COMP 3511 Operating System (Spring 2024)

Midterm Exam

Date: 18-March-2024 (Monday)

Time: 7:00 pm – 9:00 pm (2 hours)

Name (Surname, Othernames) Note: Please refer to your name printed on the student ID card	
Student ID	
ITSC email	@connect.ust.hk

Your signature: ____________________________ fully understands the **HKUST Academic Honor Code** and confirm to follow it during the exam.

Exam format and rules:
- It is an open-book, open-notes exam (Reference: Chapter 1)
- Electronic calculator is allowed, but no other electronic devices.
- Other details are sent to students via exam related announcements
- The exam booklet is single-sided. Please use the back pages for rough work. If needed, draw an arrow and continue your answers using the back pages.

Question			**Points**
Q1 – Multiple Choices			/20
Q2 – Short Questions			/16
Q3 – Process and Thread	PA1 fork-wait /8	PA1 Pipe /10	fork-wait /12
Q4 – CPU Scheduling	MLFQ /12	RM/EDF /12	Prememptive /10
Total			**/100**

Problem 1 Multiple Choices [20 Points]
Write down your answers in the boxes below:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10

MC11	MC12	MC13	MC14	MC15	MC16	MC17	MC18	MC19	MC20

[Introduction]

MC1 Which of the following statements is not true about *interrupt*?

A. Interrupts are exclusively used by I/O devices to coordinate the actions with CPU(s)
B. Interrupts are caued by asynchronous events to require CPU attention
C. Interrupts can have different priorities based on their importance
D. Each interrupt needs to be handed by by an *interrupt handler*

MC2. Which of the following statements is not true about secondary storage?

A. Mechanical storage like HDD tends to be slower than electrical storage like NVM
B. Mechanical storage per unit capacity is cheaper than electricity storage
C. Mechanical storage is more reliable than electricity storage
D. Mechanical storage usually has larger capacity than electricity storage

MC3. How do *multiprogramming* and *multitasking* complicate the OS design?

A. It requires memory management strategy to allocate memory to different processes in an efficient and protected manner
B. It requires CPU scheduling mechanism to determine which process to run on a CPU and for how long
C. It requires coordination in concurrent operation on I/O devices and CPU
D. All of the above

[Operating System Structures]

MC4. What is the purpose of providing *system calls* to user programs?

A. It provide program portability across different systems
B. It provides services by the OS that are not available in user programs
C. It hides the complex details from user programs
D. It offers more functionality and flexibility

MC5. Which of the major advantage of *dynamic link libraries* or *DLLs*?

A. DLL, in conjunction with a linker, combie multiple object files into a singe executed binary file
B. DLL, delays the linking of libaries into an executed file
C. DDL provides better memory efficiency by allowing sharing of libraries among multiple programs instead of linking into each executable program file
D. DLL enables dynamic loading of libraries into the memory

MC6. Which of the following statements on a *layered* OS approach is not true?

A. It is a modular approach that divides complex functionalites into modules
B. It offers the flexibility that a function can be implemented in any layer
C. It specifies well defined interfaces that a layer can interact with adjacent layers
D. It provides information hiding, in which a layer does not know how the functions are implemented in other layers. This simplifies the debugging

MC7. Which of the following satements is not an advantage in a *microkernel* OS design?

A. It provides IPC to enable different OS components to interact with each other
B. The kernel is smaller, making it easier to port to new architecture
C. It is more reliable with less codes in the kernel
D. It is easier to add new services to the OS as they are usually outside the kernel

[Processes]

MC8. Which of the following statements is not true for *medium-term scheduler*?

A. It temporarily suspends the execution of a process
B. It stores the program and data in memory
C. It reduces the degree of multiprogramming
D. It involve a technique called *swapping*

MC9. Which of the following elements doesnot belong to a *thread* within a *process*?

A. codes
B. program counter
C. stack
D. registers

MC10. Which of the following actions does not cause a mode change, i.e., from *user mode* to *kernel mode*?

