

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM EXAMINATION FOR
Semester 1 AY2010/2011

CS2106 Introduction to Operating Systems

October 2010

Time Allowed 1.5 hours

MATRICULATION NUMBER:

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|

TUTORIAL GROUP:

| |
|--|
| |
|--|

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains 15 questions and comprises 10 printed pages, including this page.
2. Answer **ALL** questions.
3. Write **ALL** your answers in the box provided. Please indicate clearly (with an arrow) if you use any space outside the box for your answer.
4. This is a **CLOSE BOOK** examination, but you are allowed to bring in **one sheet of double-sided A4 size paper** with notes.
5. Write your matriculation number and tutorial group you regularly attend in the space provided above and on top-left corner of every page.

| EXAMINER'S USE ONLY | | |
|---------------------|------|-------|
| Question | Mark | Score |
| Q1-9 | 36 | |
| Q10 | 3 | |
| Q11 | 4 | |
| Q12 | 8 | |
| Q13 | 8 | |
| Q14 | 12 | |
| Q15 | 9 | |
| TOTAL | 80 | |

Part I**Multiple Choice Questions (36 points)**

For each of the question below, select the most appropriate answer and **write your answer in the answer box**. Each question is worth 4 points. If none of the answers provided is appropriate, put an X in the answer box. If multiple answers are equally appropriate, pick one and write the chosen answer in the answer box. Do NOT write more than one answers in the answer box.

1. Which of the following is NOT the purpose of separating execution of an application into user mode and kernel mode?
- A. to make user applications run more efficiently
 - B. to protect certain memory content from user applications
 - C. to prevent user applications from executing certain instructions
 - D. to prevent buggy user applications from crashing the system

Answer:

2. Which of the following is NOT an operating system technique to improve response time?
- A. prioritization
 - B. preemption
 - C. message passing
 - D. multi-threading

Answer:

3. Compare to a normal function call, making a system call on the Linux PCs in the OS and Security Lab takes about
- A. the same time
 - B. 10 - 1000 times faster
 - C. 10 - 1000 times slower
 - D. 100,000 - 1,000,000 times slower

Answer:

4. Which of the following statements is the most accurate description of the corresponding functions/macros?
- A. `signal()` sends a signal to one or more processes.
 - B. `clone()` creates a copy of the core image of the parent process.
 - C. `free()` releases the lock to a critical region.
 - D. `assert()` exits a program if a given condition is true.

Answer:

5. Which of the following function calls will NEVER result in suspension (either busy waiting or blocked) of the calling process?
- A. `wait()` for a child to exit
 - B. `read()` data from a file
 - C. `fork()` a new child process
 - D. `down()` on a semaphore with value 0

Answer:

6. Consider two processes executing the following code. The value of semaphore X is initially 0.

| | |
|------------|------------|
| Process 1: | Process 2: |
| P; | R; |
| up(X); | down(X); |
| Q; | S; |
| | up(X); |
| | T; |

Which of the following is NOT a possible sequence of execution of code fragments P to T?

- A. P, R, Q, S, T
- B. P, R, S, Q, T
- C. R, P, Q, S, T
- D. R, S, P, T, Q

Answer:

7. Consider two threads executing the following code. The variable `mutex` is initially unlocked.

| | |
|--|--|
| Thread 1: | Thread 2: |
| <code>pthread_mutex_lock(&mutex);</code> | <code>pthread_mutex_lock(&mutex);</code> |
| P; | R; |
| <code>pthread_cond_signal(&cond);</code> | <code>pthread_cond_wait(&cond, &mutex);</code> |
| Q; | S; |
| <code>pthread_mutex_unlock(&mutex);</code> | <code>pthread_mutex_unlock(&mutex);</code> |

Assuming that these are the only two threads in the system, which of the following is NOT a possible sequence of execution of code fragments P to S?

