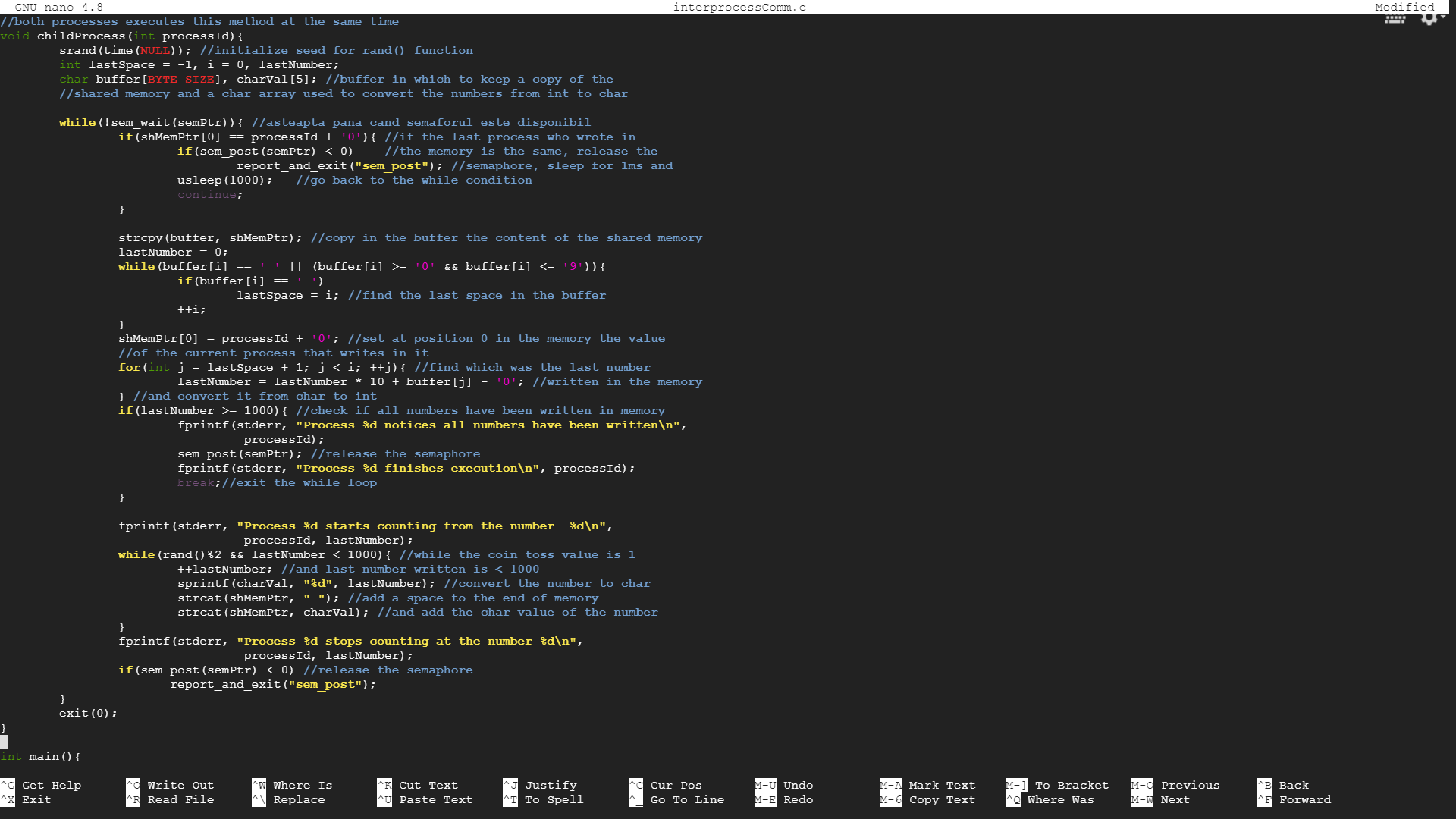
