COMP 3511 Operating System (Spring 2022)

Midterm Exam

Date: 01-Apr-2022 (Fri)

Time: 19:00 – 21:00 (2 hours)

Name (Write the name printed on your Student ID card)	**Solution**
Student ID	
ITSC email	

Exam format and rules:

- It is an open-book, open-notes exam (Reference: Chapter 1)
- 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
D	A	B	C	D	B	A	B	A	C

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

MC21	MC22	MC23	MC24	MC25
A	A	C	A	B

[Introduction]

MC1. Which is not part of the Von Neumann architecture ?
A. Control unit
B. Memory
C. Arithmetic Logic Unit
D. Operating system
Answer: D
Operating system is not a part of Von Neumann architecture.

MC2. Which of the following is false about cache ?
A. The access time of cache is faster than register
B. Cache hit ratio means the percentage of content found in cache
C. If the information required are not in cache, data will be copied from a lower storage.
D. Cache will keep the recently accessed data items closer to processor.
Answer: A
The access time of register is faster than cache.

MC3. Which of the following is not one of the advantages of multiprocessor systems compared to single process systems?
A. Economy of scale
B. Fast on-chip communication
C. More stable
D. Can share other devices such as I/O devices.
Answer: B
Fast on-chip communication is the characteristic of multicore system.

[Operating System Structures]

MC4. Which of the following is false about service of operating systems?
A. Communication service of OS is responsible for the inter process communication.
B. The logging service of OS should keep track of computer resources that each user use.
C. Protection and security service of OS means OS must take appropriate actions to ensure correct and consistent computing
D. I/O operations often use interrupt to handle requests from I/O device.
Answer: C
Protection involves ensuring that all access to system resources is controlled
Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
Take appropriate actions to ensure correct and consistent computing is the job of error detection

MC5. Which of the following is true about system calls?
A. It is better to invoke system calls directly rather than using APIs.
B. The parameters of system call can only be passed by using registers.
C. Command interpreter is a kind of system call.
D. System calls are generally available as functions written in a high-level language
Answer: D
A: It is better to use APIs
B: There are three ways to pass the parameter of system call
C： CLI is not system call
D: True

MC6. Which of the following statement is false about security and protection in operating systems?
A. OS must protect itself from user programs
B. Security means the defence of the system against all kinds of internal attacks.
C. OS should have the ability to against external attacks like viruses and malware
D. An access right is the permission to perform an operation on an object.
Answer: B
Security means the defence of the system against internal and external attacks.

MC7. Which of the following statement is false about the OS structure?
A. The little overhead in the system-call interface and fast communication in layered structure result in a good efficiency of the system
B. The stability of microkernel system is better than monolithic design because the failure of a kernel does not affect other kernels.
C. When two user-level services communicate with each other in microkernel system, message must be copied between them.
D. Hybrid systems combine multiple approaches to address performance, security, and usability needs
Answer: A
The little overhead in the system-call interface and fast communication is the characteristic of Monolithic kernels

[Processes]

MC8. We know that a process has several different states. Which of the following options is a process state change when the scheduler selects a program from the ready queue to run?
A. running → ready
B. ready → running
C. running → waiting
D. waiting → ready
Answer: B

MC9. Which of the following statements about processes and threads is false?
A. Named pipes on Windows can only be used by communicating processes on the same machine.
B. Several processes can communicate using a named pipe.
C. There is no parent-child relationship in the named pipe.
D. Named pipe can be accessed without a parent-child relationship.
Answer: A (3.53)

MC10. Which of the following is true about zombie and orphan?
A. A parent process waiting for a terminated child is called a zombie
B. If a parent process terminated without invoking wait(), it is called an orphan.
C. A zombie process must be already terminated
D. An orphan process is caused by a wait() function that calls too early.
Answer: C
A process that has terminated, but whose parent has not yet called wait(), is known a zombie.
If a parent terminated without invoking wait(), child process is an orphan.
Therefore, only C is true.

