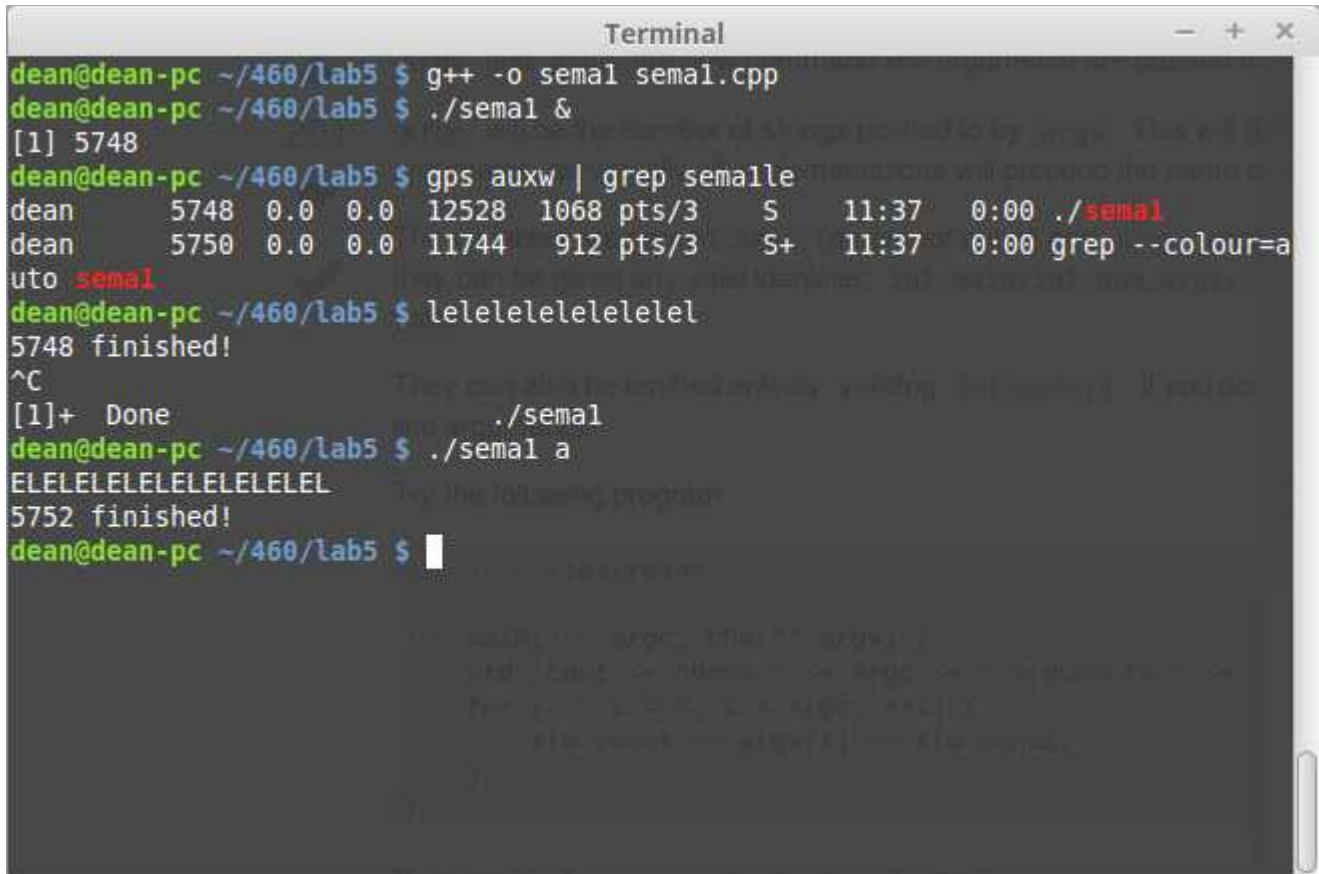


Dean Cosanella  
Flavio dos Santos-Ross  
CSE 460  
Dr. Tong Lai Yu  
Lab 5 Report