- A. P, Q, R, S
- B. R, S, P, Q
- C. R, P, Q, S
- D. R, P, S, Q

Answer:

8. Consider an operating system that implements critical regions using atomic instruction TSL. Which of the following event CANNOT occur when a process P is executing a critical region.
- A. context switch
 - B. system call by P
 - C. hardware interrupt
 - D. software interrupt

Answer:

9. Consider two computer systems A and B that are exactly identical (including scheduling algorithms and set of processes) except that the CPU of A is faster than the CPU of B . Which of the following statement is TRUE?
- A. CPU utilization of A will be higher than B .
 - B. Average turnaround time for processes in A will be higher than B .
 - C. Average response time for processes in A will be higher than B .
 - D. Throughput for processes in A will be higher than B .

Answer:

Part II**Short Questions (44 points)**

Answer all questions in the space provided. Be succinct and write neatly.

10. (3 points) Fill in the blanks. The six answers must be chosen from the following eight commands: `gdb`, `strace`, `valgrind`, `ps`, `make`, `kill`, `signal`, `man`.

(a) To examine the value of a variable in your program, you use

(a) _____

(b) To automatically compile only the necessary source files after you edited your program, you use

(b) _____

(c) You suspect that your program has a memory leak. To identify the leak, you use

(c) _____

(d) To find out what system calls are being made by a process, you use

(d) _____

(e) To find out what processes are currently running, you use

(e) _____

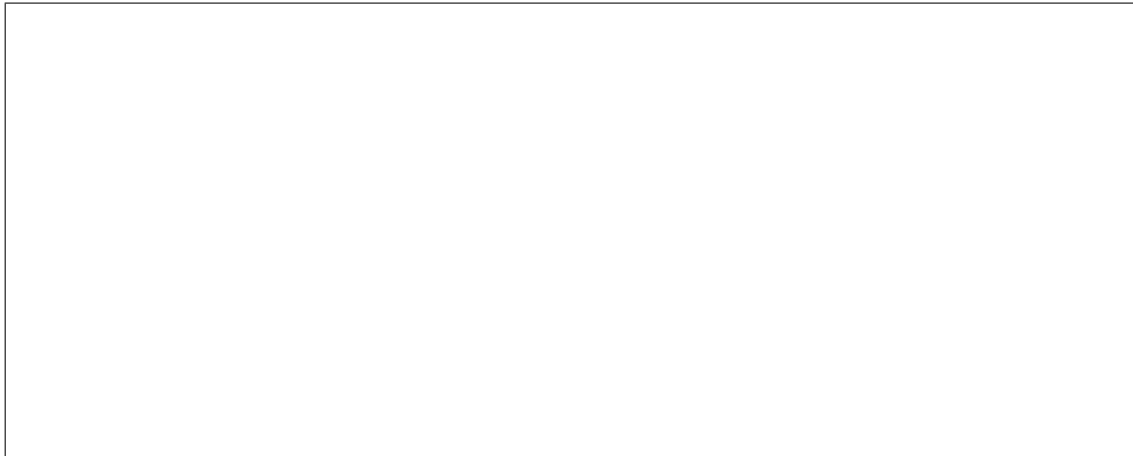
(f) To stop a running process, you can use

(f) _____

11. (4 points) **C.** Consider the following C program:

```
1  int main()
2  {
3      int *x, *y;
4      x = (int *)malloc(sizeof(int));
5      y = (int *)&x;
6      *y = 0;
7      *x = 1;
8  }
```

The program is compiled and run, but aborts immediately with an error. Trace through the program carefully and explain why segmentation fault occurs.



12. (8 points) **Process.** Consider the following C program:

```
1 int main(int argc, char *argv[])
2 {
3     fork();
4     execvp(argv[0], argv);
5 }
```

(a) (4 points) Suppose the program is compiled as `a.out`. What would happen if `a.out` is executed? Explain your answer by drawing the process hierarchy.

(b) (4 points) Suppose Line 3 is replaced with a conditional statement.

```
1 int main(int argc, char *argv[])
2 {
3     if (fork() == 0)
4         execvp(argv[0], argv);
5 }
```

What would happen if the program is compiled and executed? Explain your answer by drawing the process hierarchy.

