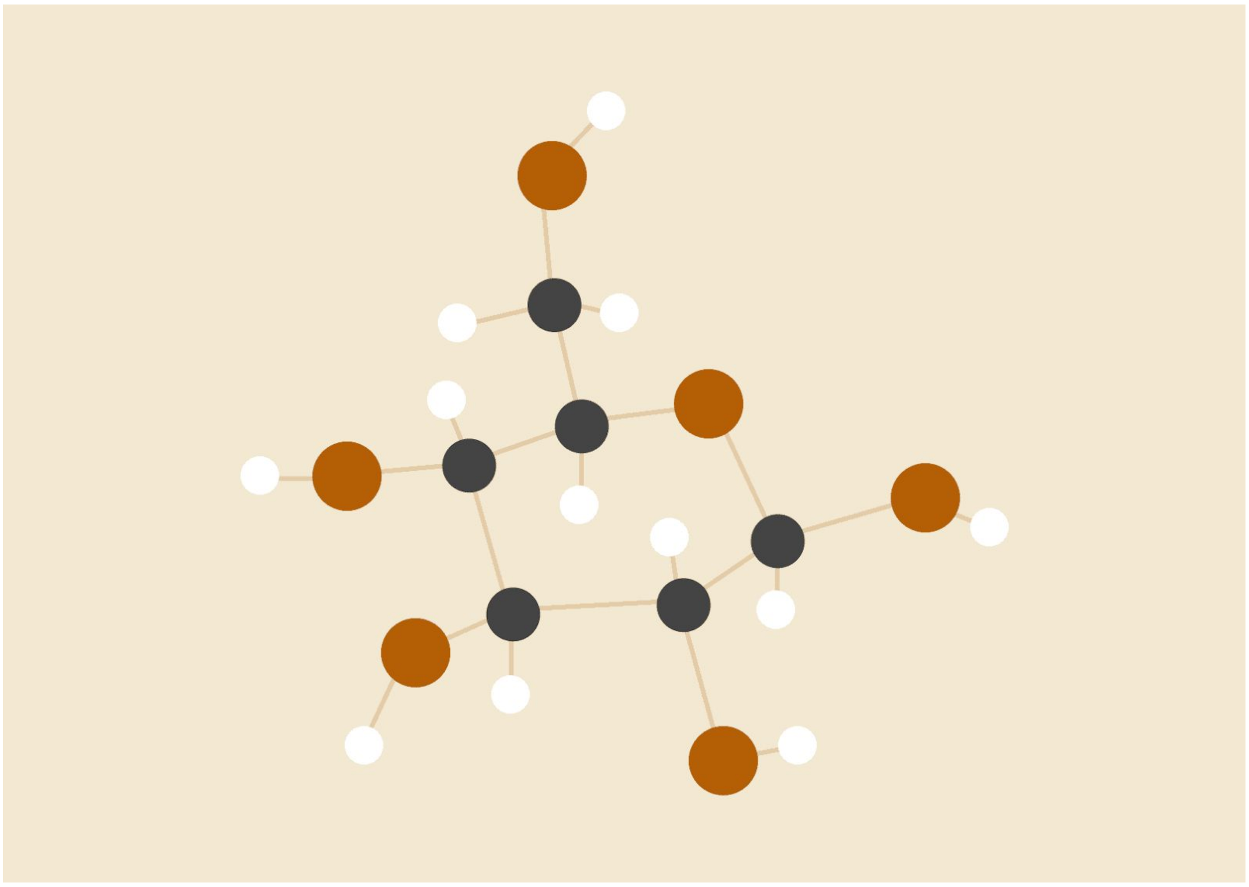
Lab Assignment 2



Shashank Kashyap

17114070

CSE

Problem Statement 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 consists of two programs which are 'server' and 'client'.

Server - The server goes the following process/stages to host one end of the connection:

- **Socket creation:** We first create the socket using domain (like AF_INET for IPv4 protocol or AF_INET6 for IPv6 protocol), communication stream type, and protocol value.
- **Setsockopt:** This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: "address already in use".
- **Bind:** After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to localhost, hence we use INADDR_ANY to specify the IP address.

Now after the socket is created, we define two methods which the server can perform:

- **Listen:** It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.
- **Accept:** It 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. At this point, the connection is established between client and server, and they are ready to transfer data.

