Christo Pettas

Project Description

A memory location is shared by three processes. Each process independently tries to increase the content of the shared memory location from 1 to a certain value by increments of one. Process 1 has target of 100000, Process 2's target is 200000 and the goal of 3 is 300000. When the program terminates, the shared memory variable will have a total of 600000. this value will be output by whichever of the three processes finishes last.
In this project, modifies the assignment1 to protect the critical section using semaphores.

After all the children have finished, the parent process release the shared memory and semaphores and then terminate. Use the **"wait"** function so that the parent knows precisely when each of the children finishes. The parent should print the process ID of each child as the child finishes execution. Then it should release shared memory and print "End of program".

output

From child 1: total = 100000.
From child 2: total = 300000.
From child 3: total = 600000.

Child1 ID
Child2 ID
Child3 ID

End of program

How It works:

First creating shared memory for each process for the value that you need to be shared in the different processes.

```
typedef struct

{

  int value;

}shared_mem;
```

```
shared_mem *total;


/* Create and connect to a shared memory segmentt*/
  if ((shmid = shmget (SHMKEY, sizeof(int), IPC_CREAT | 0666)) < 0)
    {
      perror ("shmget");
      exit (1);
    }
```

Then you need to also create semaphore and place it in shared memory. I Did not use the provided code to do this. The professor said I could do it any way I would like as long as it does the proper things for the project.

```
/* place semaphore in shared memory */
  sem_t *sem = mmap(NULL, sizeof(sem),
      PROT_READ |PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,
      -1, 0);
// initialize semaphore to 1 and checks for errors
    if ( sem_init(sem, 1, 1) < 0) {
        perror("sem_init");
        exit(EXIT_FAILURE);
  }
```

After Semaphore is also in shared memory you can now use the sem_wait() and sem_post() with the semaphore to restrict processes from entering critical section.

```
       sem_wait(sem);
        process1();
        sem_post(sem);
```

placing sem_wait(sem); infront of the process method call for each of the processes will decrement the value of sem by one and if sem value goes below 0 then that processes will be blocked preventing that process from entering critical section and allowing the process currently in critical section to exit first. This will be done by sem_post(sem);. When the process that was currently in critical section has left the critical section then it will increment sem value up by one. This will allow another process to be able to enter critical section. Providing mutual exclusion for the critical section.