Cs3103 2324A Midterm solution

Operating Systems (City University of Hong Kong)

CITY UNIVERSITY OF HONG KONG

Course code & title : CS3103 Operating Systems

Session : Semester A 2023-2024

Time allowed : 110 minutes

There are **7** questions.

---

1. Answer ALL questions.
2. Write your answers in the space provided.

---

*This is a **close-book** examination.*

Candidates are allowed to use the following materials/aids:
Approved calculators.
One page double-sides handwritten A4 note.

Materials/aids other than those stated above are not permitted. Candidates will be subject to disciplinary action if any unauthorized materials or aids are found on them.

Academic Honesty

I pledge that the answers in this exam are my own and that I will not seek or obtain an unfair advantage in producing these answers. Specifically,

- *I will not plagiarize (copy without citation) from any source;*
- *I will not communicate or attempt to communicate with any other person during the exam; neither will I give or attempt to give assistance to another student taking the exam; and*
- *I will use only approved devices (e.g., calculators) and/or approved device models.*
- *I understand that any act of academic dishonesty can lead to disciplinary action.*

I pledge to follow the Rules on Academic Honesty and understand that violations may led to severe penalties.

Student ID: _____

Name: _____

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total

Q1. [8 points] Name and describe the four different multithreading models.

Answer

- Many-to-One: Many user-level threads mapped to a single kernel thread
- One-to-One: Each user-level thread maps to the kernel thread
- Many-to-Many: Allows many user-level threads to be mapped to many kernel threads
- Two-level Model: Similar to M:M, except that it allows a user thread to be bound to the kernel thread

Q2. [8 points] What is the difference between preemptive scheduling and non-preemptive scheduling? Discuss their benefits and drawbacks.

Answer

1. Non-preemptive scheduling takes place during the state transition of switching from running to waiting state & terminate only, preemptive takes place in switching from running to waiting state, switching from running to ready state, or switching from waiting to ready or the terminates.
2. Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state; Preemptive scheduling can result in race conditions when data are shared among several processes.

Q3. [22 points]

a) Suppose we have four jobs: J0, J1, J2 and J3, and with different lengths: 5, 8, 6, 4 respectively. They are arriving roughly at the same time (Assume J[i] arrived just a hair before J[i+1]). Please calculate each $T_{turnaround}$ and $T_{response}$ of them, and also $T_{turnaround}$ and $T_{response}$ for average, by using FIFO (First In, First Out) policy.

Ans:

J0: $T_{turnaround} = 5 - 0 = 5$, $T_{response} = 0 - 0 = 0$
J1: $T_{turnaround} = 13 - 0 = 13$, $T_{response} = 5 - 0 = 5$
J2: $T_{turnaround} = 19 - 0 = 19$, $T_{response} = 13 - 0 = 13$
J3: $T_{turnaround} = 23 - 0 = 23$, $T_{response} = 19 - 0 = 19$

Average: $T_{turnaround} = (5 + 13 + 19 + 23) / 4 = 15$
$T_{response} = (0 + 5 + 13 + 19) / 4 = 9.25$

b) Suppose we have another four jobs: A, B C and D, and with different lengths: 5, 8, 6, 4 and different $T_{arrival}$: 0, 4, 5, 7, respectively. Please calculate each $T_{turnaround}$ and $T_{response}$ of them, and also $T_{turnaround}$ and $T_{response}$ for average, by using SJF (Shortest Job First non-preemptive).

Ans:

Non-Preemptive SJF:
A: $T_{turnaround} = 5 - 0 = 5$, $T_{response} = 0 - 0 = 0$
B: $T_{turnaround} = 23 - 4 = 19$, $T_{response} = 15 - 4 = 11$
C: $T_{turnaround} = 11 - 5 = 6$, $T_{response} = 5 - 5 = 0$
D: $T_{turnaround} = 15 - 7 = 8$, $T_{response} = 11 - 7 = 4$

Average: $T_{turnaround} = (5 + 19 + 6 + 8) / 4 = 9.5$, $T_{response} = (0 + 11 + 0 + 4) / 4 = 3.75$

c) If considering the system which demands fairness and responsiveness, what kinds of policy should we apply? And please discuss the pros and cons of this kind of policy.

Ans (only for reference): RR.
Pros of Round Robin: Fairness; Responsiveness; preemptive (and so on)

Cons of Round Robin: High context switch overhead:; Inefficient for long-running processes; Poor for CPU-bound tasks (and so on)

Q4. [15 points] Consider the following two threads to be run concurrently on a single-processor machine. The value at the shared memory address 3000 is initialized to 2. Assume that Thread 1 will be scheduled to execute first.

