65

California State University, Sacramento
CSC 139 Operating System Principles
Section 6, Midterm Exam 1, Spring 2019

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name: Amit Singh

Rating of Difficulty: 10/10

Multiple Choice (3 points each)

1. For a single-processor system,
   - A. processes spend long times waiting to execute.
   - B. there will never be more than one running process.
   - C. input-output always causes CPU slowdown.
   - D. process scheduling is always optimal.

2. What is a trap/exception?
   - A. hardware generated interrupt caused by an error
   - B. software generated interrupt caused by an error
   - C. user generated interrupt caused by an error
   - D. None of the above

X

3. The major difficulty in designing a layered operating system approach is
   - A. making sure each layer is easily converted to modules.
   - B. making sure that each layer hides certain data structures, hardware, and operations.
   - C. debugging a particular layer.
   - D. appropriately defining the various layers.

27

4. The text segment of a process address space contains
   - A. the statically allocated data associated with the process.
   - B. the dynamically allocated data associated with the process.
   - C. the executable code associated with the process.
   - D. the inter-process communication (IPC) messages for the process.
   - E. all of the above

5. Which of the following is *not* true about message passing?
   - A. In direct communication, the sending process specifies the receiving process (usually by ID).

B. In indirect communication, the sending process specifies a mailbox rather than a process.

C. In indirect communication a link may be associated with more than two processes.

D. In direct communication, multiple links may exist between a pair of processes.

6. What is the READY state of a process?

   A. When process is scheduled to run after some execution
   B. When process is unable to run until some task has been completed
   C. When process is using the CPU
   D. None of the above

7. Which is true about processes and threads?

   A. Threads in a process share the same stack.
   B. Threads in a process share the same file descriptors.
   C. Threads in a process share the same register values.
   D. Threads in a process share the same program counter.

8. The multithreading model supported by the Linux operating system is D.

   A. Many-to-One
   B. One-to-One
   C. Many-to-Many
   D. All of the above
   E. None of the above

9. In a system where round robin is used for CPU scheduling, which of the following is true when a process cannot finish its computation during its current time quantum?

   A. The process will terminate itself.
   B. The process will be terminated by the operating system.
   C. The process's state will be changed from running to ready.
   D. None of the above

10. When a process is accessing its heap space, it exists in the A.

   A. Running state
   B. Waiting state
   C. Terminating state
   D. Ready state

11. Which of the following scheduling algorithms will have the longest average response time after many jobs are queued and ran to completion?

   A. Round Robin with a quantum of much less than the shortest job

X B. Shortest Job First (SJF)

   C. Preemptive Shortest Job First (PSJF)

   D. First Come First Serve (FCFS)

12. Which of the following is true about multilevel queue scheduling?

   A. Processes can move between queues.

   B. Each queue has its own scheduling algorithm.

   C. A queue cannot have absolute priority over lower-priority queues.

   D. It is the most general CPU-scheduling algorithm.

13. Which of the following statements is *false* with regards to the Linux CFS scheduler?

   A. Each task is assigned a proportion of CPU processing time.

   B. Lower numeric values indicate higher relative priorities.

   C. There is a single, system-wide value or `vruntime`.

   D. The scheduler doesn't directly assign priorities.

True or False (2 points each)

X 14. True/False __T__ Non-preemptive scheduling algorithms are better for interactive jobs since they tend to favor jobs that require quick responses.

15. True/False __T__ An interrupt vector contains the addresses of the handlers for the various interrupts.

X 16. True/False __T__ System calls can be run in either user mode or kernel mode.

17. True/False __F__ Each thread of a process has its own virtual address space.

X 18. True/False __F__ All processes in UNIX first translate to a zombie process upon termination.

19. True/False __F__ In round robin scheduling, it is advantageous to give each I/O bound process a longer quantum than each CPU-bound process (since this has the effect of giving the I/O bound process a higher priority).