## Lab 5 Report

### 2. Using Semaphores

Screen shot of sema1.cpp compiling and running in the background and running with one parameter.



```
dean@dean-pc ~/460/lab5 $ g++ -o sema1 sema1.cpp
dean@dean-pc ~/460/lab5 $ ./sema1 &
[1] 5748
dean@dean-pc ~/460/lab5 $ gps auxw | grep sema1
dean      5748  0.0  0.0 12528 1068 pts/3    S   11:37   0:00 ./sema1
dean      5750  0.0  0.0 11744   912 pts/3    S+  11:37   0:00 grep --colour=a
uto sema1
dean@dean-pc ~/460/lab5 $ lelelelelelelelel
5748 finished!
^C
[1]+  Done                  ./sema1
dean@dean-pc ~/460/lab5 $ ./sema1 a
ELELELELELELELELELEL
5752 finished!
dean@dean-pc ~/460/lab5 $
```

We see the printout of letters 'e' and 'l' because we executed the program in the background without an argument. The same output would occur if we were to run this program with the following: ./sema1. The 'e' and 'l' represent the entering and leaving of the critical sections respectively.

When this program is executed with one argument, we can see the output of 'E' and 'L'. We see this output because an argument was passed into the program as a parameter. The letters 'E' and 'e' are always followed by the letters 'L' and 'l' respectively because 'E' and 'e' represent entering the critical section and 'L' and 'l' represent leaving the critical section.

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

Modification of program so that the program outputs nothing, waiting forever when it is invoked with 0 parameter (i.e. \$ ./sema1) but it outputs 'E' and 'L' as before when invoked with one or more parameters (i.e. \$ ./sema1 a ).

The modification of this program is towards the bottom inside the for loop.

```
//sema1.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <iostream>
#include <stdio.h>

using namespace std;

static int sem_id;          //semaphore id

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
    /* union semun is defined by including <sys/sem.h> */
#else
    /* according to X/OPEN we have to define it ourselves */
    union semun {
        int val;          /* value for SETVAL */
        struct semid_ds *buf; /* buffer for IPC_STAT, IPC_SET */
        unsigned short *array; /* array for GETALL, SETALL */
        /* Linux specific part: */
        struct seminfo *__buf; /* buffer for IPC_INFO */
    };
#endif

//initializes semaphore using SETVAL
static int set_semvalue ( int val )
{
    union semun sem_union; // sem_union;

    sem_union.val = val;
    if ( semctl ( sem_id, 0, SETVAL, sem_union ) == -1 ) return ( 0 );
    return 1;
}

//delete semaphore
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

```
static int del_semvalue ()
{
    union semun sem_union;// sem_union;

    sem_union.val = 1;
    if ( semctl ( sem_id, 0, IPC_RMID, sem_union ) == -1 ) return ( 0 );
    return 1;
}
```

```
static int SEM_DOWN ()
{
    struct sembuf b;

    b.sem_num = 0;
    b.sem_op = -1;      //P(), i.e. down()
    b.sem_flg = SEM_UNDO;
    if ( semop ( sem_id, &b, 1 ) == -1 ) {
        cout << "Semaphore DOWN() failed!" << endl;
        return 0;
    }

    return 1;
}
```

```
static int SEM_UP()
{
    struct sembuf b;

    b.sem_num = 0;
    b.sem_op = 1;      //V(), i.e. UP()
    b.sem_flg = SEM_UNDO;
    if ( semop ( sem_id, &b, 1 ) == -1 ) {
        cout << "Semaphore UP() failed!" << endl;
        return 0;
    }
    return 1;
}
```

```
int main ( int argc, char *argv[] )
{
    int i, pause_time;
    char ce = 'e', cl = 'l';
    //char ce = ' ', cl = ' ';
```

```
    srand ( ( unsigned int ) getpid() ); //seed RNG with process id
```

Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

```
sem_id = semget ( (key_t) 1234, 1, 0666 | IPC_CREAT );

if ( argc > 0 ) {
    if ( !set_semvalue( 1 ) ) {          //process can enter CS
        cout << "Semaphore initialized failed!" << endl;
        exit ( EXIT_FAILURE );
    }
    if ( argc > 1 ) {
        ce = 'E';
        cl = 'L';
    }
    ///////////////////////////////////
    if ( argc < 1 ) {
        cout << "true" << endl;
    }
    sleep ( 1 );
} else {
    if ( !set_semvalue( 0 ) ) {          //process will be blocked initially
        cout << "Semaphore initialized failed!" << endl;
        exit ( EXIT_FAILURE );
    }
    sleep ( 1 );
}

