**CS2106 Introduction to Operating Systems**

**Lab 3**

**Answer Book**

Please read the instructions in the main lab sheet before completing this document. Submission deadline is **1 pm, Sunday 31 March 2024**. The folder will stay open slightly after this, but once the folder closes, **absolutely no submissions will be allowed.**

**Submission checklist:** A ZIP file called AxxxxxxY.zip, where AxxxxxxY is the student ID of the student submitting. The ZIP file should contain:

* Your answer book, properly renamed.
* Your barrier.c and barrier.h
* Your sum-par.c

|  |  |
| --- | --- |
| **Student 1** | |
| Name: | Kenneth Seet |
| Student ID (AxxxxxxY): | A0258173Y |
| Group (Bxx): | B18 |
| **Student 2** | |
| Name: | Daniel Wang |
| Student ID (AxxxxxxY): | A0255689H |
| Group (Bxx): | B11 |

**Part 1**

Question 1.1 (1 mark)

The output suggests that the time quantum used by the operating system scheduler is shorter than the time it takes for a child process to complete.

Question 1.2 (1 mark)

This discrepancy occurs because each child process has its own copy of the counter variable due to the nature of process forking. When a child process modifies its own copy of the counter, it does not affect the counter in the parent process or any other child process.

Question 1.3 (1 mark)

The output suggests that the time quantum is less than 250000 milliseconds. We can observe from the output of the program that control gets handed over from child to child after the counter is incremented, and in the code, usleep(250000) is called after the counter is incremented.

Question 1.4 (1 mark)

I added lines to include the <sys/ipc.h> and <sys/shm.h> header files.

#include <sys/ipc.h>  
#include <sys/shm.h>

Next, I removed the counter variable, declaring an int\* shm instead.

*int* i;  
*int* \*shm;

Next, I created and attached the shared memory region, adding code which handled errors in getting and attaching the shared memory region. I also initialised \*shm, which replaced my counter, as 0.

// create Shared Memory Region  
*int* shmid = *shmget*(IPC\_PRIVATE, *sizeof*(*int*), IPC\_CREAT | *0600*);  
*if* (shmid == -*1*)  
{  
 *perror*("shmget");  
 *exit*(EXIT\_FAILURE);  
}  
// attach the shared memory region to this process  
shm = *shmat*(shmid, NULL, *0*);  
*if* (shm == (*int*\*) -*1*)  
{  
 *perror*("shmat");  
 *exit*(EXIT\_FAILURE);  
}  
\*shm = *0*;

In the else if block of the code which ran for child processes, I replaced the counter variable with \*shm instead, and incremented it accordingly for every iteration of the loop. I also replaced the counter variable in the printf statement with \*shm instead.

*else if* (pid == *0*)  
{  
 // Child process  
 *printf*("Child %d starts\n", i + *1*);  
 // Simulate some work  
 *for* (*int* j = *0*; j < *5*; j++)  
 {  
 (\*shm)++;  
 *printf*("Child %d increment counter %d\n", i + *1*, \*shm);  
 *fflush*(stdout);  
 *usleep*(*250000*);  
 }  
 *printf*("Child %d finishes with counter %d\n", i + *1*, \*shm);  
 *exit*(EXIT\_SUCCESS);  
}

Finally, in the final printf statement which prints the final counter value, I replaced counter with \*shm. After that, I added code which detached and removed the shared memory region, adding code which handled errors in doing so as well.

// Print the final value of the counter  
*printf*("Final counter value: %d\n", \*shm);  
// Detach the shared memory segment  
*if* (*shmdt*(shm) == -*1*) {  
 *perror*("shmdt");  
 *exit*(EXIT\_FAILURE);  
}  
  
// Remove the shared memory segment  
*if* (*shmctl*(shmid, IPC\_RMID, NULL) == -*1*) {  
 *perror*("shmctl");  
 *exit*(EXIT\_FAILURE);  
}

Output:

A screenshot of a computer program

Description automatically generated

Question 1.5 (1 mark)

Increasing NUM\_CHILDREN and the loop variable leads to the final counter variable being smaller than (NUM\_CHILDREN \* loop variable), and this is because of the increased potential for race conditions arising from the number of increments being increased and the number of children being increased, leading to the final counter variable being inaccurate and the written value of the counter variable being overwritten by other children.

Question 1.6 (1 mark)

The lock variable may fail to coordinate the processes as processes can still get preempted while they are using the lock before they get to set the lock back to 1. This leads to the processes not being able to coordinate with each other, or even result in the processes being stuck in the busy-waiting loop.

Question 1.7 (1 mark)

From question 1.4, I used a shared integer array instead of a shared integer variable, with the first integer in the array representing the counter, and the second integer in the array representing the turn variable. I initialized both as 0.

// shm[0] is the counter, shm[1] is the turn  
shm[*0*] = *0*;  
shm[*1*] = *0*;

I introduced a busy-wait loop in the else-if block which made the processes wait for their turn while it was not theirs. I also had to update the variable in the printf statement to reflect the changes I made to the shared memory.

After the process exits the for loop, I also incremented the turn variable by 1 to allow the next process to start and increment the counter.

*else if* (pid == *0*)  
{  
 // wait if it is not the process' turn  
 *while* (shm[*1*] != i);  
 // Child process  
 *printf*("Child %d starts\n", i + *1*);  
 // Simulate some work  
 *for* (*int* j = *0*; j < *5*; j++)  
 {  
 shm[*0*]++;  
 *printf*("Child %d increment counter %d\n", i + *1*, shm[*0*]);  
 *fflush*(stdout);  
 *usleep*(*250000*);  
 }  
 *printf*("Child %d finishes with counter %d\n", i + *1*, shm[*0*]);  
 // release the turn  
 shm[*1*]++;  
 *exit*(EXIT\_SUCCESS);  
}

Like earlier, I also carried on to detach and free the shared memory.

// Print the final value of the counter  
*printf*("Final counter value: %d\n", shm[*0*]);  
// Detach the shared memory segment  
*if* (*shmdt*(shm) == -*1*) {  
 *perror*("shmdt");  
 *exit*(EXIT\_FAILURE);  
}  
  
// Remove the shared memory segment  
*if* (*shmctl*(shmid, IPC\_RMID, NULL) == -*1*) {  
 *perror*("shmctl");  
 *exit*(EXIT\_FAILURE);  
}  
  
*return 0*;

Question 1.8 (1 mark)

Question 1.9 (1 mark)

Question 1.10 (1 mark)

**Part 2**

Question 2.1 (1 mark)

Question 2.2 (1 mark)

**Part 3**

Question 3.1 (1 mark)

(For grader only)

**Report: \_\_\_\_\_\_\_\_\_\_\_\_ / 13**

**Demo: \_\_\_\_\_\_\_\_\_\_\_\_\_ /7**

**TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_/20**