Student ID: ____________________

Name: ____________________

CITY UNIVERSITY OF HONG KONG

Course code & title	:	CS3103 Operating Systems
Session	:	Semester B 2021-2022
Time allowed	:	120 minutes

There are **7** questions.

1. Answer ALL questions.
2. Handwrite your answer on white paper and scan/photo the answer and submit to Canvas at the end of the exam.

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

Candidates are allowed to use the following materials/aids:
Approved calculators.

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.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total

Q1. [8 points] 1. List at least two differences between Process and Thread.

Q2. [8 points] Suppose you are developing a new thread library and need to decide what multi-threading model (i.e., the mapping between user threads and kernel threads) to be used. The target application and support platform have the following features:

(1) the applications require high throughput

(2) there are a huge number of threads, while the burst time of each thread is very short

(3) there is only one CPU core

Will you choose the one-to-one multithreading model or many-to-one multithreading model? Why?

Q3. [16 points] Consider the following processes, with the arrival and processing times as given in the table:

$$T_{turnaround} = T_{completion} - T_{arrival}$$
$$T_{response} = T_{firstrun} - T_{arrival}$$

Process Name	Arrival Time	Processing Time
A	0	9
B	1	7
C	4	2
D	5	3

a. [10 points] Calculate the turnaround time and response time of each process for each of the execution of these processes using Round Robin (RR, quantum = 1), Preemptive Shortest Time-to-Completion First (Preemptive-STCF) scheduling.

b. [3 points] Discuss the tradeoff to consider when deciding the length of a time slice (Time Quantum) in Round Robin scheduling.

c. [3 points] If these processes are for interactive applications, i.e., they need to have good responsiveness performance, how would you design the scheduling algorithm?

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 1. Assume that Thread 1 will be scheduled to execute first.

Thread 1

```
.main           # the entry point of the thread
mov $4,%ax     # %ax is initialized to 4
.top            # a label
mov 3000,%cx    # get the value from the address to register %dx
add $1,%cx      # increment the value in register %dx by 1
mov %cx,3000    # store the value back to the address
sub $2,%ax      # decrement the value in register %ax by 2
test $0,%ax     # test the value in %register %ax
jne .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,%cx      # %cx is initialized to the value in 3000
.top            # a label
mov %cx,%dx    # copy the value in cx to dx
add $2,%dx      # increment the value in register %dx by 2
mov %dx,3000    # store the value back to the address
sub $1,%cx      # decrement the value in register %cx by 1
test $0,%cx     # test the value in %register %cx
jne .top        # jump back to .top if %cx is no equal to 0
halt            # stop running this thread
```

a. [4 points] What is register %cx used for in Thread 1 and Thread 2, respectively? What is the final value of the register %cx if there is no interrupt?
b. [4 points] What is the final value at the shared memory address 3000 if there is no interrupt?
c. [7 points] 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, if Thread 1 starts to execute first, then an interrupt occurs after Thread 1 finishes its 3rd instruction "`add $1,%cx`".

Q5. (13 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 two options:

- **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.

Now suppose we have two processes. Each CPU request takes 1 time unit, and each I/O request takes 5 time units (1 time unit on CPU and 4 time units on I/O device) to complete.

Suppose that the request sequence of process 0 and process 1 are **I/O, I/O, I/O, CPU**, and **I/O, CPU, I/O, CPU**, respectively, and process 0 runs first**.** When **SWITCH_ON_END is ON**, the states of each process (RUNNING: CPU, RUNNING: IO, WAITING, READY, DONE), the number of processes using CPU, the number of processes using I/O devices and CPU utilization are provided in the following table **as an example**.

Time unit	*PID: 0*	*PID: 1*	*CPU (0, 1, or 2)*	*I/O Devices (0, 1, or 2)*
1	RUNNING: IO	READY	1	0
2	WAITING	READY	0	1
3	WAITING	READY	0	1
4	WAITING	READY	0	1
5	WAITING	READY	0	1
6	RUNNING: IO	READY	1	0
7	WAITING	READY	0	1
8	WAITING	READY	0	1
9	WAITING	READY	0	1
10	WAITING	READY	0	1
11	RUNNING: IO	READY	1	0
12	WAITING	READY	0	1
13	WAITING	READY	0	1
14	WAITING	READY	0	1
15	WAITING	READY	0	1
16	RUNNING: CPU	READY	1	0
17	DONE	RUNNING: IO	1	0
18	DONE	WAITING	0	1
19	DONE	WAITING	0	1
20	DONE	WAITING	0	1
21	DONE	WAITING	0	1
22	DONE	RUNNING: CPU	1	0

23	DONE	RUNNING: IO	1	0
24	DONE	WAITING	0	1
25	DONE	WAITING	0	1
26	DONE	WAITING	0	1
27	DONE	WAITING	0	1
28	DONE	RUNNING: CPU	1	0
29	DONE	DONE	0	0
30	DONE	DONE	0	0
CPU Utilization	8 / 28 = 28.57%			

Now suppose the system uses **SWITCH_ON_IO**, fill the following table and calculate **CPU utilization**. When you compute the CPU utilization, **DO NOT** include the time units that all the processes have finished. Note that you do not need to draw the entire table in your answer. Instead, you only need to answer the blanks of the table. (18 points)