//enter and leave critical section 10 times
for ( i = 0; i < 10; i++ ){
    if ( !SEM_DOWN () ) exit ( EXIT_FAILURE );
    ///////////////////////////////////
    if ( ce == 'e' && cl == 'l' ) {      //
        while (1) {                      // Modification of
        }                                // program
    }                                     //
    ///////////////////////////////////
    cout << ce; fflush ( stdout );        //entering critical section
    pause_time = rand() % 3;             //simulate critical section
    sleep ( pause_time );
    cout << cl; fflush ( stdout );        //leaving critical section
    if ( !SEM_UP() ) exit ( EXIT_FAILURE ); //signal other waiting process
    pause_time = rand() % 2;
    sleep ( pause_time );
}
cout << endl << getpid() << " finished!" << endl;
if ( argc > 0 ) {
    sleep ( 2 );
    del_semvalue ();
}
```

```

}
exit ( EXIT_SUCCESS );
}

```

```

Terminal
dean@dean-pc ~/460/lab5 $ g++ -o sema1 sema1.cpp
dean@dean-pc ~/460/lab5 $ ./sema1
^C
dean@dean-pc ~/460/lab5 $ ./sema1 a
ELELELELELELELELELEL
8043 finished!
dean@dean-pc ~/460/lab5 $ 

```

When we executed this program with zero parameters (./sema1), We had to force close the program using ^C because it was stuck in an infinite while loop and was never going to get out of it. When we executed with one parameter (./sema1 a), the program ran as it normally would and finished without any problems.

### 3. Shared Memory

```
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid, const void *shmaddr, int shmflg );
int shmdt(const void *shmaddr);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Use "man" to study each of the shared memory functions; **write a brief description** on the usage of each of them.

#### **shmget:**

shmget() returns the identifier of the System V shared memory segment associated with the value of the argument key. A new shared memory segment, with size equal to the value of size rounded up to a multiple of PAGE\_SIZE, is created if key has the value IPC\_PRIVATE or key isn't IPC\_PRIVATE, no shared memory segment corresponding to key exists, and IPC\_CREAT is specified in shmflg.

#### **shmat:**

shmat() attaches the System V shared memory segment identified by shmid to the address space of the calling process. The attaching address is specified by shmaddr with one of the following criteria:

If shmaddr is NULL, the system chooses a suitable (unused) address at which to attach the segment. If not Null the attach occurs at the address equal to shmaddr rounded down to the nearest multiple of SHMLBA. Otherwise shmaddr must be a page-aligned address at which the attach occurs.

If SHM\_RDONLY is specified in shmflg, the segment is attached for reading and the process must have read permission for the segment. Otherwise the segment is attached for read and write and the process must have read and write permission for the segment.

#### **shmdt:**

shmdt(): Detaches the shared memory segment located at the address specified by shmaddr from the address space of the calling process. The to-be-detached segment must be currently attached with shmaddr equal to the value returned by the attaching shmat() call.

On a successful shmdt() call the system updates the members of the shmid\_ds structure associated with the shared memory segment as follows: shm\_dtime is set to the current time, shm\_lpid is set to the process-ID of the calling process and shm\_nattch is decremented by one. If it becomes 0 and the segment is marked for deletion, the segment is deleted.

