COP 4600-002: Operating Systems
Fall 2017

Project #2: Semaphores

Alan Rodriguez, U86831061

I. INTRODUCTION

FOR this project 4 processes were created. These processes modify a shared variable "total". The processes increment until their respective bounds, 100,000, 200,000, 300,000 and 500,000.
A Semaphore is introduced in order to ensure that only one process can be in its critical code at a time. This is applied by calling Pop before the critical code, and Vop after.

Child processes are created and modify the shared variable as well. The parent process does not terminate until all child processes have exited. This is accomplished by using the waitpid function.

Passing in our semaphore key into the function semget returns our semaphore id. Passing our semaphore union into the semctl along with the sem_id initializes our semaphore.
Once all the parent processes have terminated upon completion, the shared memory must be detached. This is done by passing our shared variable to the function call, shmdt(total).
Now that shared memory is detached, it can be removed with shmctl(shmid, IPC_RMID, NULL); This ensures that shared memory is removed and there is no memory leakage.

After the processes have terminated and memory is detached, the semaphore is deallocated using the semctl function and passing in the sem_id.

The program was executed several times, the results were similar:

II. RESULTS

```
From Process 1: counter = 399937
Child 21324 pid has just exited.
From Process 2: counter = 699995
Child 21325 pid has just exited.
From Process 3: counter = 900003
Child 21326 pid has just exited.
From Process 4: counter = 1100000
Child 21327 pid has just exited.
End of Program.
```

```
From Process 1: counter = 399928
Child 21353 pid has just exited.
From Process 2: counter = 699996
Child 21354 pid has just exited.
From Process 3: counter = 899999
Child 21355 pid has just exited.
From Process 4: counter = 1100000
Child 21356 pid has just exited.
End of Program.
```

```
From Process 1: counter = 399917
Child 21358 pid has just exited.
From Process 2: counter = 700004
Child 21359 pid has just exited.
From Process 3: counter = 899999
Child 21360 pid has just exited.
From Process 4: counter = 1100000
Child 21361 pid has just exited.
End of Program.
```

III. CONCLUSION:

From this data, I was able to conclude that the program would always produce similar results. The results of the counter only vary by a small margin which implies that the semaphore is working properly and only allowing one process at a time to enter its critical code.
Therefore, the counter value is preserved. In the critical section of the process, the shared memory, total, is modified. The value of total is accessed by only one process at a time. This is ensured by surrounding our critical code with the function calls Pop and Vop.