Assignment 1

Instructions

1. This assignment consists of **nine** questions.
2. **Due Date: Monday, October 9th, 2023**. Submit your answers to Canvas.
3. Before you begin, please take the time to review the course policy on academic integrity at: https://www.cityu.edu.hk/ah/ . Please write your own answer. All submitted answer will be scanned by anti-plagiarism software.

Please use the answer sheet provided to write all your answers.

1. Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

Ans) One could issue periodic timer interrupts and monitor what instructions or what sections of code are currently executing when the interrupts are delivered. A statistical profile of which pieces of code were active should be consistent with the time spent by the program in different sections of its code. Once such a statistical profile has been obtained, the programmer could optimize those sections of code that are consuming more of the CPU resources.

2. Describe the operating system's two modes of operation.

Ans) In order to ensure the proper execution of the operating system, most computer systems provide hardware support to distinguish between user mode and kernel mode. A mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), it must transit from user to kernel mode to fulfill the request.

3. Describe the relationship between the API, the system-call interface, and the operating system. Why use APIs rather than system calls?

Ans) The system-call interface of a programming language serves as a link to system calls made available by the operating system. This interface intercepts function calls in the API and invokes the necessary system call within the operating system. Thus, most of the details of the operating-system interface are hidden from the programmer by the API and are managed by the run-time support library.

4. What are the differences between user-level threads and kernel-support threads? Name and describe the four different multithreading models.

Ans) a) User-levels thread has no kernel support, so they are very inexpensive (in terms of resources demand) to create, destroy, and switch among threads does not cause interrupt to CPU.

Kernel support thread is more expensive (in resources) because system calls are needed to create and destroy them, and the kernel must schedule them to share access to CPU. They are more powerful because they are independently scheduled and block individually.

b)

- Many-to-One: Many user-level threads mapped to single kernel thread
- One-to-One: Each user-level thread maps to 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 kernel thread

5. Considering the following four algorithms: FCFS, non-preemptive SJF, preemptive SJF and RR. Please answer the following questions. (Each question may have more than one answer).

a. Which algorithm(s) has/have the maximum CPU utilization (suppose the context switch overhead can be ignored)?

b. Which algorithm(s) has/have the minimum average waiting time?

c. Which algorithms(s) is/are the fairest?

Ans)

a. All are the same (if do not consider context switching)

b. Preemptive-SJF

c. RR

6. Consider the following process A, B, C, D with arrival and processing times as given in the table:

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

Compute the finish time, response time, and turnaround time for each process for the following CPU Scheduling. Insert the values into the table in the answer sheet:

1) First-Come-First-Serve (FCFS)
2) Shortest-Job-First (SJF) (preemptive)
3) Round-Robin (RR) (time quantum = 1)

Note: For the answer of RR, we assume that the new coming job will be put into the end of the waiting queue.

		A	B	C	D
FCFS	Finish	7	11	15	18
	Response	0	5	7	6
	Turnaround	7	9	11	9
SJF	Finish	18	6	10	13
	Response	0	0	2	1
	Turnaround	18	4	6	4
RR	Finish	15	12	17	18
	Response	0	1	2	3
	Turnaround	15	10	13	9

Draw the schedule for each scheduling scheme by filling in the name of the current running process. Each square represents one time unit.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
FCFS	A	A	A	A	A	A	A	B	B	B	B	C	C	C	C	D	D	D	
SJF	A	A	B	B	B	B	C	C	C	C	D	D	D	A	A	A	A	A	
RR	A	A	A	B	A	B	C	A	B	C	A	B	D	C	A	D	C	D	

For RR, other reasonable answers also get points.

7. Consider the following two threads in the same process executing on the same core. The value at memory 2000 is initialized to 0. We also assume that Thread 1 is scheduled to execute first.

Thread_1

```
.main
mov $3,%ax
.top
mov 2000,%dx
add $1,%dx
mov %dx,2000
sub $1,%ax
```

```
test $0,%ax
jgt .top
halt
```

Thread_2

```
.main
mov $5,%ax
.top
mov 2000,%dx
add $2,%dx
mov %dx,2000
sub $1,%ax
test $0,%ax
jgt .top
halt
```

Note: the above-used assembly instructions are the same as the x86 assembly introduced in tutorial 3.

(a) What is the final value at the memory 2000 if the interrupt is disabled?

Ans) 13

(b) If the interrupt occurs in each instruction, can we get the same value at the shared memory 2000? If yes, please explain the reason; if not, what part of instructions in the code block of Thread 1 and Thread 2 needs to be protected?

Ans) No.

```
T1：mov 2000,%dx
    add $1,%dx
    mov %dx,2000
T2：mov 2000,%dx
    add $2,%dx
    mov %dx,2000
```

The following two questions are based on running the same `x86.py` simulator provided in tutorial 3.

Copy the `x86.py` simulator and program `wait-for-me.s`, which are in /public/cs3103/assignment1/ directory, to your working directory on gateway server.

8. Run: `./x86.py -p wait-for-me.s -a ax=1,ax=2 -R ax -M 2000`. This sets the `%ax` register to 1 for thread 0, and 0 for thread 1, and watches `%ax` and memory location 2000. How should the code behave? How is the value at location 2000 being used by the threads? What will its final value be?

Ans) First, thread 0 tests if the %ax register equals to 2. This is not satisfied for the given initial %ax value. Thread 0 continues to execute waiter section, which gets the value in memory location 2000 (value 0) back to %cx register and check if it isn't equal to 3, and if so, jumps back to the top of the waiter section again. This condition holds until it is interrupted. So thread 0 loops. After the condition gets set by thread 1 by executing the signaller section, the last “test” in the waiter section finds %cx and 3 to be equal, and thus the subsequent jump does NOT take place, and the program finally halts.The value at location 2000 is being used as a flag to determine whether the signaller or waiter should be executed. Its final value is 3.

9. Now switch the inputs: `./x86.py -p wait-for-me.s -a ax=2,ax=1 -R ax -M 2000`. How do the threads behave? What is thread 0 doing? How would changing the interrupt interval (e.g., `-i 1000`, or perhaps to use random intervals) change the trace outcome? Is the program efficiently using the CPU?

Ans) First, thread 0 tests if the %ax register equals to 2. This condition is true for the given value of ax for thread 0, so it jumps to the signaller section, which stores value 3 to memory location 2000. After thread 0 halts, thread 1 starts running, the condition for jumping to signaler section is not satisfied, so it continues to execute the waiter section, which loads the value from memory location 2000 to %cx register and tests if it isn't equal to 3. Because the memory address 2000 is already set to 3 by thread 0, so it will not jump and stop running this thread.

For question 8, changing the interrupt interval to a large number will increase loop iterations of thread 0 and the waiting time of thread 1. For this question, there is no difference with larger interval and small interrupt interval adds more interrupts.

The program in question 8 is not efficiently using the CPU as thread 0 spins and wastes CPU time. But this question is more efficient.