```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>

#define SERVER_PORT 10101
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
        struct sockaddr_in sin;
        char buf[MAX_LINE];
        int len;
        int s, new_s;
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(SERVER_PORT);

        struct sockaddr_in so;
        int so_len;
        char so_addr[INET_ADDRSTRLEN];

        pid_t child_pid;

        /* setup passive open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex−talk: socket");
                exit(1);
        }

if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
                perror("simplex−talk: bind");
                exit(1);
        }
        listen(s, MAX_PENDING);
        /* expera pelas conexÃµes e imprime o que receber dos clientes */
        while(1) {
                /* aceita a conexÃ£o de um cliente por um socket novo new_s */
                if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
                        perror("simplex−talk: accept");
                        exit(EXIT_FAILURE);
                }
                /* faz o fork do processo */
                child_pid = fork();
                if (child_pid < 0) {
                        perror("simplex−talk: fork");
                        exit(EXIT_FAILURE);
                }
                /* caso o processo seja o filho */
                if (child_pid == 0) {
                        /* fecha o socket que estÃ¡ esperando por novos clientes
 */
                        close(s);
                        /* coleta as informaÃ§Ãµes do socket e imprime na stdout
 */
                        so_len = sizeof(so);
                        if (getpeername(new_s, (struct sockaddr *)&so, &so_len)
< 0) {
                                perror("simplex−talk: getpeername");
                                close(s);
```

```
                                exit(EXIT_FAILURE);
                        }
                        inet_ntop(AF_INET, &(so.sin_addr), so_addr, INET_ADDRSTR
LEN);
                        printf("IP address: %s; Port number: %d\n", so_addr, ntohs(so.sin
_port));
                        /* entra em loop imprimindo as mensagens do cliente */
                        while (len = recv(new_s, buf, sizeof(buf), 0)) {
                                fputs(buf, stdout);
                        }
                        close(new_s);
                }
                /* caso o processo seja o pai, fecha o novo socket */
                close(new_s);
        }
        close(s);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <arpa/inet.h>

#define SERVER_PORT 10101
#define MAX_LINE 256

int main(int argc, char * argv[])
{
        FILE *fp;
        struct hostent *hp;
        struct sockaddr_in sin;
        char *host;
        char buf[MAX_LINE];
        int s;
        int len;

        struct sockaddr_in so;
        int so_len;
        char so_addr[INET_ADDRSTRLEN];

        if (argc==2) {
                host = argv[1];
        }
        else {
                fprintf(stderr, "usage: ./client host\n");
        exit(1);
}

/* translate host name into peerâM−^@M−^Ys IP address */
        hp = gethostbyname(host);
        if (!hp) {
                fprintf(stderr, "simplex−talk: unknown host: %s\n", host);
                exit(EXIT_FAILURE);
        }
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        bcopy(hp−>h_addr, (char *)&sin.sin_addr, hp−>h_length);
        sin.sin_port = htons(SERVER_PORT);
        /* active open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex−talk: socket");
                exit(EXIT_FAILURE);
        }
        if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
                perror("simplex−talk: connect");
                close(s);
                exit(EXIT_FAILURE);
        }
        /* get socket information and prints it on the stdout */
        so_len = sizeof(so);
        if (getsockname(s, (struct sockaddr *)&so, &so_len) < 0) {
                perror("simplex−talk: getsockname");
                close(s);
                exit(EXIT_FAILURE);
        }
        inet_ntop(AF_INET, &(so.sin_addr), so_addr, INET_ADDRSTRLEN);
        printf("IP address: %s; Port number: %d\n", so_addr, ntohs(so.sin_port));
        /* main loop: get and send lines of text */
        while (fgets(buf, sizeof(buf), stdin)) {
                if (strcmp(buf, "exit\n") == 0) {
```

```c
                        break;
                }
                buf[MAX_LINE−1] = ’\0’;
                len = strlen(buf) + 1;
                send(s, buf, len, 0);
        }
        close(s);
}
```