COMP 3511 Operating System (Fall 2021)

Midterm Exam

Date: 26-Oct-2021 (Tue)

Time: 19:00 – 21:00 (2 hours)

Name (Write the name printed on your Student ID card)	**Solution** **Last modified: 8/11/2021 8:43:00 AM**
Student ID	
ITSC email	

Exam format and rules:

- It is an open-book, open-notes exam (Reference: Chapter 1.10)
- For on-site exam:
  - Physical calculator is allowed
  - Other electronic devices are not allowed
  - Write your answers in the spaces provided
- For online exam:
  - For approved students only
  - Allowed devices: a device to record your Zoom video and a computer
  - In the last 15 minutes (i.e. starts at 8:45 pm), you can remove the camera to take pictures of your hand-written answers
  - Submit your work via Canvas using doc/docx/pdf format
- Other details are already sent to students via midterm related emails

Part I. Multiple Choices [25 * 1 points]

Write down your answers in the boxes below:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10
B	C	B	C	D	D	A	D	A	D

MC11	MC12	MC13	MC14	MC15	MC16	MC17	MC18	MC19	MC20
D	C	C	D	D	C	D	B	B	D

MC21	MC22	MC23	MC24	MC25
C	A	Canceled	D	C

[Introduction]

MC1. Which of the following statement is not true for a multicore system?

A. Communication between processors on the same chip is faster than processors residing on separate chips
B. Processors on the same chip communicate with each other through a common bus
C. It utilizes less power
D. It places multiple computing cores on a single chip

Answer: B

MC2. Which of the following technology reduces the overhead for large amount of data movement between main memory and devices?
A. Non-uniform Memory Access (NUMA)
B. Storage Area Network (SAN)
C. Direct Memory Access (DMA)
D. Symmetric Multiprocessing (SMP)

Answer: C

[Operating System Structures]

MC3. Which of the following statement is true?
A. iOS is open-sourced, Android is closed-sourced
B. iOS is a hybrid system consisting of Mach microkernel and BSD UNIX kernel
C. Both iOS and Android run Java programs in a virtual machine
D. All of the above

Answer: B

MC4. Which of the following statement is not an advantage of a microkernel design?

A. It provides more security and reliability
B. The kernel is easier to port to a different platform
C. Inter Process Communication (IPC) enables easier communication between different components
D. New services are added to user space without modification on the kernel
Answer: C

MC5. What is relationship between library call *printf()* and *write()* system call?
A. *printf()* and *write()* system call are concurrent
B. *write()* system call is an alternate version of *printf()*
C. *write()* system call invokes *printf()* to perform the print function
D. *printf()* invokes *write()* system call to get service from operating system
Answer: D

MC6. Which of the following statement is true for loadable kernel module design?
A. It combines the benefits of both the layered and microkernel design
B. It is flexible to enable the modules communicate with one another
C. The user can dynamically load modules into the kernel if more functionality is required
D. all of the above
Answer: D

[Processes]

MC7. Which of the following component is not considered to be part of address space of a process?
A. registers
B. program or text section
C. data section
D. heap
Answer: A

MC8. Which of the following events trigger a change from user mode to kernel mode?
A. system call
B. trap or exception
C. interrupt
D. All of the above
Answer: D

MC9. Which of the following statement about ordinary pipes is not true?
A. Ordinary pipes allow bi-directional communication
B. Only the parent and child processes can use ordinary pipes for communication.

C. Ordinary pipes cease to exist after the communicating processes have finished.
D. Reading and writing to ordinary pipes are performed like file I/O
Answer: A

MC10. Which of the following events might be able to force a process to leave the CPU?
A. make a fork system call
B. make an I/O request
C. interrupt
D. all of the above
Answer: D

MC11. How many child processes will be created by the following C code fragment? (suppose all *fork()* are successful)

```
for( int i = 0; i < n ; ++i ){
   fork();
}
```

A. n
B. 2n
C. 2^n
D. $2^n - 1$
Answer: D

MC12. How many processes will be created by the following C code fragment? (suppose all *fork()* are successful)

```
for( int i = 0; i < 2 ; ++i ){
   if (fork() != 0) fork();
}
```

