COMP 3511 Operating System (Fall 2023)

Midterm Exam

Date: 21-October-2023 (Saturday)

Time: 1:30 pm – 3:30 pm (2 hours)

Name (Write the name printed on your Student ID card)	**Solution (v6 – with detailed marking scheme after grading)** **Last modified: 10/24/2023 4:06:00 PM**
Student ID	
ITSC email	@connect.ust.hk

Exam format and rules:

- It is an open-book, open-notes exam (Reference: Chapter 1)
- Electronic devices
- Other details are sent to students via exam related emails

Question	Marks
Q1 – Multiple Choices	/25
Q3 – Process and Thread	/35
Q4 – CPU Scheduling	/40
Total	**/100**

Problem 1 Multiple Choices [25 Points]

Write down your answers in the boxes below:

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

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

MC21	MC22	MC23	MC24	MC25
B	**C**	**C**	**C**	**B**

[Introduction]
MC1. Which of the following events does not generate an interrupt?
A. An I/O operation completes
B. CPU fetches instruction or data from memory
C. An arithmetic error occurs
D. A timer expires
Answer: B

MC2. Which of the following statements is not true for a multicore system?
A. It consumes significantly less power
B. Communication between CPU cores on the same physical chip is faster than CPU cores residing on separate chips
C. Each CPU core has its own local cache
D. CPU cores on the same chip communicate with each other through a common bus
Answer: D

MC3. During DMA transfer, the following four events occur:
1. Disk controller initializes DMA transfer
2. DMA controller sends an interrupt to CPU upon transfer completion
3. Disk controller is informed with transfer addresses and transfer bytes
4. ~~MDA~~ DMA *(Note: clarified during the midterm exam)* controller carry out transfer between memory and an I/O device

The correct order of execution is ____.
A. 1→4→3→2
B. 2→4→3→1

C. 3→1→4→2

D. 3→4→2→1

Answer: C

MC4. Which of the following statements about cache is incorrect?
A. In order for cache to work, access must exhibit *temporal* and *spatial* locality
B. Cache stores a subset of the data in a faster device
C. Cache keeps the recently accessed data closer to a processor.
D. Cache improves the utilization of a slower device
Answer: D

[Operating System Structures]
MC5. Which of the following statements is true on APIs?
A. It can replace system calls
B. It hides the complex details in system calls.
C. It intercepts function calls and invokes the necessary system calls
D. It offers better efficiency than calling system calls directly
Answer: B

MC6. Which of the following statements on policy is incorrect?
A. The policy usually cannot be changed
B. The policy is not concerned with implementation
C. The policy specifies what needs to be done
D. The policy is separated from mechanism to minimize changes in implementation
Answer: A

MC7. Which of the following statements is not true in LKM design?
A. LKM combines the benefits of the layered and microkernel design
B. LKM is primarily used to support device driver and file systems
C. LKM enables functionalities be added to and removed from the kernel while it is running
D. In LKM design, only core services are provided in the kernel, other services are dynamically implemented and added to user space
Answer: D

[Processes]
MC8. Which of the following resources cannot be shared between a parent process and its child process?
A. I/O and opened files
B. Stack as part of running state
C. Program
D. Global and local variables
Answer: B (Slide 3.32)

MC9. In process creation with `fork()`, which of the following events is impossible?
A. The child process runs concurrently with the parent process.

B. The child process calls wait() to wait for the completion of the parent process
C. The child process calls exec() to load a new program into it.
D. The child process runs an identical program with the parent process
Answer: B

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(), the parent process becomes an orphan.
C. A zombie process is a process that has been terminated, but not yet taken care by the parent process
D. An orphan process is caused by a wait() function that calls too early.
Answer: C

MC11. Which of the following statements is incorrect on *ordinary pipe*?
A. One ordinary pipe enables a parent and a child process to exchange messages with each other
B. Reading and writing to ordinary pipes are performed like file I/O
C. An ordinary pipe ceases to exist after the communicating processes have completed
D. An ordinary pipe cannot support communications between two arbitrary processes
Answer: A

MC12. How many processes will be created by the following code snippet including the original process? (Suppose all fork() are normal)

```
int main()
{
    int i=0;
    for (i=0;i<4;i++)
        fork();
    return 0;
}
```

