WEEK 6: Task

Date: 15/09/2019

4	Threads: Build a Multi-Threaded Server to transfer a file from server to client using Producer-Consumer Model.

OBJECTIVE:

- **Familiarizing with basic system calls open(), read(), write() and close() to do System Programming using C language.**
- **Understand the concepts of Threads and its functionalities in comparison with processes.**
- **Understand the concepts of Semaphores used to synchronize the processes.**
- **Understand socket inter process communication and its system calls used. (Skeleton socket client server program will be shared for lab tasks)**

WEEK 6 IOS Program Execution Steps for

Multithreaded Server to transfer a file from Server to Client using Producer-Consumer Model

1. We have 2 C programs the **client.c** and **multiserver.c**.
2. We call the server as multiserver because we have 2 threads on the server side to produce data to the buffer and another to consume data from the buffer.
3. Use the same **client.c** program shared in WEEK 2 Task.
4. The **multiserver.c** program with skeleton socket program code is shared for WEEK 6.
5. Client sends request 'get' to get the contents of the file specified after ‘get’ as third argument. **./client get 'filename'.**
6. No need to use ‘list’ request. Not required.
7. Server responds to client by displaying the contents of the file by using basic system calls.

- open() - Open the file requested.
- read() - Read the contents of the file.
- write() - Write the read contents of the file to the buffer created on the server side.
- close() - Closes the file opened.

8. A bounded buffer is created with fixed length on the server side to hold the contents of the read file and also to transfer the same contents from the buffer to the client back.
9. The buffer created on the server side is controlled by 3 different mutex variables.
   1. **buf_mutex** – mutex variable to say buffer is FREE or NOT FREE to produce/consume data.
      Value 0 says buffer LOCKED/NOT FREE and
      Value 1 says buffer UNLOCKED/FREE
   2. **empty_lock** – mutex variable to say buffer is EMPTY or NOT EMPTY to produce/write data.
      Value 0 says buffer NOT EMPTY and
      Value 1 says buffer EMPTY
   3. **fill_lock** – mutex variable to say buffer is FULL or NOT FULL to consume/read data
      Value 0 says buffer FULL and
      Value 1 says buffer NOT FULL
10. The read() acts as producer() function in multiserver.c, who keeps producing/writing the data into the buffer by reading the client requested file.
11. The write() acts as consumer() function in multiserver.c, who keeps consuming/reading the data from the buffer and then writing the same to the client file back.
12. Both producer() and consumer() are 2 threads synchronized by semaphore to avoid race condition occurring between 2 threads.
13. Semaphore functions sem_wait(), waits for the other thread to release and sem_post(), signals the other thread about its release.
14. Both threads should ensure to wait for each other's task completion before terminating.

15. **Terminal 1: Run MultiServer first to accept request from Client:**

Compile the multiserver.c using –lpthread option (lightweight POSIX thread).

$gcc multiserver.c –lpthread –o multiserver && ./multiserver

(Compile server.c and redirect the output to server. Later run/execute server. Both compilation and execution happens in the same command using &&)

15. **Terminal 2: Run Client second to send request to Server:**

$gcc client.c -o client
$./client get 'filename'

('get' is a command line argument that can be passed to the client)

Week 6 Tasks OUTPUT: (OUTPUT IS SAME AS WEEK 2 TASKS)

TERMINAL 1: SERVER PROGRAM

```
dell@dell-Latitude-E7240:~/Desktop$ ls
client.c  multiserver.c  temp.txt
dell@dell-Latitude-E7240:~/Desktop$ gcc multiserver.c -lpthread -o multiserver
dell@dell-Latitude-E7240:~/Desktop$ ./multiserver

Socket created


Bind Done

Waiting for incoming connections...

Successfully Created Producer

Successfully Created Consumer Thread

###temp.txt

In producer@@

In consumer
dell@dell-Latitude-E7240:~/Desktop$
```

TERMINAL 1: CLIENT PROGRAM

```
dell@dell-Latitude-E7240:~/Desktop$ ls
client.c  multiserver.c  temp.txt
dell@dell-Latitude-E7240:~/Desktop$ gcc client.c -o client
dell@dell-Latitude-E7240:~/Desktop$ ./client get 'temp.txt'
@@temp.txtBytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 256

 Read Complete
```