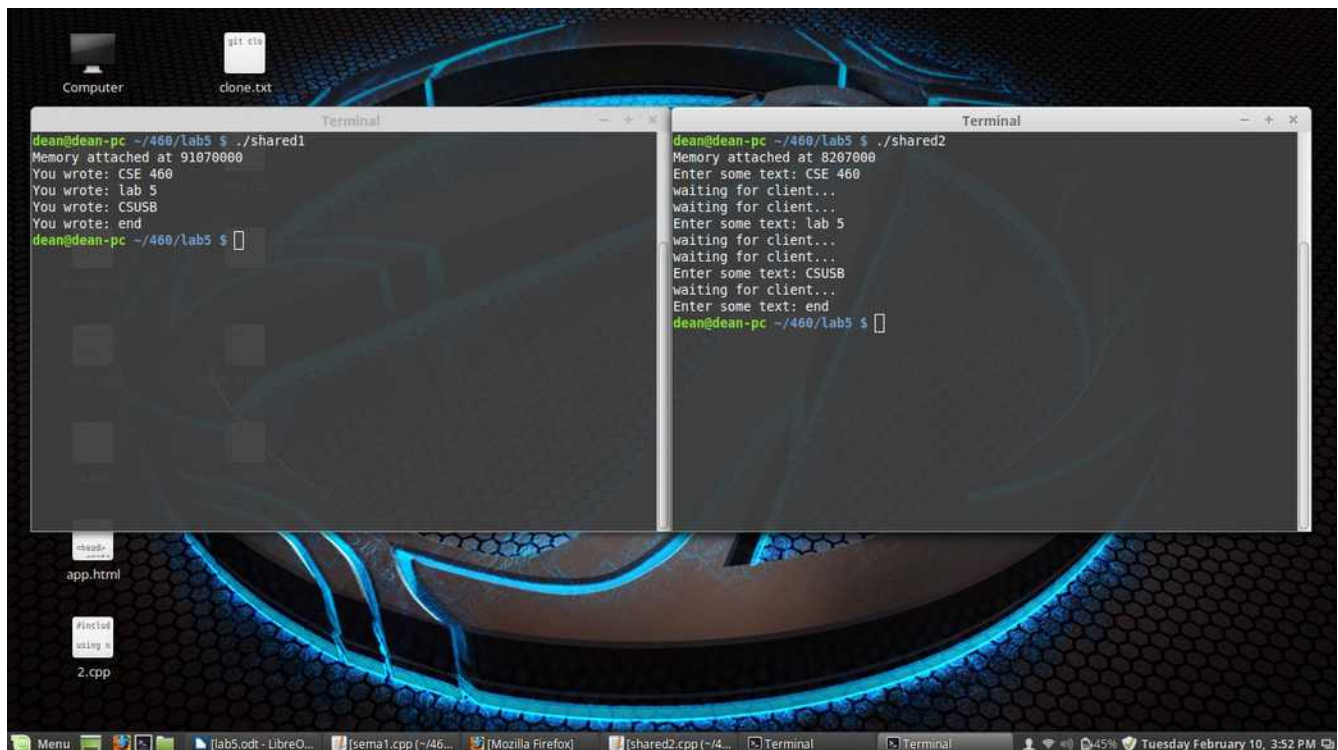
Dean Cosanella  
Flavio dos Santos – Ross  
CSE 460

### shmctl:

System V shared memory control

shmctl() performs the control operation specified by cmd on the System V shared memory segment whose identifier is given in shmid.

Screen shot of shared1.cpp and shared2.cpp running simultaneously in different terminal windows.



The screenshot shows a desktop environment with two terminal windows. The left terminal window, titled 'Terminal', shows the execution of `shared1`. It displays the memory address `91070000` and the user's input: `CSE 460`, `lab 5`, `CSUSB`, and `end`. The right terminal window, also titled 'Terminal', shows the execution of `shared2`. It displays the memory address `8207000` and the user's input: `CSE 460`, `lab 5`, `CSUSB`, and `end`. The desktop background is a dark blue pattern with a large, glowing blue 'S' shape. The taskbar at the bottom shows various icons, including a menu, a file manager, and several open applications like `lab5.odt`, `LibreOffice`, `sema1.cpp`, `shared2.cpp`, and `Firefox`. The system clock in the bottom right corner indicates it is Tuesday, February 10, 3:52 PM.