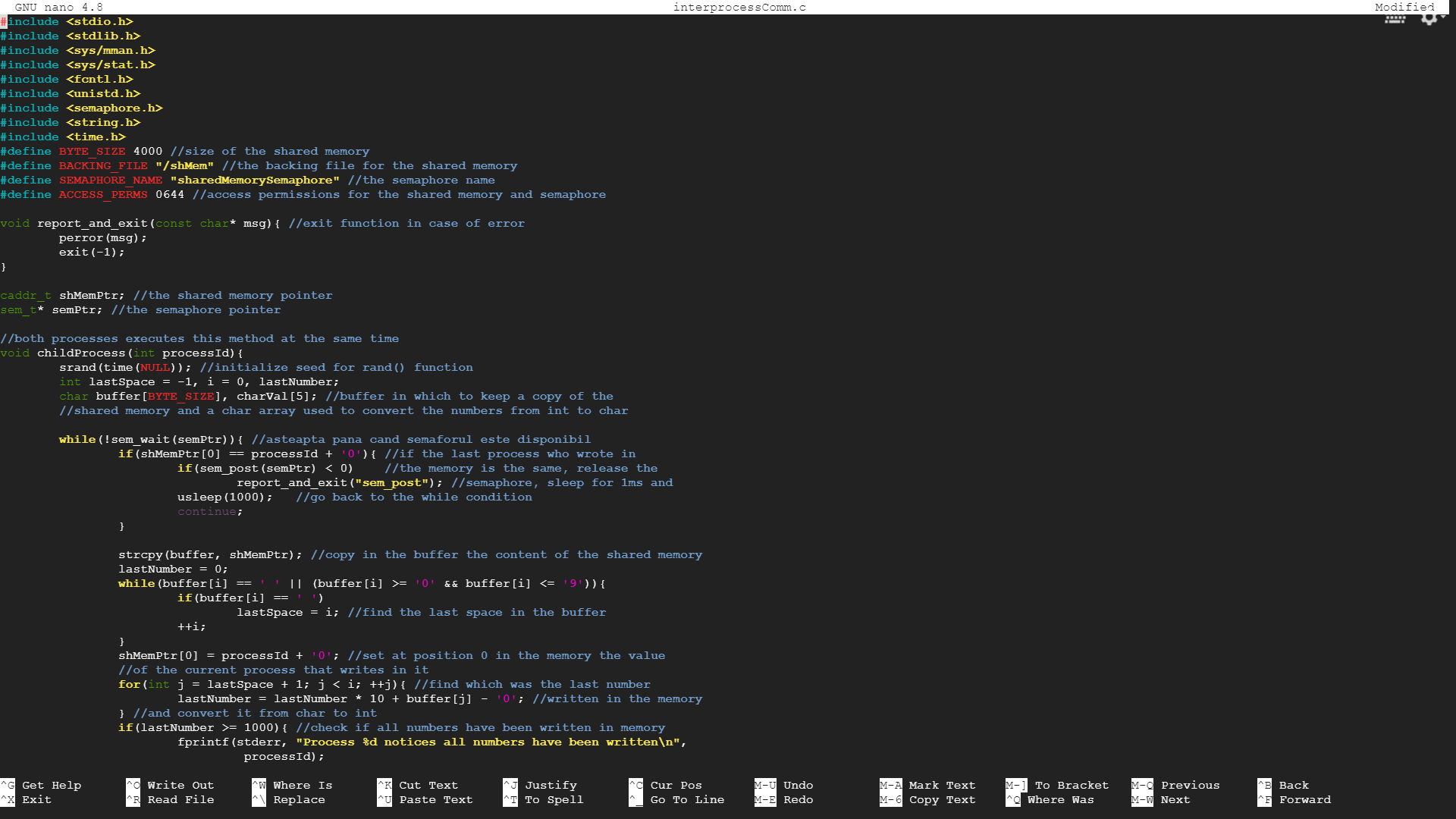