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

```c
#include <netinet/in.h>
#include <string.h>
#include <time.h>

#define PORT 8080

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

       int sockfd, nSocket, valRead;
       struct sockaddr_in addr;
       int opt = 1;
       int addresslen = sizeof(addr);
       char buffer[1024] = {0};
       char *response = "Server responded!"; // Response message from server

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

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

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

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

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

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

       // Wait for client to make approach. Wait for 3.
      if (listen(sockfd, 3) < 0) {
          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 ((nSocket = accept(sockfd, (struct sockaddr *)&addr,
(socklen_t*)&addresslen))<0) {
          perror("accept");
          exit(EXIT_FAILURE);
      }

      valRead = read( nSocket , buffer, 1024);
      printf("%s\n",buffer );
      send(nSocket , response , strlen(response) , 0 );
      time_t seconds;
      seconds = time (NULL);
      printf("Response message sent at %ld \n", seconds/3600);
      return 0;
}
```

Client

- **Socket connection:** Exactly same as that of server's socket creation

Now after the socket is created, we define a method which the client can perform:

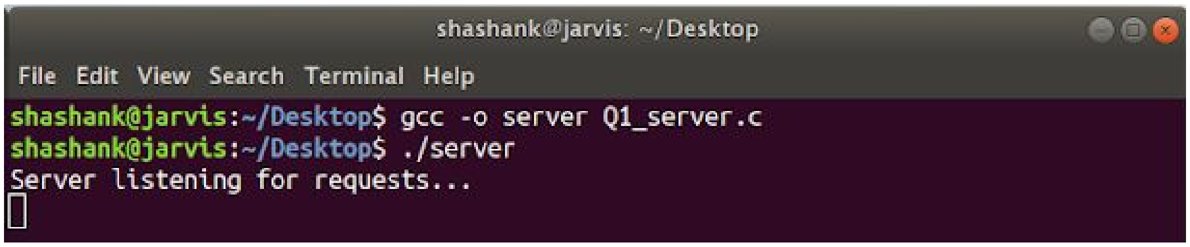
- **Connect:** The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

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

#define PORT 8080

int main(int argc, char const *argv[]) {
      int Socket = 0, valueread;
      struct sockaddr_in serv_addr;
      char *request = "Client requesting...";
      char buffer[1024] = {0};

      if ((Socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          printf("\n Socket creation error \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(Socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    time_t seconds;
    seconds = time (NULL);
    send(Socket , request , strlen(request) , 0 );
    printf("Request message sent at %ld \n", seconds/3600);
    valueread = read( Socket , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

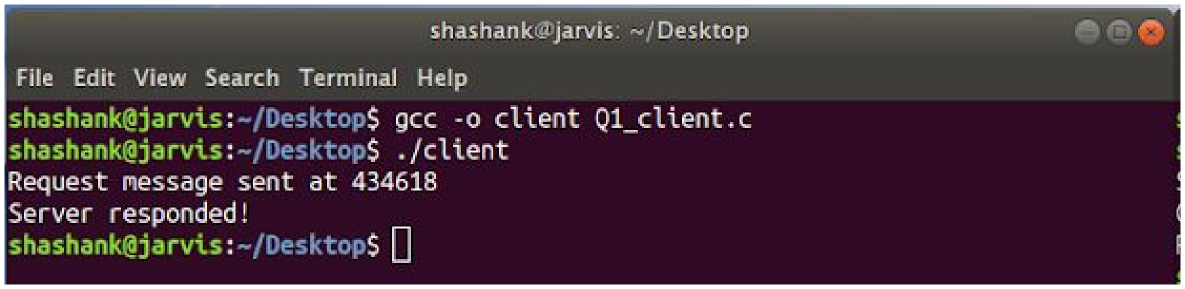
Execution:

- First we compile and run the server program. This sets up the server and the node starts listening for any calls.


shashank@jarvis: ~/Desktop
File Edit View Search Terminal Help

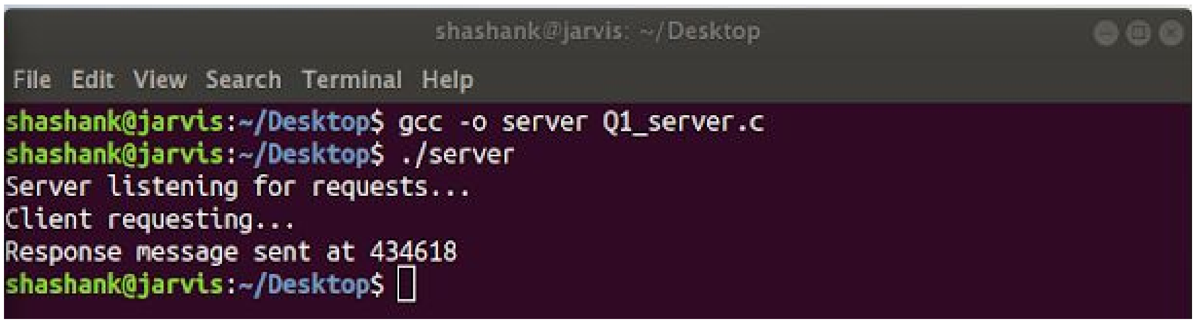
```
shashank@jarvis:~/Desktop$ gcc -o server Q1_server.c
shashank@jarvis:~/Desktop$ ./server
Server listening for requests...
```

- Then the client program is compiled and run on a new terminal. As soon as the program is executed, it sends a message to the server and gets a response from the server.