A. 3
B. 6
C. 9
D. 16
Answer: C

MC13. Which might be the possible output for the following C code? (suppose all *fork()* are successful)

```
void fork_demo()
{
  int x = 2;
  if (fork() == 0)
        printf("x1 = %d ", ++x);
  else
        printf("x2 = %d ", --x);
}
int main()
{
    fork_demo();
    return 0;
}
```

A. x1 = 2 x2 = 2
B. x1 = 3 x2 = 2
C. x2 = 1 x1 = 3
D. x2 = 2 x1 = 3
Answer: C

[Threads]

MC14. Which of the following is unique for different threads within a process?
A. heap
B. data section
C. text section
D. stack
Answer: D

MC15. Which of the following would be an acceptable signal handling scheme for a multithreaded program?
A. Deliver the signal to the thread to which the signal applies.
B. Deliver the signal to every thread in the process.
C. Deliver the signal to only certain threads in the process.
D. All of the above
Answer: D

MC16. According to Amdahl's Law, what is the speedup for an application that is 80% parallel and we run it on a machine with 2 processing cores?
A. 1.60
B. 0.60
C. 1.67
D. 2
Answer: C

MC17. Which of the following statement is true for *clone()* ?
A. It creates a process that can share the address space of the calling process.
B. It uses a set of flags to determine the extent of sharing
C. It loads a new function to execute in the child process.
D. All of the above
Answer: D

[CPU Scheduling]

MC18. In a non-preemptive scheduling scheme, which of the following events will not trigger scheduling?
A. A process completes its CPU burst time
B. A process arrives on the ready queue
C. A process initiates an I/O operation
D. A process waits on a signal and blocks itself
Answer: B

MC19. Which of the following statement is not true about priority scheduling with round-robin or RR scheme when a higher priority process arrives and the CPU is currently occupied with a lower priority process?
A. There are multiple queues, each associated with a priority
B. The process running on CPU will be preempted and placed at the end of the queue associated with its priority
C. The preempted process will not get a new quantum but continues with its remaining quantum next time when it is scheduled to run.
D. The higher priority process will be scheduled to run immediately
Answer: B

MC20. A significant problem with any priority scheduling algorithms is _____.
A. complexity
B. determining the length of the next CPU burst
C. determining the length of the time quantum
D. starvation
Answer: D

MC21. In multilevel feedback queue scheduling, which of the following statement is true?
A. It requires the knowledge on the next CPU burst time
B. Processes are assigned to a queue permanently
C. It handles interactive jobs well by delivering similar performance as SJF.
D. It suffers from belady's anomaly
Answer: C

MC22. Which of the following statement about EDF scheduling is not true?
A. It requires processes' processing time to be known
B. It does not require processes to be periodic
C. It requires the deadlines of processes to be known
D. It does not require a process processing time (CPU burst time) to be a constant
Answer: A

[Synchronization Tools]

MC23. A solution to the critical section problem does not have to satisfy which of the following requirements?
A. mutual exclusion

B. atomicity
C. process
D. bounded waiting
Answer: ~~B~~ (Canceled – give 1 mark for all answers)
TA: We made a typo mistake in the option C (process => progress). All students should get full marks in this question.

MC24. When using semaphores, a process invokes the *wait()* operation before accessing its critical section, followed by the *signal()* operation upon completion of its critical section. Consider reversing the order of these two operations—first calling *signal()*, then calling *wait()*. What would be a possible outcome of this?
A. Starvation is possible.
B. Mutual exclusion is still guaranteed.
C. Deadlock is possible.
D. Several processes could be active in their critical sections at the same time.
Answer: D

MC25. Suppose the binary variable *lock* is initialized to be 0, which of the following can be an implementation of the entry section to solve the critical-section problem?
A. while (compare_and_swap(&lock, 0, 1) == 0), do nothing;
B. while (compare_and_swap(&lock, 1, 0) != 0), do nothing;
C. while (compare_and_swap(&lock, 0, 1) != 0), do nothing;
D. while (compare_and_swap(&lock, 0, 0) != 0), do nothing;
Answer: C

Part II. Calculations [75 points]

1. [30 Points] Process and Thread

1) (12 points) Consider the following program, where M and N will be replaced by different values in the following questions:

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

int main() {
        int i,j;
        for (i=0; i<M; i++) {
                fork();
                for (j=0; j<N; j++)
                        fork();
        }
        return 0;
}
```

(a) (4 marks) What is the total number of process(es) if M = 1 and N = 1. Briefly explain your answer

Solution: 4 (2 marks)

Explanation (2 marks). When M = 1 and N = 1, the program is equivalent to:

```
int main() {
   fork();
   fork();
   return 0;
}
```

Thus, the total number of processes is 2^2 = 4

(b) (4 marks) What is the total number of process(es) if M = 2 and N = 3. Briefly explain your answer.

Solution: 256 (2 marks)

Explanation (2 marks). When M = 2 and N = 3, the program is equivalent to:

```
int main() {
   fork();
   fork();
   fork();
   fork();
   fork();
   fork();
   fork();
   fork(); // 8 fork() function calls by expanding the loop
   return 0;
}
```

Thus, the total number of processes is 2^8 = 256

(c) (4 marks) What is the total number of process(es) in term of M and N? Briefly explain your answer:

Answer: (2 marks) 2^ ((N+1) * M) process(es)

Note: ^ means to the power of.
For example, 2^1 = 2, 2^2 = 4, 2^3 = 8, ….

(2 marks) For the explanation

By expanding the inner loop, the equivalent program without the inner loop is:

```
for (i=0; i<M; i++) {
     fork();
     fork();
     …
     fork(); // N+1 fork()
}
```

Next, we expand the outer loop. After the expansion, you should have M*(N+1) fork() functions:

```
fork();  // the first fork
…
…
…
fork(); // the (N+1)*M fork
```

Each fork() creates 2 processes. So, the total number of processes is 2^((N+1)*M)

Other equivalent answers are accepted, for example:

- (2^(N+1))^M
- 2^(MN+N)
- (2^N x 2)^M
- ….

2) (8 points) You have learned a number of system calls to handle input and output redirection. The following program will execute the following command:

```
wc -l < input.txt > output.txt
```

The above command counts the number of lines in the file input.txt and stores the result to the file output.txt. For example, if input.txt contains the following lines:

COMP3511
Operating System
Fall 2021

After running the program, output.txt becomes: 3
Assume input.txt exists in the same directory of the program, and output.txt does not exist. You cannot include extra header files and cannot add extra variables

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main() {
     char* args[256] = {"wc", "-l", NULL};
     int perm = S_IRUSR | S_IWUSR;
     int fd_in, fd_out;
     fd_in = open("input.txt", O_RDONLY, perm);
     fd_out = open("output.txt", O_CREAT | O_WRONLY, perm);
     close(1);
     BLANK1;
     close(0);
     BLANK2;
     execvp(BLANK3, BLANK4);
     return 0;
}
```

Answer: (For BLANK1 and BLANK2, errors will occur when swapping order!)

BLANK1 (2 marks)	dup(fd_out) // or dup2(fd_out, 1)
BLANK2 (2 marks)	dup(fd_in) // or dup2(fd_in, 0)
BLANK3 (2 marks)	args[0] // or "wc"(wc is also okay for a minor mistake) Note: "wc" is not a good programming practice args[1] is wrong. Index should start at 0
BLANK4 (2 marks)	args

Note: File descriptor ID for stdout is 1, File descriptor ID for stdin is 0
(Reference: lab3 slides and PA1)