20. True/False __F__ Processes in a microkernel architecture operating system usually communicate using shared memory.

Short Answer (5 points each)

21. What needs to be saved and restored on a context switch between two threads in the same process? What if two are in different processes? Be brief and explicit.

in the same process nothing ~~needs~~ to be saved or restored.

3

when CPU switches to a diff process the system must save the old process and restore the saved state

what are they?

22. Describe the difference between fork() and exec().

exec() system call follows the fork(), exec() overwrites a process and fork() creates a new process.

5

23. Why would two processes want to use shared memory for communication instead of using message passing?

- shared memory is faster than message passing.
- message passing is implemented using system calls.

5

5

24. Describe the difference between "preemptive" scheduling and "non-preemptive" scheduling.

Preemptive scheduling is when a running process may be forced to release the CPU even though it is not yet completed or blocked while in non-preemptive scheduling each running process keeps the CPU until it completes or it switches to the waiting state.

California State University, Sacramento
CSC 139 Operating System Principles
Section 6, Midterm Exam 1, Spring 2019 (Continued)

Long Questions

25. (15 points) Here is a table of processes and their associated arrival and CPU burst times.

QT = 1

Process ID	Arrival Time	CPU Burst Time
Process 1	0	5
Process 2	1	5
Process 3	5	3
Process 4	6	2

(a) Draw Gantt charts for these processes under First Come First Serve (FCFS), Shortest Job First (SJF), and Round Robin (RR) with a quantum = 1 time unit. Assume that the context switch overhead is 0 and new processes are added to the *head* of the queue except for FCFS.

FCFS

P1	P2	P3	P4
0	5	10	13

15 ✓

SJF

~~P4~~	~~P3~~	P1	P2
0	2	5	10

15

P4 arrives at time 6

RR

P1	P2	~~P3~~	~~P4~~	P1	P2	P3	P4	P1	P2	P3	P1	P2	P1	P2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

15

P3 and P4 are not available at time 2 and 3

6

(b) For each process in each schedule above, show the waiting time and turnaround time. The waiting time is the total time a process spends in the wait queue (a.k.a. ready state). The turnaround time is defined as the time a process takes to complete after it first arrives.

	Wait	Turnaround
FCFS	$\frac{0+5+10+13}{4}$ $\frac{28}{4} = 7$	$\frac{5+10+13+15}{4} = \frac{43}{4} = 1.75$
SJF	$\frac{0+2+5+10}{4} = \frac{17}{4} = 4.25$	$\frac{2+5+10+15}{4} = \frac{32}{4} = 8$
RR	$\frac{0+1+2+3+4+5+6+7+8+9+10+11+12+13+14}{15}$ $= \frac{105}{15} = 7$	$\frac{1+2+3+4+5+6+7 \ldots + 15}{15}$ $= \frac{120}{15} = 8$

26. (6 points) Assume the following code is compiled and run on a modern Linux machine.

```
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!" be displayed?

once

(b) What will be the largest value of a displayed by the program?

a++
a++
a++

2

4

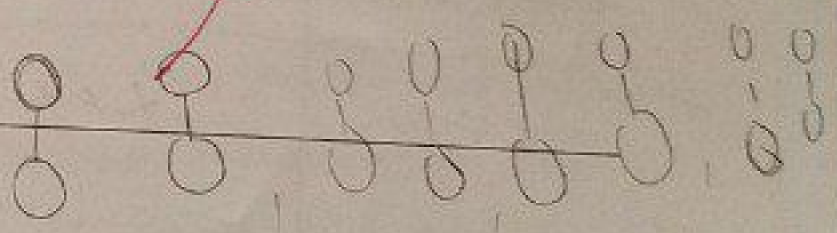
27. (6 points) What is the number of child processes created? Draw a simple tree diagram to show the parent-child hierarchy of the spawned processes.

```
#include <stdio.h>
#include <unistd.h>

int main() {
    int i;
    for (i = 0; i < 3; i++) {
        fork();
        fork();
    }
    return 0;
}
```

$2^3 = 8$ processes

2



End of exam.