HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

BBM342 OPERATING SYSTEMS
PROJECT I

Subject : Thread, Socket, and Semaphore programming
Submission Date : 06.05.2021
Deadline : 26.05.2021, 23:59 pm
Programming Language : C, Java
Advisors : Assoc. Prof. Dr. Kayhan İMRE, Assoc. Prof. Dr. Ahmet Burak CAN, R. A. Feyza Nur Kılıçaslan

1. INTRODUCTION / AIM:

Operating systems provide a conceptual model consisting of sequential processes running in parallel. Processes can be created and terminated dynamically. Each process has its own address space. For some applications, it is useful to have multiple threads of control within a single process. These threads are scheduled independently and each one has its own stack, but all the threads in a process share a common address space. Because threads share a common address space, they can communicate and share data in this address space.

When multiple processes/threads try to reach same memory address, data consistency should be protected. For consistency, if there is any writing operation, the read/write operations should be organized with primitives such as locks, semaphores, monitors and messages. These primitives are used to ensure that no two processes/threads are ever in their critical regions at the same time, a situation leads to chaos.

The aim of this experiment is to make you familiar with some concepts of processes, threads, sockets, semaphores, and have practical experience with these concepts. You are expected to develop an ASCII art video streaming server and a client application which enables to view streamed content. The server named ***sserver*** will be implemented with C programming language in UNIX and the client named ***sclient*** will be implemented in Java programming language. An example for ASCII art video streaming can be found on http://www.asciimation.co.nz/.

2. EXPERIMENT

The streaming server will stream at most three video channels. Multiple clients cana connect to the streaming server to watch these channels. More than one client can connect to the same video channel, which means multiple client can watch the same channel at the same time. The streaming server takes the following parameters at the start:

```
sserver –p port –s streams –ch1 videofile [–ch2 videofile] [–ch3 videofile]
```

-p port : specifies the port number that the server will listen.
-s streams: specifies the number of videos to be streamed. It can take values between 1 and 3.
-ch1 videofile: specifies the filename of the first video
-ch2 videofile: specifies the filename of the second video, which is optional
-ch3 videofile: specifies the filename of the third video, which is optional

When the server is started, the server starts to listen the specified port by opening a TCP socket. Then, it opens the video files for streaming, and creates threads, buffers and semaphores needed for video streaming operations. The server supports upto 3 streaming channels, where each one streams different video files. The server must be started with at least one channel, but other channels are optional.

The client application connects to a channel of the streaming server by opening a TCP socket. Then, it displays the video stream on a text box or canvas. Until the termination of the client, streaming continues infinitely. The client application is started as follows:

```
sclient  –a address –p port –ch channelID
```

- a address: specifies the IP address/hostname of the streaming server to be connected
- p port: specifies the port number of the streaming server
- ch channel: specifies the channel number to be connected on the server, which can be 1, 2, or 3

The server needs to implement a buffering mechanism for a good streaming experince. For each channel, the server will create a producer thread, which reads the video file content and feeds a buffer with the frames of the video stream. When the file is finished, the producer thread will start over from the beginning of the video file. In other words, the producer feeds the buffer in an infinite loop until terminating the server process. The server must manage multiple clients connecting to the same channel or different channels. More than one client can connect to the same streaming channel, so the server should not delete a frame from the channel's buffer(s) until all the connected clients read the frame.

A client process can connect to only one channel of the streaming server. The client should implement a buffering mechanism to be able to view video with a constant rate. The expected frame rate of video is 20 fps (frame per seconds).

A simplified view of the whole system is given in Figure 1. While the processes are shown with ellipses, threads are shown with wavy symbols. PortListener thread listens the server port and accepts the client connections. For each connected client, a worker thread is assigned for streaming data, such as Worker1, Worker2. Channel consumers (Channel1, Channel2, and Channel3) consume streaming data of each channel from related buffers and feed this data to worker threads. You will need to implement a mechanism to support multiple clients connecting to the same channel, such as Client1 and Client2 in the figure. Your actual design and implementation can be different, the figure just provides a sample architecture.

