```c
/*
*   This program is free software: you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*    the Free Software Foundation, either version 3 of the License, or
*    (at your option) any later version.
*
*    This program is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*    GNU General Public License for more details.
*
*    You should have received a copy of the GNU General Public License
*    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <time.h>
#include <fcntl.h>

#define BUF_LEN 256
#define PORT 9000

static void error_hndlr(const char *get) {
    fputs(strerror(errno), stderr);
    fputs(": ", stderr);
    fputs(get, stderr);
    fputs("\n", stderr);

    exit(EXIT_FAILURE);
}

void timestamp(char *buf) {
    time_t td;
    char time_buf[128];
    int n;

    time(&td);
    n = (int)strftime(time_buf, sizeof time_buf, "%H %M %S", localtime(&td
));
    // error handling

    memcpy(buf, time_buf, sizeof time_buf);
}


int main() {

    int z;  // temp return value
    int reuse = 1; // optval true
    int fd; // fd for iteration
    int i; // temp variable for loop
    int flags; // flags for fctnl
    struct sockaddr_in srvr_addr, clnt_addr;
    fd_set master_fds, other_fds;
    int max_fd;
    int server_socket, client_socket;
    size_t server_len, client_len;
    char buf[BUF_LEN];
    char msg[BUF_LEN];
    char time[BUF_LEN];
```

```
67
68      memset(&buf, 0, sizeof buf); // zero out
69      memset(&msg, 0, sizeof msg);
70      memset(&time, 0, sizeof time);
71
72      // create server socket
73      server_socket = socket(AF_INET, SOCK_STREAM, 0);
74      if (server_socket == -1)
75          error_hndlr("Could not open socket()");
76
77      // enable address reuse
78      z = setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof
    reuse);
79      if (z == -1)
80          error_hndlr("Could not reuse address with setsockopt()");
81
82      memset(&srvr_addr, 0, sizeof srvr_addr);
83      srvr_addr.sin_family = AF_INET;
84      srvr_addr.sin_addr.s_addr = INADDR_ANY;
85      srvr_addr.sin_port = htons(PORT);
86      server_len = sizeof srvr_addr;
87
88      // bind addr to socket
89      z = bind(server_socket, (struct sockaddr*)&srvr_addr, server_len);
90      if (z == -1)
91          error_hndlr("Could not bind()");
92
93      // listen
94      z = listen(server_socket, 10);
95      if(z == -1)
96          error_hndlr("Could not listen()");
97
98      // saves flags before manipulating them, => should create a
    set_nonblock()
99      if ((flags = fcntl(server_socket, F_GETFL, 0)) < 0)
100         error_hndlr("Could not save fcntl() flags");
101
102     if (fcntl(server_socket, F_SETFL, flags | O_NONBLOCK) < 0)
103         error_hndlr("Could not set fcntl() NONBLOCK flag");
104
105     // initialize and set the file descriptors
106     FD_ZERO(&master_fds);
107     FD_ZERO(&other_fds);
108
109     FD_SET(server_socket, &master_fds);
110
111     max_fd = server_socket;
112
113     for(;;) {
114         // copy master_fds to other_fds
115         memcpy(&other_fds, &master_fds, sizeof master_fds);
116
117         // Synchronous I/O Multiplexing, monitoring a set of fds
118         z = select(max_fd+1, &other_fds, 0,0,0);
119         if (z == 0)
120             error_hndlr("select() timeout");
121         else if (z == -1)
122             error_hndlr("select() failed");
123
124         for(fd = 0; fd <= max_fd; fd++) {
125             // if the given fd is part of other set
126             if( FD_ISSET(fd, &other_fds) ) {
127             // if fd is server -> accept
128                 if(fd == server_socket) {
129                     client_len = sizeof clnt_addr;
130                     client_socket = accept(server_socket, (struct sockaddr
    *)&clnt_addr, &client_len);
```

```
131                     if (client_socket == -1)
132                         error_hndlr("Could not accept() client socket");
133 
134                     // saves flags before manipulating them
135                     if ((flags = fcntl(client_socket, F_GETFL, 0)) < 0)
136                         error_hndlr("Could not save fcntl() flags");
137 
138                     if (fcntl(client_socket, F_SETFL, flags | O_NONBLOCK)
    < 0)
139                         error_hndlr("Could not set fcntl() NONBLOCK flag");
140 
141                     // add to the master file descriptors set
142                     FD_SET(client_socket, &master_fds);
143                     // if the client's fd is higher than the highest fd
144                     if(client_socket > max_fd)
145                         max_fd = client_socket;
146 
147 
148                 } else { // if fd is not the server socket
149                     z = recv(fd, buf, sizeof buf, 0); // could use read
150                     if (z <= 0) { // if nothing's been read
151                         close(fd);
152                         FD_CLR(fd, &master_fds);
153                         //error_hndlr("recv() nothing");
154                         printf("recv() nothing, somebody's quit \n");
155                     }
156 
157                     timestamp(time);
158                     // copy timestamp and recv buffer inmsg
159                     snprintf(msg, sizeof msg, "[%s] %s", time, buf);
160                     // sends message to everybody except to server
161                     for(i = 0; i <= max_fd; i++) {
162                         if(FD_ISSET(i, &master_fds)) {
163                             if(i != server_socket) {
164                                 z = send(i, msg, sizeof msg, 0); // could
    use write
165                                 if (z == -1)
166                                     error_hndlr("could not send() message");
167                             }
168                         }
169                     }
170                 }
171             }
172         }
173     }
174 
175 
176     close(server_socket);
177 
178     return 0;
179 }
180 
```