Time unit	*PID: 0*	*PID: 1*	*CPU (0, 1, or 2)*	*I/O Devices (0, 1, or 2)*
1		READY	1	0
2	WAITING			1
3	WAITING	WAITING	0	
4	WAITING	WAITING	0	
5	WAITING	WAITING	0	
6			1	1
7			1	1
8	WAITING		1	1
9	WAITING	WAITING	0	
10	WAITING	WAITING	0	
11		WAITING	1	1
12	WAITING	WAITING	0	2
13	WAITING		1	1
14	WAITING	DONE	0	1
15		DONE	0	1
16	RUNNING: CPU	DONE	1	0
17	DONE	DONE	0	0
18	DONE	DONE	0	0
19	DONE	DONE	0	0
CPU Utilization				

Q6. (15 points)

The TestAndSet() instruction can test a boolean variable (stored in the memory) and set its value to be TRUE, which is one of special atomic hardware instructions provided by modern machines. Here is the semantic of the TestAndSet( ) instruction.

```
boolean TestAndSet (boolean *target)
{
        boolean rv = *target;
        *target = TRUE;
        return rv;
}
```

The following is the pseudocode for implementing wait() and signal() semaphore operations using the TestAndSet() instruction to implement a mutex lock to ensure the atomicity of the operations in wait() and signal().

```
//the pseudocode for implementing wait() and signal() semaphore operations using
//the TestAndSet() instruction.

1  int target = 0;
2  int semaphore value = 0;
3
4  wait()
5  {
6
7    while (TestAndSet(&target) == 1);
8    if (semaphore value == 0)
9      {
10        atomically add process to a queue of processes
11        waiting for the semaphore and set target to 0;
12     }
13    else {
14       semaphore value--;
15      target = ?;
16    }
17
18 }
19
20 signal()
21 {
22
23     while (TestAndSet(&target) == 1);
24     if (semaphore value == 0 && there is a process on the wait queue)
25     wake up the first process in the queue of waiting processes
26     else
27     semaphore value++;
28     target = ?;
29
30 }
```

a. [3 points] Which mode is the TestAndSet () instruction get executed in? (Kernel mode or User mode) Why?

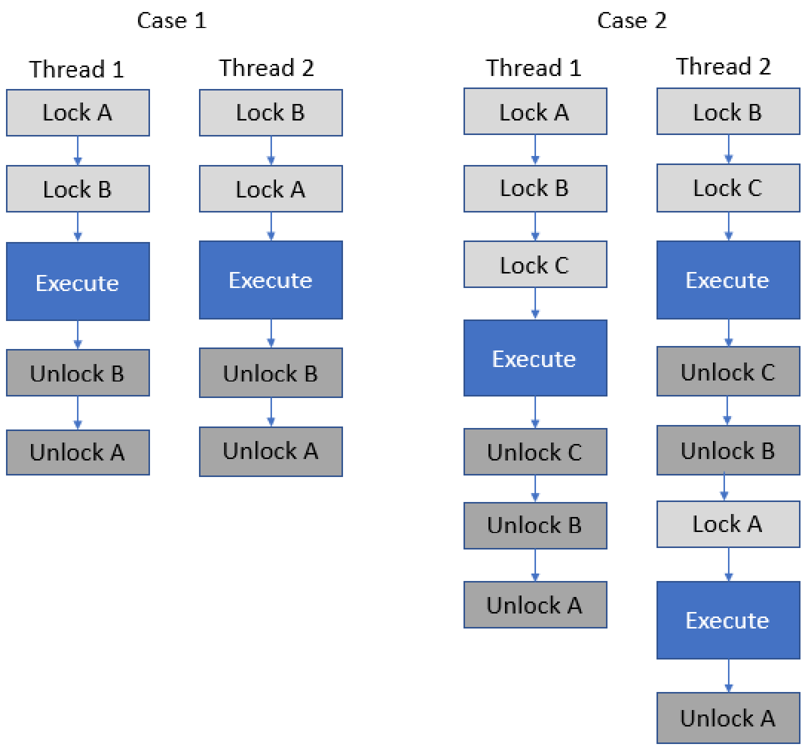
b. [6 points] In the above pseudocode, what are the values of *target* in line 15 and line 28, respectively? Why?

c. [6 points] The TestAndSet() instruction is used to implement a mutex lock to ensure the atomicity of the operations inside wait() and signal(). When does a process acquire the mutex lock and when does it release the mutex lock in wait() and signal() (please list the line number in the pseudocode)? Why?

Q7. (25 Points)

a. [5 points] Please explain what are *Deadlock* and *Starvation*.

b. [8 points] Suppose the system has two threads Thread 1 and Thread 2, and three mutex locks A, B and C. Consider the two cases, where the execution sequence of each thread is shown as in the following figure. Please answer whether there could be a deadlock in Case 1 and Case 2, respectively. Why?



c. [12 points] Suppose the system has two types of resource A and B. **Initially, A has 6 instances, and B has 6 instances**. There are 4 processes in the system, P0~P3. The maximal number of instances of each type of resource that may requested by the process, and the number of instances of each type of resource that has already been allocated to each process, is shown in the following figure. Now suppose P0 request 1 more instance of A and 2 more instances of B. According to Banker's algorithm, should we grant the requested resources to P0? Please write down the procedure of your analysis/calculation.

	Maximal request		Allocated	
	A	B	A	B
P0	3	5	1	1
P1	2	4	0	1
P2	1	1	1	0
P3	3	1	1	0