

Maharshi Shah

Valentina Korzhova

Operating Systems - COP 4600

09 September 2018

## Shared Memory Project 1

Shared memory is a method by which program processes can exchange data rapidly by reading and writing using the regular operating system services. For example, a client process may have data to pass to a server process that the server process is to modify and return to the client. In this project, we were supposed to use the concept of shared memory to create 4 interdependent processes sharing a variable called 'total'. Each process increments total by 100,000, 200,000, 300,000 and 500,000 times. `fork()` calls are used to invoke the child processes. This report discusses the results and conclusion of running the code developed for creating, sharing and terminating shared memory between processes.

## RESULTS

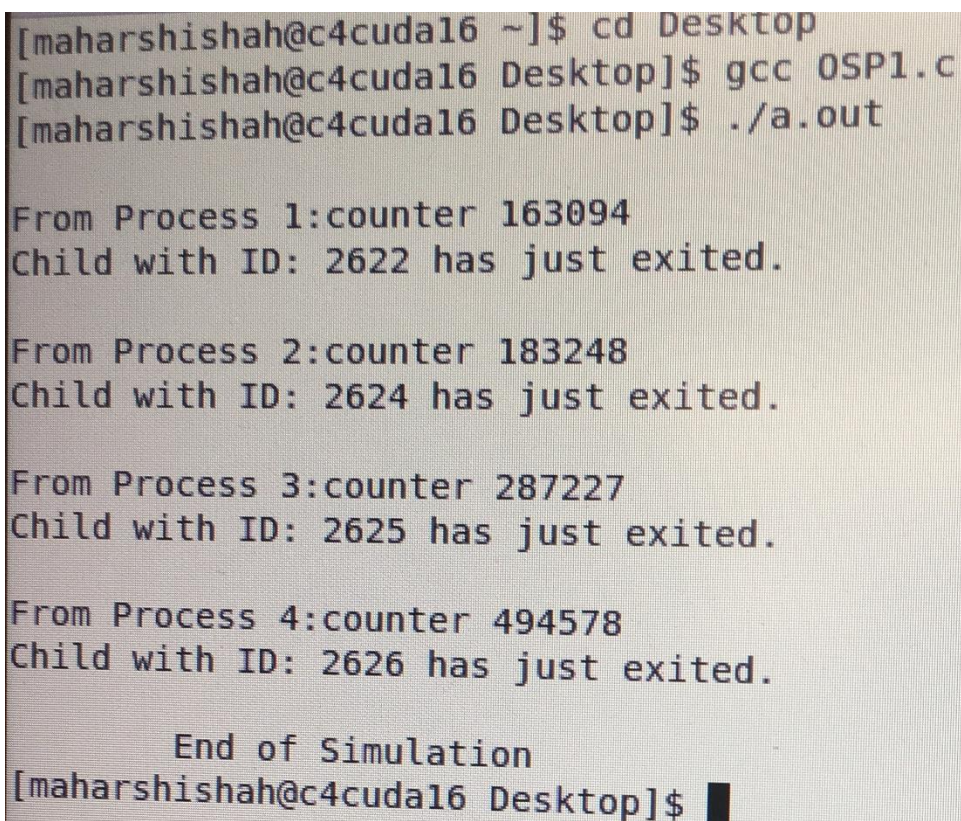
The following page shows the images after running the code developed that creates 4 processes and produces child processes. The output has the counter of a process and the child ID of a particular parent process that returns it. In order to run the code on the C4 machines, following are the steps:

- **gcc OSP1.c**                      To compile
- **./a.out**                          To run the code

More information on compiling and running is provided in the readme file attached. Performance was not evaluated since there is no comparison to be made between the processes or algorithms

running them. Following images are the result of running the code using the commands mentioned on the previous page. The timing of the print statement reflects the corresponding processes exit. The processes do not exit at the same time, instead, they exit immediately after reporting the value of the total. That is the reason why the output(s) below differ from the sample output given in the project description. The timing of the process is still accurate with each process exiting and printing the child ID only after repeating the addition a certain number of times (based on the process number).

- Sample Output 1:



```
[maharshishah@c4cuda16 ~]$ cd Desktop
[maharshishah@c4cuda16 Desktop]$ gcc OSP1.c
[maharshishah@c4cuda16 Desktop]$ ./a.out

From Process 1:counter 163094
Child with ID: 2622 has just exited.

From Process 2:counter 183248
Child with ID: 2624 has just exited.

From Process 3:counter 287227
Child with ID: 2625 has just exited.

From Process 4:counter 494578
Child with ID: 2626 has just exited.

      End of Simulation
[maharshishah@c4cuda16 Desktop]$
```

- Sample Output 2:

```
From Process 1:counter 96446
Child with ID: 2839 has just exited.

From Process 2:counter 215070
Child with ID: 2840 has just exited.

From Process 3:counter 322811
Child with ID: 2841 has just exited.

From Process 4:counter 528389
Child with ID: 2842 has just exited.

      End of Simulation
[maharshishah@c4cuda16 Desktop]$
```

## CONCLUSION

Although the way the print statements are ordered make it seem like they are getting printed before waiting for all children to exit, the `wait(NULL)` statement used inside the for loop takes care of this not happening. Overall, the code runs correctly by producing 4 processes and 4 Child IDs after execution of each process. The shared variable gets incremented by 1 several different numbers of time and the data is not lost or altered during the interprocess communication (IPC). The data is made directly accessible to both processes without having to use the system services.