A. Interrupt
B. Trap
C. System calls
D. Context switch

MC11. How many processes will be created by the following code snippet? (Suppose all *fork()* are successful)

```
int main() {
    int i=0;
    fork();
    for (i=0;i<5;i++)
        fork();
    return 0;
}
```

A. 64
B. 32
C. 16
D. 6

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

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

A. 3
B. 4
C. 5
D. 8

MC13. Which of the following statements is not an advantage of *name pipe* over *ordinary pipe* in Unix?

A. Name pipe enables process communications across different machines
B. Name pipe exists independently of the processes using it
C. Name pipe does not require parent-child relationship
D. Name pipe allows more than two processes to use it for communications

[Threads]

MC14. Which of the following statements is true for *thread cancellation*?

A. Thread cancellation can be disabled
B. Thread cancellation can be deferred or delayed
C. Thread cancellation can occur immediately
D. All of the above

MC15. According to the *Amdahl's Law*, if the serial portion of a program is 10%, which of the following speedup is impossible?

A. 5
B. 7
C. 9
D. 11

[CPU Scheduling]

MC16. Which of the following statements is not true about FCFS vs. SJF?

A. SJF produces the minimum average waiting time with no preemption
B. FCFS suffers from Convoy effect while SJF does not
C. FCFS always results in larger average turnaround time than SJF
D. SJF usually has shorter average response time than FCFS

MC17. Which of the following statements is not true on *chip multithreading* or *CMT*?

A. CMT assigns each core multiple hardware threads.
B. CMT enables user-level threads to be mapped to a kernel thread
C. CMT is made possible because of the *memory stall*
D. CMT appears as multiple logical CPUs for each core to run software threads

MC18. Suppose that there are five processes arriving at a ready queue at the time 0, 1, 2, 3 and 4. Their CPU burst times are all 5 time units. Please derive the *average response time* under FCFS and RR with a quantum 2 time units, respectively. (Note: In order to break a tie in RR, if a new arrival comes at the same time as a process returns to the ready queue upon quantum expiration, the new arrival is placed at the front.)

A. FCFS 13 and RR 4
B. FCFS 12.6 and RR 4
C. FCFS 13 and RR 4.8
D. FCFS 12.6 and RR 4.8

MC19. Suppose that there are five processes arriving at a ready queue at the time 0, 1, 2, 3 and 4. Their CPU burst times are 6, 2, 3, 4, and 2 time units. Please derive the *average waiting time* under FCFS and SRTF, respectively. (Note: An newly arrived process preempts an existing process only when its CPU burst time is shorter than the remaining CPU burst time of the process running in SRTF.)

A. FCFS 4 and SRTF 3.2
B. FCFS 6 and SRTF 3.8
C. FCFS 5 and SRTF 2.4
D. FCFS 7 and SRTF 2.0

MC20. Which of the following statements is true for *EDF* scheduling?

A. EDF does not need processes be periodic
B. EDF does not require the processing time of each process be a constant
C. EDF uses a dynamic priority
D. All of the above

Problem 2 Short Questions [16 points]

1) (4 points) What are *zombie* process and *orphan* process, and how they could be related to each other?

2) (4 points) What does CPU scheduler or short-term scheduler do, and how does hyperthreading affect that?

3) (4 points) What are the major steps in `fork()`? Which one is most time-consuming?

4) (4 points) Can you describe a scenario that RR can produce a better and worse average turn-around time than FCFS for a give set of processes? (Hint: you can illustrate with concrete examples)

Problem 3 Process and Thread [30 Points]
Suppose all *fork()* are successful and necessary header files are included.

1) **(8 points)** You have implemented a simplified Linux shell in PA1. Consider the following code snippet from the main function of PA1. Suppose all *fork()* are successful and necessary header files are included.

```
// necessary header files are included

// Implementation of process_cmd(char *)
// and other related helper functions

// Only the main function is shown
int main() {
    // A character array storing the command
    char cmdline[MAX_CMDLINE_LENGTH];

    // … lines before the loop are skipped

    // The main event loop
    while (1) {

        // Get the input command from user

        // Some lines are skipped here

        pid_t pid = fork();
        if (pid == 0){
            BLANK1; // child
        }
        else {
            BLANK2; // parent
        }

    } // end of the main event loop

    return 0;
}
```

Please answer the sub-questions (a) and (b) on the next page

a) Suppose a function `process_cmd(char *)` is used for processing the input command. Please fill the 2 blanks (one line of code for each blank) for the code to process the input commands correctly and normally. Your code needs to be correct C code, pseudocode is not accepted. (2 points)

BLANK1 (1 point)	
BLANK2 (1 point)	

b) What is the purpose of BLANK2? What happen if BLANK2 is missing? Please give a short answer for each question.

