CS474 Operating System

Project #2 (A shared Protected Circular Queue and Communication between threads))

October 19, 2020

Project Objectives:

The purpose of this project is to give students an opportunity to experiment with process synchronization mechanisms.

Total points Available: 100

Due: November 5, 2020

Project Description:

The purpose is to learn how to use semaphores to protect a limited size resource. 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.

Submitting your assignment

- Submission via Canvas's Assignment.
    - It is your responsibility to submit these assignments in a timely fashion.
- All files should be zipped together.
- There should be a readme file explaining in detail the exact steps to be taken to compile and execute the code files and the title page.
- Testing of this work should be done only on the CS lab machines. Please make sure these machines are not locked up due to your code. The execution for grading purposes will be done on the lab machines.
- In case of any code errors, partial credit may be offered based on the code and documentation.
- A report that presents the performance evaluation of your solution.
    - The report should be properly formatted (an academic format style, such as ACM or IEEE being preferred) and contain quantitative data along with your analysis of these data.

Sample Output:

The output your program produces will be reconstruction of the original thread contained in "mytest.dat". It need not include the "*" character.

Grading Criteria:

1) Minus 90% if code does not compile
2) Minus 70% if run time error
3) Minus 10% if the shared buffer is not created or initialized properly
4) Minus 20% if the semaphores are not created or initialized properly
5) Minus 20% if the creation of threads does not work properly
6) Minus 20% for choosing wrong critical section
7) Minus 20% if Producer thread does not work properly
8) Minus 20% if Consumer thread does not work properly (remember Consumer runs slower, do not miss 1 sec sleep). Also you will have to ensure that neither Producer nor Consumer adds extra or misses any characters than there are in the file
9) Minus 10% (for each) if there are unreleased semaphores or shared memory
10) Minus 10% if the report is not written
11) Minus 10% if no comments
12) Minus 5% if your name is not included as comments at the beginning of your program

Use the following header for your program

```
#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>
```

You will require 3 semaphores;
The buffer should be treated as circular buffer

Below is a piece of "C" code that gives 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);
```

To compile program use the command:

gcc name_of_program.c -lpthread -lrt

The semaphore functions:

```
sem_t  sem1;
sem_wait(&sem1);
sem_post(&sem1);
sem_init(&sem1, …,…);
sem_destroy(&sem1);
```

Useful commands for threads:

```
pthread_t       tid1[1];    /* process id for thread 1 */
pthread_t       tid2[1];    /* process id for thread 2 */
pthread_attr_t  attr[1];    /* attribute pointer array */


        fflush(stdout);
/* Required to schedule thread independently.*/
pthread_attr_init(&attr[0]);
pthread_attr_setscope(&attr[0], PTHREAD_SCOPE_SYSTEM);
        /* end to schedule thread independently */

/* Create the threads */
 pthread_create(&tid1[0], &attr[0], thread1, NULL);
 pthread_create(&tid2[0], &attr[0], thread2, NULL);

/* Wait for the threads to finish */
 pthread_join(tid1[0], NULL);
 pthread_join(tid2[0], NULL);
```

……

Terminate threads
pthread_exit(NULL);

Some useful Unix commands:

Releasing shared memory:

ipcs command gives you shared memory id of the shared memory unreleased by you, if you have any.

Type command *ipcrm* -m id to remove shared memory,

or *ipcrm* –s id to remove semaphores.

Process Control Commands:

kill - send a signal to a process

suspend - shell built-in function to halt the current shell

jobs, *fg*, *bg*, *stop*, *notify* - control process execution

halt, *poweroff* - stop the processor

Other Commands:

limit, *ulimit*, *unlimit* - set or get limitations on the sys-

ping - send ICMP ECHO_REQUEST packets to network hosts

man - find and display reference manual pages

Development Environment

You may write your program using any available editor Nano, Emacs, Vi or whatever editor you are most comfortable with, BUT, it must compile with gcc and be executable on one of the CS machines.

Hints:

Build your project in an incremental fashion. Attempt to meet each objective before moving on to the next.