INDIAN STATISTICAL INSTITUTE

Course: M.Tech (CS)

Subject: Computer Networks (Lab)

December 11, 2023

Duration: 1 hour **Total: 20 Marks**

This is an open book, open net exam. You may use any available material with proper attribution to the sources

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`, with port number as `543XX`, where `XX` is your roll number/day of birth.

Answer any one of the following four questions

1. **Building a Socket Program for Counting Runs of Palindromes** **[20]**
   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.

   i. Write a client program that takes a positive 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. [8]

   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 successive palindromes seen so far (current run length of consecutive palindromes). The server checks whether the received number is palindrome or not, and updates the run value accordingly. The server sends the run 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 palindromes.] [12]

   Example:

input sequence:	12	13	121	5	33	19	131	15	0
run length counter:	0	0	1	2	3	0	1	0	`close`

2. **Building an RPC Program for Computing Vector Convolution** **[20]**
   Recall that the convolution of two vectors A and B, is a vector of length $|A| + |B| - 1$, where the k-th term of the resultant vector $C = \texttt{convolution}(A, B)$, is computed as follows:

   $$C[k] = \sum_{i+j=k} A[i] \times B[j]$$

   i. Suppose we need to perform a vector convolution operation using the Remote Procedure Call APIs. Here, we need to send two vectors (possibly of different lengths) to the remote procedure and get back another vector after the computation is done. Write the specification file (.x file) for this purpose. Assume that the vectors are declared as fixed sized (max size = 20) integer arrays. [Note that a vector can be of smaller size also, say 5, as well.] [6]

   ii. Use `rpcgen` to generate the required files. [Modify the generated `makefile` if required.] [2]

   iii. Modify the generated client and server programs so that the client first takes size of the vectors n and m, followed by elements of the two vectors $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ and $B = \langle b_0, b_1, \ldots, b_{m-1} \rangle$ from the user. Then the client invokes the remote procedure with this A and B (and also n and m). The remote procedure performs the computation on the two vectors and returns another vector $C = \langle c_0, c_1, \ldots, c_{n+m-2} \rangle$, where $c_k = \sum_{i+j=k} a_i \times b_j$. The client also displays the returned vector. [6+6]

3. **Building a Socket Program for Simulating Flow Control Over a Noisy Channel** **[20]**
Simulate the stop-and-wait protocol for a noisy channel, where every alternate packet is lost while transmitting. Assume no frame delay occurs, and all acknowledgement packets are delivered without fail. Let n be the number of packets to be sent (taken as user input on the client side). Write the client and server socket programs to simulate this. After the simulation, the client program would print the total number of transmission attempts made to successfully deliver all n packets (considering both successful as well as failed attempts). [You are free to make any other reasonable assumptions.]

4. **Simulation of Routing Protocol through ns-3** **[20]**
Consider the network topology in the figure below, where the link states are given as $\langle delay, datarate \rangle$. Here node 0 is the source and node 5 is the sink. Assume it uses the OLS routing protocol. Modify the ns-3 code `exp11.cc` to implement this topology. Run the simulation and name the trace file as `lsr.tr`. Use the Python script `node.py` provided to you to find the most used route by data packets.