Purpose of BLANK2 (3 points)	
What happen if BLANK2 is missing (3 points)	

2) (10 points) The following example is like the PA1-related lab example. It demonstrates the execution of a pipe command, which involves connecting the output of one command (ls) as the input of another command (wc):

```
// Assume necessary header files are included
// Example: Execute ls | wc command
int main() {
   int pfds[2];
   pipe(pfds);
   pid_t pid = fork();
   if ( pid == 0 ) {
      close(1);
      dup(pfds[1]);
      close(pfds[0]);
      execlp("ls", "ls", NULL);
   } else {
      close(0);
      dup(pfds[0]);
      close(pfds[1]);
      wait(0);
      execlp("wc", "wc", NULL);
   }
   return 0;
}
```

It is very hard to rewrite the above code to support multi-level pipe. Can you change the above program to a loop with only 2 iterations?

The loop structure described below revolves around a key idea:

- The if-clause signifies the child process of the current iteration. Its primary role is to handle the pipe input/output by utilizing dup2 system calls. Additionally, it executes the command specified in the current iteration using execlp

- The else-clause signifies the parent process (i.e., the original process). Its main objective is to update the variable prev_in, which serves as a memory for the input file descriptor in the subsequent iteration.

Based on the design, we can fill in the missing blanks to rewrite the 2-level pipe into a for loop. This modification allows us to effortlessly extend the 2-level pipe concept to accommodate multi-level pipes:

Write your answers in the following table. Your code needs to be correct C code. Pseudocode is not accepted in this question.

The code and the answer table are shown on the next page for easy reference.

```
// Assume necessary header files are included
// Goal: Convert a 2-level pipe example into
// a for loop with 2 iterations
// Note: 0 (stdin), 1 (stdout)
int main() {
    int pfds[2], i;
    pid_t pid;
    int prev_in = 0; // assume prev_in is stdin
    char commands[2][50];
    strcpy(commands[0], "ls"); // The first command
    strcpy(commands[1], "wc"); // The second command
    for (i = 0; i < 2; i++) {
        pipe(pfds);
        pid = fork();
        if (pid == 0) {
            close(0);
            dup2( BLANK1 , 0);
            if (i < 1) { // not the last segment
                close(1);
                dup2( BLANK2 , 1);
            }
            close( BLANK3 );
            execlp(commands[i], commands[i], NULL);
        }
        else if (pid > 0) {
            wait(0); // wait for the child
            close( BLANK4 );
            prev_in =  BLANK5 ; // save it for the next
        }
    }
}
```

BLANK1 (2 marks)	
BLANK2 (2 marks)	
BLANK3 (2 marks)	
BLANK4 (2 marks)	
BLANK5 (2 marks)	

3) **(12 points)** Consider the following code snippet. Assume all fork() calls are successful, and necessary header files are included. Use standard input/output functions like printf() for output. Address the following questions.

```
int main() {
    int outer_loops, inner_loops;
    // Assume outer_loops and inner_loops
    // are assigned values earlier in the code
    for (int i = 0; i < outer_loops; i++) {
        if (fork() == 0) {
            // Child process
            for (int j = 0; j < inner_loops; j++) {
                if (fork() == 0) {
                    printf("A");
                    fflush(stdout);
                    exit(0);
                }
                wait(NULL);
            }
            exit(0);
        }
        wait(NULL);
    }
    return 0;
}
```

Please write the answers of sub-questions (a) and (b) on the next page:

a) Assuming outer_loops = 2 and inner_loops = 3, <u>draw a tree diagram</u> and give the <u>output of this program</u>. (5 points)

