IT5002 Computer Systems and Applications
Assignment Answer Book

Student Number: A0255954R

Name: Wu Tong

Total Marks: _____ / 20

DEADLINE:

PDF this file, naming it AxxxxxxY.pdf (where AxxxxxY is your student ID), and submit to Canvas by **Monday 6 November 2023, 2359 hrs**.

Question 1. (3 marks)

Examine the code you have typed in, and explain how this code works. That is, why it produces two threads that print once per second and once every two seconds.

For thread 1, it receives one parameter. The code in this function first passes the global variable "x" into the thread function. After that, the thread goes into an infinite loop, which means it will run forever until the user interrupts it. In the loop, it first plus one to this variable. Then, it prints a message to indicate the thread number and the current value of "x". Finally, it uses the "sleep" function in the time module to idle for specific seconds, which is determined by the passing variable, "param", which is 1 in this case, then goes to the start of the loop.

For thread 2, it receives two parameters. The code in this function first passes the global variable "x" into the thread. After that, the thread goes into an infinite loop. In the loop, it first multiplies the global variable "x" by 2. Then, it prints a message to indicate the thread number and prints the input variable "param2" and the current value of "x". Finally, it uses the "sleep" function in the time module to idle for specific seconds, which is determined by variable "param1", which is 2 in this case.

Outside the function defining code, the global variable "x" is initiated as 1. After the function defining code, two threads are created and started: "th1" and "th2". These two threads have a target to function "thread1" and "thread2" respectively. Function "thread1" is passed value 1, which is mapped to the variable "param", and function "thread2" is passed two values 2, 123, which is mapped to the variable "param1" and "param2".

According to the code, since thread "th1" is first created and started, the first line of output is created by "th1", and the global variable "x" is updated. Then, the thread "th1" waits for 1 second, while the thread "th2" is created and started within much faster than 1 second, so the second line of the output is created by "th2" and global variable "x" is updated. The thread "t1" prints once per second, and "t2" prints once every two seconds. Since the other arithmetic and console output operations' running time can be seen as negligible, the running time of these two threads can be determined by their sleep time, which is 1 second and 2 seconds, respectively.

Question 2a. (2 marks)

Run the code with the following values of “n”, recording whether you get the correct result:

n = 10, Correct: Y
n = 1000, Correct: Y
n = 1000000, Correct: N
n = 10000000, Correct: N

Question 2b. (4 marks)

Some of the answers in 2a are incorrect. Explain why you get incorrect results.

The first thread calculates the sum of the first “n” positive integers and stores the result in the “result” variable, which has many arithmetic operations. Those large numbers of operations can take a significant amount of time to CPU. It will grow significantly when “n” is increased exponentially.

The second thread multiplies the “result” by 3, which is the target variable of the first thread, which means the first thread’s result will be stored in the variable “result”. However, these two threads are running concurrently, and there is no synchronization between two threads, like mutex. No synchronization means that the second thread may perform a multiplication operation before the first thread stores the result in the “result” variable, which means the second thread may get the wrong number from “result”. It especially will happen when the first thread needs a long CPU time to finish.

In conclusion, the second thread may get the wrong number from the variable “result” because it may fetch this variable before the first thread finishes since these two threads do not have synchronization like mutex.

Question 3. (4 marks)

t3.py now shows the correct result even for large n. Explain, using the idea of queues, how this program works to give the correct result. You may Google for what queues are.

In the code, “queue” has been used in variables “resultQ” and “finalQ”. Class “queue” provides a FIFO queue, which can use “put()” function to put the data into the queue and use “get()” to fetch the data in the queue. It is thread-safe, meaning multi-threading access cannot fetch the wrong data.

In this case, two variables are created as a FIFO queue. Thread1 uses the function “put()” to put the result into “resultQ” when the calculation is completed. At the same time, thread2 is running concurrently with thread1. However, the “resultQ” variable has no data until thread1 completes the calculation and puts the data into the queue, so thread2 has to wait until thread1 completes. Hence, it proves that this code is thread-safe and that this program works to give the correct result.

Question 4. (2 marks)

Since p1.py and t1.py are essentially the same program, they should produce the same results, but do not. Explain why.

This is caused by the difference between the threading module and the multiprocessing module.

In the threading module, it creates different threads under one process. Threads can be seen as holes in the process, which can share the same memory space while concurrently running. Hence, in t1.py, two threads can access and modify the global variable "x".

In the multiprocessing module, it creates different processes. Processes are parallel and do not share the memory space. Hence, the modification in a process will not affect the variable's value in other processes.

Question 5. (5 marks)

Detail the changes you made to p1.py to make it give the same results at t1.py below and explain why it works.

Add Queue class from multiprocessing: `from multiprocessing import Queue`
Modification of the input variables of two functions:

1. For function "process1", add input variable "queue1", "queue2":
   `def process1(param, queue1, queue2):`
   Pass two queue to process1 so that the function can access them.
2. For function "process2", add input variable "queue1", "queue2":
   `def process2(param1, param2, queue1, queue2):`
   Same explanation as above.

Modification of the structure of two process functions and the main function:

1. In function "process1", add code after "`x = x + 1`":
   ```
   if x not in (2, 5):
         print("This is process 1. x is %d" % x)
         x += 1
   ```
   This is because the first two rounds of two process output is process1 output 1 line and process2 output 1 line. This is caused by their sleep time and start order. After the first two rounds, the variable "x" plus another 1 since the process1 will run two times before the process2 runs, and these two runs has no inter-processes communication.
2. In function "process1", add code before "`time.sleep(param)`":
   ```
   queue1.put(x)
   x = queue2.get()
   ```
   This is because process1 needs to put the calculated "x" into the queue so that process2 can get it. Then, get "x" from the process2 output for the following loop calculation.
3. In function "process2", add code after "`while True:`":
   ```
   x = queue1.get()
   ```
   This is to get "x" from the output of process1.
4. In function "process2", add code before "`time.sleep(param1)`":
   ```
   queue2.put(x)
   ```
   This is to put the output of process2 into the queue in order to let process1 to able to get it.
5. In the main function, modify the original code to:
   ```
   queue1 = Queue()
   queue2 = Queue()
   ```

```
# Create the first thread
p1 = multiprocessing.Process(target=process1, args=[1, queue1, queue2])
p1.start()
p2 = multiprocessing.Process(target=process2, args=[2, 123, queue1, queue2])
p2.start()
```

This is to create two queues for queue1 and queue2. Queue1 stores the result of process1 and pass to process2. Queue2 stored the result of process2 and pass to process1. Then, change the process creation code by adding the queue1 and queue2 into argument field.

After modifying these, the code can perform the same result as the previous program t1. Two queues "queue1" and "queue2" achieve the communication between the processes. For each loop that process1 and process2 runs, they must wait the previous process's task finished and put the result into the queue. In detail, according to the starting order, so in first two round process1 and process2 will both run 1 time respectively. After that, process1 will run two times and process2 will run 1 time in each round. The output in console is below:

```
This is process 1. x is 2
This is process 2. Param 2 is 123. x is 4
This is process 1. x is 5
This is process 2. Param 2 is 123. x is 10
This is process 1. x is 11
This is process 1. x is 12
This is process 2. Param 2 is 123. x is 24
This is process 1. x is 25
This is process 1. x is 26
This is process 2. Param 2 is 123. x is 52
……
```