A. 4
B. 6
C. 8
D. 16
Answer: D

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

```
void fork_demo()
{
  int x = 3;
  pid_t pid = fork();
  if (pid == 0)
  {
```

```
        x += 3;
        printf("x1 = %d   ", x);
  }
  else if (pid>0)
  {
        x -= 2;
        printf("x2 = %d   ", x);
  }
}
int main()
{
    fork_demo();
    return 0;
}
```

A. x1 = 6 x2 = 1
B. x1 = 1 x2 = 6
C. x2 = 1 x1 = 3
D. x2 = 3 x1 = 1
Answer: A

[Threads]
MC14. Which of the following elements is shared by threads in mult-threaded process?
A. Running state
B. Scheduling information
C. Memory-management information
D. CPU registers
Answer: C

MC15. Which of the following statements is incorrect on *user* and *kernel threads*?
A. A user thread must be mapped to a kernel thread before execution
B. The operating system manages and schedules only kernel threads
C. A user thread is uniquely represented by a TCB in the kernel
D. Multiple kernel threads of a process enable multicores to run a program in parallel
Answer: C

MC16. Based on the *Amdahl's Law*, if the serial portion is 20%, at least how many cores are needed to obtain a speedup factor of 4?
A. 4
B. 8
C. 16
D. 32
Answer: C

MC17. Which of the following statements is correct on a *task* in Linux?
A. A task in Linux can behave like a single-threaded process
B. A task in Linux can behave like a thread within a multi-threaded process
C. A task in Linux can be configured to share a subset of resources with the parent

D. All of the above
Answer: D

[CPU Scheduling]
MC18. Which of the following events can trigger CPU scheduling?
A. A process completes its CPU burst time
B. A process waits for I/O operation
C. An interrupt occurs
D. All of the above
Answer: D

MC19. Which of the following statements about MLFQ is incorrect?
A. It is possible under certain special cases that MLFQ delivers better performance in term of the average waiting time than SRTF
B. MLFQ always produces better response time than RR when the top queue uses the same quantum as that in RR
C. MLFQ does not require any prior knowledge of the next CPU burst time
D. MLFQ overcomes the convoy effect in FCFS
Answer: A

MC20. Consider a system running ten I/O-bound processes and one CPU-bound processes, and all of them are long-running processes. Assume that each I/O-bound process issues an I/O operation once for every 10 unit of times and that each I/O operation takes 100 unit of times to complete. Also assume that the context-switching cost is 1 unit of time. What is the effective CPU utilization (excluding context-switching overhead) for a round-robin scheduler with the quantum=10.
A. 85%
B. 88%
C. 91%
D. 94%
Answer: C
Irrespective of which process is scheduled, the scheduler incurs a 1 unit of time context-switching overhead after each quantum (10). The effective CPU utilization of 10/ (10+1) = 91%.

MC21. Consider a system running ten I/O-bound processes and one CPU-bound processes, and all of them are long-running processes. Assume that each I/O-bound process issues an I/O operation once for every 10 unit of times and that each I/O operation takes 100 unit of times to complete. Also assume that the context-switching cost is 1 unit of time. What is the effective CPU utilization (excluding context-switching overhead) for a round-robin scheduler with the quantum=100.
A. 85%
B. 94%
C. 75%
D. 91%

Answer: B
The time quantum is 100 unit of time. The I/O-bound process incurs a context switch after using up only 10 unit of the time quantum. The time required to cycle through all the processes is therefore 10x(10+1) + 10.1 (as each I/O-bound process executes for 10 unit of times and then incurs the context switch task, whereas the CPU-bound task executes for 100 unit of times before incurring a context switch). The CPU utilization is therefore 200/211 = 94%.

MC22. Which of the following queue arrangements naturally provides *processor affinity* in SMP systems?
A. Multi-level queues
B. A common ready queue for all CPU cores
C. Per-processor or per-core ready queue
D. Multi-level feedback queues
Answer: C (Slide 5.44)

MC23. Suppose that there are five processes arriving at a ready queue at the same time. Their CPU burst times are all 10 time units. Please derive the average waiting times under FCFS and RR with a quantum 4 time units, respectively.
A. FCFS 20 and RR 38
B. FCFS 30 and RR 36
C. FCFS 20 and RR 36
D. FCFS 30 and RR 38
Answer: C
FCFS (0+10+20+30+40)/5 = 20
RR(4) (32+34+36+38+40)/5 = 36

