NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

CS2106—Introduction to Operating Systems

Semester 1: 2021/2022

Time Allowed: 1 hour

Instructions to students:

1. This assessment paper contains TWO sections with SEVENTEEN questions.
2. Section 1 comprises of **14 MCQ questions to be answed in the MCQ form**. The MCQ form should be filled out using pencil.
3. Section 2 contains **THREE short-answer questions to be answered in this booklet.**
4. Section 2 questions must be answered in the space provided; no extra sheets will be accepted as answers.
5. This is an OPEN BOOK assessment.
6. You are allowed to use pencils.
7. Using pencil, shade the MCQ form with your student number.
8. Using ink or pencil, write down your **student number** and shade the corresponding circle **completely** in the grid for each digit or letter. DO NOT WRITE YOUR NAME!

STUDENT NO:

STUDENT NUMBER

A								
U ○	0	0	0	0	0	0	0	A N
A ●	1	1	1	1	1	1	1	B R
HT ○	2	2	2	2	2	2	2	E U
NT ○	3	3	3	3	3	3	3	H W
	4	4	4	4	4	4	4	J X
	5	5	5	5	5	5	5	L Y
	6	6	6	6	6	6	6	M
	7	7	7	7	7	7	7	
	8	8	8	8	8	8	8	
	9	9	9	9	9	9	9	

Part A: MCQ questions (2 marks each question)

Questions 1-14 from this section should be answered using the MCQ bubble form provided. Answers given in the midterm paper will not be considered for grading.

1. Which of the statement(s) is/are TRUE regarding OS, user programs, and hardware?
   (i) You need an OS to run any program.
   (ii) An OS manages hardware resources for user programs.
   (iii) An OS can protect a user program from other malicious applications.

   A. (i), (ii), and (iii)
   B. (i) and (ii)
   C. (ii) and (iii)
   D. (iii) only
   E. None of the above

2. Which of the following statement(s) regarding the difference between library calls and system calls is/are TRUE?
   (i) Library calls are programming language dependent, while system calls are dependent on the operating system.
   (ii) A system call may be invoked during a library call.
   (iii) If a system call and library call both execute approximately the same code, the library call is likely to take a longer amount of time to complete, as the system call is lower-level.

   A. (i) only.
   B. (i) and (ii) only.
   C. (ii) and (iii) only.
   D. (i), (ii) and (iii).
   E. None of the above.

3. Consider the following program:

```
Line#  Code
1      int main() {
2         f1(10);
3      }
4
5      void f1(int x) {
6          if (x == 0) {
7              /* How many stack frames? */
8              return;
9          }
10         return f2(x);
11     }
12
13     void f2(int x) {
14         f1(x/2);
15         return;
16     }
```

Given that before main() calls f1(10), there is one stack frame (for the main() function) on the stack, how many stack frames are on the stack at line 7?

A. 5
B. 6
C. 9
D. 10
E. None of the above

4. Consider the following code:

Line#	Code
1	`while (fork() != 0) {`
2	`   execl("a.out", "a.out", NULL);`
3	`   execl("b.out", "b.out", NULL);`
4	`}`

Recall that a “fork bomb” occurs when more and more processes are uncontrollably created. Assume that the code compiles correctly, and assume if a.out and b.out are valid executables, they simply print a line and exit. Which statement is TRUE?

A. When a.out is a valid executable and b.out is invalid, this code is a “fork bomb”
B. When b.out is a valid executable and a.out is invalid, this code is a “fork bomb”
C. Regardless of whether a.out or b.out are valid executables, this code is a “fork bomb”
D. Regardless of whether a.out or b.out are valid executables, this code is NOT a “fork bomb”
E. None of the above

5. Consider the following code:

Line#	Code
1	`int globalVar = 0;`
2	
3	`for (int i = 0; i < 1000; i++) {`
4	`  if (fork() == 0) {`
5	`    globalVar++;`
6	`    break;`
7	`  } else {`
8	`    globalVar--;`
9	`  }`
10	`}`
11	`// Point α`

Note that `break` exits the closest enclosing loop. What is/are possible final value(s) of `globalVar` at Point α?