MC11. How many processes will be created by the following C code fragment? (suppose all *fork()* are successful and necessary header files are included)

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

A. 2
B. 3
C. 4
D. 6
Answer: B
There are two processes at the end of the first fork, but only one of them fulfils the condition and only one new process will be generated.
Therefore, there are in total three processes.

MC12. What might be the total number of '+' that this program print? Note that this code might generate different outputs.
(suppose all *fork()* are successful and necessary header files are included. Function `fflush(stdout)` is used to print the data in the buffer immediately, so that fork() won't copy these unprinted result to the child process.)

```
int main()
{
  while(fork()%2!=0){
        printf("++");
        fflush(stdout);//print the data in the buffer
  }
  printf("+");
  return 0;
}
```

A.1
B. 3
C.4
D.5

Answer: D
fork() will return 0 and child's process id. If the child's process ID is even, no process will goes into the iteration and there will be two '+' printed.
If the child's process ID is odd, it will print '++' and goes to the next iteration. If the next child's process ID is even, they will not goes into the next iteration and there are in total 3 processes in this program. Therefore, there are 2+3 '+' be printed.

[Threads]

MC13. Which of the following statement about many-to-one threading model is true?
A. When a thread in a multithreaded process is blocked, the process in the kernel is blocked because the result of the thread is not available.
B. Multithreading may not run in parallel on a multicore system.
C. All threads share one kernel process.
D. All of the above
Answer: D

MC14. In a system that supports multithreading, what cannot be shared among several threads created by process P?
A. The code segment of process P
B. The file opened in process P
C. The global variable of process P
D. The stack pointer of a thread in process P
Answer: D (4.15)

MC15. Which of the following statements about user threads and kernel threads is false?
A. User threads cannot be scheduled to run on the CPU
B. The operating system only manages and schedules kernel threads
C. Kernel thread calls are made when calling the Windows threading library
D. There is a mapping between user threads and kernel threads.
Answer: C (4.20)

MC16. According to Amdahl's Law, what is a possible combination of percentage of parallel components and the number of processing cores if the speedup gain is at most 1.25?
A. percentage of parallel components: 50%, processing cores: 2
B. percentage of parallel components: 50%, processing cores: 4
C. percentage of parallel components: 40%, processing cores: 2
D. percentage of parallel components: 40%, processing cores: 4
Answer: C
speedup of A: 1.33, B: 1.61, C: 1.25, D: 1.43

[CPU Scheduling]

MC17. The CPU scheduling algorithm is divided into preemptive and non-preemptive. In this question we discuss the following 4 scheduling algorithms, which of them must be non-preemptive?
A. SJF
B. RR
C. FCFS
D. priority algorithms
Answer: C

MC18. Consider context switching of processes. Assume there are two processes P0 and P1. PCB0 and PCB1 are the process control block of P0 and P1 respectively. Which of the following time units are included in the dispatch latency during context switching between P0 and P1?
A. P0 executing
B. Save state into PCB0, and restore state from PCB1
C. P1 executing
D. All of the above
Answer: B (5.7)

MC19. Which of the following statements about round-robin is true?
A. The round-robin (RR) scheduling algorithm is similar to FCFS scheduling, but preemption is added to enable system to switch between processes.
B. If the time quantum is very short, the scheduler will schedule processes frequently, increasing the system overhead.
C. The average turnaround time does not necessarily improves as the time quantum size increases.
D. All of the above
Answer: D

MC20. Which of the following can be used for the scheduling strategy of the ready queue?
A. FCFS
B. Round-Robin
C. Priority scheduling
D. All of the above
Answer: D

MC21. Which of the following is true of the rate-monotonic (RM) and earliest-deadline-first (EDF) scheduling algorithms?
A. The rationale for RM is to assign a higher priority to tasks requiring CPU more often.
B. Both RM and EDF are non-preemptive.
C. RM uses a dynamic priority policy.
D. EDF uses a static priority policy.
Answer: A