MC24. Suppose that there are five processes arriving at a ready queue at the same time with the order P1, P2, P3, P4 and P5. Their CPU burst times are 4, 2, 6, 4, and 2. Please derive the average waiting times under FCFS and SJF, respectively.
A. FCFS 7.6 and SJF 5.4
B. FCFS 7.2 and SJF 5.2
C. FCFS 7.6 and SJF 5.2
D. FCFS 7.6 and SJF 5.6
Answer: C
FCFS(0+4+6+12+16)/5 = 7.6
SJF (4+0+12+8+2)/5 = 5.2

MC25. Which of the following statements is not true for RM scheduling?
A. RM scheduling requires processes be periodic
B. RM scheduling requires that the CPU burst time of each process is a constant
C. RM scheduling uses a static priority
D. The priority of a process is dictated by the length of the period of the process
Answer: B

Problem 2 Process and Thread [35 Points]

Suppose all *fork()* are successful and necessary header files are included.

1) (10 points) Consider the following C program.

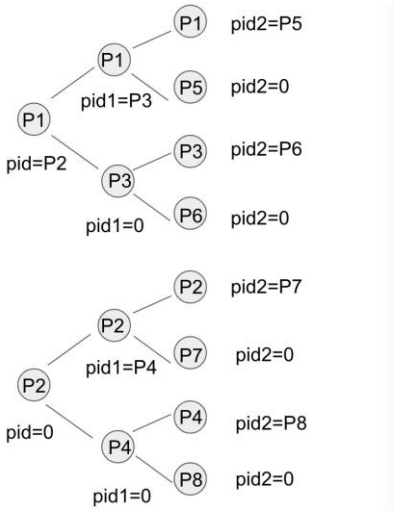
```
int main()
{
    pid_t pid, pid1, pid2;
    pid = fork();
    pid1 = fork();
    pid2 = fork();

    if(pid==0)
    {
         printf("%d\n", getpid());
         fflush(stdout);

    }
    else if(pid>0)
    {
         printf("%d\n", pid);
         fflush(stdout);
         wait(NULL);
    }
    if(pid1==0)
    {
         printf("%d\n", getpid());
         fflush(stdout);

    }
    else if(pid1>0)
    {
         printf("%d\n", pid1);
         fflush(stdout);
         wait(NULL);
    }
    if(pid2==0)
    {
         printf("%d\n", getpid());
         fflush(stdout);
    }
    else if(pid2>0)
    {
```

```
            printf("%d\n", pid2);
            fflush(stdout);
            wait(NULL);
    }
    return 0;
}
```

How many lines will be output by the program? How many different numbers will be output by the program? Briefly explain your answer.
Answer: 24 lines (3 marks), 7 different numbers (3 marks).

Then all 8 processes process (1 mark) the 3 if-else statements (assuming no fork errors), each process printing 3 lines. So 8 processes times 3 lines is 24 lines. (1 mark)

The output is like (order is not fixed, each Px is a line):
P2 P7 P4 P8 P2 P2 P2 P2 P3 P6 P4 P8 P3 P3 P4 P4 P5 P6 P7 P8 P5 P6 P7 P8
So there are 7 different numbers. (2 marks)



2) (15 points) Consider the following code snippet.

```
int x = 5;

void *subtract(void *arg) {
    int *data = arg;
    x -= *data;
    return NULL;
}

void *multiply(void *arg) {
    int *data = arg;
    x *= *data;
    return NULL;
}

int main(int argc, char *argv[]) {
    pid_t pid;
```

```
    pthread_t tid1, tid2;
    pthread_attr_t attr1, attr2;
    int num1 = 3, num2 = 4;

    printf("ORIGINAL: x = %d \n", x);
    fflush(stdout); /* clear output buffer */

    pid = fork();

    if (pid > 0) { /* parent process */
        pthread_attr_init(&attr1);
        pthread_create(&tid1, &attr1, subtract, &num1);
        pthread_join(tid1, NULL);
        printf("PARENT: x = %d \n", x);
        fflush(stdout);

        wait(NULL);

        pthread_attr_init(&attr2);
        pthread_create(&tid2, &attr2, multiply, &num2);
        pthread_join(tid2, NULL);
        printf("PARENT: x = %d \n", x);
        fflush(stdout);
    }
    else if (pid == 0) { /* child process */
        num1 = 2;
        subtract(&num1);
        printf("CHILD: x = %d \n", x);
        fflush(stdout);

        num2 = 5;
        multiply(&num2);
        printf("CHILD: x = %d \n", x);
        fflush(stdout);
        return 0;
    }

    printf("FINAL: x = %d \n", x);
    fflush(stdout);

    return 0;
}
```

What is the output of the above program? For each line of output, please justify your answer. You only need to write down **one possible answer** for this question.

