Real-time Operating System - 48450

Assignment – 2

Mutex/Semaphore and Pipe for teal time file reading/writing

Student Name: Smit Patel

Student ID: 11819164

**Table of Contents**

[I. Introduction 3](#_Toc481415762)

[II. Theory of operation 3](#_Toc481415763)

[III. Operating condition 3](#_Toc481415764)

[IV. Implementation 4](#_Toc481415765)

[V. Experiments 5](#_Toc481415766)

[1. Hypothesis 5](#_Toc481415767)

[2. Results 5](#_Toc481415768)

[VI. References 5](#_Toc481415769)

1. Introduction

The purpose of this assignment is to develop understanding on Application program developments and applying skills of dealing with semaphores, pipe and threads for real time reading and writing on file.

A thread is a sequence of instructions which can be executed solely inside the process and behaves like a sub process. Generally, processes have multiple threads running simultaneously, this increases the data. When this happen, the program needs to create multiple threads for reading/writing kind of operations.

1. Theory of operation

As the assignment specification states that program will involve concepts of threads and pipes One Process Multiple threads model is the best suitable multithreading model for this problem. The user level threads are managed by the thread library and switching threads does not require kernel mode privileges and scheduling is specific to certain applications.

Semaphores on the other hand are non-negative integer variables which coordinate and synchronise multiple threaded programs in which all threads are running on same address space and have access to same variables and data. These are shared memory blocks. Semaphores have very efficient waiting mechanism where in threading a thread cannot proceed until some changes take place so it is unnecessary to loop multiple times and checking for change of state. In conditions like this semaphore are set as signals which represents the right of a thread to proceed ahead. A non-zero value means the thread can proceed and must wait when zero.

1. Operating condition

For this assignment three threads needs to be created to read data from one file and write to another through pipe-line impression. This mechanism resembles to internet file transmission. There are few points which are must to fulfil this assignment’s requirement.

* Data.txt is given as data file which will pass through pipe.
* Thread A should write one line at a time from data file to the pipe.
* Thread B should be able to read the data written to pipe by Thread A and store somewhere so Thread C can work on it.
* Thread C must analyse the data from Thread B and determine is the given line is from file header or from the content region of the data file.
* File headers are rejected and content region lines are written into net text file called src.txt

Because there is memory block which is accessed by multiple threads there needs to be waiting algorithm for resource allocation. Semaphore are the most suitable for this scenario. It will allow only one thread to read/write on shared memory block and utilizes time by ownership of the resource.

1. Implementation

This assignment will be completed under POSIX (Portable Operating System Interface) standard. POSIX is a family of standards defining the API, with software compatibility to the command line for various Unix and other operating systems. The requirement of this assignment is to create three threads A, B & C and perform reading/writing operations mentioned in previous section.

There are also some conditions for the data input and arguments like every parameter should go via structure to the threads and threads should be processed in synchronized activity with the help of semaphore.

Primarily global variables are created which will be accessed by every thread and main program. That is buffer size must be greater than the header detection character array so 1024 is ideal for that. Header files are vital for program so include all the headers whose3 functions will be used like pthread.h, semaphore.h, unistd.h etc. up next is the structures of the Threads. This is not necessary but as stated in assignment specification we must pass the parameters to the threads using struct. As every thread has different aim, semaphore and flags vary between threads.

The Thread A has file descriptor variable that stores return value of get file. That thread also looks for semaphore waiting flag and exit flag of the file. Few error handling statements gives this thread robustness.

Thread B has read variable of type nread which reads the shared memory or pipe and then the read data goes to infinite loop and exits at classified conditions, this has exit flags, read file from pipe, close the pipe and handle errors like buffer overflow and posting semaphore.

The last and most important thread is Thread C which has subroutine to detect the “end\_header” line from the shared buffer and create new text file called src.txt in which all the lines after “end\_header” detection will be written. By doing this complex operation there are few possibilities of errors in execution which will be handled by semaphore set-up in main function.

The main function is where things get started and allocated to desired functions to manipulate the data. It creates a shared pipe file descriptors array then initialize the semaphore followed by creation of all three threads. And at the end it closes the semaphores initialized before.

(Program code can be viewed in source file submitted)

1. Experiments
2. Hypothesis

The Experiment question is what will be the scenario if no mutex/semaphore is applied and only three threads handles the pipe to get the job done.

1. Results

After running the code, it came into my knowledge that threading makes the program very efficient. However, coordinating multiple threads is much more difficult that single threaded system. Coming to the experiment semaphores are used for good utilization of time and least wait time for efficient performance. But by not applying it in multithreaded model the pipe file descriptor and all three threads were assessed randomly and no anticipated outcomes took place.

1. References
2. Silberschatz, P. B. Galvin & G. Gagne, 2012, Operating System Concepts, 9 th edn, John Wiley & Sons, New York.
3. B. W. Kernighan & D. Ritchie, The C Programming Language, 2nd edn, AT&T Bell Laboratories, New Jersey.