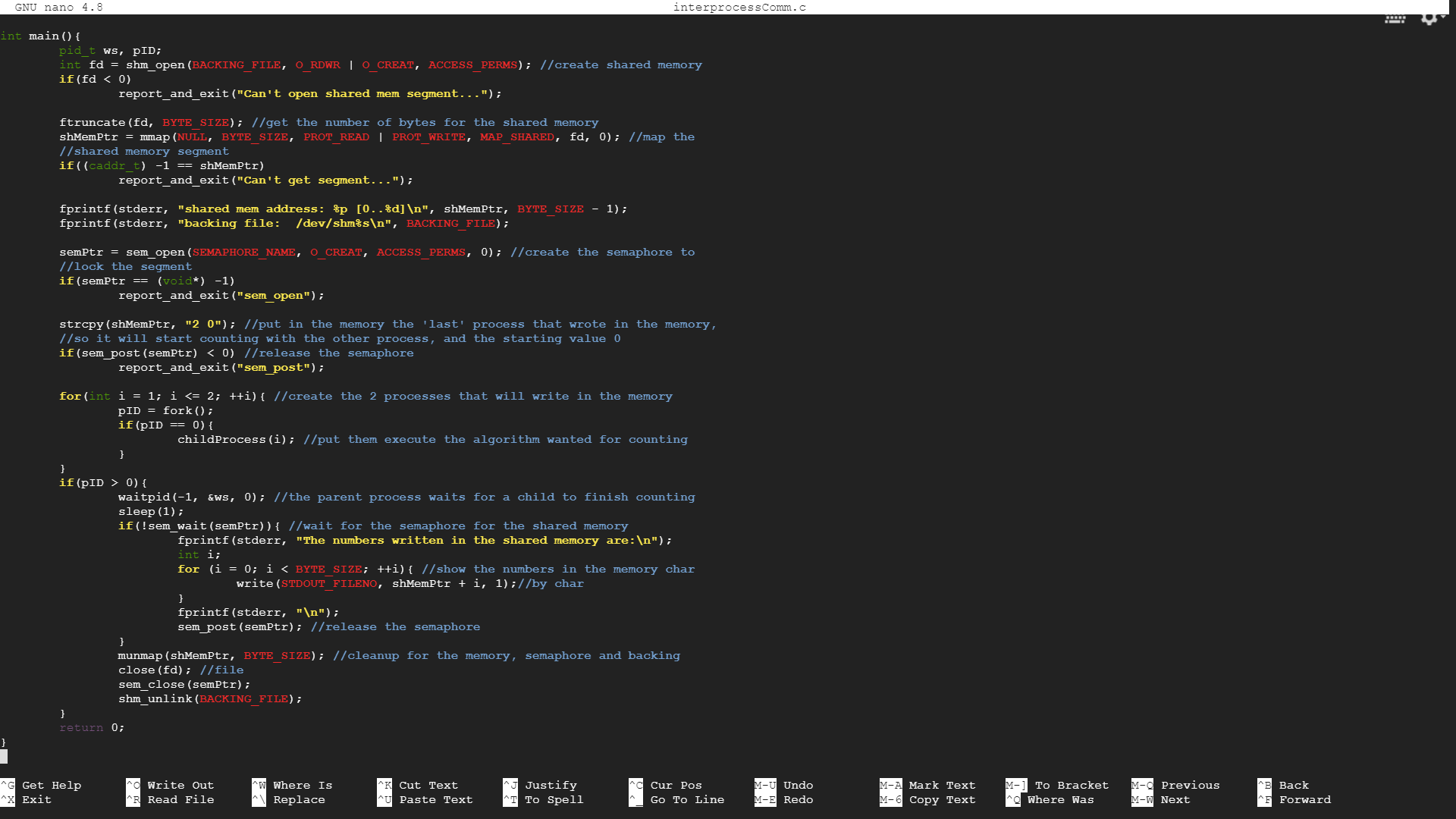
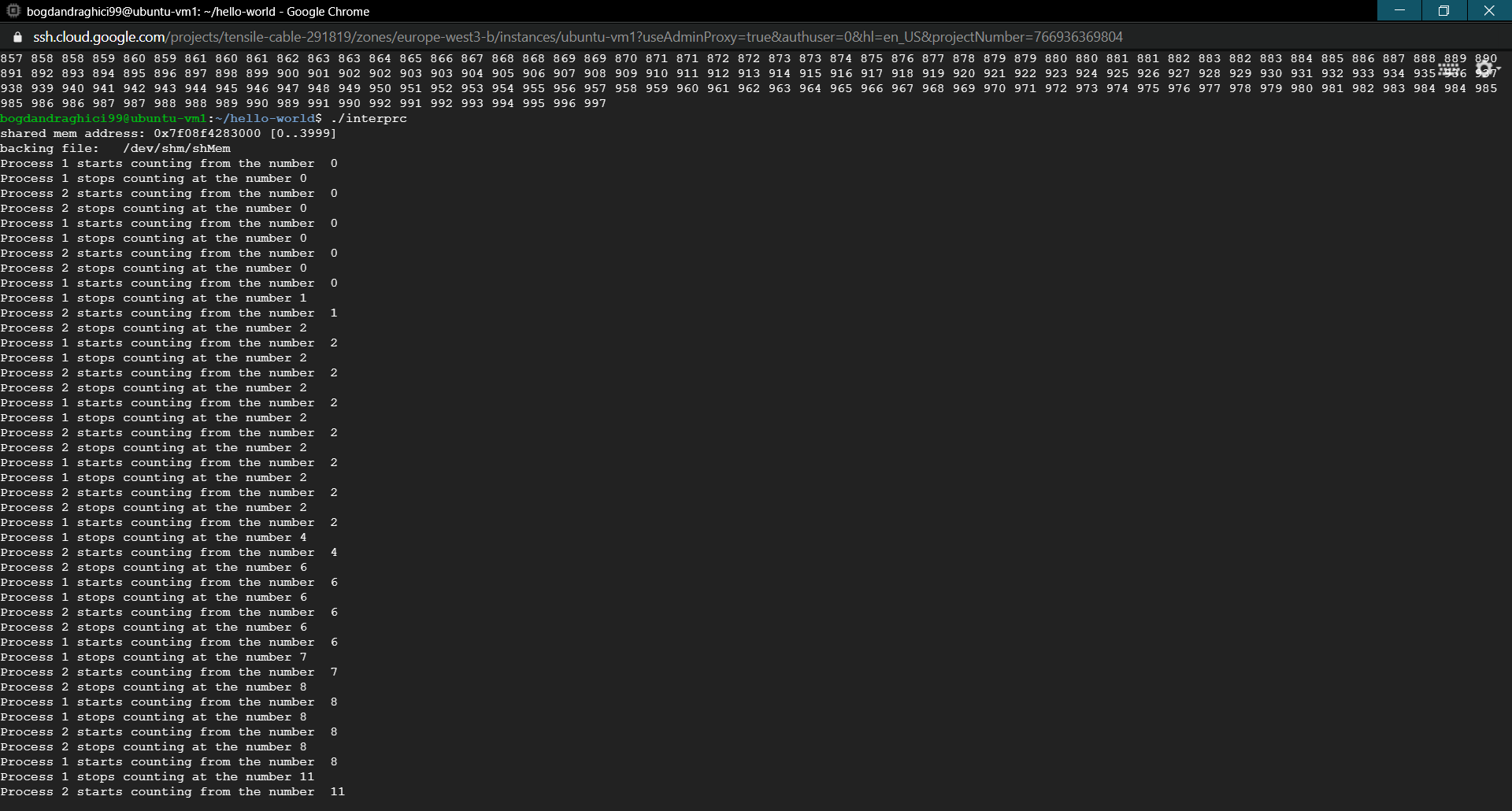
**Inter-process communication**