For example, if your program has *N* lines of output, you should write your answers in the following format:

Output of line 1 (Explanation for line 1)
Output of line 2 (Explanation for line 2)

…
Output of line N (Explanation for line N)
Answer:
Note: The output ordering may be different.
ORIGINAL: x = 5 (original value)
PARENT: x = 2 (in the parent process, the child thread calculates x = 5 – 3 = 2)
CHILD: x = 3 (in the child process, after the subtract function, x = 5 – 2 = 3)
CHILD: x = 15 (in the child process, after the multiply function, x = 3 * 5 = 15)
PARENT: x = 8 (in the parent process, there is another child process, after the multiply function, x= 2 * 4 = 8)
FINAL: x = 8 (the child process has been existed, the final value is the x in the parent process, x = 8)

3) (10 points) Programming question using fork and pipe.
In lectures and labs, we use an ordinary pipe for unidirectional one-way communication.
We can create 2 ordinary pipes to achieve a bi-directional communication.

Variable Name	Usage
`int p2c[2]`	It is used to send a message from the parent process to the child process
`int c2p[2]`	It is used to send a message from the child process to the parent process

Here is the expected output of the program:

```
Parent received the message: Child
Child received the message: Parent
```

Necessary header files are included.
You should only use read, write, and pipe syscalls in the following blanks:

```
int main() {
    char buffer[10] = {}; // initialize the array as 0
    int p2c[2], c2p[2]; // variables for 2 ordinary pipes
    BLANK1;
    BLANK2;
    pid_t pid = fork();
    if ( pid == 0 ) { // child process
        const char msg[] = "Child";
        close(c2p[0]);
        BLANK3;
        close(p2c[1]);
        BLANK4;
        printf("Child received the message: %s\n", buffer);
} else { // parent process
        const char msg[] = "Parent";
        close(c2p[1]);
```

```
            BLANK5;
            close(p2c[0]);
            BLANK6;
            printf("Parent received the message: %s\n", buffer);
        }
        return 0;
}
```

BLANK1 (1 mark)	(See the answers and the explanations of the solution code below)
BLANK2 (1 mark)	
BLANK3 (2 marks)	
BLANK4 (2 marks)	
BLANK5 (2 marks)	
BLANK6 (2 marks)	

Partial credit scheme:

BLANK3-6 has 3 input parameters.
-1 mark for each wrong / missing parameter. (Max: 2 marks deduction)

Solution code:

```
int main() {
    // initialize all characters as NULL
    char buffer[10] = {};
    int p2c[2], c2p[2];
    pipe(p2c);  // blank1
    pipe(c2p);  // blank2 - The order of blank1,2 can be swapped
    pid_t pid = fork();
    if ( pid == 0 ) { // child process
        const char msg[] = "Child";

        // The order of blank3,4 can be swapped

        close(c2p[0]);

        // blank3
        // The last value: 5-10 are okay as all other chars are 0
        write(c2p[1], msg, 5);

        close(p2c[1]);

        // blank4
        // The last value: 6-10 are okay as all other chars are 0
        read(p2c[0], buffer, 6);


        printf("Child received the message: %s\n", buffer);
    } else { // parent process
```

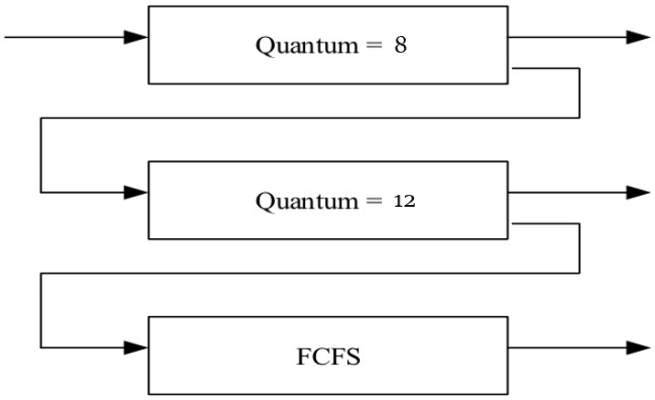
```
        // The order of blank5,6 can be swapped

        const char msg[] = "Parent";
        close(c2p[1]);

        // blank5, The last value: 5-10
        read(c2p[0], buffer, 5);
        close(p2c[0]);

        // blank6, The last value: 6-10
        write(p2c[1], msg, 6);
        printf("Parent received the message: %s\n", buffer);
    }
    return 0;
}
```

