Southern University of Science and Technology
College of Engineering
Department of Computer Science and Engineering – CSE

Spring 2019 Dr. Bo Tang & Ms. Yun Shen

# First Midterm Exam

March 27, 2019
CS302 Operating Systems

| | |
|---|---|
| **Your Name:** | |
| **Student ID:** | |

General Information:
This is a **closed book and one 2-sided handwritten note (A4-size)** examination. You have 120 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it! Good Luck!

| **QUESTION** | **POINTS ASSIGNED** | **POINTS OBTAINED** |
|---|---|---|
| **P1: CPU scheduling** | **14** | |
| **P2: Multi-threading** | **15** | |
| **P3: System call** | **21** | |
| **P4: Short answer** | **30** | |
| **P5: Multiple choices** | **20** | |
| **TOTAL** | **100** | |

**P1 (14 points) CPU Scheduling.**
Consider the following **single-threaded** processes, and their arrival times, estimated CPU costs and their priorities (a process with a higher priority number has priority over a process with lower priority number):

| Process | Estimated CPU Cost | Arrives | Priority |
|---|---|---|---|
| A | 4 | 1 | 1 |
| B | 1 | 2 | 2 |
| C | 2 | 4 | 4 |
| D | 3 | 5 | 3 |

Please note:

- Priority scheduler is preemptive.
- Newly arrived processes are scheduled last for RR. When the RR quanta expires, the currently running thread is added at the end of to the ready list before any newly arriving threads.
- Break ties via priority in Shortest Remaining Time First (SRTF).
- If a process arrives at time x, they are ready to run at the beginning of time x.
- Ignore context switching overhead.
- The quanta for RR is 1 unit of time.
- Average turn-around time is the average time a process takes to complete after it arrives.
- Highest response ratio next (HRRN) is a non-preemptive scheduling algorithm. In HRRN, the next job is not that with the shorted estimated run time, but that with the highest response ratio defined as: 1 + waiting time / estimated CPU time.

Given the above information please fill in the following table.

| Time | HRRN | FIFO/FCFS | RR | SJF | Priority |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| Avg.Turn-around Time | | | | | |

**P2 (15 points) Multithreading.**

Consider the following two threads, to be run concurrently in a shared memory (all variables are shared between the two threads):

| Thread A | Thread B |
|---|---|
| for (i=0; i<5; i++) {<br>    x = x + 2;<br>} | for (j=0; j<5; j++) {<br>    x = x + 3;<br>} |

Assume a single-processor system, that load and store are atomic, that x is initialized to 0 *before either thread starts*, and that x must be loaded into a register before being incremented (and stored back to memory afterwards). The following questions consider the final value of x after both threads have completed.

a) (3 points) Give a *concise* proof why x≤25 when both threads have completed.

b) (3 points) Give a *concise* proof why x≠2 when both threads have completed.

c) (3 points) Suppose we replace '**x = x+3**' in Thread B with an atomic double increment operation **atomicIncr2(x)** that cannot be preempted while being executed. What are all the possible final values of x? Explain.

d) (3 points) What needs to be saved and restored on a context switch between two threads in the same process? What if the two threads are in different processes? Be explicit.

e) (3 points) Under what circumstances can a multithreaded program complete more quickly than a non-multithreaded program? Keep in mind that multithreading has context-switch overhead associated with it.

**P3 (20 points, system call) (a) [4pts]:** Assume that we have the following piece of code:

```
int main() {
    printf("Starting main\n");
    int file_fd = open("test.txt", O_WRONLY | O_TRUNC | O_CREAT, 0666);
    dup2(file_fd, STDOUT_FILENO);
    pid_t child_pid = fork();
    if (child_pid != 0) {
          wait(NULL);
          printf("CS302 SUSTech, in parent\n");
    } else {
          printf("CS302 SUSTech, in child\n");
    }
    printf("Ending main: %d\n", child_pid);
}
```

What is the output of this program? You can assume that no syscalls fail and that *the child's PID is 123.* Fill in the following table with your prediction of the output:

| Standard out | test.txt |
|---|---|
| | |

**(b) [4pts]:** Next, assume that we have altered this code as follows:

```
int main() {
    printf("Starting main\n");
    int file_fd = open("test.txt", O_WRONLY | O_TRUNC | O_CREAT, 0666);
    int new_fd = dup(STDOUT_FILENO);
    dup2(file_fd, STDOUT_FILENO);
    pid_t child_pid = fork();
    if (child_pid != 0) {
          wait(NULL);
          printf("CS302 SUSTech, in parent \n");
    } else {
          dup2(new_fd, STDOUT_FILENO);
          printf("CS302 SUSTech, in child \n");
    }
    printf("Ending main: %d\n", child_pid);
}
```

What is the output this time? Fill in the following table with your prediction of the output:

| Standard out | test.txt |
|---|---|
| | |

**(c) [5pts]:** Consider the following fragment of a program (which is missing lines of code). Fill in the blanks in the code to ensure that the output of this program is always the following two lines in the following order (under all interleavings or schedules):

```
Val = 0
Val = 1
```

*No more than one function call (or system call) per blank! Your solution is not allowed to make assignments to the "`val`" variable. Also, no use of `printf()` or other print statements!*

```
 void apple () {
         pid_t oval = _________________________________;
         kill(oval, SIGTEST);
 }

 void orange(int i) {

         val = 1;
 }
int val = 0;
int main() {
         pid_t pid = fork();
         ______________________________________;

         if (pid == 0) {
              ___________________________________;

         } else {
              ___________________________________;
         }
          printf("val=%d\n",val);
 }
```

**(d) [3pts]:** If a child process sends a SIGKILL signal to its parent, what happens to the parent process and the child process? Can the parent prevent this from happening?

