Computer Networks Laboratory CSN361

# Lab Assignment 2

Ashutosh Chaubey (17114015)

# 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.

## Algorithm:

Server:-

- Create a new socket
- If socket() returns 0, then socket creation fails -> Abort
- Bind the socket with port 8080
- If port already in use -> Abort
- Ready the port to listen requests
- Send response after accepting connection from client

Client:-

- Create a new socket
- If socket() returns 0, then socket creation fails -> Abort
- Send request to the server
- Read server response

## Data Structures:

- int sockfd : creates a socket with IPv4 communication domain and TCP communication type.
- sockaddr_in : struct for all syscalls and functions that deal with internet addresses

## Code:

problem1_client.c | problem1_server.c | problem2.cpp

```c
/** @file problem1_client.c
 *  @brief Problem Statement 1 : Program in C to connect two nodes on a network
 *  @author Ashutosh Chaubey
 */
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

#define PORT 8080

/** @brief Problem Statement 1 entrypoint.
 */
int main(int argc, char const *argv[])
{
    int sock = 0, val_read;
    struct sockaddr_in serv_addr;
    char *request = "Client requesting...";
    char buffer[1024] = {0};

    if ((sock = 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(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");
    val_read = read( sock , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

problem1_client.c | problem1_server.c | problem2.cpp | problem2-2.cpp

```c
/** @file problem1_server.c
 *  @brief Problem Statement 1 : Program in C to connect two nodes on a network to communicate with each other, where one socket
 *  @author Ashutosh Chaubey
 */
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>

#define PORT 8080
/** @brief Problem Statement 1 entrypoint.
 */
int main(int argc, char const *argv[]) {

    int sockfd, new_socket, val_read;
    struct sockaddr_in address;
    int opt = 1;
    int len_addr = sizeof(address);
    char buffer[1024] = {0};
    char *response = "Response from the server!";   // Response message from server

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

    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);
    }

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

    // Bind the socket to port 8080 of local machine
    if (bind(sockfd, (struct sockaddr *)&address, sizeof(address))<0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

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

    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 ((new_socket = accept(sockfd, (struct sockaddr *)&address, (socklen_t*)&len_addr))<0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

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

## Output

```
djikstra@helios: ~
File  Edit  View  Search  Terminal  Help
(base) djikstra@helios:~$ cd Academic/CSN361/L2/
(base) djikstra@helios:~/Academic/CSN361/L2$ gcc problem1_client.c -o problem1_c
lient
(base) djikstra@helios:~/Academic/CSN361/L2$ ./problem1_client
Request message sent
Response from the server!
(base) djikstra@helios:~/Academic/CSN361/L2$
[ashu] 0:bash- 1:bash*                                  "helios" 02:42 01-Aug-19
```

```
djikstra@helios: ~
File  Edit  View  Search  Terminal  Help
(base) djikstra@helios:~$ cd Academic/CSN361/L2
(base) djikstra@helios:~/Academic/CSN361/L2$ gcc problem1_server.c -o problem1_s
erver
(base) djikstra@helios:~/Academic/CSN361/L2$ ./problem1_server
Server listening...
Client requesting...
Response message sent
(base) djikstra@helios:~/Academic/CSN361/L2$
[ashu] 0:bash* 1:bash-                                  "helios" 02:42 01-Aug-19
```

# Problem Statement 2

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

## Algorithm:

A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as reaping the zombie process.

Orphan processes are those processes that are still running even though their parent process has terminated or finished. A process can be orphaned intentionally or unintentionally.

## Data Structures:

- Int child

## Code:

problem1_client.c | problem1_server.c | problem2.c

```c
/** @file problem2.cpp
 *  @brief Problem Statement 2 :  C program to demonstrate both Zombie process and Orphan Process
 *  @author Ashutosh Chaubey
 */

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

/** @brief Problem Statement 2 (ii) entrypoint.
 */
int main() {
    int child = fork(); //create child process
    if (child > 0) //if child is non zero, then it is parent process
        printf("Inside Parent---- PID is : %d\n", getpid());
    else if (child == 0) { //for chile process child will be 0
        sleep(50); //wait for some times
        child = fork();
        if (child > 0) {
            printf("Inside Child---- PID :%d and PID of parent : %d\n", getpid(), getppid());
            while(1)
                sleep(1);
            printf("Inside Child---- PID of parent : %d\n", getppid());
        }else if (child == 0)
        printf("Inside grandchild process---- PID of parent : %d\n", getppid());
    }
    return 0;
}
```

## Output

```
(base) djikstra@helios:~/Academic/CSN361/L2$ gcc problem2.c -o problem2
(base) djikstra@helios:~/Academic/CSN361/L2$ ./problem2
Inside Parent---- PID is : 8255
(base) djikstra@helios:~/Academic/CSN361/L2$ Inside Child---- PID :8256 and PID of parent : 2292
Inside grandchild process---- PID of parent : 8256
```