Executing "ls –l" requires more disk I/Os than execting "ls". Why?**

Because 'ls –l' is a "long listing", which includes file size, owner, last modified date, etc. This information is obtained from the file inodes. A simple directory listing need only read the directory contents (the contents of the file listing the directory entries).

7) **Suppose a directory has 10 files in it. Using the basic UNIX filesystem described in class, how many disk I/Os will it take to execute "ls –l" ?**

Beyond the disk I/Os for listing the directory entries, it will require reading the inode for each file in the directory. This means reading ~2 disk blocks for the directory (one for the directory inode, and one for the data block containing the directory entries), and then another ~10 disk blocks – one for each file, for approximately 12 disk reads. These numbers are approximate, since inodes are not necessarily a full disk-block in size, and therefore filesystems bunch inodes together. Moreover, we did not discuss the specific layout of a directory file – so we are assuming here one for the inode and one for the data. In grading this problem, I would mainly be looking for acknoweldgement that you need to read each of the inodes for each of the 10 files. So answers like 11 or 12 would very likely receive full credit, provided they are properly justified. Similarly, simply stating that you would need to read the directory listing from the disk, then each of the 10 inodes from the disk, would also receive full credit.

8) **If a machine has 2 CPUs, how many processes can consecutively be in the "running state"? How about the ready state?**

2 can be in the running state – one per CPU. For counting how many can be in the ready state, I would have accepted a variety of answers, such as (1) the OS limit, if it contains a hard limit; (2) limited by the number of processes that can fit in memory / that exhausts some system resource; and (3) any number. For the grading of this problem, I am looking for an understanding that the number of processes in the ready state is not related to the number of CPUs. It's also true that two other things might affect the number in the "running" state. First – the OS should be configured to support multiple CPUs – otherwise the limit would be 1. Second, if hyperthreading were enabled (we haven't discussed what hyperthreading is), the answer would actually be 4. Both of these answers, if properly justified, could receive credit. However, they are not answering the intended question. 2 is the preferred answer for the maximum number of processes in the running state.