**(e) [2pts]:** Consider the following scenario. A process forks a child process and the parent process waits on it. Then, the child exits normally and the parent is unblocked. Finally, the parent makes another wait call on the child process. What happens to this last wait call and to the parent process?

**(f) [3pts]:** Why do we switch from the user's stack to a kernel stack when we enter the kernel (e.g. for a system call)? Why do we associate a *unique* kernel stack for *each* user thread?

**P4 (30 points, 6 points for each problem) Short answers, please answer the question within 6 sentences or less**

(a)What is the difference between "Parallelism" and "Concurrent" in job scheduling.

(b) Please draw the life cycle of process and thread, then compare the different of process and thread

(c) Please describe the actions in operating system when "fork()" function executed.

(d) Please list the storage unit with its access latency in a modern commodity personal computer (as many as possible), and analyze the underlying trade-off strategy in current storage system.

(e) Please give the clear definition of user time, system time and real time, then give two concrete examples to show 1) real time < user time + system time and 2) real time > user time + system time.

## P5 Multiple Choice (20 points, 1 for each question)

1. In which condition(s), a new process will be created in Linux?
A. User login by remote ssh    B. execute `ls` on a shell
C. call exec*() in a program    D. call pthread_create() in a program

2. Which of follows is(are) not shared by different thread in the same process?
A. PID    B. File descriptor    C. Execution Stack    D. PC

3. Which threading model(s) is(are) not exist?
A. One to many    B. Many to many    C. Many to one    D. One to one

4. Which system calls are employed to implement a C libary call "system()"?
A. fork()    B. exec*()    C. wait()    D. pipe()

5. () is the interface that Operating System provides to application.
A. system call    B. interrupt    C. library function    D. atomic operation

6. Which is the proper time to reduce the priority of process?
A. the time slice of the process is used up.
B. the process just finished I/O and enter into ready queue.
C. the process stays in ready queue for a long time.
D. the state of the process changes from ready to running.

7. In following options, () is the scheduling algorithm that satisfies Shortest Job First(SJF) and will not cause starvation.
A. First Come First Serve(FCFS)    B. Highest Response Ratio Next(HRRN)
C. Round-robin    D. Non-preemptive SJF

8. Which is executed during user space?
A. command line interpreter (shell)    B. page fault handler
C. process scheduler    D. clock interrupt handler

9. Which does not happen in user space?
A. system call B. external interrupt C. context switch    D. page fault

10. Both interrupt handling and subroutine calling need protection by stack pushing. Which is the content that interrupt handling has to save but subroutine calling not?
A. PC    B. flag register    C. data register    D. address register

11. If a user process read data from certain disk file by read() system call, then which of the following statement(s) is correct?
    I. If the data of file does not exist in main memory, then this process will change to waiting state
    II. Invoking read() system call will lead CPU change from user space to kernel space

III. The argument of read() should include file name

A. Ⅰ and Ⅱ B. Ⅱ C. Ⅲ D. Ⅰ, Ⅱ and Ⅲ

12. In a multiprogramming system, only two jobs P1 and P2 exist. The arrival time of P2 is 5ms later than that of P1, and their orders of CPU computation and I/O operation are shown as follows:

P1: Computation 60ms, I/O 80m, Computation 20ms

P2: Computation 120ms, I/O40ms, Computation 40ms

If not considering scheduling and switch time, then the least time to finish both jobs is ()

A. 240ms B. 260ms C. 340ms D. 360ms

14. In a single-processor multi-process system, if there are several ready processes, then the wrong statement about process scheduling is ().

A. process scheduling can execute when a process terminates

B. process scheduling can execute when a process creates

C. process scheduling can execute when a process is in critical section

D. process scheduling can execute when system call finished and system returns to user space

15. For the statements about thread and process, which one is correct?

A. No matter the system supports threads or not, process is the basic unit of resource allocation.

B. Thread is the basic unit of resource allocation and process is the basic unit of scheduling.

C. the switches of user-level and kernel-level threads all need the support of kernel.

D. every thread in the same process has distinct address space.

16. Which will lead a user process switch from user space to kernel space?

I. division by zero II. sin() function call III. read() system call

A. I and II B. I and III C. II and III D. I, II and III

17. After computer boosting, Operating System will be finally loaded to ()

A. BIOS B. ROM C. Disk D. RAM

18. A system is running three processes P1, P2 and P3, the time ratios of CPU computation and I/O are shown as follows

| Process | CPU time | I/O time |
|---|---|---|
| P1 | 90% | 10% |
| P2 | 50% | 50% |
| P3 | 15% | 85% |

To improve the efficiency of resource usage, which priority setting for processes is more reasonable?。

A. P1 >P2 >P3 B. P3>P2 >P1 C. P2>P1 =P3 D. P1>P2=P3

19. Suppose the following instructions have been loaded to instruction registers, which instruction cannot lead CPU to switch from user space to kernel space?
    A. DIV R0， R1; // (R0)/(R1) → R0
    B. INT n ； // generate soft interrput
    C. NOT R0 ； // do NOT operation for contents in R0
    D. MOV R0,addr ； // move the data in addr from main memory to R0

20. Which will lead the state of a process to change from running to ready?
    A. execute wait() B. fail to malloc()
    C. start I/O device D. preempt by high priority process

[This page intentionally left blank]