MC22. We have 6 processes arriving at the ready queue at the same time. Their estimated runtimes are: 1, 2, 3, 4, 5 and 6 seconds. Their priorities are 2, 3, 5, 6, 1, and 4. A smaller number means higher priority (e.g., priority 1 has a higher priority than priority 2). Ignoring process switching overhead, calculate the average waiting time for priority scheduling.
A. 8.33s
B. 8.67s
C. 8s
D. 7.67s
Answer: A

[Synchronization]

MC23. Which of the following statements about race condition is true?
A. A race condition will happen only if the outcome of execution does not depend on the order in which instructions are executed.
B. A race condition happens when several threads try to access the same data concurrently.
C. A race condition happens when several threads try to access and modify the same data concurrently.
D. None of the above
Answer: C

MC24. Which of the following statements is false?
A. Spinlocks can be used to prevent busy waits in semaphore execution.
B. Counting semaphores can be used to control access to resources with a limited number of instances.
C. Two standard operations modify Semaphore: wait() and signal().
D. If a semaphore value is negative, its magnitude is the number of processes currently waiting on the semaphore.
Answer: A

MC25. What is the correct sequence of operations to protect a critical section with a mutex?
A. release() followed by acquire()
B. acquire() followed by release()
C. wait() followed by signal()
D. signal() followed by wait()
Answer: B

Part II. Calculations [75 points]

1. [25 Points] Process

Suppose all *fork()* are successful and necessary header files are included.
Function `fflush(stdout)` is used to print the data in the buffer immediately, so that fork() won't copy these unprinted result to the child process.

1) (8 points) Consider the following program.

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

int main() {
   for (int i=0; i<4; i++) {
      fork();
      printf("-");
      fflush(stdout); //print the data in the buffer

   }
   return 0;
}
```

(a) (4 points) What is the total number of process(es) in this program? Briefly explain your answer.

Solution: 16 (2 marks)

Explanation (2 marks).
There are in total 2^4 process.

(b)(4 points) How many '-' will be printed on the screen? Briefly explain your answer.

Solution: 30 (2 marks)

Explanation (2 marks).
There are 2 process at the end of the first iteration; 2^2 process at the end of the second iteration; 2^3 process at the end of the third iteration; 2^2 process at the end of the forth iteration.
Therefore, there are in total 2+2^2+2^3+2^4=30 '-' be printed.

2) (10 points) Consider the following program, where **X** and **Y** will be replaced by different code in the following questions (Assume the fork will return pid > 1000):

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

int main() {
    for (int i=0; i<Y; i++) {
        if(___X____){
            break;
        }
    }
    return 0;
}
```

(a) (5 points) What is the total number of process(es) if the code in **X** is 'fork()' and **Y** is '3'? Briefly explain your answer.

Solution: 4 (2 marks)

In part (a), the code is:

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

Explanation 1: (3 marks).
If fork() returns 0, it will break.
If fork() does not return 0, it will go to the next iteration
Therefore, 3 child processes will be created and there is a parent process. There are in total 3+1 = 4 processes

Explanation 2: (3 marks).
In the first iteration, there are 2 processes, but only one process goes into the second iteration.
In the second iteration, 1 new process is generated, and only one process goes into the next iter.
In the second iteration, 1 new process is generated, iteration stops.
There are in total 2 + 1 + 1 = 4 process.

(b) (5 points) What is the total number of process(es) if the code in **X** is 'i==3' and **Y** is 'fork()+1'? Briefly explain your answer.

Solution: 8 (2 marks)

In part (b), the code is:

```
int main() {
  for (int i=0; i<fork()+1; i++) {
    if(i==3){
        break;
    }
  }
  return 0;
}
```

Explanation (3 marks).
It can be regarded as the following code which excute twice:

```
int main() {
  for (int i=1; i<fork()+1; i++) {
    if(i==3){
        break;
    }
  }
  return 0;
}
```

If fork() returns 0, it stop the iteration.
If fork() does not return 0, it will go to the next iteration until i equals to 3.
The maximum iteration of this new code is 3, so there will be 3 child processes and 1 parent process when you execute this program.
Therefore, there will be 2 * (3 + 1)=8 program in total.

