INDIAN STATISTICAL INSTITUTE

Course: M.Tech (CS)

Subject: Computer Networks (Lab)

November 1, 2023

Assume that all numbers are unsigned 32-bit integers and no computation results into an overflow/underflow

Properly organize your codes and put comments as applicable

Run your server locally at `127.0.0.1`. Take port number as `543XX`, where `XX` is your roll number/day of birth.

Part A: Socket Programming

Consider a client-server system that can send numbers back and forth. The client iteratively takes a number from the user and sends it to the server. Upon receiving a number the server does some computation on it, and then sends back another number as a reply. The client then prints the reply, and this process starts over. The process goes on until user enters a special value, say 0, as the input.

A1. Building a Socket Program for Counting Primes

i. Write a client program that takes an integer from the user and sends it to a server using Socket APIs. The client awaits for a reply from the server, and then prints that reply value. The client then takes another integer from the user and repeats this process. If the input value is 0, the client closes the connection. [3]

ii. Write a corresponding server program that can receive a number from a client and sends another as a reply. The server maintains a running *counter* for the number of primes seen so far. The server checks whether the received number is prime or not, and updates the counter value accordingly. The server sends the counter value as a reply. If the received value is 0, the server closes the connection. [Try to be as efficient as possible while testing for primality.] [7]

A2. Building a Socket Program for Computing a Moving Average

i. Write a client program that takes an integer from the user and sends it to a server using Socket APIs. The client awaits for a reply from the server, and then prints that reply value. The client then takes another integer from the user and repeats this process. If the input value is 0, the client closes the connection. [3]

ii. Write a corresponding server program that can receive a number from a client and sends another as a reply. The server maintains an *integral moving average* for all the numbers seen so far. The server includes the received number, and updates the average value accordingly. The server sends the new average as a reply. If the received value is 0, the server closes the connection. Assume that the integral moving average at i-th step is computed as $\overline{x}_i = \lfloor \alpha x_i + (1-\alpha)\overline{x}_{i-1} \rfloor$, where x_i is the received number at this step and consider $\alpha = 0.9$. [Note that the floor function $\lfloor x \rfloor$ is automatically computed when a floating point value is stored into an integral datatype in C[1].] [7]

A3. Building a Socket Program for Simulating Flow Control

i. Simulate the Go-Back-N protocol for flow control where frame delay may happen. The user specifies the window size for the client. Assume no frame is lost, but it may be delayed. Use the `sleep()` to introduce delays and the `select()` API to detect a timeout event. [You are free to make any reasonable assumptions.] [10]

ii. Simulate the Go-Back-N protocol where some packets may be lost/corrupted while transmitting. Devise some mechanism to introduce packet loss scenarios and accordingly update the solution developed in A3.i. [You are free to make any reasonable assumptions.] [10]

[1] For A2.ii, you may see this discussion: `https://stackoverflow.com/questions/12240228/c-integer-division-and-floor`

Part B: Remote Procedure Call

Consider a client-server system that can perform some computation on a remote machine. The client takes some input from the user and invokes a remote procedure at the server with it. The server executes the invoked procedure on the given parameters, and then returns a reply. The client then prints the return value.

B1. **Building an RPC Program for Vector Scalar Multiplication**

i. Suppose we need to perform a vector scalar multiplication operation using the Remote Procedure Call APIs. Therefore, we need to send a vector along with a scalar value to the remote procedure and get back another vector after the multiplication has been done. Write the specification file (.x file) for this purpose. Assume that the vectors are stored as fixed sized (max size = 20) integer arrays, and the scalar multiplier to be a positive integer. [Note that a vector can be of smaller size also, say 5.] [3]

ii. Use `rpcgen` to generate the required files. [Modify the generated `makefile` if required.] [1]

iii. Modify the generated client and server programs so that the client first takes size of the vector n, followed by elements of the vector $A =< a_1, a_2, \ldots, a_n >$ and a positive integer k from the user. Then the client invokes the remote procedure with this A and k. The remote procedure does the scalar multiplication and returns another vector, say $B =< b_1, b_2, \ldots, b_n >$, where $b_i = a_i \times k$. The client also displays the returned vector. [3+3]

B2. **Building an RPC Program for Vector Addition**

i. Suppose we need to perform a vector addition operation using the Remote Procedure Call APIs. Therefore, we need to send two vectors to the remote procedure and get back another vector after the addition is done. Write the specification file (.x file) for this purpose. Assume that the vectors are stored as fixed sized (max size = 20) integer arrays. [Note that a vector can be of smaller size also, say 5.] [3]

ii. Use `rpcgen` to generate the required files. [Modify the generated `makefile` if required.] [1]

iii. Modify the generated client and server programs so that the client first takes size of the vector n, followed by elements of the two vectors $A =< a_1, a_2, \ldots, a_n >$ and $B =< b_1, b_2, \ldots, b_n >$ from the user. Then the client invokes the remote procedure with this A and B (and also n). The remote procedure adds the two vector returns another vector $C =< c_1, c_2, \ldots, c_n >$ where $c_i = a_i + b_i$. The client also displays the returned vector. [3+3]

B3. **Building an RPC Program for Identification of Primes in an Array**

i. Consider a mapping f, where we map prime numbers to 1 and others to 0. Given an array A of positive integers, we want to compute another array B where $B = f(A)$. More specifically the i-th element of B is 1 whenever the i-th element of A is a prime, and 0 otherwise. This can be mathematically written as: [3]

$$B[i] = f(A[i]) = \begin{cases} 1 & \text{if } A[i] \text{ is prime} \\ 0 & \text{otherwise} \end{cases}$$

Given the array A, we wish to compute this array B through the Remote Procedure Call APIs. Therefore, we need to send the array A to the remote procedure and get back another array B after the above mapping has been done. Write the specification file (.x file) for this purpose. Assume that both arrays are statically defined and contains exactly 10 integers. [Try to be as efficient as possible while testing for primality.]

ii. Use `rpcgen` to generate the required files. [Modify the generated `makefile` if required.] [1]

iii. Modify the generated client and server programs so that the client first takes number elements in the array n, followed by elements of the array A from the user. Then the client invokes the remote procedure with this A. The remote procedure applies the above mapping and returns the array B. The client also displays the returned array. [3+3]