Thread 1

```
.main            # the entry point of the thread
mov $1,%ax       # %ax is initialized to 1
.top             # a label
mov 3000,%cx     # get the value from the address to register %cx
sub $1,%cx       # decrement the value in register %cx by 1
mov %cx,3000     # store the value back to the address
sub $1,%ax       # decrement the value in register %ax by 1
test $0,%ax      # test the value in %register %ax
jgt .top        # jump back to .top if %ax is greater than 0
halt             # stop running this thread
```

Thread 2

```
.main            # the entry point of the thread
mov 3000,%dx     # get the value from the address to register %dx
sub $1,%dx       # decrement the value in register %dx by 1
mov %dx,3000     # store the value back to the address
test $1,%dx      # test the value in %register %dx
jgt .main       # jump back to .main if %dx is greater than 1
halt             # stop running this thread
```

a) What is the final value at register ax in Thread 1 and the shared memory address 3000 if there is no interrupt?

Answer: 0 and 0.

b) If we change the instruction "mov 3000, %dx" into "mov %ax, %dx" for Thread 2, can we know the final value at the shared memory address 3000 if there is no interrupt? If yes, give the specific value. If no, give the reason.

Answer: We cannot know the exact value of memory address 3000, because Thread 1 and Thread 2 has independent register values, and we do not know the initial value of register ax in Thread 2.

c) What is the final value of 3000 if an interrupt occurs after every 3 instructions ? (the scheduler switches from a thread to the other whenever an interruption happens. For example, Thread 1 starts to execute first,then an interrupt occurs after Thread 1 finishes its 3rd instruction "`sub $1,%cx`").

Answer: 1.

Q5 [18 points] Consider system running on one CPU where each process has four states: *RUNNING* (the process is using the CPU right now), *READY* (the process could be using the CPU right now, but some other process is using the CPU), *WAITING* (the process is waiting on I/O), and *DONE* (the process is finished executing). The scheduler has several options:

SWITCH_ON_IO and **SWITCH_ON_END**

IO_RUN_LATER and **IO_RUN_IMMEDIATE**

a) Please explain the function of **SWITCH_ON_IO** and **SWITCH_ON_END.**

SWITCH_ON_IO: the system switches to another process whenever a process issues an IO request and start to waits for I/O completes;
SWITCH_ON_END: the system can NOT switch to another process while one is doing I/O; instead, it can switch to another process when the currently running process is completely finished.

b) Now suppose that we have three processes. process 0 is **I/O, I/O, I/O**, process 1 is **CPU, CPU, CPU, I/O**, and process 2 is **CPU, CPU, CPU, CPU**. Each CPU request takes 1 time unit and each I/O requests takes 5 time units (1 time unit on CPU and 4 time units on I/O device) to complete.

Fill the **states of each process** (RUNNING: CPU, RUNNING: IO, WAITING, READY, DONE), **the number of processes using CPU**, and **the number of processes using I/O devices** in the following tables and calculate **CPU utilization**. When you compute the CPU utilization, **DO NOT include the time units that all the processes have done.**

SWITCH_ON_IO and **IO_RUN_LATER** PROCESS 0: I/O, I/O, I/O PROCESS 1: CPU, CPU, CPU, I/O PROCESS 2: CPU, CPU, CPU, CPU					
Time unit	*PID: 0*	*PID: 1*	*PID: 2*	*CPU (0, 1, or 2)*	*I/O Devices (0, 1, or 2)*
1	RUNNING: IO	READY	READY	1	0
2	WAITING	RUNNING: CPU	READY		
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					

14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
CPU Utilization					

NOTE: This table may have extra lines.

c) What happened if we change **IO_RUN_LATER** to **IO_RUN_IMMEDIATE**? Give the CPU utilization and explain why this happened.

Ans:

b)

SWITCH_ON_IO:
CPU UTILIZATION:

SWITCH_ON_IO and **IO_RUN_LATER** PROCESS 0: I/O, I/O, I/O PROCESS 1: CPU, CPU, CPU, I/O PROCESS 2: CPU, CPU, CPU, CPU					
Time unit	*PID: 0*	*PID: 1*	*PID: 2*	*CPU (0, 1, or 2)*	*I/O Devices (0, 1, or 2)*
1	RUNNING: IO	READY	READY	1	0
2	WAITING	RUNNING: CPU	READY	1	1
3	WAITING	RUNNING: CPU	READY	1	1
4	WAITING	RUNNING: CPU	READY	1	1
5	WAITING	RUNNING: IO	READY	1	1
6	READY	WAITING	RUNNING: CPU	1	1
7	READY	WAITING	RUNNING: CPU	1	1
8	READY	WAITING	RUNNING: CPU	1	1
9	READY	WAITING	RUNNING: CPU	1	1
10	RUNNING: IO	DONE	DONE	1	0
11	WAITING	DONE	DONE	0	1
12	WAITING	DONE	DONE	0	1
13	WAITING	DONE	DONE	0	1

