



Assignment Report

This report contains work from Assignment 2 of course CSN 361.

Name- Kaustubh Trivedi
Enrollment Number - 17114044
Class - CSE B.Tech. 3rd Year
Submission Files - [Repository Link](#)

QUESTION 1: Write a socket program in C to connect two nodes on a network to communicate with each other, where one socket listens on a particular port at an IP, while other socket reaches out to the other to form a connection.

SOLUTION: The solution has 2 programs:

Server: Program creates a node which has a socket to listen on the port 8080 (localhost).

`int sockfd = socket(AF_INET, SOCK_STREAM, 0)` Socket creation with IPv4 communication domain and TCP communication type.

`setsockopt()` Helper Method in manipulating options for the socket referred by the file descriptor sockfd.

`bind()` method binds the socket to the address and port number specified in `addr`(custom data structure).

`listen()` method puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.

`accept()` method extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket.

`sockaddr_in` is a struct for all syscalls and functions that deal with internet addresses.

ServerCode:

```
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#define PORT 8080

int main(int argc, char const *argv[]) {
```

```
int serverfd, newSocket, valRead;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
char buffer[1024] = {0};
char *response = "Server responded!"; // Response message from
server

serverfd = socket(AF_INET, SOCK_STREAM, 0); // File descriptor
for socket with IPv4 and TCP.

if(serverfd == 0) {
    perror("socket failed!");
    exit(EXIT_FAILURE);
}

// Attach the socket to port 8080 forcefully
if(setsockopt(serverfd, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT, &opt, sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY; // For localhost
address.sin_port = htons(PORT);

// Bind the socket to localhost 8080
if (bind(serverfd, (struct sockaddr *)&address,
sizeof(address))<0) {
    perror("bind failed");
    exit(EXIT_FAILURE);
}

printf("Server listening...\n");

if (listen(serverfd, 3) < 0) { // Wait for client to make
approach. Backlog here is 3 (max size of wait queue).
    perror("listen");
```

```

        exit(EXIT_FAILURE);
    }

    // Extract the first connection from pending queue and establish
    connection b/w client and server by creating a new socket.
    if ((newSocket = accept(serverfd, (struct sockaddr *)&address,
(socklen_t*)&addrlen))<0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    valRead = read( newSocket , buffer, 1024);
    printf("%s\n",buffer );
    send(newSocket , response , strlen(response) , 0 );
    printf("Response message sent\n");
    return 0;
}

```

Client: This program creates a node which sends a request on port 8080 to be read by the server program. The **connect()** system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

ClientCode:

```

#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <string.h>

#define PORT 8080

int main(int argc, char const *argv[])
{
    int sock = 0, valread;

```

```
struct sockaddr_in serv_addr;
char *request = "Client Request";
char buffer[1024] = {0};

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("\n Error in socket creation. \n");
    return -1;
}

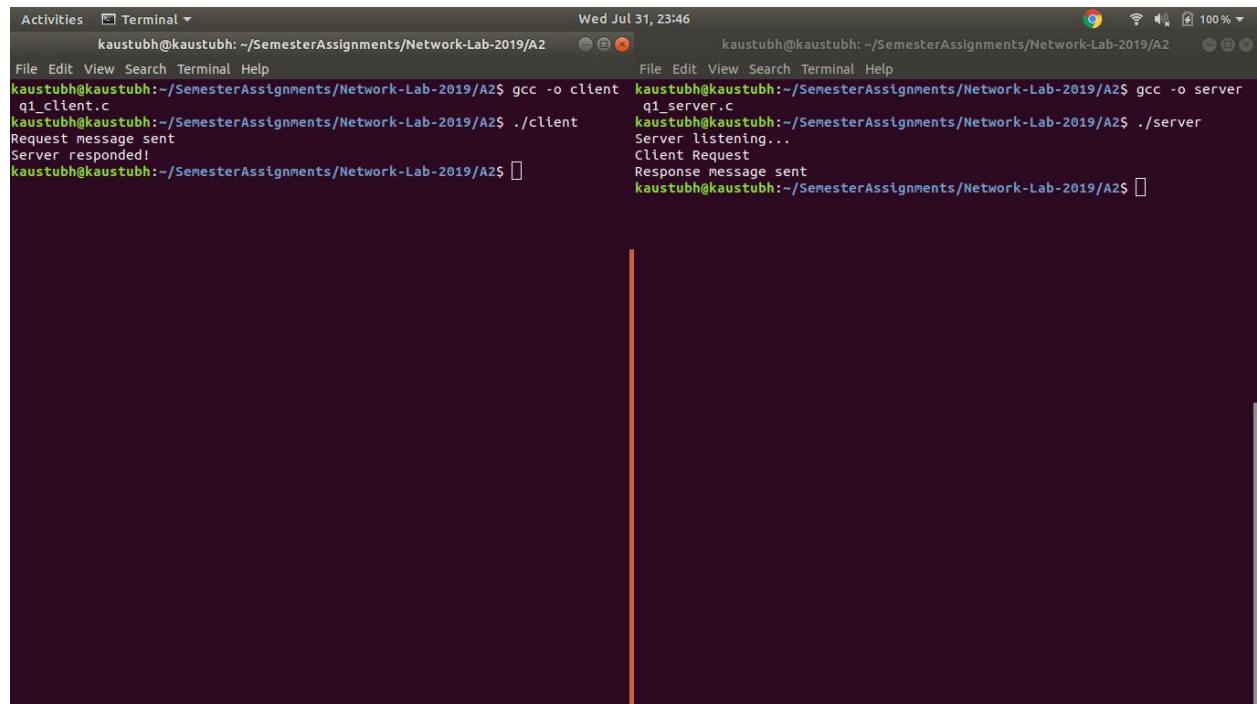
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
{
    printf("\nInvalid address/ Address not supported \n");
    return -1;
}

if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) <
0)
{
    printf("\nConnection Failed \n");
    return -1;
}

send(sock , request , strlen(request) , 0 );
printf("Request message sent\n");
valread = read( sock , buffer, 1024);
printf("%s\n",buffer );
return 0;
}
```

