Midterm Examination
CS 111, Winter 2020
2/5/2020, 2 – 3:50pm

Name: _____ Student ID: _____

This is a closed book, closed notes test. One double-sided cheat sheet is allowed.

1. What is the benefit of using the copy-on-right optimization when performing a *fork* in the Linux system? **8 points**

With copy-on-right, when a fork occurs, the parent process's pages are not copied for the child process. Instead, the pages are shared between the child and the parent process. Whenever a process (parent or child) modifies a page, a separate copy of that particular page alone is made for that process (parent or child) which performed the modification. This process will then use the newly copied page rather than the shared one in all future references. The other process (the one which did not modify the shared page) continues to use the original copy of the page (which is now no longer shared). This saves a lot of unnecessary data copying effort.

2. Round Robin, First come First Serve, and Shortest Job First are three scheduling algorithms that can be used to schedule a CPU. What are their advantages and disadvantages? Which one is likely to have the largest overhead? Why? **10 points**

Round robin: ensures fairness but introduces much context switch overhead
First come first serve: easy to implement, but does not provide any guarantee
Shortest Job First: reduces average wait time, but does not provide fairness

Round robin has the largest overhead.

3. Assume you have a system with three processes (X, Y, and Z) and a single CPU. Process X has the highest priority, process Z has the lowest, and Y is in the middle. Assume a priority-based scheduler (i.e., the scheduler runs the highest priority job, performing preemption as necessary). Processes can be in one of five states: RUNNING, READY, BLOCKED, not yet created, or terminated. Given the following cumulative timeline of process behavior, indicate the state the specified process is in AFTER that step, and all preceding steps, have taken place. Assume the scheduler has reacted to the specified workload change. **15 points**

For all questions in this Part, use the following options for each answer:
a. RUNNING
b. READY
c. BLOCKED
d. Process has not been created yet
e. Not enough information to determine
OR None of the above

(a) Process X is loaded into memory and begins; it is the only user-level process in the system. Process X is in which state?

Running. X is only process so it will be scheduled.

(b) Process X calls fork() and creates Process Y. Process X is in which state?

X: Running. X is highest priority process so it is scheduled
Y: Ready. Y could be scheduled, but it is lower priority than X.

(c) The running process issues an I/O request to the disk. Process X is in which state? Process Y is in which state?

X: Blocked. X is waiting for I/O to complete.
Y: Running. Y is now the only available ready process to schedule.

(d) The running process calls fork() and creates process Z. Process X is in which state? Process Y is in which state? Process Z is in which state?

X: Blocked. X is waiting for I/O to complete.
Y: Running. Y is higher priority than Z
Z: Ready. Z is lower priority than Y

(e) The previously issued I/O request completes. Process X is in which state? Process Y is in which state? Process Z is in which state?

X: Running. X is ready and at higher priority than others.
Y: Ready. Y could be scheduled, but it is lower priority than X
Z: Ready. Z is lower priority than X.

4. For the next two questions, assume the following code is compiled and run on a modern linux machine (assume any irrelevant details have been omitted): **16 points**

```
main() {
    int a = 0;
    int rc = fork();
    a++;
    if (rc == 0) { rc = fork(); a++; }
    else { a++; }
    printf("Hello!\n");
    printf("a is %d\n", a);
}
```

(a) Assuming fork() never fails, how many times will the message "Hello!\n" be displayed? Explain how you get this result. 8 points

After first fork(): 2 processes. Only child calls second fork() creating a third process (call 'grandchild').

(b) What will be the largest value of "a" displayed by the program? Explain how you get this result. 8 points

Parent process: a = 0; a++; a++; => a=2.
Child process: a=0 (after fork()); a++; a++; => a=2.
Grandchild: a=1 (after fork()); a++; => a =2.

5. **Scheduling.** Consider the following set of processes, with associated processing times and priorities:

Process Name	Processing Time	Priority
A	4	3
B	1	1
C	2	3
D	1	4
E	4	2

For each scheduling algorithm, fill in the table with the process that is running on the CPU (for time slice-based algorithms, assume a 1 unit time slice). **18points**

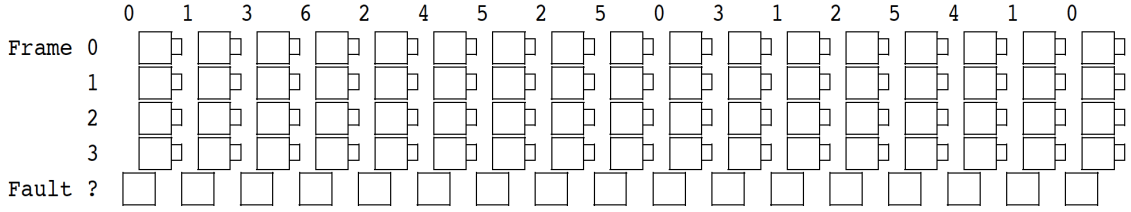
Notes: • A smaller priority number implies a higher priority.
• For RR and Priority, assume that an arriving thread is run at the beginning of its arrival time, if the scheduling policy allows it.