14	WAITING	DONE	DONE	0	1
15	RUNNING: IO	DONE	DONE	1	0
16	WAITING	DONE	DONE	0	1
17	WAITING	DONE	DONE	0	1
18	WAITING	DONE	DONE	0	1
19	WAITING	DONE	DONE	0	1
20	DONE	DONE	DONE	0	0
CPU Utilization	11 / 19 = 0.5789				

c)

SWITCH_ON_IO and **IO_RUN_IMMEDIATE** PROCESS 0: I/O, I/O, I/O PROCESS 1: CPU, CPU, CPU, I/O PROCESS 2: CPU, CPU, CPU, CPU					
Time unit	*PID: 0*	*PID: 1*	*PID: 2*	*CPU (0, 1, or 2)*	*I/O Devices (0, 1, or 2)*
1	RUNNING: IO	READY	READY	1	0
2	WAITING	RUNNING: CPU	READY	1	1
3	WAITING	RUNNING: CPU	READY	1	1
4	WAITING	RUNNING: CPU	READY	1	1
5	WAITING	RUNNING: IO	READY	1	1
6	RUNNING: IO	WAITING	READY	1	1
7	WAITING	WAITING	RUNNING: CPU	1	2
8	WAITING	WAITING	RUNNING: CPU	1	2
9	WAITING	WAITING	RUNNING: CPU	1	2
10	WAITING	DONE	RUNNING: CPU	1	1
11	RUNNING: IO	DONE	DONE	1	0
12	WAITING	DONE	DONE	0	1
13	WAITING	DONE	DONE	0	1
14	WAITING	DONE	DONE	0	1
15	WAITING	DONE	DONE	0	1
16	DONE	DONE	DONE	0	0
CPU Utilization	11 / 15 = 0.733				

Answer: System resource (CPU) is effectively utilized. Because all these three instructions issued by process 0 are I/O operations. After issuing an I/O operation, we have to wait for it to complete. Other processes can be scheduled to run simultaneously. So it would be better if we could run the process that just completed an I/O again. It will issue another I/O instruction, then release CPU for other processes.

Q6 [12 points]

Consider two processes, P1 and P2, running concurrently on the same core, and they have the same priority. S1 and S2 are two semaphores, both with initial values of 0. The pseudocode for P1 and P2 is shown below, where x, y, and z are three shared variables between P1 and P2 with initial values of 1, 2 and 3, respectively. The execution of each line of the code is atomic.

Line	P1	P2
1	x=x+2	z=x+z
2	wait(S1)	signal(S1)
3	y=y+x	z=z+y
4	signal(S2)	wait(S2)
5	y=y+z	x=x+z

a) Assume that interruption only occurs when wait() is blocked, and P1 runs first. Please calculate the final values of x, y, and z after the two processes have finished.

x, y, z = 11, 13, 8

b) Repeat the same question above, but we now assume that P2 runs first.

x, y, z = 9,11,6

c) If the interrupt may occur after any line of the code and P1 runs first, please list all the possible values of x, y, and z after the processes have finished. Additionally, explain the conditions under which these values can be obtained.

The execution of Line 3 is out-of-order as signal(S1) wakes P1 up. After the execution of Line 3, there are two possible results.

- Type 1 (y=y+x runs first): x=3, y=5, z=11
- Type 2 (z=z+y runs first): x=3, y=5, z=8

Although the execution of Line 5 is also out-of-order, but it does not affect the final results.

Therefore, the possible values of x, y and z after running the two processes are:

- Type 1 (y=y+x runs first): x=14, y=16, z=11
- Type 2 (z=z+y runs first): x=11, y=13, z=8

Q7 (16 points)

There are two code examples of using fork() function, which is a popular way to create a subprocess. Please answer the following questions about these two codes.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
    int i;
    for (i = 0; i < 3; i++) {
        fork();
        magic();
    }
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
    printf("A");
    int is_parent = fork();
    printf("B");
    if (is_parent)
        wait(NULL);
    printf("C");
}
```

a) Which space does the fork() function get executed? (Kernal or User) Why?

Kernel space, since it will handle many privileged requests.

b) In the left code example, how many times will the function magic() be executed. How many subprocesses will be created. Please explain the detailed calculation steps.

24 times. 7 sub-processes are created.

c) What is the possible output of the right code example? If there are multiple possible outputs, please list them all.

Two possible answers. ABBCC or ABCBC.