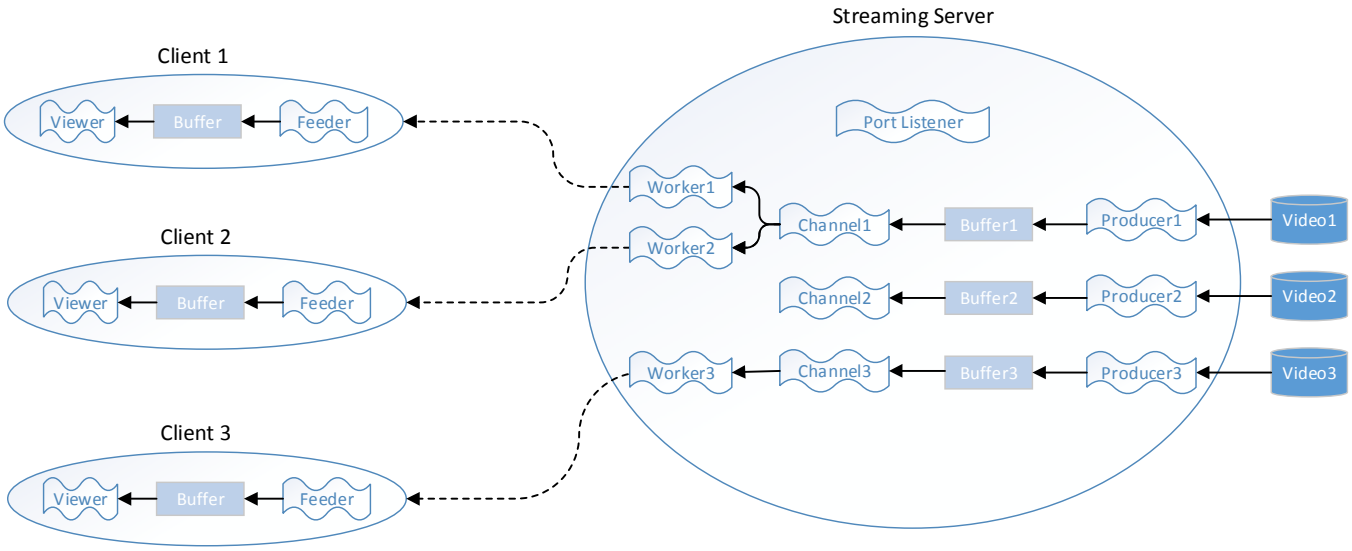

Figure 1. A simplified view of the streaming architecture

Some channels may be idle (i.e. no client is connected), such as Channel 2 in Figure 1. Clients may connect to (or disconnect from) the streaming server on arbitrary times. While the server is streaming data to a client on a channel, new clients may connect to the same channel. Similarly, while multiple clients are connected to the same channel of the server, a client may disconnect from the channel and server. As the new clients connect to the server or disconnect from the server, other connected clients must continue streaming operation without having any delay or problem.

GRADING:

- The source code of program have 90 points: 20 points for Coding Convention and Design, and Comments and 70 points for Execution.
- The report have 10 points. In the report, you are expected to explain your solution method, important definition or data structures and limitations (such as maximum number of clients per channel, etc).
- If your code only supports a single client connection per channel, you can get at most 70% of the full credit.
- Your final grade will be calculated depending on the execution of your program. To be able to get full points from report and other parts of source code, you should get at least 30 points from execution (30/60).
- As understood from grading you should model your system accurately. Attributes' and functions' names must be understandable. You should write plenty of comment lines in your code.

LAST REMARKS:

- Your submission code file structure must implement this template:
  <student_number>.zip
  |--- report.pdf | docx
  |--- src
  |--- sserver.c
  |--- sclient.java
  |--- *.c | *.h
  |--- *.java

- **You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ Deadline is: 23:59 pm.**
- Do not submit any file via e-mail related with this assignment.
- **SAVE** all your work until the assignment is graded.
- The assignment must be original, **INDIVIDUAL** work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms or source codes.
- You can ask your questions through course's communication channel and you are supposed to be aware of everything discussed in the channel: https://piazza.com/hacettepe.edu.tr/spring2021/bbm342/.

REFERENCES:

1. A Simple Java socket programming example: https://www.baeldung.com/a-guide-to-java-sockets
2. Java socket programming examples: https://cs.lmu.edu/~ray/notes/javanetexamples/
3. Java multithread programming: https://www.javatpoint.com/multithreading-in-java
4. UNIX TCP Server programming example: https://www.tutorialspoint.com/unix_sockets/socket_server_example.htm
5. A tuturial on UNIX TCP programming: https://medium.com/swlh/getting-started-with-unix-domain-sockets-4472c0db4eb1
6. A multithreaded UNIX TCP Server example: https://dzone.com/articles/parallel-tcpip-socket-server-with-multi-threading
7. Another multithreaded UNIX TCP Server code: https://gist.github.com/oleksiiBobko/43d33b3c25c03bcc9b2b
8. UNIX Pthread programming examples: https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html