3) (7 points) Consider the following C program.

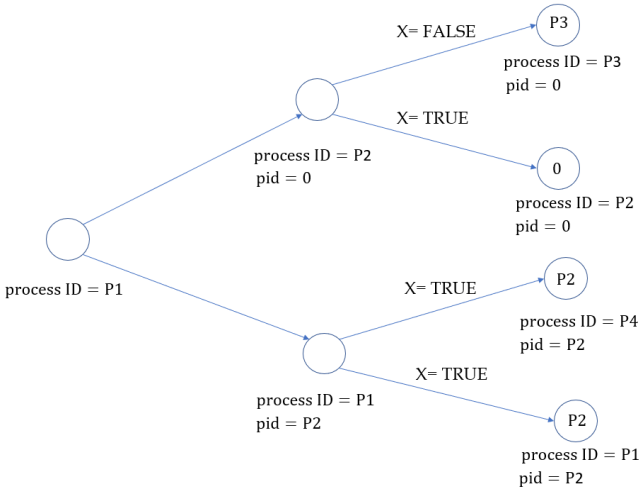
```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main() {
        pid_t pid;
        pid = fork();
        if (pid!=fork()) {
                printf("%d \n",pid);

        } else {
                printf("%d \n",getpid());
        }
        return 0;
}
```

How many different numbers will be output by the program? Briefly explain your answer.

Answer: 3 different number will be output by the program (2 marks)

Explanation (5 marks).
There are in total 4 processes in this program (2 marks).
The return value of fork of the parent process is the process id of the child process (2 marks).

Let X represents '`pid!=fork()`', the condition in the program.
When pid is not equal to zero, the condition `pid!=fork()` is true, they will generate same output, print the same pid.
When pid is equal to zero, the condition is true in one process and is false in another process, so they will generate different output.
Therefore, there will be in total 3 different output (1 marks).



2. [35 Points] CPU Scheduling

1) (13 points) The following table contains 5 processes (P1-P5). Each process has its arrival time, burst time, and priority.

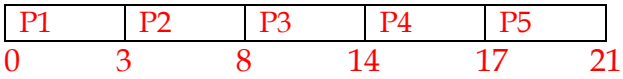
Process	Arrival Time	Burst Time	Priority(Only use in RR)
P_1	0	3	2
P_2	1	5	3
P_3	5	6	2
P_4	8	3	1
P_5	10	4	2

- Whenever there is a tie among processes (the same arrival time, remaining time, etc), they are inserted into the ready queue in the ascending order of process id.
- A smaller number means a higher priority (e.g., priority 1 has a higher priority than priority 2)

For each of the following scheduling algorithms, draw the Gantt charts depicting the sequence of the process execution and calculate the average turnaround time.

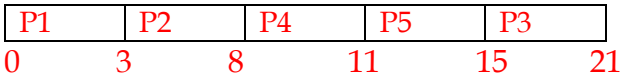
1. FCFS
2. SJF (non-preemptive)
3. Preemptive priority scheduling with RR (with quantum of 2 milliseconds)
4. ~~SRJF~~ SRTF (Shortest Remaining Time First)

- **FCFS** (3 points)



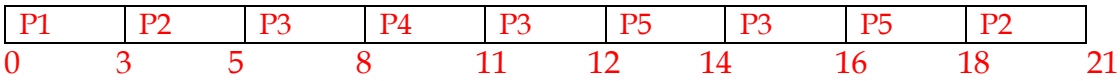
Avg turnaround: {3 + (8-1) + (14-5) + (17-8) + (21-10)}/5 = 39/5 = 7.8
Note: This chart is quite simple. No partial credits for the Gantt Chart (i.e., 0 or 2 marks for the Chart)

- **SJF** (3 points)