SCREENSHOTS:



The image displays two side-by-side terminal windows from a Linux system, showing the compilation and execution of a client-server program. Both windows are titled 'kaustubh@kaustubh: ~/SemesterAssignments/Network-Lab-2019/A2' and show the date 'Wed Jul 31, 23:46'.

Left Terminal Window:

```
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ gcc -o client q1_client.c
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ ./client
Request message sent
Server responded!
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$
```

Right Terminal Window:

```
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ gcc -o server q1_server.c
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ ./server
Server listening...
Client Request
Response message sent
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$
```

QUESTION 2: Write a C program to demonstrate both Zombie and Orphan process.

SOLUTION: The following program uses the `fork()` and `sleep()` system calls to create a simulation of orphan and zombie processes.

SOLUTION CODE:

Here I create a simulation of a parent and child process for demonstration.

```
#include <stdio.h>
#include <sys/wait.h>
#include <bits/stdc++.h>
#include <unistd.h>

using namespace std;

int main() {

    cout << "Parent's process id: " << getpid() << endl << endl;

    pid_t child_pid = fork();

    if (child_pid > 0) {
        cout << "Parent active..." << endl;
        sleep(4);
        cout << "Parent terminated" << endl;
    }
    else if (child_pid == 0) {
        cout << "Child created with pid "<< getpid() << " from parent pid "
        << getppid() << endl;
        child_pid = fork();
        if(child_pid > 0) {
            sleep(1);
            cout << "Child sleeping..." << endl;
            sleep(2);
            cout << "Child awake again and active!" << endl;
            sleep(2);
            cout << "Child is now orphan!" << endl << endl;
        }
        else if(child_pid == 0) {
            cout << "Grandchild created with pid "<< getpid() << " from
            parent pid " << getppid() << endl << endl;
            sleep(1);
            cout << "Terminating grandchild" << endl;
        }
    }
}
```

```

        cout << "Grandchild is now zombie" << endl << endl;
    }
}

return 0;
}

```

SCREENSHOTS:

The screenshot displays a terminal window on the left and a code editor on the right, both showing the execution of a C++ program named q2.cpp.

Terminal Output:

```

kaustubh@kaustubh: ~/SemesterAssignments/Network-Lab-2019/A2$ g++ -o q2 q2.cpp
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ ./q2
Parent's process id: 28920
Parent active...
Child created with pid 28921 from parent pid 28920
Grandchild created with pid 28922 from parent pid 28921
Child sleeping...
Terminating grandchild
Grandchild is now zombie
Child awake again and active!
Parent terminated
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$ Child is now orphan!
kaustubh@kaustubh:~/SemesterAssignments/Network-Lab-2019/A2$

```

Code Editor Content (q2.cpp):

```

1  #include <stdio.h>
2  #include <sys/wait.h>
3  #include <bits/stdc++.h>
4  #include <unistd.h>
5
6  using namespace std;
7
8  int main() {
9
10     cout << "Parent's process id: " << getpid() << endl << endl;
11
12     pid_t child_pid = fork();
13
14     if (child_pid > 0) {
15         cout << "Parent active..." << endl;
16         sleep(4);
17         cout << "Parent terminated" << endl;
18     }
19     else if (child_pid == 0) {
20         cout << "Child created with pid "<< getpid() << " from parent pid " <<
21         child_pid << endl;
22         if (child_pid > 0) {
23             sleep(1);
24             cout << "Child sleeping..." << endl;
25             sleep(2);
26             cout << "Child awake again and active!" << endl;
27             sleep(2);
28             cout << "Child is now orphan!" << endl << endl;
29         }
30         else if (child_pid == 0) {
31             cout << "Grandchild created with pid "<< getpid() << " from paren
32             sleep(1);
33             cout << "Terminating grandchild" << endl;
34             cout << "Grandchild is now zombie" << endl << endl;
35         }
36     }
37
38     return 0;
39 }
40

```