*Abstract*—**Group Project: A shared Protected Circular Queue and Communication between threads. The purpose is to learn how to use semaphores to protect a limited size resource.** **The output of the program produces will be reconstruction of the original thread contained in “mytest.dat”. It need not include the “\*” character. To compile program use the command: “gcc name\_of\_program.c -lpthread –lrt”.**

Final Programming Project

Lukyan M. Hritsko, Sean T. Dunford, and Mahmoud Ibrahim, Students, University of South Florida

*Index Terms- Unix, processes, shared memory, fork*

# INTRODUCTION

A circular buffer with 15 positions (each position stores 1 character) is to be used to communicate information between two threads (producer and consumer). The producer thread will read characters, one by one from a file and place it in the buffer and continue to do that until the “end-of-file” (EOF) marker is reached. The name of the file must be “mytest.dat” when you are submitting the program – of course you can use your own file while individually testing your program. There should be no more than 150 characters in the file. The producer must inform the consumer when it has finished placing the last character in the buffer. The producer could do this by placing a special character for example, ”\*” in the shared buffer or by using a shared memory flag that the producer sets to true and the consumer reads at the appropriate time

Consumer thread will read the characters, one by one, from the shared buffer and print it to the screen. A parent process will create both producer and consumer threads and will wait until both are finished to destroy semaphores. The consumer should run slower than producer. So, place a one second sleep in the consumer thread between “reads” from the shared memory.

# Procedure

Below is a piece of “C” code that give you some idea of how to open file and read from file

char newChar;

FILE\* fp;

fp= fopen("mytest.dat", "r");

while(fscanf(fp,"%c",&newChar) != EOF)

………………………………………………..

close(fp);

**The semaphore functions:**

sem\_t sem1;

sem\_wait(&sem1);

sem\_post(&sem1);

sem\_init(&sem1, …,…);

sem\_destroy(&sem1);

# Code

/\*

\*

\* Final Programming Assignment

\* Operating Systems

\*

\*

\* Purpose: To demonstrate the use of threads and semaphores

\*

\* Authors: Lukyan Hritsko, Sean Dunford, Mouhmad(sp?) Ibrahim(sp?)

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <sys/types.h>

#include <unistd.h>

char array[15] = {'\00'}; //This buffer will be treated as a circular buffer

sem\_t sem;

char output[151] = {'\00'}; //150 + 1(null terminator)

int overallcount = 1;

void \* producer() {

char c;

FILE \* fp;

int count = 0;

fp = fopen("mytest.dat", "r");

while(1) {

if(sem\_wait(&sem) == 0) { //if statement makes sure no errors occur

//with semaphore

if(array[count] == '\00') {

if(fscanf(fp, "%c", &c) == EOF || overallcount == 150) {

sem\_post(&sem);

count = (count + 1) % 15;

break;

}

array[count] = c; //Critical sec

overallcount++;

//below assures count will never go above 14

count = (count + 1) % 15;

}

sem\_post(&sem);

}

else {

perror("semaphore");

exit(1);

}

}

if(sem\_wait(&sem) == 0) {

while(1) {

if(array[count] == '\00') {

array[count] = '\*';

break;

}

count = (count + 1) % 15;

}

sem\_post(&sem);

}

else {

perror("semaphore");

exit(1);

}

}

void \* consumer() {

int i;

int count = 0;

char temp[2] = {'\00'};

while(1) {

sleep(1);

if(sem\_wait(&sem) == 0) {

temp[0] = array[count];

if(temp[0] != '\*') {

strncat(output, temp, 1);

array[count] = '\00';

count = (count + 1) % 15;

}

else {

while(1) {

temp[0] = array[count];

if(temp[0] != '\00' && temp[0] != '\*') {

strncat(output, temp, 1);

}

else {

break;

}

count = (count + 1) % 15;

}

break;

}

}

else {

perror("semaphore");

exit(1);

}

sem\_post(&sem);

}

}

int main(int argc, char \*\* argv) {

pthread\_t thread1, thread2;

pthread\_attr\_t attr[1];

printf("1 - I am here in pid %d\n", getpid());

fflush(stdout);

sem\_init(&sem, 1, 2); //Initializes the semaphore

pthread\_attr\_init(&attr[0]);

pthread\_attr\_setscope(&attr[0], PTHREAD\_SCOPE\_SYSTEM);

pthread\_create(&thread1, &attr[0], &producer, NULL);

pthread\_create(&thread2, &attr[0], &consumer, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

sem\_destroy(&sem); //This destroys the semaphore used

printf("Amount of characters read from file: %d\n", overallcount);

fflush(stdout);

printf("%s\n", output);

return 0;

}

# Output

1 - I am here in pid 2487

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2592

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2595

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2598

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2601

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2604

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2607

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2611

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2614

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2633

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2636

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2640

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2643

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2653

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2657

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2661

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2671

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2674

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2678

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2696

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2714

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2718

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2721

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2731

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2742

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2766

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2778

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2781

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2822

Amount of characters read from file: 24

This is the test file.

1 - I am here in pid 2826

Amount of characters read from file: 24

# Analysis of Solution and Data

The code above returns consistent and correct data under the requirements that were given. The use of shared memory is carefully watched. The threads that are spun up are carefully written so they do not execute elsewhere in the program. Strict adherence to detaching shared mem from the process before it was killed was upheld. This example shows us how necessary it is to be careful when dealing with shared memory, threads, semaphores, and processes. Avoiding caution can result in garbage data, infinite loops, floating blocks of shared memory, and other problems.

.

1. Acknowledgment

We would like to acknowledge our Professor, the TA, and the writers of the Unix/Linux Man pages. Without them none of this was possible.

References

1. Operating Systems Comcepts 8th edition Galvin, Gagne, Silberschatz
2. Prof. Korzhova
3. Samuel Hawkins TA
4. B. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.
5. <http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.perf.doc/perf106.htm>
6. <http://linux.die.net/man/2/shmctl>
7. <http://uw714doc.sco.com/en/SDK_sysprog/_Getting_Shared_Memory_Segments.html>
8. <http://pubs.opengroup.org/onlinepubs/009695399/functions/shmget.html>
9. <https://computing.llnl.gov/tutorials/pthreads/>
10. <http://www.cs.utsa.edu/~whaley/teach/cs4823/LEC/pthreads_ho.pdf>
11. <https://www.cs.drexel.edu/~jjohnson/2010-11/winter/cs676/lectures/lec4.html>
12. <http://www.csc.villanova.edu/~mdamian/threads/posixsem.html>
13. <http://stackoverflow.com/questions/2065747/pthreads-mutex-vs-semaphore>
14. <http://linux.about.com/library/cmd/blcmdl2_shmdt.htm>
15. [http://www.kernel.org/doc/man-pages/online/pages/man2/semop.2.html#ERRORS](#ERRORS)
16. <http://pubs.opengroup.org/onlinepubs/009696699/functions/semop.html>
17. <http://linux.about.com/library/cmd/blcmdl3_perror.htm>
18. [http://www.inf.ufes.br/~zegonc/material/Sistemas%20Operacionais/Semaphores.pdf](http://www.inf.ufes.br/~zegonc/material/Sistemas Operacionais/Semaphores.pdf)
19. <http://bytes.com/topic/c/answers/779925-system-function-not-executed-correctly>
20. <http://pubs.opengroup.org/onlinepubs/009696699/functions/semop.html>
21. <http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/semaphore.htm>
22. <http://www.cis.upenn.edu/~lee/07cis505/Lec/SemaphoreOperations.pdf>