A. 1000
B. -1000
C. 990
D. 0
E. All of the above

6. Which of the following statement(s) about zombie and orphan processes is/are TRUE?
   (i) A process after exit() will always transition to zombie state.
   (ii) If the parent process calls wait() (and wait() returns successfully) after forking exactly one child, its child process will never become an orphan.
   (iii) A process can stay in zombie state forever.

A. (i), (ii), and (iii)
B. (i) only
C. (i) and (ii)
D. (ii) and (iii)
E. None of the above

7. A process executes the following code:

Line#	Code
1	**`fork();`**
2	**`if (fork() != 0) { exit(0); }`**
3	**`fork();`**
4	**`fork();`**
5	
6	**`// Point α`**

How many processes will reach **Point α**?

A. 4
B. 8
C. 16
D. 32
E. None of the above

8. Which of the following statement(s) about system calls and interrupts is/are TRUE?
    (i) System calls and interrupts all require mode switches (user <-> kernel).
    (ii) System calls, like interrupts, can happen asynchronously.
    (iii) System calls and interrupt handlers use a different hardware context as user programs.

    A. (i), (ii), and (iii)
    B. (i) only
    C. (i) and (ii)
    D. (ii) and (iii)
    E. None of the above

9. Which of the following statement(s) about scheduling algorithms is/are TRUE?
    (i) Using a priority based scheduling algorithm allows a higher priority job to always pre-empt a lower priority job
    (ii) Shortest Remaining Time (SRT) can improve the average turnaround time over Shortest Job First (SJT)
    (iii) Round Robin (RR) can improve the average turnaround time over First-Come First-Serve (FCFS)

    A. (i), (ii), and (iii)
    B. (ii) only
    C. (i) and (ii)
    D. (ii) and (iii)
    E. None of the above

10. Which of the following statements is/are TRUE about changing the time quantum and ITI for a round-robin scheduler? Assume that the time quantum is always a multiple of the ITI.

    A. Decreasing the ITI while keeping the total time quantum constant reduces the percentage of CPU time that the scheduler takes to run its own code.
    B. Increasing the total time quantum while keeping the ITI constant is likely to reduce the response time for interactive processes.
    C. A system with only long running CPU-bound tasks would prefer a short ITI to maximize throughput.
    D. All of the above
    E. None of the above

11. A Shortest Remaining Time (SRT) scheduler is running on a batch-processing system, using estimated task durations through exponential averaging. New tasks can arrive at any time. Which of the statements below are TRUE in this scenario?
    (i) Tasks may starve
    (ii) The scheduler guarantees the smallest average waiting time
    (iii) Priority inversion is possible

    A. (i) only.
    B. (i) and (ii) only.
    C. (ii) and (iii) only.
    D. (i), (ii) and (iii).
    E. None of the above.

12. Consider the following scenario.

    Assume that Process A is performing a long-running computation that generates one integer result periodically. Each time a result is generated, Process A wants to transmit a single integer to Process B, with a total of 10,000 integers over the lifetime of the program. Both processes know that 10,000 integers will be generated and transmitted.

    Process A may transmit *each* of the 10,000 integers by either:

    a) Placing it in a shared memory region that Process B can access.
    b) Using `send` in Process A and a blocking `receive` in Process B.

    Which of the statements below are TRUE?

    (i) It is possible for Process B to receive all 10,000 integers from Process A correctly over shared memory without using semaphores.
    (ii) After all 10,000 integers are transmitted, the total number of syscalls executed using message passing with send/receive is less than using shared memory.
    (iii) If Process B is waiting for an integer with the receive call, it is actively running on the CPU.

    A. (i) only.
    B. (i) and (ii) only.
    C. (ii) and (iii) only.
    D. (i), (ii) and (iii).
    E. None of the above.

13. Assume we have a single-core system that supports simultaneous multithreading. Process P executes 1 CPU intensive user thread that can be mapped to any of 25 kernel threads. Assume that process P is the only process in the system. How many threads can, at most, be in the RUNNING state at the same time?

