CSCI 4061: Introduction to Operating System

Fall 2017 (Instructor: Anand Tripathi)

Assignment 5: Inter-process Communication using TCP/IP
Due: December 9, 2017

This assignment can be done either individually or in a team of up to two students.

Instruction: This assignment contains 2 parts. Solution of part 1 must be saved in a PDF file and put in the folder named 'Theory'. Solution of part 2 must be put in the folder 'Project'. Create another file named teaminfo.txt in the same directory containing the name and student IDs of the team members. These two folders must be zipped (.zip or .tar or .tar.gz) in a single file and submitted using the Homework Submission link on Moodle

PART - 1 : Theory (20 points)

Problem 1: (5 points) Consider a system with three processes, named P, Q, and R. Each process continuously prints its color. We now require that the printing by the processes be coordinated such that the output produced would be

P:RED Q:WHITE R:BLUE P:RED Q:WHITE R:BLUE P:RED Q:WHITE R:BLUE …

Using semaphores, synchronize these three process such that processes P, Q, and R execute their print statements in the above order. Insert appropriate synchronization code before and after the each print statement:

```
Process P   {
     while ( true ) {  // Insert synchronization code
          print("P:RED");
     }
  }  //end of process P
Process Q  {
      while ( true ) {  // Insert synchronization code
          print("Q:WHITE");
    }
}  //end of process Q
Process R  {
      while ( true ) {  // Insert synchronization code
           print("R:BLUE");
     }
}  //end of process R
```

Problem 2: **(5 points)** Suppose that a system has two processes, each of which needs 6 seconds of CPU time and 12 seconds for I/O. I/O operations are processed sequentially.

(a) How long will it take to complete both these processes if they run sequentially?
(b) What are the best case and worst case completion times for these processes if they run concurrently, using multiprogramming?

Question 3: (5 points) Consider a system in which a page can store 200 integers. On this system a small program that operates on a two-dimensional matrix *A* is executed. The program code resides in page 0, which corresponds to addresses 0 through 199. This page is always kept in the physical memory.

A is defined as: int A[ ] [ ] = int [10][200];
where A[0][0] is at the logical address 200 and the matrix is stored in the memory in the row-major form.

Consider the following two ways to initialize this matrix. For each of these two cases answer the following:

Identify the pattern in *page reference strings* these two initialization routines would generate for accessing matrix A. You may just identify the pattern that gets repeated because there may be several hundred page references.

Initialization 1:
```
for (int i =0; i< 10;  i++)
    for (int k =0; k < 200;  k++)
        A[ i ] [ k ] = 0;
```

Initialization 2:
```
for (int k =0; k < 200;  k++)
    for (int i =0; i< 10;  i++)
        A[ i ] [ k ] = 0;
```

Problem 4 (5 points): Consider a Unix style file allocation on disk with total 13 storage pointer entries in the inode. In the inode, the first 10 storage pointers point directly to file data blocks, the other three are pointers to indirect pointer blocks, doubly-indirect pointer blocks, triple-indirect pointer blocks. Assume that the file data - block size is 1028 bytes, and an indirect block contains 256 file data-block addresses.
What's the maximum file size which this system can store?

PART – 2 : Programming Project (100 points)

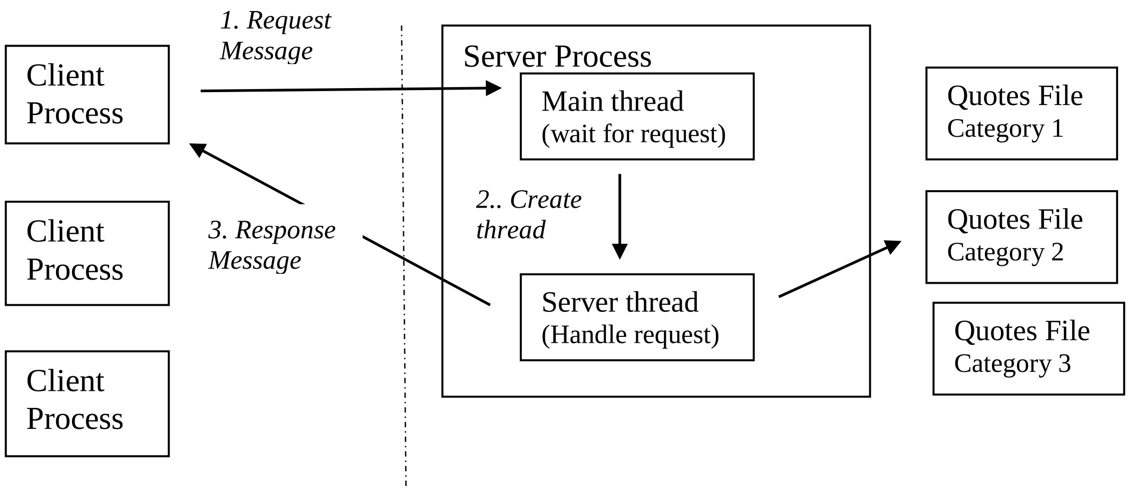
1. Objectives

The objective of this assignment is to learn basic concepts in inter-process communication using TCP sockets. You will learn the basic concepts in programming with the TCP protocol using the C language to implement a multi-threaded server.

This assignment has to be done either individually **or** in a group of two students.

2. Assignment Statement

In this assignment you will implement a multi-threaded client-server application. The architecture of the application is shown in the figure below. You have to implement a quote server in C (**quote_server.c**) that will communicate with a client (**quote_client.c**) using TCP sockets. **The client code is provided to you.** You need to implement only the server side.