I created the following C program for Linux:

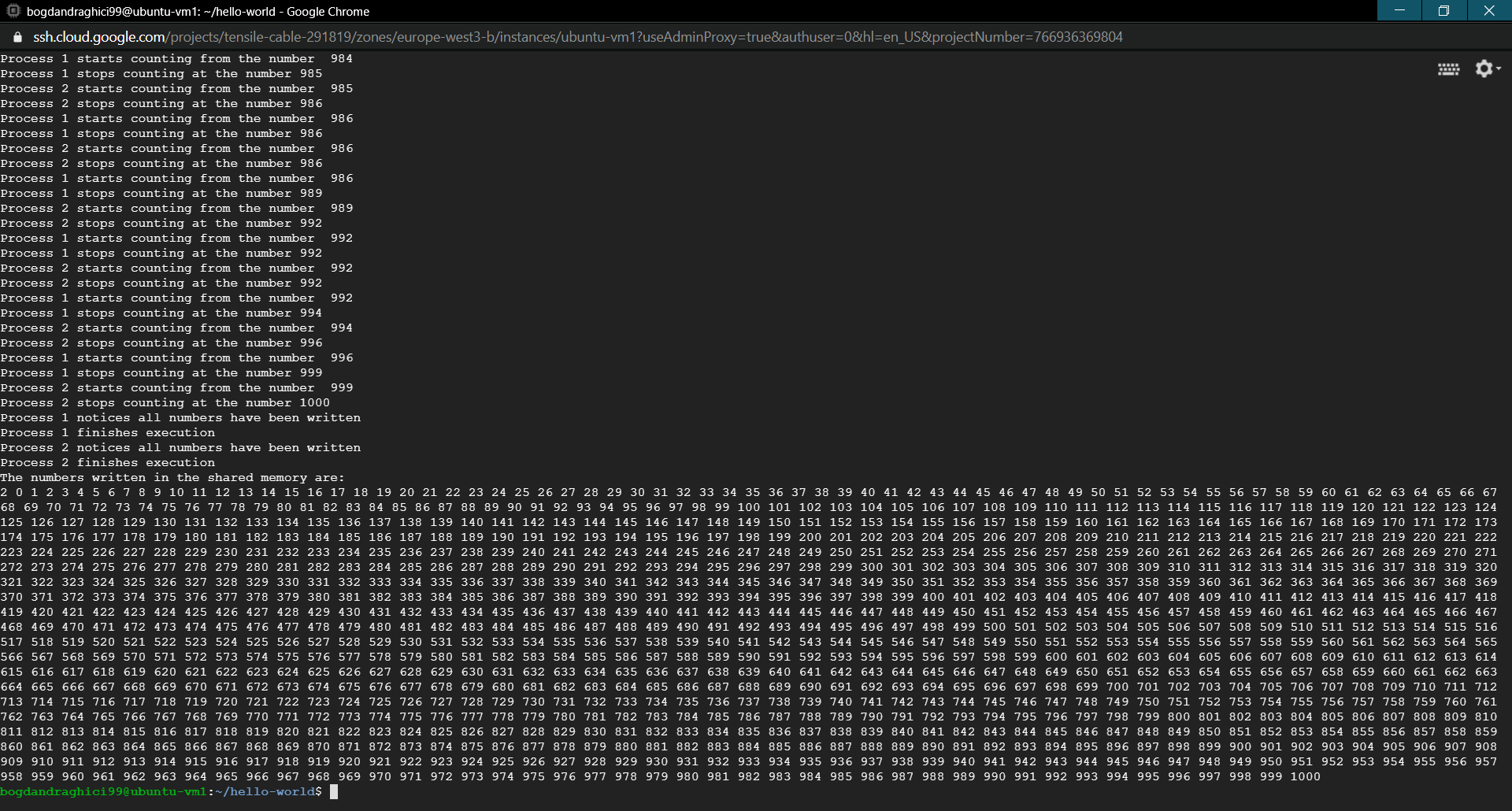




After compiling the program using the command gcc -o interprc interprocessComm.c -lrt -lpthreadD, and executed the file interprc, I obtaine the following result:



……………………………………………….



Moved the interprocessComm.c file in the git repository, added it to stage, commit with the message “added inter-process communication program”, then tried to push it in the remote, but the connection failed.

The code in text is:

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <semaphore.h>

#include <string.h>

#include <time.h>

#define BYTE\_SIZE 4000 //size of the shared memory

#define BACKING\_FILE "/shMem" //the backing file for the shared memory

#define SEMAPHORE\_NAME "sharedMemorySemaphore" //the semaphore name

#define ACCESS\_PERMS 0644 //access permissions for the shared memory and semaphore

void report\_and\_exit(const char\* msg){ //exit function in case of error

perror(msg);

exit(-1);

