**CS2106 Introduction to Operating Systems**

**Lab 3**

**Answer Book**

|  |  |
| --- | --- |
| **Name: Ling Guan Yu** | **Name: NIL** |
| **Student ID: A0308292A** | **Student ID: NIL** |
| **Lab Group: B20** | **Lab Group: NIL** |

**Part 1**

Question 1.1 (1 mark)

All the “<some number> I am child X” appears before most of the numbers as each child process blocks when it calls “usleep(250000);” which allows a child process created later to print its “<some number> I am child X”. Also, “fflush(stdout);” is called before “usleep(250000);” which causes the 1 number from “printf("%d ", j);” stored in the output buffer of stdout (stream) to be flushed (need since printing without ‘\n’) so all the “<some number> I am child X” appears before most of the numbers instead of all of the numbers.

This suggests that the time quantum is longer than the total time needed to call “printf("I am child %d\n", i);”, “printf("%d ", j);” and “fflush(stdout);” in succession.

The interleaving happens due to the combination of “printf("%d ", j);”, “fflush(stdout);” and “usleep(250000);” called in succession which causes a number to be printed to the screen from each child process before it blocks and gives up the CPU due to usleep.

The operating system’s scheduling affects the interleaved order of output.

Question 1.2 (1 mark)

The lock variable is not atomic, meaning any operation involving the lock variable can be interrupted. Hence trying to use it as a synchronization mechanism will fail and a race condition will still occur.

Question 1.3 (1 mark)

Step 1:

I created the shared variable “turn” and initialized it to 0 at the start of main().

Step 2.1:

I added “while(turn[0] != i);” above “break;” in the loop used to create processes.

Step 2.3:

I added “turn[0] = turn[0] + 1;” as the last statement to execute in the if-block for “pid == 0”.

Extra Step:

I added code to detach and free the shared variable “turn” as the last statements to execute in the else-block for “pid == 0”.

Turn variables might be less efficient of multi-core systems due to busy waiting occurring on each core from checking the turn variable every frame, wasting CPU cycles per core which could have been used to make progress in other computations.

Question 1.4 (1 mark)

sem\_init params (from left to right):

1st param (pass in address of the semaphore to initialize)

2nd param (pass in 0 for semaphore shared between threads of a process or non-0 for semaphore shared between processes)

3rd param (pass in value to initialize semaphore with)

sem\_wait param:

1st param (pass in address of semaphore, value of semaphore is decremented if it is non-0 and call blocks if value of semaphore is 0 until it becomes non-0 or a signal handler interrupts the call)

sem\_post param:

1st param (pass in address of semaphore, value of semaphore is incremented, resulting value is non-0 if no threads or processes were blocked waiting for semaphore to be unlocked or 0 if a thread or process was unblocked and allowed to return from sem\_wait)

Question 1.5 (1 mark)

Since the semaphore is not part of shared memory, when the child process calls sem\_wait, it will block indefinitely as the semaphore has the value of 0 on initialization. Since the parent process calls wait and the child process never exits, the parent process will also be blocked indefinitely, causing the program to hang.

Question 1.6 (1 mark)

**Part 2**

Question 2.1 (1 mark)

Question 2.2 (1 mark)

**Part 3**

Question 3.1 (1 mark)

Question 3.2 (1 mark)

Question 3.3 (1 mark)

**TOTAL:** \_\_\_\_\_\_ / 11