This application is structured as one server process that communicates with any arbitrary number of client processes. A client process sends a request message to the server to receive a quote (*one sentence statement of some kind of philosophical value*) from the server. The server process would maintain quotes in several different categories, and all quotes in a given category would be stored in one text-file.

On receiving a client request for a quote in some specific category, server picks up a quote in the requested category and sends it as the response to the client request. The server reads quotes from *a* set of files at the server side containing quotes for different categories. These files are also provided to you. The format of the files containing quotes would be that each quote is stored in two consecutive lines of the file.

The server will also maintain a *configuration file* containing the names of all the quote categories, and the name of the file for each of the category. We will provide you the following **config.txt** file and the corresponding quotes file.

```
Einstein: einstein.txt
Twain: twain.txt
Computers: computers.txt
```

2.1 Server startup

The server will be started from the command line. It will take the name of the *configuration* file as an argument. The server process will go in a loop listening for client session requests on a specified port (*which will be 6789* in your assignment code).

% quote-server config.txt

2.2 Server side request handling

Upon receiving a session connection request on port 6789, the server should create a thread that will handle that client session. This thread should handle the request as following:

- The thread should open all the files listed in the *configuration* file in read mode.
- Based on the category specified by the client, the thread would pick up the next quote from the file that contains quotes for that category, starting with the first quote in the beginning. Here "next" is to be considered only with respect to that client.
- If the mentioned category does not have a matching file in the file database of the server, the server should return an error message to the client.
- If the client asks for a list of available categories, the server should return the list of the categories read from the *configuration* file.
- If the client sends "BYE", the server thread should close the connection and terminate itself.
- After handling a request other than "BYE", the thread should go back in a loop and wait for next request from the client.

2. 3 Client-side Session

The client will establish connection to the server running on the specified host. The client will be invoked from the command line with the server's host name (DNS name) as the parameter. It displays a prompt for taking the request from the user. The request might contain a category (a string) or "LIST" or "BYE" or nothing. If the client does not give any input and just presses <ENTER>, the server should pick up a random category from those available and send the next quote from the corresponding file. A typical run of the client should look like this.

Example of a Client Session:

Suppose that the server is running on a host name one.cselabs.umn.edu and a port number. The user will start a client as shown below, assuming that the server is listening on port 5001.:

% quote_client one.cselabs.umn.edu 5001
Connection to the quote server established

Below is an example of a client session. Text in **boldface** indicates user input in the session, and non-boldface text indicates the text output of the program. Text in italics is a comment

Example when the user selects the LIST command:

```
   Press <ENTER> for any quote
  'LIST' for a list of categories
   <category name> for a quote in a specific category
 ' BYE' to quit
Enter your selection:  LIST

Einstein
Twain
Computers
```

Example when user selects category name 'Computers':

```
   Press <ENTER> for any quote
  'LIST' for a list of categories
   <category name> for a quote in a specific category
  ‘ BYE’ to quit
Enter your selection:  Computers

 "C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole
leg."    -Bjarne Stroustrup
```

Example when user selects category name ‘Einstein’:

```
    Press <ENTER> for any quote
   ‘LIST’ for a list of categories
    <category name> for a quote in a specific category
  ‘ BYE’ to quit
Enter your selection:  Einstein

"Only a life lived for others is a life worthwhile."
-Albert Einstein
```

Example when user input is BYE:

```
    Press <ENTER> for any quote
   ‘LIST’ for a list of categories
    <category name> for a quote in a specific category
  ‘ BYE’ to quit
Enter your selection:  BYE

Connection to server terminated. Exiting…
```

2. 4 Requirements

- The quote_client provided to you should be able to talk to your quote_server and obtain
  - A list of available categories
  - Quote from a specified category *or a random category if the client presses <ENTER>.*
  - If the client asks for a quote from the same category again, the server should return the next quote from the file.
- Multiple instances of the quote_client should be able to talk to the server at the time.

2.5 Hints

- Each server thread should open all the quote files in read mode on creation. After that you can have a switch statement on the client request. Depending on the category you should get two lines from the corresponding file using fgets(). This way if the client asks for the same category later, it will get the next two lines, since fgets() advances the offset pointer every time it reads a line. If you ever reach the end of some file (when fgets() returns zero), you can start again from the beginning of the file.
- If the client specifies no category (by simply pressing <ENTER>) then you should pick a random category and do an fgets() on the corresponding file.
- Set attributes of server threads as PTHREAD_CREATE_DETACHED and PTHREAD_SCOPE_SYSTEM

C Functions that can be used

- Sockets: socket() , bind() , accept() , listen() , close()

2.6 Things to Submit:

- Submit only **quote_server.c** file and any other header files that you might be using.
- Please provide a Makefile
- At the TOP of the program file, please include names of each of the students in your group and student Id numbers.

2.7 Note on Comments
Please comment your code. It is a good practice to comment your code so that it is understandable to you and others. Proper commenting will save you lot of time while debugging. Also see to it that you do not have excessive comments. Comments should be crisp clear and to the point.

3.0 Grading Criteria:

- (20 points) Creation of server thread on each new client session connection, with multiple clients at the same time
- (15 points) Correctly handling quote requests for each of the three categories
- (12 points) Correctly delivering the next quote when multiple times requests are made for the same category.
- (12 points) Selecting a random category when the user presses <ENTER>
- (10 points) Correctly handling the case when the user enters a non-existing category name.
- (12 points) Correctly handling the LIST request
- (9 points) Correctly handling of BYE by terminating the session connection and termination of the server thread.
- (5 points) Correctly functioning Makefile
- (5 points) Commenting of the code