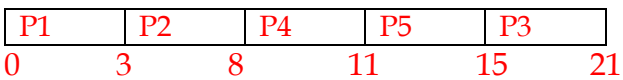
Avg turnaround: {3 + (8-1) + (21-5) + (11-8) + (15-10)}/5 = 34/5=6.8
Note: This chart is quite simple. No partial credits for the Gantt Chart (i.e., 0 or 2 marks for the Chart)

- **Preemptive priority scheduling with RR (TQ=2)** (4 points)



Avg turnaround: {3 + (21-1) + (16-5) + (11-8) + (18-10)}/5 = 45/5=9
Partial credits on the Gantt chart:
Correct up to t=8 (1 point), because it is a key point to preempt P3
Correct up to t=12 (2 points), because it is a key point to resume the remaining time of P3 after the pre-emption. Many students wrongly choose to restart the whole time quantum (instead of resuming the remaining time) in this step.
The whole Gantt chart is correct (3 points)

- ~~**SRJF**~~ (3 points) It should be **SRTF** (Shortest Remaining Time First):

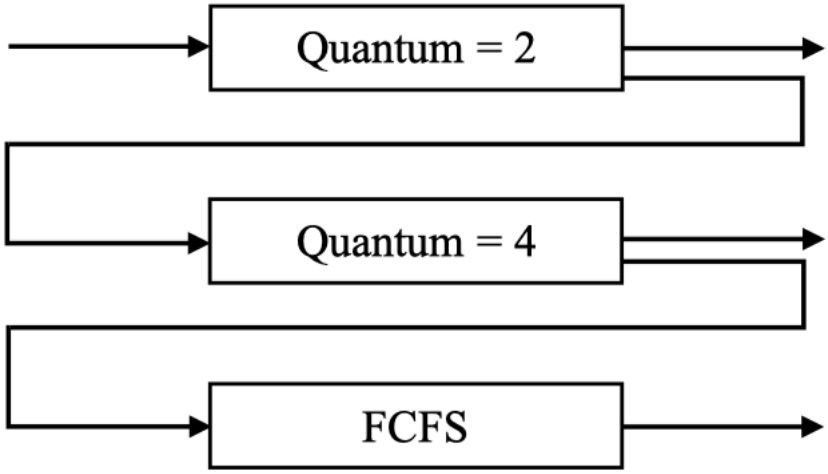


Avg turnaround: {3 + (8-1) + (21-5) + (11-8) + (15-10)}/5 = 34/5=6.8
Note: This chart is quite simple. No partial credits for the Gantt Chart (i.e., 0 or 2 marks for the Chart)

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

Q0 – RR with time quantum 2 milliseconds
Q1 – RR with time quantum 4 milliseconds
Q2 – FCFS

Process	Arrival Time	Burst Time
P_1	0	5
P_2	2	15
P_3	8	7
P_4	9	4
P_5	15	16
P_6	20	6



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

P1	P2	P1	P2	P3	P4	P2	P5	P3	P6	P3	P4	P5	P6	P2	P3	P5

0 2 4 7 8 10 12 15 17 20 22 23 25 29 33 42 43 53

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

P1: 2
P2: (7-4) + (12-8) + (33-15) = 25
P3: (17-10) + (22-20) + (42-23) = 28
P4: 1 + (23-12) = 12
P5: (25-17) + (43-29) = 22
P6: 29-22 = 7
average waiting time = (2+25+28+12+22+7)/6=16

Grading Critirion		
2.2 a	Understanding the basics, with several mistakes	2
	Almost correct, with one minor mistake	4
	correct	6
2.2 b	Understanding the basics, with several mistakes	1
	Almost correct, with one minor mistake	2
	correct	4

3) (12 points) Consider the following single-thread processes, 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 20 timeslots, and show how many deadlines are missed in each scheduler.

