select_client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <netinet/in.h>  // For sockaddr_in
#include <string.h>      // For string handle function
#include <arpa/inet.h>   // For address conversion function
#include <unistd.h>
#include <fcntl.h>


// Used to print error messages
#define PANIC(msg) { perror(msg); exit(-1); }
#define STDIN 0


#define BUFSIZE 256 // Define the buffer size
#define DEFAULT_PORT 9999
#define FILE_BUFSIZE 1024

int main(int argc, char *argv[]){
    int sockfd;
    int retval;                 // The return value of select()
    int i;                      // Just for "for" loop
    int fdmax;
    int fd;
    int recvnum;
    int flag=0;
    char trans_buf[BUFSIZE];    // Put the data that you will send
    char recv_buf[BUFSIZE];     // Put the data that you received
    char filename[BUFSIZE];
    char file_buf[FILE_BUFSIZE];
    struct sockaddr_in dest;
    struct timeval tv;          // Set the timeout time
    fd_set read_fds;            // fd set

    // Generate an socketfd of IPv4 + TCP connection
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd == -1) PANIC("socket");

    // Empty the struct of server_addr
    memset(&dest, 0, sizeof(dest));
    dest.sin_family = AF_INET;                      // IPv4
    dest.sin_port = htons(DEFAULT_PORT);            // Set port number
    dest.sin_addr.s_addr = inet_addr("127.0.0.1");  // Set IP address
```

```c
puts("Login...");
puts("If you want to download files from server, "
  "you can key [require+filename]");

// Build the connection with Server, and we will be accepted
if(connect(sockfd, (struct sockaddr*)&dest, sizeof(dest))==-1)
    PANIC("connect")

fdmax=sockfd;

for(;;){
    FD_ZERO(&read_fds);         // Empty the read_fds
    FD_SET(STDIN,&read_fds);    // Put STDIN into read_fds
    FD_SET(sockfd,&read_fds);   // Put sockfd into read_fds

    retval = select(fdmax+1, &read_fds, NULL, NULL, NULL);
    switch(retval){
        case -1: // When retval=-1, it means that some errors occur
            perror("select");
            continue;

        case 0: // When retval=0, it means overtime
            printf("Time Out...\n");

            // Close all fd in master_fds,
            // but except "stdin" "stdout" "stderr".
            // Then exit the process
            for(i=3; i<= fdmax; i++)
                if (FD_ISSET(i,&read_fds))
                    close(i);
            exit(0);
    }

    // If you key a string in your terminal
    if(FD_ISSET(STDIN, &read_fds)){
        memset(trans_buf, 0, BUFSIZE); // Empty the buffer

        fgets(trans_buf, BUFSIZE, stdin); // Get a string from your terminal

        if(strstr(trans_buf, "require") != NULL){
            // Parse filename
            sscanf(trans_buf, "require %s\n", filename);
            printf("You require the [%s] file from Server\n", filename);
            flag = 1;
        }
```

```c
            // Send the data of trans_buf to the server
            send(sockfd, trans_buf, BUFSIZE, 0);
        }

        // If the server sends me a message
        if(FD_ISSET(sockfd, &read_fds)){
            memset(recv_buf, 0, BUFSIZE);   // Empty the buffer

            // Receive the message from server
            recvnum = recv(sockfd, recv_buf, sizeof(recv_buf), 0);
            if(recvnum == -1) perror("recv");

            printf("%s", recv_buf);

