

TP2

Programmation des sockets en C sous Linux

Serveur d'écho en mode connecté

```
/* client.c*/

#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
int main() {
    const char* server_name = "localhost";
    const int server_port = 8877;
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    // creates binary representation of server name
    // and stores it as sin_addr
    // http://beej.us/guide/bgnet/output/html/multipage/inet_ntopman.html
    inet_pton(AF_INET, server_name, &server_address.sin_addr);
    // htons: port in network order format
    server_address.sin_port = htons(server_port);
    // open a stream socket
    int sock;
    if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("could not create socket\n");
        return 1;
    }
    // TCP is connection oriented, a reliable connection
    // **must** be established before any data is exchanged
    if (connect(sock, (struct sockaddr*)&server_address,
        sizeof(server_address)) < 0) {
        printf("could not connect to server\n");
        return 1;
    }
    // send
    // data that will be sent to the server
    const char* data_to_send = "Gangadhar Hi Shaktimaan hai";
    send(sock, data_to_send, strlen(data_to_send), 0);
    // receive
    int n = 0;
    int len = 0, maxlen = 100;
    char buffer[maxlen];
```

```

char* pbuffer = buffer;
// will remain open until the server terminates the connection
while ((n = recv(sock, pbuffer, maxlen, 0)) > 0) {
    pbuffer += n;
    maxlen -= n;
    len += n;
    buffer[len] = '\0';
    printf("received: '%s'\n", buffer);
}
// close the socket
close(sock);
return 0;
}
}
/* server.c*/

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
/**
 * TCP Uses 2 types of sockets, the connection socket and the listen socket.
 * The Goal is to separate the connection phase from the data exchange phase.
 * */
int main(int argc, char *argv[]) {
    // port to start the server on
    int SERVER_PORT = 8877;
    // socket address used for the server
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    // htons: host to network short: transforms a value in host byte
    // ordering format to a short value in network byte ordering format
    server_address.sin_port = htons(SERVER_PORT);
    // htonl: host to network long: same as htons but to long
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    // create a TCP socket, creation returns -1 on failure
    int listen_sock;
    if ((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("could not create listen socket\n");
        return 1;
    }
    // bind it to listen to the incoming connections on the created server
    // address, will return -1 on error
    if ((bind(listen_sock, (struct sockaddr *)&server_address,

```

```

        sizeof(server_address))) < 0) {
    printf("could not bind socket\n");
    return 1;
}
int wait_size = 16; // maximum number of waiting clients, after which
                    // dropping begins
if (listen(listen_sock, wait_size) < 0) {
    printf("could not open socket for listening\n");
    return 1;
}
// socket address used to store client address
struct sockaddr_in client_address;
int client_address_len = 0;
// run indefinitely
while (true) {
    // open a new socket to transmit data per connection
    int sock;
    if ((sock =
        accept(listen_sock, (struct sockaddr *)&client_address,
               &client_address_len)) < 0) {
        printf("could not open a socket to accept data\n");
        return 1;
    }
    int n = 0;
    int len = 0, maxlen = 100;
    char buffer[maxlen];
    char *pbuffer = buffer;
    printf("client connected with ip address: %s\n",
           inet_ntoa(client_address.sin_addr));
    // keep running as long as the client keeps the connection open
    while ((n = recv(sock, pbuffer, maxlen, 0)) > 0) {
        pbuffer += n;
        maxlen -= n;
        len += n;
        printf("received: '%s'\n", buffer);
        // echo received content back
        send(sock, buffer, len, 0);
    }
    close(sock);
}
close(listen_sock);
return 0;
}

```