Process	Processing Time	Deadline	Period
P_1	1	4	5
P_2	3	7	10
P_3	2	13	16
P_4	2	8	8

RM	1	4	4	2	2	1	2	3	4	4	1	2	2	2	3	1	4	4	3	3
EDF	1	2	2	2	4	4	1	3	3	4	1	4	2	2	2	1	4	4	3	3
Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

RM: 1(P3), EDF: 0 (2 marks)

RM(4 marks): [0, 5], [5, 10], [10, 15], [15, 20] 1 mark per interval
EDF(6 marks): [0, 4], [4, 7], [7, 10], [10, 15], [15, 18], [18, 20] 1 mark per interval

3. [15 points] Synchronization

1) (5 points) Consider two threads perform non-atomic operations counter++ and counter-- respectively. Assume the counter equals to 10 initially. What are possible values of counter when both threads terminate? Please list all possible values.

"counter ++" and "counter --" can be implemented as:

`//counter ++ :`
 	
`register1 = counter`
`register1 = register1 + 1`
`counter = register1`
	`//counter --:`
	
	`register2 = counter`
	`register2 = register2 - 1`
	`counter = register2`

Answer: The concurrent execution of "counter--" and "counter++" is equivalent to a sequential execution in which the lower-level statements presented previously are interleaved in some arbitrary order.

A case that the final value of counter is 9:

T0: counter++ executes register1 = counter {register1 = 10}
T1: counter++ executes register1 = register1 + 1 {register1 = 11}
T2: counter-- executes register2 = counter {register2 = 10}
T3: counter-- executes register2 = register2 – 1 {register2 = 9}
T4: counter++ executes counter = register1 { counter = 11 }
T5: counter-- executes counter = register2 { counter = 9}

By changing the order of the above steps we can have counter = 9,10,11;

one correct: 1 mark, two correct: 3 marks, three correct: 5 marks

2) (10 points) Readers-Writers Variant Problem

Recall the Readers-Writers Problem. There are some readers and writers share some data. Readers only read the data without any modification, while writers modify the data. Any number of readers can simultaneously read the data, whereas only one writer at a time can write the data. If a writer is writing the data, no reader or writer can access the data. If there is at least one reader reading the data, no writer can write to it.

```
int read_count = 0;
semaphore mutex = 1;
semaphore rw_mutex =1;


writer(){
    while (1){
        wait(rw_mutex);
        ...
        /* writing is performed */
        ...
        signal(rw_mutex);
    }
}
```

```
reader () {
    while (1){
        wait(mutex);
        read_count++;
        if (read_count == 1)
            wait(rw_mutex);
        signal(mutex)
        ...
        /* reading is performed */
        ...
        wait(mutex);
        read_count--;
        if (read_count == 0)
            signal(rw_mutex);
        signal(mutex);
    }
}
```

The above code segment may starve writers in the queue. This question asks you to solve the Readers-Writers Variant Problem which adds the constraint that no Reader or Writer shall be allowed to starve(ie. FCFS).

Note: We assume semaphores preserve "First Come First Serve" ordering when block and wakeup processes.

1) Please fill in the following code.
   The code for writer is given. You need an extra semaphore **w**. The semaphore **w** is already defined in the code segment.

2) Please describe the role of the semaphore **w**.

```
int read_count = 0;
semaphore mutex = 1;
semaphore rw_mutex = 1;
semaphore w = 1;

void writer(){
    while (1){
        wait(w);
        wait(rw_mutex);
        signal(w);
        ...
        /* writing is performed */
        ...
        signal(rw_mutex);
    }
}

void reader() {
    while (1){
        wait(w);
        wait(mutex);
        read_count++;
        if (read_count == 1)
            wait(rw_mutex);
        signal(mutex);
        signal(w);
        ...
        /* reading is performed */
        ...
        wait(mutex);
        read_count--;
        if (read_count == 0)
            signal(rw_mutex);
        signal(mutex);
    }
}
```

(one mark per line)

Reference: https://rfc1149.net/blog/2011/01/07/the-third-readers-writers-problem/

2) Semaphore w in order not to starve the writer. It can guarantee the queue follow FCFS rule.(2 marks)