Project 1 Performance Evaluation

Tiffany Nguyen
Gallogly College of Engineering, University of Oklahoma
tiffanybnguyen@ou.edu

Abstract– This document presents the performance evaluation of my solution of Fall 2024 Introduction to OS Project 1. It will contain my solution to the programming assignment as well as an explanation and analysis.

I. INTRODUCTION

The purpose of project 1 was to introduce the concept of shared memory and protection of shared memory. The main objective of this project is to develop a program that creates 4 children and increments a shared variable by one to 100,000, 200,000, 300,000, 400,000 and 500,000 respectively. After all the children have finished, the parent process will print the process ID of each child as they finish execution, release the memory, and terminate the program. This paper will address the solution, observations, how the code functions, and the thought process behind the code.

A. Solution

The solution to the coding problem posed in the introduction is shown below.

```
nguy0850@gpel13:~/cs3113/project1$ ./project1
From process 1: counter = 96803
Child with ID 3526717 has just exited.
From process 2: counter = 218503
Child with ID 3526718 has just exited.
From process 3: counter = 341043
Child with ID 3526719 has just exited.
From process 4: counter = 520462
Child with ID 3526720 has just exited.
End of Program
nguy0850@gpel13:~/cs3113/project1$
```

The program creates four child processes, increments the shared memory total a respective amount until the child process reaches its intended limit, prints the amount it incremented the total to (e.g. “From process 1: counter = 96803”), and exits. For every child that exits, the parent will print their ID and state that the child has exited (e.g. “Child with ID 3526717 has just exited.”). At the very end of the program, the parent releases the memory and terminates with the phrase “End of Program.”

B. Observations

The order in which the processes print varied every now and then. In the screenshot shown below, the program prints the processes and some of the exit statements out of order.

```
nguy0850@gpel13:~/cs3113/project1$ ./project1
From process 1: counter = 97781
Child with ID 3525697 has just exited.
From process 4: counter = 346232
From process 2: counter = 216546
Child with ID 3525700 has just exited.
Child with ID 3525698 has just exited.
From process 3: counter = 333006
Child with ID 3525699 has just exited.
End of Program
nguy0850@gpel13:~/cs3113/project1$
```

Some reasons I thought of include the fact that the processes run in parallel and therefore have no specific order in which the statements must be printed. For example, if the process 1 and process 2 increment and reach their specified limit at the same time, then it is possible that they will print at almost the same time, causing a misalignment in the statements. The program will run differently depending on the OS of the machine it is running on because different OS’s schedule their OS’s in different ways. However, no matter how many times the program is run, the very end of the program will always be symbolized by the phrase “End of Program”.

C. How the Code Functions

This program contains one main function and four helper functions for the child processes. The main function declares all the necessary variables, uses shmget() to create a shared memory segment, uses shmat() to attach the shared memory segment to the process’s address space, and initializes the shared memory value to 0. It then forks a total of four children which then call their respective helping

functions, and increment the shared memory value until they reach their respective capacities. The parent waits for each child to terminate, and then prints the PID of the child as it exits. After all four children have died, the program will detach the shared memory segment from the process's address space using shmdt(), mark the segment for deletion using shmctl(), and terminate the program after printing "End of Program". The four child processes are spawned in sequentially, but run in parallel. Process one increments the shared total by one each time until the total reaches the limit of 100,000. Process two does the same thing for 200,000, process three for 300,000, and process four for 500,000. Since the shared memory is constantly being accessed by four different children, there will be some discrepancies in the total.

D. Thought Process Behind the Code

I initially approached this problem with mmap(), however the professor released slides and additional information for the project using sys/shm.h instead. Moving forward with the professor's technique, it took multiple attempts to create the forks for each child. I had originally made a for loop to create the forks which created the appropriate amount of children, however I was unable to specify what functions I wanted each child to run. Eventually, I settled and made four if-statements, which were able to provide the correct functions for each child. Then, I created the functions for each process and implemented the loop for the parent process to print the PID of the children. The hardest part of this was actually comprehending programming assignment details, but once those were figured out I had no trouble implementing the rest. Overall, the project was not difficult as the professor provided many resources and code hints for the shared memory segment. I followed the code hints for creating, attaching, detaching, and marking the memory segment, making sure that I understood each part of the process and that I was implementing methods of sys/shm.h correctly.