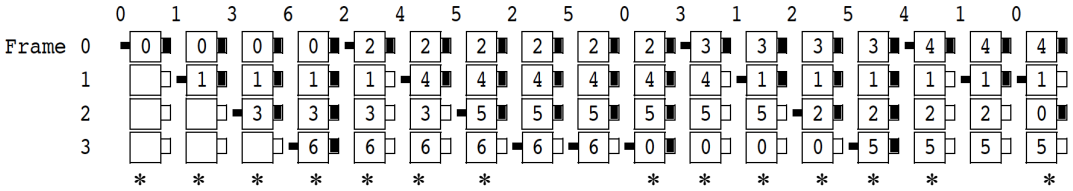
• All of the processes arrive at time 0 in the order Process A, B, C, D, E.
• Assume the currently running thread is not in the ready queue while it is running.
• Turnaround time is defined as the time a process takes to complete after it arrives
• SRTF, known as shortest remaining time first (SRTF), is another scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute.

Time	FIFO	Round Robin	SRTF	Priority
0	A	A	B	B
1	A	B	D	E
2	A	C	C	E
3	A	D	C	E
4	B	E	A	E
A	C	A	A	A
6	C	C	A	A
7	D	E	A	A
8	E	A	E	A
9	E	E	E	C
10	E	A	E	C
11	E	E	E	D
Average turnaround time	((4-0)+(5-0)+(7-0)+(8-0)+(12-0))/5 = 7.2	((11-0)+(2-0)+(7-0)+(4-0)+(12-0))5 = 7.2	(8+1+4+2+12)/ 5 = 5.4	(9+1+11+12+5) /5= 7.6

6. **Clock Algorithm**. The clock algorithm is an approximation of LRU based on using one use bit for each page. When a page is used its use bit is set to 1. We also use a pointer to the next victim, which is initialized to the first page/frame. When a page is loaded, it is set to point to the next frame. The list of pages is considered as a circular queue. When a page is considered for replacement, the use bit for the next victim page is examined. If it is zero [that page is replaced] otherwise [the use bit is set to zero, the next victim pointer is advanced, and the process repeated until a page is found with a zero use bit].

Consider the reference string shown along the top of the following graphical structure. The system has four frames. Use the clock algorithm described in the previous paragraph. The narrow boxes to the right of the page number boxes can be used to keep up with use bits. Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur. **16points**





There are totally 14 faults.

7. In the early 1990s, SUN Microsystems, the maker of the Solaris Operating System, wanted to move from the engineering desktop, where it was well established, to a broader market for personal productivity tools. The best personal productivity tools were all being written for Windows platforms, and SUN was on the wrong side of the applications/demand/volume cycle, which made getting those applications ported to Solaris a non-option. **17 points**

   One approach to their problem was to modify the version of Solaris that ran on x86 processors (the popular hardware platform for Windows) to be able to run Windows binaries without any alterations to those binaries. This would allow Sun to automatically offer all of the great applications that were available for Windows.

   (a) What would have to be done to permit Windows binaries to be loaded into memory and executed on a Solaris/x86 system? 4 points

   1. A translator that supports the windows ABI – it can recognize the Windows load module and provide translation between Windows and Solaris.
   2. A system call emulator that emulates windows system calls; the user-level instructions can still be executed directly on x86.

   (b) What would have to be done to correctly execute the system calls that the Windows programs requested? 3 points

   A new 2nd level trap handler would be written to intercept the Windows system calls, and pass it on to an emulation layer, which would try to simulate the effects of each Window's system call, using Solaris mechanisms.

   (c) How good might the performance of such a system be? Justify your answer. 3 points

   User-level instructions do not need to be emulated; system calls do. The performance depends on the number of system calls made in the program.

   (d) List another critical thing, besides supporting a new load module format and the basic system calls, that the system would have to be prepared to simulate? How might that be done? 3 points

We would also have to emulate the functionality of the Windows device drivers (specifically displays and printers). This could be extremely complex. It might be easier to provide our own DLLs to directly implement the higher level functionality on a Solaris display server. Other answers to this part could deal with the Windows registry or special Windows networking code. Other answers are also possible.

(e) Could a similar approach work on a Solaris/PowerPC or Solaris/SPARC system? Why or why not? 3 points

No. Now all instructions need to be emulated due to the different ISAs used.