3) (10 points) Consider the following C program,

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main() {
        pid_t pid;
        pid = fork();
        if ( pid > 0 ) {
                sleep(10);
                wait(0);
        } else {
                sleep(1);
        }
        return 0;
}
```

System call sleep() temporarily suspends the current process for a specified number of seconds. For example, sleep(1) means suspending the current process for 1 second. Except the sleep() system call, assume that the running time of all other programming statements is neglectable.

a) (5 points) In the above program, are there any zombie process? If yes, write down the occurrence time using the format: "Yes, the zombie process occurs at t=??s". If no, write down "No, there is no zombie process". Briefly explain your answer.

Answer: Yes, the zombie process occurs at t=1s (1+1=2 marks)

At t=1s, the child process is terminated. The parent has not yet called wait() to revamp the child process, because the parent only wakes up at t=10s. (3 marks, MUST answer YES in the first part)

b) (5 points) In the above program, are there any orphan process? If yes, write down the occurrence time using the format: "Yes, the orphan process occurs at t=??s". If no, write down "No, there is no orphan process". Briefly explain your answer.

Answer: No, there is no orphan process. (1+1=2 marks)

Before the parent terminates, it waits for the child process using the wait() system call. Thus, the child process will be properly taken care by the parent process at t=10s. No orphan process will be created. (3 marks, MUST answer NO first in the first part)

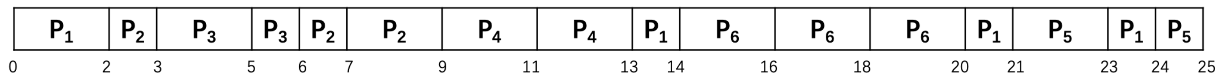
2. [30 Points] CPU Scheduling

1) (10 points) Consider the following single-thread process, arrival times, burst time and priority.

Process	Arrival Time	Burst Time	Priority
P_1	0	5	3
P_2	2	4	2
P_3	3	3	1
P_4	9	4	2
P_5	11	3	3
P_6	14	6	1

a) (6 points) Draw the Gantt chart depicting the scheduling procedures for these processes using a priority scheduling with Round-Robin (RR) scheme, and RR time quantum is 2 ms.

Answer and the partial credits:



- 1 mark – correct up to t=3
- 2 marks – correct up to t=6
- 3 marks – correct up to t=13
- 4 marks – correct up to t=14
- 5 marks – correct up to t=20
- 6 marks – all correct

b) (4 points) Calculate the average waiting time.

Answer:

P_1 : 11+6+2=19
P_2 : 3
P_3 : 0
P_4 : 0
P_5 : 10+1=11
P_6 : 0
Total: 33
Avg:11/2 ms

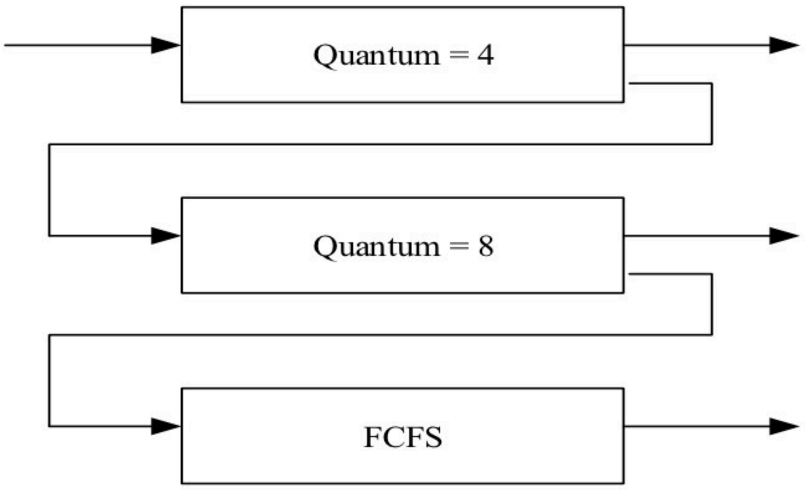
Partial credits:

- 2 correct (1 mark)
- 3-4 correct (2 marks)
- 5 correct (3 marks)
- All correct (4 marks)

2) (10 points) Consider the following single-thread process, arrival times, burst time and the following three queues:

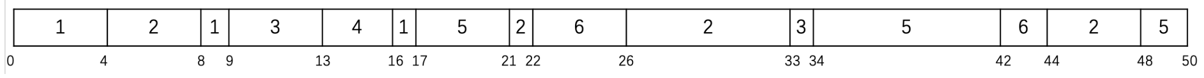
Q0 – RR with time quantum 4 milliseconds
Q1 – RR with time quantum 8 milliseconds
Q2 – FCFS

Process	Arrival Time	Burst Time
P_1	0	6
P_2	3	16
P_3	9	5
P_4	11	3
P_5	17	14
P_6	22	6



a) (6 points) Draw the Gantt chart depicting the scheduling procedures for these processes.

Answer and the partial credit scheme:
- 1 mark – correct up to t=8
- 2 marks – correct up to t=17
- 3 marks – correct up to t=26
- 4 marks – correct up to t=34
- 5 marks – correct up to t=44
- 6 marks – correct up to t=50



b) (4 points) Calculate the average waiting time.

Answer:

P_1 : 4+7 = 11
P_2 : 1+13+4+11=29
P_3 : 20
P_4 : 2
P_5 : 13+6=19
P_6 : 16
Total: 97
Avg: 97/6 ms

Partial credits :

- For P1-P6, any 2 correct (1 mark)
- For P1-P6, any 3-4 correct (2 marks)
- For P1-P6, any 5 correct (3 marks)
- For P1-P6, all correct (4 marks)
- If total and average time is wrong, but P1-P6 correct, deduct 1 mark

3) (10 points) Consider the following single-thread process, executing times, deadlines and periods. Assume all processes arrive at timeslot 0. Fill in the table with the ID of the process that is running on the CPU with Rate-Monotonic (RM) scheduling and Earliest Deadline First (EDF) scheduling in the first 16 timeslots, and show how many deadlines are missed in each scheduler.

Process	Processing Time	Deadline	Period
P_1	1	2	4
P_2	2	5	6
P_3	3	7	9

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
RM																
EDF																

Answer:

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
RM	1	2	2	3	1	3	2	2	1	3	3	3	1	2	2	3
EDF	1	2	2	3	1	3	3	2	1	2	3	3	1	3	2	2

RM: 1, EDF: 0

For the chart (8 marks)

- Each group of 4 numbers = 1 mark
  - For example, t=1 to 4, RM, the 4 numbers are 1 2 2 3
  - If all 4 numbers are correct, you will get 1 mark
- For the whole table, you should get at most 8 marks

For the total number of deadlines missed (2 marks)

- Each answer worth 1 mark

3. [15 points] Synchronization
Consider the following C code, we use test_and_set to emulate locks:

- Please note that this program will not always give the same output
- The final output is dependent on the execution sequence of the threads

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
// boolean is used in our lecture notes. Here, we use int
int test_and_set(int *value) {
    int result = *value;
    *value = 1;
    return result;
}
int value = 0;
int hello = 0;
void thread1_hello() {
    while (test_and_set(&value));
    hello += 1;
    printf("Thread 1: %d\n", hello);
    value = 0;
    pthread_exit(0);
}
void thread2_hello() {
    while (test_and_set(&value));
    hello += 1;
    printf("Thread 2: %d\n", hello);
    value = 0;
    pthread_exit(0);
}
int main() {
    pthread_t thread1, thread2;
    pthread_create(&thread2, NULL, (void*)&thread2_hello, NULL);
    usleep(1); // sleep for a very short period of time
    pthread_create(&thread1, NULL, (void*)&thread1_hello, NULL);
    while (test_and_set(&value));
    printf("Parent thread: %d\n", hello);
    value = 0;
    usleep(1); // sleep for a very short period of time
    return 0;
}
```

