COP 4600 Operating Systems

Project 2 - Semaphores

Gregory O'Marah - Net ID: gdo - U34240613

The purpose of this project was to show how we could modify a shared memory segment between three processes. In order to achieve the protection on the critical section of the process, a semaphore was employed to determine when a process could update the shared variable in memory, or when the process should wait. Some experimentation with different methods and placement of the components was required to achieve optimal results.

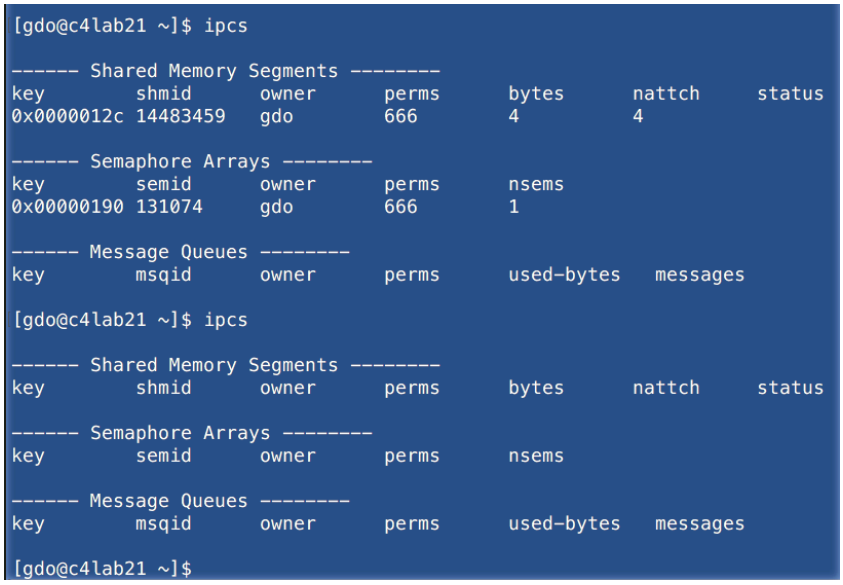
The code from the first project was used as a reference, and then it was updated to include three child processes. Then the semaphore was created and implemented to protect the critical section of each child process. The main job of the children was to increment a counting variable of type integer, by a factor of 1 each time it executes. The counting variable is accessible by each child as it is stored in shared memory. Child process 1 increments the count 100,000 times, while child process 2 increments the count 200,000 times, and child process 3 increments the count 300,000 times. Thus we expect the count to increment from 0 up to 600,000.

At the same time while the child processes are running, the parent process is waiting and will be tasked with printing out a statement for each child process that finishes, along with its Process ID. This was accomplished by using the wait command. The code was written and tested about 100 times. A sampling of 10 runs is included in the table below. According to my analysis, I have concluded that the semaphore did its job and protected the critical section of each process. This allowed the processes to increment a variable in shared memory up to a total of 600k.

Run	P1 Count	P2 Count	P3 Count
1	298062	503135	600000
2	300531	502807	600000
3	294879	504029	600000
4	293499	505368	600000
5	275849	507802	600000
6	291241	492933	600000
7	288191	499780	600000
8	299795	500077	600000
9	295103	507930	600000
10	299461	489998	600000
Average	293661	501386	600000

The average count at the time each process exited shows that the processes were in fact running concurrently and taking turns in order to update the shared count variable. To further confirm that no two processes were able to update the shared memory at a time, I placed print statements inside and outside each critical section and observed that at no time would any of them execute until the other (active process) had finished.

I also observed in the Linux environment, that I could check the resources while the program was executing and verify that a shared memory segment was allocated, and also a semaphore array was created. At the termination of the program both of these resources were freed in order to prevent memory or resource leaks. Attached below are screen shots from the executing of the program and the observation of the resource allocation and freeing.



```
[gdo@c4lab21 ~]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000012c 14483459   gdo        666        4          4

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000190 131074     gdo        666        1

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

[gdo@c4lab21 ~]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

[gdo@c4lab21 ~]$
```

```
From Process 1: counter = 298062.
From Process 2: counter = 503015.
From Process 3: counter = 600000.

Child with ID 13801 has exited.
Child with ID 13802 has exited.
Child with ID 13803 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 300531.
From Process 2: counter = 502807.
From Process 3: counter = 600000.

Child with ID 13704 has exited.
Child with ID 13705 has exited.
Child with ID 13706 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 294879.
From Process 2: counter = 504029.
From Process 3: counter = 600000.

Child with ID 13783 has exited.
Child with ID 13784 has exited.
Child with ID 13785 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 293499.
From Process 2: counter = 505368.
From Process 3: counter = 600000.

Child with ID 13792 has exited.
Child with ID 13793 has exited.
Child with ID 13794 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 275849.
From Process 2: counter = 507802.
From Process 3: counter = 600000.

Child with ID 13774 has exited.
Child with ID 13775 has exited.
Child with ID 13776 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 291241.
From Process 2: counter = 492933.
From Process 3: counter = 600000.

Child with ID 13765 has exited.
Child with ID 13766 has exited.
Child with ID 13767 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 288191.
From Process 2: counter = 499780.
From Process 3: counter = 600000.

Child with ID 13756 has exited.
Child with ID 13757 has exited.
Child with ID 13758 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 299795.
From Process 2: counter = 500077.
From Process 3: counter = 600000.

Child with ID 13719 has exited.
Child with ID 13720 has exited.
Child with ID 13721 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 295103.
From Process 2: counter = 507930.
From Process 3: counter = 600000.

Child with ID 13729 has exited.
Child with ID 13730 has exited.
Child with ID 13731 has exited.

    End of  Simulation.
```

```
From Process 1: counter = 299461.
From Process 2: counter = 489998.
From Process 3: counter = 600000.

Child with ID 13738 has exited.
Child with ID 13739 has exited.
Child with ID 13740 has exited.

    End of  Simulation.
```