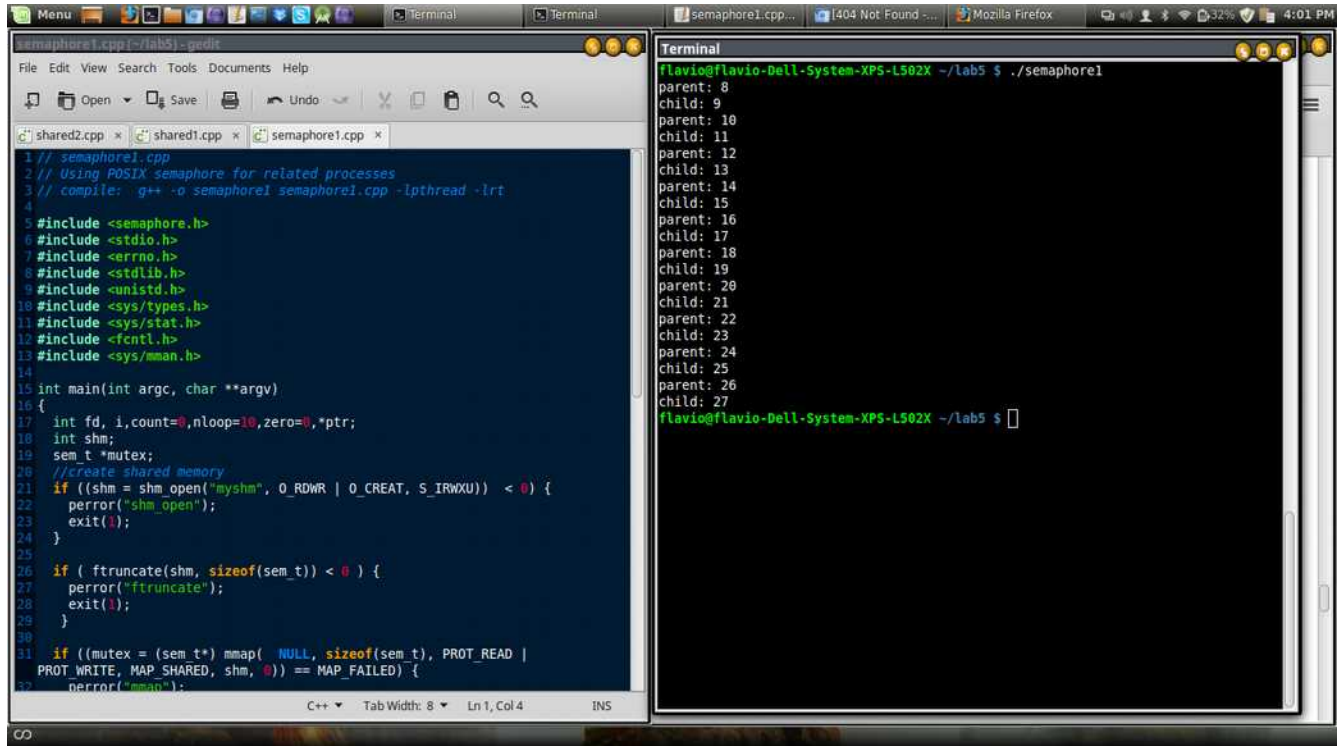
With this program we see that when we run `shared2.cpp`, it asks the user to enter some text. The user enters text and presses enter and we see that the text appears when `shared1.cpp` is running. Once the text outputs from `shared1.cpp`, `shared2.cpp` asks for more text, this process continues until the user types the text “end” into `shared2.cpp`.

We were not able to figure out how to modify this code to protect the shared memory so that when a process accesses the shared area, the other is excluded from doing so. We did not understand exactly what this problem was asking and what the output should look like when it runs correctly.



## 4. POSIX Semaphores

Screen shot of semaphore1.cpp program running.



