LAB ASSIGNMENT-2

CSN-361

Submitted by:
Ritik Kumar 17114063

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.

Server

Algorithm:

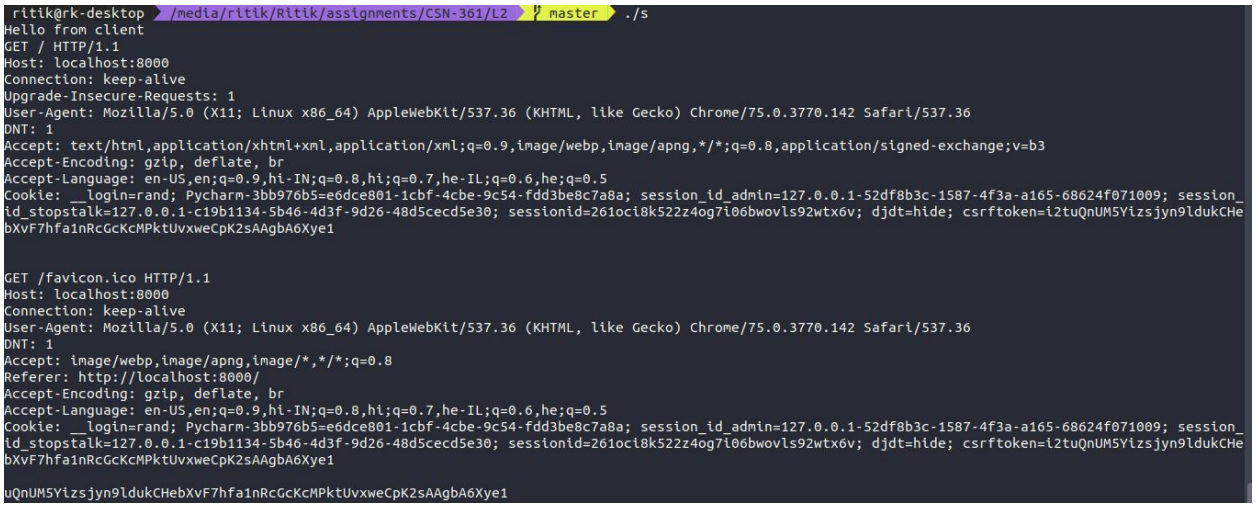
1. Get a socket file descriptor
2. Configure the socket with options
3. Bind the socket with the IP and port
4. Start listening to the socket
5. Accept and respond to each connection with a message

Data Structure:

- Socket: int
- Address: sockaddr_in
- Buffer: char array
- message: char pointer

```c
/** @file Q1_server.c
 *  @brief Socket program in C for server.
 *
 *  @author Ritik
 *
 * @date 7/31/2019
 */

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8000

char *server_message =
"HTTP/1.1 200 OK \n\
\n\
<html>\n\
<head>\n\
    <title>Title: CSN 361 - L2</title>\n\
<head>\n\
<body>\n\
    <center>\n\
        <h1>\n\
            Hello from server!\n\
        </h1>\n\
        <h3>\n\
            Submission for Lab assignment 2\n\
        </h3>\n\
    </center>\n\
</body>\n\
</html>";

/** @brief Q1 entrypoint for server.
 */
int main(int argc, char const *argv[]) {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port
    if (setsockopt(server_fd, 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;
    address.sin_port = htons(PORT);

    // Bind the socket to localhost
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 5) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    while(1){
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (
            socklen_t*)&addrlen))<0) {
```

Running code:



Client

Algorithm:

1. Get a socket file descriptor
2. Get the localhost address to binary
3. Establish a connection
4. Send the message
5. Get and print the response

Data Structure:

- Socket: int
- Address: sockaddr_in
- Buffer: char array
- message: char pointer

```c
/** @file Q1_client.c
 *  @brief Socket program in C for client.
 *
 *  @author Ritik
 *
 * @date 7/31/2019
 */

#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8000

/** @brief Q1 entrypoint for client.
 */
int main(int argc, char const *argv[])
{
    struct sockaddr_in address;
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    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 , hello , strlen(hello) , 0 );
    valread = read( sock , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

Working code:

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

Zombie process

Algorithm:

1. Fork the parent
2. The child exits while the parent is in sleep.

Data Structure:

- Int to store PIDs

```c
/** @file Q2_zombie.c
 *  @brief C program to demonstrate Zombie process.
 *  use 'ps aux' command to verify whether child process exist in process
 table.
 *
 *  @author Ritik
 *
 * @date 7/31/2019
 */

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

/** @brief Q2 entrypoint.
 */
int main(){
    printf("Parent's PID : %d.\n", getpid());
    int k = fork();
    if(k != 0){
        sleep(100);//parent went to sleep
        printf("Parent's done\n");
    }
    else
    {
        printf("Child's PID : %d.\n", getpid());
        printf("Child's done\n");
        exit(0); //child process terminated
    }
    return 0;
}
```

Working code:

```
ritik@rk-desktop /media/ritik/Ritik/assignments/CSN-361/L2  master ./qz
Parent's PID : 25576.
Child's PID : 25577.
Child's done

Parent's done
```

Orphan

Algorithm:

1. Fork the parent
2. The child sleeps while the parent exits.

Data Structure:

- Int to store PIDs

```c
/** @file Q2_orphan.c
 *  @brief C program to demonstrate Orphan process.
 *  use 'ps aux' command to verify whether parent process exist in process
 table.
 *
 *  @author Ritik
 *
 * @date 7/31/2019
 */

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

/** @brief Q2 entrypoint.
 */
int main(){
    printf("Parent's PID : %d.\n", getpid());
    int k = fork();
    if(k == 0){
        // Child process
        printf("Child created with PID : %d.\n", getpid());
        sleep(100);//child went to sleep
        // Orphan child process
        printf("Child's done\n");
        exit(0);
    }
    else
    {
        // Parent process
        printf("Parent's done\n");
        exit(0);
        //Parent process terminated
    }
    return 0;
}
```

Working code:

```
ritik@rk-desktop /media/ritik/Ritik/assignments/CSN-361/L2  master ./qo
Parent's PID : 27509.
Parent's done
Child created with PID : 27510.
```