Nepal College of Information Technology

Balkumari, Lalitpur

(Affiliated to Pokhara University)

A Lab Report

On

Concurrent TCP Echo Server and Client Using Fork in C

Submitted as partial fulfillment of requirement of the curriculum of Bachelor's of Engineering in Software Engineering (6th Semester)

Submitted by:

Harsh Chaudhary Kalwar

(221715)

Submitted to:

Madan Bhandari

Date:

28th June, 2025

Objective:

The objective of this lab was to implement a concurrent TCP echo server capable of handling multiple clients simultaneously using the fork() system call. Each client sends a message, and the server echoes it back. The server creates a new process for each client connection to allow concurrent handling.

Lab Tasks and Execution:

1. Concurrent TCP Echo Server (conServer.c):

Functionality:

· Listens on a specified port for incoming TCP connections.
· For each client connection, spawns a child process using fork().
· In the child, handles receiving data and sending the echoed message back.
· The parent continues accepting new connections.

Corrections and Completion in Skeleton:

· Replaced: connfd = accept(); //change this
· with: connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &clilen);
· Added the echo handling logic in the child process:

```
while ((n = read(connfd, buf, MAXLINE)) > 0) {
buf[n] = '\0';
printf("Echoing back to client: %s", buf);
write(connfd, buf, strlen(buf));
}
close(connfd);
exit(0);
```

Key Concepts Used:

· socket(), bind(), listen(), accept()
· fork() for concurrency
· read() and write() for communication

2. TCP Echo Client (conClient.c):

Functionality:

· Connects to the server using IP and port provided as command-line arguments.
· Accepts user input from the keyboard.
· Sends the message to the server and prints the echoed response.

Filled Inside While Loop:

```
while (fgets(sendline, MAXLINE, stdin) != NULL) {
write(sockfd, sendline, strlen(sendline));
if (read(sockfd, recvline, MAXLINE) == 0) {
perror("Server terminated prematurely");
exit(4);
}
printf("Echo from server: %s", recvline);
}
```

Key Concepts Used:

· socket(), connect()
· fgets() for user input
· write() and read() for sending and receiving

Output / Observations:

· Server terminal output showed:

```
Server running at Port: 3000
Received request...
No of Child: 1
Child created
Echoing back to client: Hello Server
```

· Client terminal output showed:

```
Enter message:
Hello Server
Echo from server: Hello Server
```

· Multiple clients were able to connect concurrently. Each got their own child process and echoed messages correctly.

Conclusion:

This lab effectively demonstrated the creation of a concurrent TCP server using fork() for handling multiple clients. Each child process handled one client, allowing simultaneous interactions. It reinforced key concepts of socket programming and inter-process communication in UNIX/Linux.