Assume the following sequence of execution:
In this program, we have 3 threads: MainThread, Thread1, and Thread2

1. MainThread starts and creates 2 threads, then MainThread is interrupted
2. Thread2 starts and run it increments hello, then Thread2 is interrupted
3. Thread1 starts, until it is interrupted
4. MainThread resumes, until it is interrupted
5. Thread2 runs to completion
6. MainThread runs to completion, but it is not yet exit
7. Thread1 runs to completion

1) (4 points) Is the above sequence of execution possible? Write Yes/No and briefly explain:
Yes. (2 marks)
In step 3 and 4, MainThread and Thread1 make no progress. They can only run to completion after Thread2 sets the value to 0 (2 marks)

2) (4 points) Fill in the table of the return value of test_and_set(&value)

Answer: (Note: True/False won't give marks because the function return an integer)

The execution steps	**Return value of test_and_set(&value)**
Step 3: Thread1 starts, until it is interrupted	1
Step 6: MainThread runs to completion, but it is not yet exit	0

3) (4 points) Given the above execution sequence, what will the program print?

Answer:
Thread 2: 1
Parent thread: 1
Thread 1: 2

OR

Thread 2: 1
Thread 1: 2
Parent thread: 1

No marks are given for any non-sense output

4) (3 points) Is this implementation better than using locks? Write Yes/No and briefly justify your anwer.
Answer:
No or The Same (1 mark)
(2 marks) Reason: while (test_and_set(&value)); will causes a ton of busy waiting
If the answer is "The same", the student needs to say that both this implementation and locks are busy waiting.
Other reasonable explanation is also accepted.
No marks are given for non-sense answers