Output of the program: ____________________________

Draw a tree diagram here:

b) Assuming inner_loops=m and outer_loops=n, how many processes are there in total? <u>Express the result using m and n</u>. Briefly <u>explain your answer</u>.(7 points)

Problem 4 CPU Scheduling [34 Points]

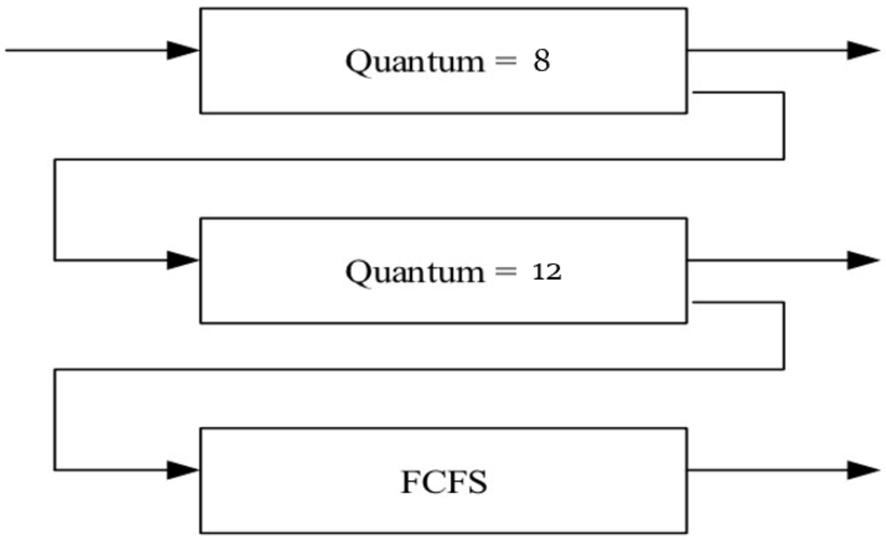
1) (12 points) Consider the following single-thread processes, arrival times, CPU burst times and the following three queues:

Q0 – RR with time quantum 8 milliseconds

Q1 – RR with time quantum 12 milliseconds

Q2 – FCFS

Process	Arrival Time	Burst Time
P_1	0	10
P_2	6	50
P_3	26	5
P_4	38	40
P_5	50	28
P_6	60	12



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

b) (3 points) Calculate the waiting time for each process and the average waiting time.

c) (3 points) Calculate the response time of each process and the average response time. (Hint: Pay attention to the response time of P4).

2) (12 points) Consider the following two processes (P1 – P2) 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	4	6	9
P_2	3	7	7

Write down the process numbers (e.g, 1, 2) on the following Gantt chart. If there are no processes running, write a letter X.

RM																				
EDF																				

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

How many deadlines are missed for RM: ____________________________

How many deadlines are missed for EDF: ____________________________

3) (10 points) Consider a *preemptive* scheduling algorithm that uses priority-based scheduling with dynamic priorities. Each process is assigned a numerical priority number, and **a larger number indicates a higher priority**. When a process is waiting for the CPU (in the ready queue but not running), its priority changes at a rate α; when it is running, its priority changes at a rate β. For example, if a process is assigned an initial priority 0 and it waits for 1 time unit and runs 2 time units, its priority becomes $\alpha+2\beta$. Assume the system has four processes:

Process	Arrival Time	Burst Time
P_1	0	10
P_2	3	6
P_3	5	4
P_4	8	5

a) Assume $\alpha = 1$ and $\beta = 2$. All processes are given a priority of 0 when they enter the ready queue. Draw the Gantt chart to show the scheduling order of the processes. (3 points)

b) Assume $\alpha = -2$ and $\beta = -1$. All processes are given a priority of 50 when they enter the ready queue. Draw the Gantt chart to show the scheduling order of the processes. (3 points)

c) Assume $\alpha = 100$ and $\beta = 0$, and only consider P1 and P2 in the table. All processes are given a priority of 0 when they enter the ready queue. This system **only** checks the priority of processes every 2 time unit to determine which process should be executed. Draw the Gantt chart to show the scheduling order of the processes. (4 points)