Problem 3 CPU Scheduling [40 Points]

1) (14 points) Consider the following single-thread processes, arrival times, burst times and the following three queues:

Q0 – RR with time quantum 8 milliseconds
Q1 – RR with time quantum 12 milliseconds
Q2 – FCFS

Process	Arrival Time	Burst Time
P_1	0	12
P_2	5	50
P_3	24	3
P_4	30	23
P_5	33	26
P_6	55	10



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

Answer:

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

0 8 16 20 24 27 30 38 46 51 55 63 71 83 85 115 118 124

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

Answer:

P1: 16-8=8
P2: (8-5) + (20-16) + (27-24)+(46-30)+(85-51) = 60
P3: 0
P4: (51-38)+(63-55)+(115-71)=65
P5: (38-33)+(71-46)+(118-83)=65

P6: 83-63=20
average waiting time = (8+60+0+65+65+20)/6=36.33

Marking:
a) 0-27:2pts, 3-63:2pts, 71-124:2pts
b)/c) the time of each process: 2 pts (all correct, -1/1 wrong) , avg: 2pts,

c) (4 points) Calculate the average response time.
Answer:
P1: 8
P2: 16-5=11
P3: 27-24=3
P4: 38-30=8
P5: 46-33=13
P6: 63-55=8
average response time = 8.5

2) (12 points) Consider the following three single-thread processes (P1 – P3) with the executing times, deadlines and periods shown in the table. 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 under each scheduler.

Process	Processing Time	Deadline	Period
P_1	2	4	9
P_2	3	8	8
P_3	5	15	15

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

Answer:

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

RM: 1 (first P1), EDF: 0 (2 marks). Gantt charts with arrows (like lecture examples) will also be accepted.

RM(5 marks): [0, 4], [4, 8], [8, 12], [12, 16], [16, 20] 1 mark per interval
EDF(5 marks): [0, 4], [4, 8], [8, 12], [12, 16], [16, 20] 1 mark per interval

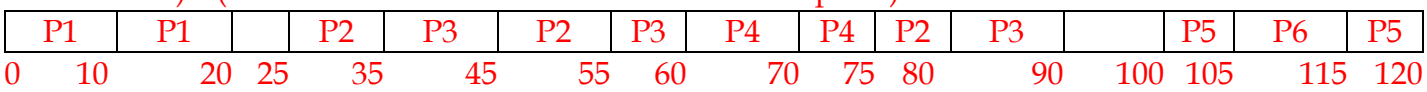
3) (14 points) Assume a priority-based preemptive scheduling scheme is used. Each process is assigned a numerical priority number, with a smaller number indicating a higher priority. Processes with the same priority utilize round-robin scheduling with the quantum = 10. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue of the same priority with a new quantum.
Consider the following 6 processes with the numerical priority, CPU burst time and arrival time given.

Process	Priority	Arrival Time	Burst Time
P1	1	0	20
P2	3	25	25
P3	3	30	25
P4	2	60	15
P5	5	100	10
P6	4	105	10

a) (5 points) Show the scheduling order of the processes using a Gantt chart. (Hint: we may have some blanks on the Gantt chart.)
b) (3 points) What is the turnaround time for each process?
c) (3 points) What is the waiting time for each process?
d) (3 points) What is the CPU utilization rate?

Answer:

a) (Each three continuous blocks worth for 1 point)



b) (each answer for 0.5 point) p1: 20-0 = 20, p2: 80-25 = 55, p3: 90 - 30 = 60, p4: 75-60 = 15, p5: 120-100 = 20, p6: 115-105 = 10
c) (each answer for 0.5 point) p1: 0, p2: 30, p3: 35, p4: 0, p5: 10, p6: 0 (turn-around time – CPU burst time)
d) (three points for the answer) 105/120 = 87.5 percent.