shashank@jarvis: ~/Desktop
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop$ gcc -o client Q1_client.c
shashank@jarvis:~/Desktop$ ./client
Request message sent at 434618
Server responded!
shashank@jarvis:~/Desktop$
```

- The server program responds to the client when the request is made.


shashank@jarvis: ~/Desktop
File Edit View Search Terminal Help

```
shashank@jarvis:~/Desktop$ gcc -o server Q1_server.c
shashank@jarvis:~/Desktop$ ./server
Server listening for requests...
Client requesting...
Response message sent at 434618
shashank@jarvis:~/Desktop$
```

Problem Statement 2

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

Solution:

The following program uses the following system calls to create a simulation of orphan and zombie processes:

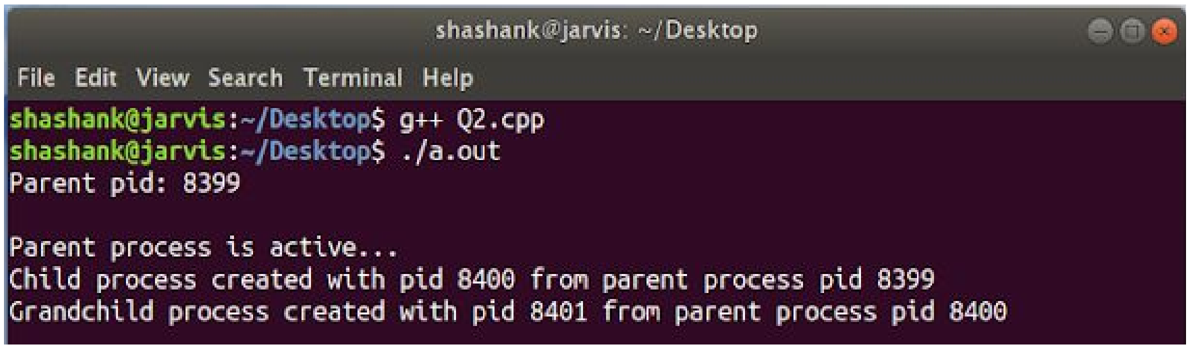
- fork()
- sleep()

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

using namespace std;

int main() {

    cout << "Parent process id: " << getpid() << endl << endl;
    pid_t Ch_pid = fork();

      if (Ch_pid > 0) {
       cout << "Parent active..." << endl;
       sleep(4);
       cout << "Parent terminated" << endl;
      }
      else if (Ch_pid == 0) {
       cout << "Child created with pid "<< getpid() << " from parent pid " << getppid()
<< endl;
       Ch_pid = fork();
       if(Ch_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(Ch_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;
}
```

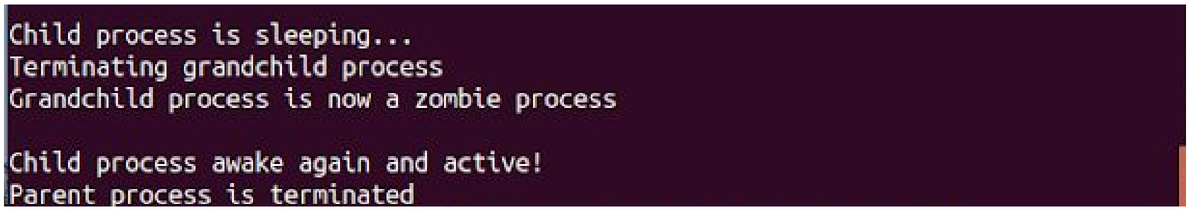
Execution:

- When the program is executed, a parent, a child and a grandchild are created.



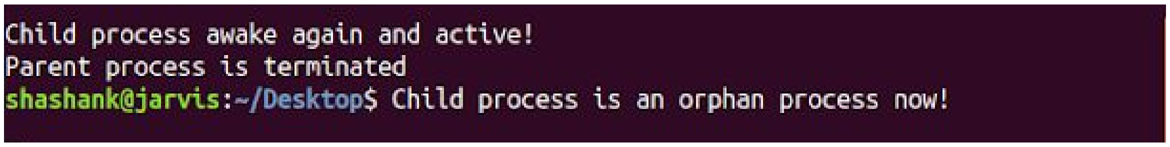
```
shashank@jarvis:~/Desktop$ g++ Q2.cpp
shashank@jarvis:~/Desktop$ ./a.out
Parent pid: 8399

Parent process is active...
Child process created with pid 8400 from parent process pid 8399
Grandchild process created with pid 8401 from parent process pid 8400
```

- Then the child process is put to sleep and due to this , the grandchild terminates. Now there is no grandchild process in the parent process, but it exists in the table of the child process as its child.



```
Child process is sleeping...
Terminating grandchild process
Grandchild process is now a zombie process

Child process awake again and active!
Parent process is terminated
```

- In some time, the parent process terminates, thereby bringing up the terminal again to accept commands, but the child process is still running as an orphan.



```
Child process awake again and active!
Parent process is terminated
shashank@jarvis:~/Desktop$ Child process is an orphan process now!
```