    A. 1
    B. 2
    C. 25
    D. 50
    E. None of the above

14. Which of the following statement(s) about threads is/are **FALSE**?

    (i) Unlike parent-child processes, for two threads in the same process, updates to function local variables from one thread are visible to the other thread.

    (ii) In the pure user-thread model, when the OS interrupts a process (e.g., timer interrupt), the OS can pick a different (ready) thread in the process to run.

    (iii) Using a hybrid thread model, if we bind N user threads to a kernel thread, when one of the N user threads is blocked on I/O, the remaining N-1 user threads can still run.

    A. (i) only
    B. (i) and (ii)
    C. (ii) and (iii)
    D. (i), (ii), and (iii)
    E. None of the above

Part B. Short Questions

15. Answer each of the following questions briefly. State your assumptions, if any.

(i) [2 marks] If an OS using Shortest Job First (SJF) **cannot** predict task CPU time accurately, explain (or give a simple example) why SJF can result in longer average wait time than First-Come First-Serve (FCFS).

(ii) [2 marks] There are 2 threads, and both threads need to acquire mutex A, mutex B, and mutex C before they can enter a critical section. How do you ensure the 2 threads never result in a deadlock?

16. Consider the following program consisting of a `main` function and `f`.

Line#	**Code for part (a)**	
1	**`void f() {`**	**`void main() {`**
2	**`  int i;`**	**`  int i;`**
3	**`  i = 1234;`**	
4	**`}`**	**`  for (i = 0; i <= 1000; i++)`**
5		**`    f();`**
6		**`}`**

a. [2 marks] When we run `main()`, what is the **maximum number** of stack frames of function **`f()`** that are on the stack at the same time? Explain your answer.

Line#	**Code for part (b)**	
1	`void f(int i) {`	`void main() {`
2	`  if (i == 0)`	`  int i;`
3	`    return 1;`	
4	`  else`	`  for (i = 0; i <= 1000; i++)`
5	`    return i * f(i-1);`	`    f(i);`
6	`}`	`}`

b. [2 marks] Assume we modify `f()` and `main()` as above. When we run `main()`, what is the **maximum number** of stack frames of function **`f()`** that are on the stack at the same time? Explain your answer.

Line#	**Code for part (c)**	
1	`void factorial(int i, int acc){`	`void main() {`
2	`  if (i == 0)`	`  int i;`
3	`    return acc;`	`  factorial(1000, 1);`
4	`  else`	`}`
5	`    factorial(i - 1, acc * i);`	
6	`}`	

c. [2 marks] Assume we create a factorial function by modifying f as above. We claim that a smart compiler can create and reuse only one stack frame for all invocations of **`factorial`**, e.g., if we call **`factorial(1000, ...)`**, **`factorial(999, ...)`** will reuse the stack frame that **`factorial(1000, ...)`** used.

   Why can a compiler do this for the code in part (c) but not part (b)?

17. [2 marks] Dr. Gru has invented a new atomic function:

```
int fetch_and_increment(int *addr);
```

The function increments the integer stored at the memory location and returns the value before the increment, in an **atomic** step. For example, if *a= 5, fetch_and_increment(a) will return 5 and increment *a to 6 atomically. You can assume the function is implemented correctly.

With this new atomic function, Dr. Gru came up with a new lock implementation:

```
Line#  Dr. Gru's new lock implementation
1      struct lock {
2          int ticket;
3          int turn;
4      }

5      void lock_init(struct lock *l) {
6          l->ticket = 0;
7          l->turn = 0;
8      }

9      void lock_acquire(struct lock *l) {
10         int my_ticket = fetch_and_increment(&l->ticket);
11         while (my_ticket != l->turn) {
12         }
13     }

14     void lock_release(struct lock *l) {
15         l->turn++;
16     }
```

Does Dr. Gru's lock implementation ensure mutual exclusion? If yes, explain how it enforces mutual exclusion. If not, give an example of how mutual exclusion can be violated.

- End of paper -