}

caddr\_t shMemPtr; //the shared memory pointer

sem\_t\* semPtr; //the semaphore pointer

//both processes executes this method at the same time

void childProcess(int processId){

srand(time(NULL)); //initialize seed for rand() function

int lastSpace = -1, i = 0, lastNumber;

char buffer[BYTE\_SIZE], charVal[5]; //buffer in which to keep a copy of the

//shared memory and a char array used to convert the numbers from int to char

while(!sem\_wait(semPtr)){ //asteapta pana cand semaforul este disponibil

if(shMemPtr[0] == processId + '0'){ //if the last process who wrote in

if(sem\_post(semPtr) < 0) //the memory is the same, release the

report\_and\_exit("sem\_post"); //semaphore, sleep for 1ms and

usleep(1000); //go back to the while condition

continue;

}

strcpy(buffer, shMemPtr); //copy in the buffer the content of the shared memory

lastNumber = 0;

while(buffer[i] == ' ' || (buffer[i] >= '0' && buffer[i] <= '9')){

if(buffer[i] == ' ')

lastSpace = i; //find the last space in the buffer

++i;

}

shMemPtr[0] = processId + '0'; //set at position 0 in the memory the value

//of the current process that writes in it

for(int j = lastSpace + 1; j < i; ++j){ //find which was the last number

lastNumber = lastNumber \* 10 + buffer[j] - '0'; //written in the memory

} //and convert it from char to int

if(lastNumber >= 1000){ //check if all numbers have been written in memory

fprintf(stderr, "Process %d notices all numbers have been written\n",

processId);

sem\_post(semPtr); //release the semaphore

fprintf(stderr, "Process %d finishes execution\n", processId);

break;//exit the while loop

}

fprintf(stderr, "Process %d starts counting from the number %d\n",

processId, lastNumber);

while(rand()%2 && lastNumber < 1000){ //while the coin toss value is 1

++lastNumber; //and last number written is < 1000

sprintf(charVal, "%d", lastNumber); //convert the number to char

strcat(shMemPtr, " "); //add a space to the end of memory

strcat(shMemPtr, charVal); //and add the char value of the number

}

fprintf(stderr, "Process %d stops counting at the number %d\n",

processId, lastNumber);

if(sem\_post(semPtr) < 0) //release the semaphore

report\_and\_exit("sem\_post");

}

exit(0);

}

int main(){

pid\_t ws, pID;

int fd = shm\_open(BACKING\_FILE, O\_RDWR | O\_CREAT, ACCESS\_PERMS); //create shared memory

if(fd < 0)

report\_and\_exit("Can't open shared mem segment...");

ftruncate(fd, BYTE\_SIZE); //get the number of bytes for the shared memory

shMemPtr = mmap(NULL, BYTE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0); //map the

//shared memory segment

if((caddr\_t) -1 == shMemPtr)

report\_and\_exit("Can't get segment...");

fprintf(stderr, "shared mem address: %p [0..%d]\n", shMemPtr, BYTE\_SIZE - 1);

fprintf(stderr, "backing file: /dev/shm%s\n", BACKING\_FILE);

semPtr = sem\_open(SEMAPHORE\_NAME, O\_CREAT, ACCESS\_PERMS, 0); //create the semaphore to

//lock the segment

if(semPtr == (void\*) -1)

report\_and\_exit("sem\_open");

strcpy(shMemPtr, "2 0"); //put in the memory the 'last' process that wrote in the memory,

//so it will start counting with the other process, and the starting value 0

if(sem\_post(semPtr) < 0) //release the semaphore

report\_and\_exit("sem\_post");

for(int i = 1; i <= 2; ++i){ //create the 2 processes that will write in the memory

pID = fork();

if(pID == 0){

childProcess(i); //put them execute the algorithm wanted for counting

}

}

if(pID > 0){

waitpid(-1, &ws, 0); //the parent process waits for a child to finish counting

sleep(1);

if(!sem\_wait(semPtr)){ //wait for the semaphore for the shared memory

fprintf(stderr, "The numbers written in the shared memory are:\n");

int i;

for (i = 0; i < BYTE\_SIZE; ++i){ //show the numbers in the memory char

write(STDOUT\_FILENO, shMemPtr + i, 1);//by char

}

fprintf(stderr, "\n");

sem\_post(semPtr); //release the semaphore

}

munmap(shMemPtr, BYTE\_SIZE); //cleanup for the memory, semaphore and backing

close(fd); //file

sem\_close(semPtr);

shm\_unlink(BACKING\_FILE);

}

return 0;

}