Indian Institute of Technology Jodhpur
Operating Systems Lab (CSL 3030)
Assignment 3

Dated 31st August, 2021 Total marks: 20

Part-I needs to be finished in the lab session. Part-II is a take-away assignment component for further evaluation.

Part-I

Write a C program to implement the simple producer-consumer problem using a bounded buffer of defined size. For implementing the shared buffer, you have to use POSIX API for shared memory implementation.

1. Create two processes- the producer and the consumer.
2. Create a shared memory segment to implement the bounded buffer.
3. Let the processes communicate using the buffer.

You don't have to use synchronization tools as of now. It is okay if your solution shows race condition or does not utilize the buffer capacity fully.

Part-II

In this assignment you are going to learn the basics of UNIX socket programming. You will implement a simple client-server model where a client program can chat with a dummy math server. The communication between client and server goes as follows:

- The server gets started first and listens to a known port.
- The client program is started as server IP and port is provided in the command line.
- The client connects to the server and asks for user input. The user enters simple arithmetic expression as "5+5" or "6-2". The input is sent to the server through connected socket.
- The server receives the input from the client socket, evaluate the expression and sends the result back to the client.
- The client should display the result from the server to the user and prompts for the next input, until the user terminates the client program by Ctrl+C.

You are provided with the Client skeleton source code. Write C/C++ code for **three** versions of server.

1. Your server program 'Server1' is a single process server that can handle only one client at a time. If another client tries to chat with the server while one client session is going on the second client will receive an error message like "line busy!"
2. Your server program "Server2" is a multi-process server that forks a process whenever it receives a new client request. Multiple clients will be able to chat with the server concurrently.

3. Your server program “Server3” is a single process (multi-threaded) server that uses select system call to handle multiple clients concurrently.

At the bare minimum the server should be able to handle addition, subtraction, multiplication and division operations on two integer operands.

The Client:

The Client code is first compiled. The server runs on port 5555 in another terminal. The client program is then given the IP (localhost 127.0.0.1 in this case) and port (5555) as command line inputs. Sample runs of the client goes as follows.

```
$ gcc client.c -o client
$ ./client 127.0.0.1 5555
Connected to server
Please enter the message to the server: 2 + 4
Server replied: 6
Please enter the message to the server: 3 * 3
Server replied: 9
...
...
```

The Server:

The sample run of the server is as follows.

```
$ gcc server1.c -o server1
$ ./server1 5555
Connected with client socket number 4
Client socket 4 sent message: 2 + 4
Sending reply: 6
Client socket 4 sent message: 3 * 3
Sending reply: 9
...
...
```

Skeleton code for Client.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define MAX_INPUT_SIZE 256

int main(int argc, char *argv[])
{
    int sockfd, portnum, n;
    struct sockaddr_in server_addr;

    char inputbuf[MAX_INPUT_SIZE];
```

```
    if (argc < 3) {
       fprintf(stderr,"usage %s <server-ip-addr> <server-port>\n", argv[0]);
       exit(0);
    }

    portnum = atoi(argv[2]);

    /* Create client socket */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
      {
       fprintf(stderr, "ERROR opening socket\n");
       exit(1);
      }

    /* Fill in server address */
    bzero((char *) &server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    if(!inet_aton(argv[1], &server_addr.sin_addr))
      {
       fprintf(stderr, "ERROR invalid server IP address\n");
       exit(1);
      }
    server_addr.sin_port = htons(portnum);

    /* Connect to server */
    if (connect(sockfd,(struct sockaddr *)&server_addr,sizeof(server_addr)) <
0)
      {
       fprintf(stderr, "ERROR connecting\n");
       exit(1);
      }
    printf("Connected to server\n");

    do
      {
       /* Ask user for message to send to server */
       printf("Please enter the message to the server: ");
       bzero(inputbuf,MAX_INPUT_SIZE);
       fgets(inputbuf,MAX_INPUT_SIZE-1,stdin);

       /* Write to server */
       n = write(sockfd,inputbuf,strlen(inputbuf));
       if (n < 0)
         {
           fprintf(stderr, "ERROR writing to socket\n");
           exit(1);
         }

       /* Read reply */
       bzero(inputbuf,MAX_INPUT_SIZE);
       n = read(sockfd,inputbuf,(MAX_INPUT_SIZE-1));
       if (n < 0)
         {
           fprintf(stderr, "ERROR reading from socket\n");
           exit(1);
         }
```

```
        printf("Server replied: %s\n",inputbuf);

      } while(1);

    return 0;
}
```

Evaluation Guidelines:

Submit three versions of the server code along with the client code. Include a readme file to reflect your understanding about the comparative performance analysis among all three versions.

- **For three versions of the server codes 3*5=15 marks**
- **Readme file 5 marks**

Deadline: 7th September, 2021