13. (8 points) **Race Conditions.** Consider the following program:

```
int x = 0;
void *A (void *arg) {
    int i;
    for (i = 0; i < 5; i++)
        x = x + 1;          // Line A
}
void *B (void *arg) {
    int i;
    for (i = 0; i < 5; i++)
        x = x - 1;
}
int main() {
    pthread_t a, b;
    pthread_create(&a, NULL, A, NULL);
    pthread_create(&b, NULL, B, NULL);
    pthread_join(a, NULL);
    pthread_join(b, NULL);
    printf("%d\n", x);
}
```

(a) (4 points) What are the possible values of x after both threads has ended? Explain.

(b) (4 points) Suppose we replace the line indicated by **Line A** with an *atomic* operation `INCR(x)` that adds one to the value of x atomically. What are the possible values of x after both threads has ended? Explain.

14. (12 points) **Process Synchronization.**

- (a) (4 points) A critical region is a fragment of code where only one process can enter the region at a time. Now, consider a generalized concept of critical region, called k -critical region, where at most k processes can enter the region at a time.

A process calls `enter_k()` before entering a k -critical region, and calls `leave_k()` when the process leaves the k -critical region.

Write the pseudocode for a semaphore-based implementation of `enter_k()` and `leave_k()`. Indicate the semaphore variable and its initial value clearly.

- (b) (4 points) Consider the dining philosopher problem with N philosophers ($N > 1$). Argue that, if there are at most $N - 1$ philosophers who are hungry at one time, then deadlock will not occur.

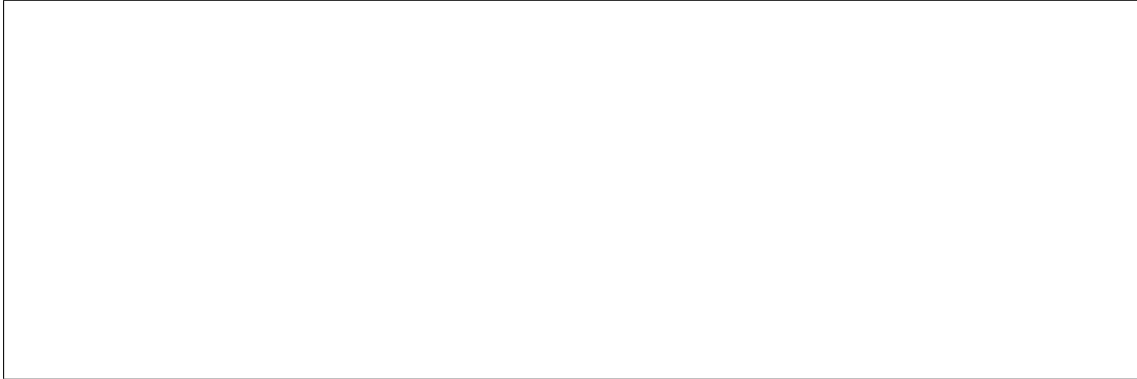
- (c) (4 points) Using Parts (a) and (b) above, implement a symmetric solution to the dining philosopher problem using semaphores that will not lead to deadlock.

15. (9 points) **Scheduling.** There are only three processes in a uni-processor system: A , B , and C , in *increasing* order of priority (C has the highest priority and A has the lowest priority). The system uses preemptive, static priority scheduling with a time quantum of 1 unit time. All three processes are created at the same time and no new process is created.

- A is a completely compute-bound process with no I/O and running time of 5 unit time.
- B is a compute-bound process that computes for 4.5 unit time, followed by an I/O (for 1 unit time), followed by a 1.5 unit time computation.
- C is an I/O-bound process that repeatedly computes (for a 0.5 unit time) and performs I/O (for 1.5 unit-time) in an infinite loop.

The CPU schedules the next process as soon as the current process blocks for I/O, is preempted, or exits. We ignore the time taken for context switching.

- (a) (5 points) Draw the sequence of process execution until both A and B have completed their execution.



- (b) (2 points) What is the turnaround time for A ?



- (c) (2 points) What is the response time for B ?



END OF PAPER