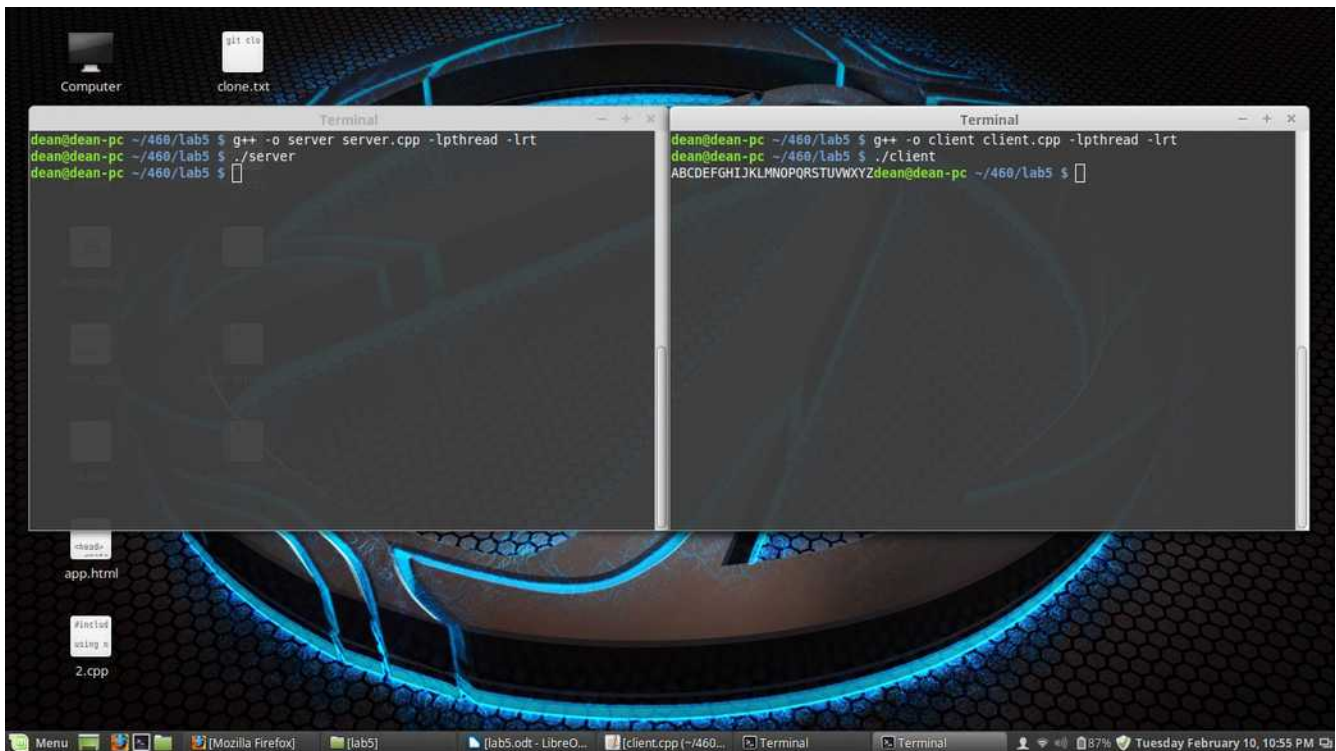
```
semaphore1.cpp (~/.lab5) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Cut Copy Paste Find
shared2.cpp x shared1.cpp x semaphore1.cpp x
1 // semaphore1.cpp
2 // Using POSIX semaphore for related processes
3 // compile: g++ -o semaphore1 semaphore1.cpp -lpthread -lrt
4
5 #include <semaphore.h>
6 #include <stdio.h>
7 #include <errno.h>
8 #include <stdlib.h>
9 #include <unistd.h>
10 #include <sys/types.h>
11 #include <sys/stat.h>
12 #include <fcntl.h>
13 #include <sys/mman.h>
14
15 int main(int argc, char **argv)
16 {
17     int fd, i, count=0, nloop=10, zero=0, *ptr;
18     int shm;
19     sem_t *mutex;
20     //create shared memory
21     if ((shm = shm_open("myshm", O_RDWR | O_CREAT, S_IRWXU)) < 0) {
22         perror("shm open");
23         exit(1);
24     }
25
26     if (ftruncate(shm, sizeof(sem_t)) < 0) {
27         perror("ftruncate");
28         exit(1);
29     }
30
31     if ((mutex = (sem_t*) mmap( NULL, sizeof(sem_t), PROT_READ |
32         PROT_WRITE, MAP_SHARED, shm, 0)) == MAP_FAILED) {
33         perror("mmap");
34     }
35 }
C++ Tab Width: 8 Ln 1, Col 4 INS

Terminal
flavio@flavio-Dell-System-XPS-L502X ~/Lab5 $ ./semaphore1
parent: 8
child: 9
parent: 10
child: 11
parent: 12
child: 13
parent: 14
child: 15
parent: 16
child: 17
parent: 18
child: 19
parent: 20
child: 21
parent: 22
child: 23
parent: 24
child: 25
parent: 26
child: 27
flavio@flavio-Dell-System-XPS-L502X ~/Lab5 $
```

The program compiles properly and presents the process of relationship between parent and child in ascending order. The parent process is always one number less than the child process because the child process is created from the parent process. The smaller number indicates that process was created first.

## Unrelated Process

The compiling and execution of server.cpp and client.cpp.



```
dean@dean-pc ~/460/lab5 $ g++ -o server server.cpp -lpthread -lrt
dean@dean-pc ~/460/lab5 $ ./server
dean@dean-pc ~/460/lab5 $

dean@dean-pc ~/460/lab5 $ g++ -o client client.cpp -lpthread -lrt
dean@dean-pc ~/460/lab5 $ ./client
ABCDEFGHIJKLMNOPQRSTUVWXYZdean@dean-pc ~/460/lab5 $
```

When we run this code, we can see that client.cpp outputs the alphabet from 'A' to 'Z'. The reason why this happens is because the server.cpp and client.cpp programs are both accessing the same memory and the client is the one that is reading the shared memory data so it is the one that outputs the data that is stored there.

The server.cpp program has to be executed first because it is the program that creates the semaphore that is shared between the two programs. If we try to run the client.cpp program first we get the following output: “reader:unable to execute semaphore: No such file or directory”. This is saying that a semaphore has not been created therefore it cannot read anything.

We were not able to complete the modification of these two programs so that the server sits in a loop to accept string inputs from users and send them to the client, which then prints out the string. We did not know where to start for this problem and we ran out of time.

Since we were not able to complete some of the modifications, we give ourselves a score of 17/20 on this lab. We did learn a lot about semaphores in the process of doing this lab even though we were not able to finish everything completely.