            if(flag == 1){
                // Create the file which I required.
                fd = open(filename, O_WRONLY|O_CREAT, 0666);
                switch(fd) {
                    case -1:
                        perror("open");
                        flag=0;
                        continue;
                    default:
                        // Save the data to the file
                        write(fd, recv_buf, recvnum);
                        printf("[%s] save successfully.\n", filename);
                        flag=0;
                        continue;
                }
            }
        }
    } // for(;;), Client loop.
    close(sockfd);
    return 0;
} // main()
```

select_server.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>  // select()
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <fcntl.h>      // File ctrl.
#include <unistd.h>     // select()
#include <pthread.h>


// Used to print error messages
#define PANIC(msg) { perror(msg); exit(-1); }
#define STDIN 0

#define BUFSIZE 256
#define DEFAULT_PORT 9999
#define FILE_BUFSIZE 1024

// Thread Function, used to handle file transmission
void* threadFunc(void *threadArgs);
char filename[BUFSIZE];


int main(int argc, char *argv[]){
    int server_fd;
    int client_fd;
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    fd_set master_fds;
    fd_set read_fds;
    int fdmax;                          // record the maximum of fd
    struct timeval tv;                  // Set the timeout time
    int port = DEFAULT_PORT;            // Port number
    int len;                            // sizeof(client_addr)
    int i, j;                           // For for loop.
    char buf[BUFSIZE];
    int nbytes;                         // Read the size of data
    int yes=1;                          // For setsockopt()
    int retval;                         // The return value of select()
    pthread_t threadid;
```

```
// SIGPIPE
  struct sigaction action;
  action.sa_handler=SIG_IGN;
  sigaction(SIGPIPE, &action, 0);

// Get port nums from args
if(argc > 1){
   int argsBuff = atoi(argv[1]);
   if(argsBuff <= 0 || argsBuff > 65535) perror("Args");
   else port = argsBuff;
}

  // Empty two fd sets
  FD_ZERO(&master_fds);
  FD_ZERO(&read_fds);

// Create a socket. use IPv4 + TCP connection.
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if(server_fd < 0) PANIC("Socket()");

  // Set IP Port to be reused
  if (setsockopt(server_fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1)
      PANIC("setsockopt");

  memset(&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family = AF_INET;           // IPv4
  server_addr.sin_addr.s_addr = INADDR_ANY;   // 0.0.0.0
  server_addr.sin_port = htons(port);         // Set port number

  // Bind srever_fd to a specific IP and port
  if (bind(server_fd, (struct sockaddr *)&server_addr,
       sizeof(server_addr)) == -1)
      PANIC("bind");

  // Monitor the number of connections at the same time, up to 10 connection
  if (listen(server_fd,10) == -1) PANIC("listen");

  FD_SET(server_fd, &master_fds); // Put server_fd into master_fds (fd set)
  FD_SET(STDIN, &master_fds);     // Put STDIN into master_fds (fd set)

  fdmax = server_fd; // Set the maximum of fd

  // Server Loop
  for(;;) {
      int *threadArgs = calloc(1, sizeof(int));
```

```
tv.tv_sec = 60; // Set the time out time to 60 seconds
tv.tv_usec = 0;

// Copy the fd in master_fds to read_fds
read_fds=master_fds;

// Use the select function to monitor multiple fds at the same time
retval = select(fdmax+1, &read_fds, NULL, NULL, &tv);
switch(retval){
    case -1:    // Some errors occur
        perror("select");
        continue;

    case 0:
        printf("Time Out...\n");

        // Close all fd in master_fds, but except "stdin" "stdout"
        //     "stderr". Then exit the process.
        for(i=3; i<= fdmax; i++)
            if (FD_ISSET(i,&master_fds))
                close(i);
        exit(0);
}

for(i = 0; i <= fdmax; i++) {
// When a fd get a I/O.
    if (FD_ISSET(i, &read_fds)) {

        // When "i" is equal to server_id, it means that there is a
  //     client want to connect.
        if (i == server_fd) {          // Handle New Connection
            len = sizeof(client_addr); // Get the length of client_addr
            // Accept the connection for client, and return a client_fd
            client_fd = accept(server_fd,
                 (struct sockaddr *) &client_addr,
                 (socklen_t*) &len);
     if (client_fd == -1) {
                perror("accept");
                continue;
            } else {
        // Add client_fd to the fd set
                FD_SET(client_fd, &master_fds);

                // Used to find the maximum of fd
                if (client_fd > fdmax) fdmax = client_fd;
                printf("New connection from %s on socket %d\n",
            inet_ntoa(client_addr.sin_addr), client_fd);
    }
} // if (i == server_fd)

// Handle data from the clients and the STDIN
else {
    // Read the data into buffer
    if ((nbytes = read(i, buf, sizeof(buf))) > 0) {
        // If client require file
        if(strstr(buf, "require") != NULL){
            // Parse the filename
            sscanf(buf, "require %s\n", filename);
            printf("Client %d require the [%s] file\n",
      i, filename);

            *threadArgs = i;

            //puts("1");

            // Cteate a thread
            if(pthread_create(&threadid, NULL, threadFunc,
             threadArgs) != 0){
                PANIC("pthread_create()");
            } else {
                // After finishing the thread function,
                // the resource will automatically freed.
                pthread_detach(threadid);
            }
        }

        // If client not require file, just send message to me.
        else {
  // Write the data of buffer to your console
            write(0, buf, nbytes);

            for(j = 0; j <= fdmax; j++) {
                if (FD_ISSET(j, &master_fds))
                    // In addition to me, STDIN, and the client
      //     which sended date, we should send a
                    //     copy to the other clients.
                    if (j!=server_fd && j!=i &&j!=0)
                        if (send(j, buf, nbytes, 0) == -1)
                            perror("send");
            }
        } // if(strstr(buf, "require") != NULL)
    } else {
        perror("read");
        close(i);
```

```
                        FD_CLR(i, &master_fds); // Clean "i" from master_fds
                    }
                } // if (i == server_fd)
            } // if (FD_ISSET(i, &read_fds))
        } // for(),  scan all fd
    } // for(;;), Server loop
    return 0;
} // main()


// Used to handle file transmission
void* threadFunc(void *threadArgs) {
    int fd;
    int recvnum;
    char file_buf[FILE_BUFSIZE];
    int clientfd = *(int*) threadArgs;
    free(threadArgs);

    // Open the file which client require
    fd=open(filename, O_RDONLY, 0666);
    switch(fd){
        case -1:
            perror("open");
            write(clientfd, "File not exists", sizeof("File not exists"));

        default: // File exists and Open successfully.
            printf("%s exists.\n", filename);

            // Read the file data
            recvnum = read(fd, file_buf, sizeof(file_buf));
            if(recvnum == -1) perror("recv");

            // Send file date to client
            write(clientfd, file_buf, recvnum);
    }
} // void* threadFunc(void *threadArgs)
```