

```
server.c      *      daytimeserver.c      *      concurrentechoserver.c      *      client.c      *      client4.c      *
132     if(FD_ISSET(fileno(fp), &rset)){
133         fgets(sendbuffer, MAXLINE, fp);
134         if(strcmp(sendbuffer, "EXIT\n") == 0){
135             for(i=0; i<backlog; i++){
136                 if(friend[i] >= 0){
137                     close(friend[i]);
138                 }
139             }
140             exit(0);
141         }
142         strcpy(receiver, strtok(sendbuffer, " "));
143         if(receiver[strlen(receiver) - 1] == '\n') receiver[strlen(receiver) - 1] = '\0';
144         ;
145         strcpy(sendMessage, sendbuffer + strlen(receiver) + 1);
146         if(sendMessage[strlen(sendMessage) - 1] == '\n') sendMessage[strlen(sendMessage) - 1] = '\0';
147
148         if(strcmp(receiver, "ALL") == 0){
149             for(i=0; i<backlog; i++){
150                 if(friend[i] >= 0){
151                     sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
152                     write(friend[i], message, sizeof(message));
153                 }
154             }
155         }else{
156             for(i=0; i<backlog; i++){
157                 if(friend[i] >= 0 && strcmp(friendNAME[i], receiver) == 0){
158                     sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
159                     write(friend[i], message, sizeof(message));
160                     break;
161                 }
162             }
163             if(i >= backlog){
164                 sprintf(message, "No user\n");
165                 fputs(message, stdout);
166             }
167         }
168     //user input
169
170     for(i=0; i<=maxFriend; i++){
171         //printf("some has thing to say\n");
172         if(friend[i] < 0) continue;
173         if(FD_ISSET(friend[i], &rset)){
174             ssize_t len;
175             len = read(friend[i], receivebuffer, MAXLINE);
176             if(len == 0){
177                 printf("%s has left the chat room\n", friendNAME[i]);
178                 close(friend[i]);
179                 FD_CLR(friend[i], &allset);
180                 friend[i] = -1;
181                 free(friendNAME[i]);
182                 continue;
183             }
184             fputs(receivebuffer, stdout);
185             numReady--;
186             if(numReady <= 0)break;
187         }
188     }
189 }
190 }
191 }
```

```
server.c          daytimeserver.c      concurrentechoserver.c    client1.c      client4.c
114     read(friendfd, &receivebuffer, MAXLINE);
115 //printf("receive name %s\n", receivebuffer);
116 friendNAME[i] = malloc(MAXLINE * sizeof(char));
117 strcpy(friendNAME[i], receivebuffer);
118
119 FD_SET(friendfd, &allset);
120 if(friendfd > maxfd) maxfd = friendfd;
121 if(i > maxFriend) maxFriend = i;
122
123 //printf("friend number is %d\n", maxFriend);
124 //printf("friend max fd is %d\n", maxfd);
125
126 numReady--;
127 if(numReady <= 0) continue;
128 }
129 //save friend info
130
131 if(FD_ISSET(fileno(fp), &rset)){
132     fgets(sendbuffer, MAXLINE, fp);
133     if(strcmp(sendbuffer, "EXIT\n") == 0){
134         for(i=0; i<backlog; i++){
135             if(friend[i] >= 0){
136                 close(friend[i]);
137             }
138         }
139         exit(0);
140     }
141     strcpy(receiver, strtok(sendbuffer, " "));
142     if(receiver[strlen(receiver) - 1] == '\n') receiver[strlen(receiver) - 1] = '\0';
143     ;
144     strcpy(sendMessage, sendbuffer + strlen(receiver) + 1);
145     if(sendMessage[strlen(sendMessage) - 1] == '\n') sendMessage[strlen(sendMessage) - 1] = '\0';
146
147     if(strcmp(receiver, "ALL") == 0){
148         for(i=0; i<backlog; i++){
149             if(friend[i] >= 0){
150                 sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
151                 write(friend[i], message, sizeof(message));
152             }
153         }
154     }else{
155         for(i=0; i<backlog; i++){
156             if(friend[i] >= 0 && strcmp(friendNAME[i], receiver) == 0){
157                 sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
158                 write(friend[i], message, sizeof(message));
159                 break;
160             }
161         }
162         if(i >= backlog){
163             sprintf(message, "No user\n");
164             fputs(message, stdout);
165         }
166     }
167 }
168 //user input
169
170 for(i=0; i<=maxFriend; i++){
171     //printf("some has thing to say\n");
172     if(friend[i] < 0) continue;
173     if(FD_ISSET(friend[i], &rset)){
174         ssize_t len;
175         len = read(friend[i], receivebuffer, MAXLINE);
176         if(len == 0){
177             printf("%s has left the chat room\n", friendNAME[i]);
```

```
server.c      x   daytimeserver.c      x   concurrentechoserver.c  x   client1.c      x   client4.c      x
57 //printf("try to connect port %d\n", friendPORT);
58 friendfd = socket(AF_INET, SOCK_STREAM, 0);
59
60 bzero(&friendAddr, sizeof(friendAddr));
61 friendAddr.sin_family = AF_INET;
62 friendAddr.sin_port = htons(friendPORT);
63 inet_pton(AF_INET, friendIP, &friendAddr.sin_addr);
64
65 if(connect(friendfd, (struct sockaddr *)&friendAddr, sizeof(friendAddr)) < 0){
66     printf("Connect error\n");
67     exit(1);
68 }
69 for(j=0; j<backlog; j++){
70     if(friend[j] < 0){
71         friend[j] = friendfd;
72         break;
73     }
74 }
75
76 //printf("connect success\n");
77 write(friendfd, myNAME, sizeof(myNAME));
78 read(friendfd, &receivebuffer, MAXLINE);
79 //printf("port %d's name is %s\n", friendPORT, receivebuffer);
80 friendNAME[j] = malloc(MAXLINE * sizeof(char));
81 strcpy(friendNAME[j], receivebuffer);
82 //printf("copy name done\n");
83 if(friendfd > maxfd) maxfd = friendfd;
84 if(j > maxFriend) maxFriend = j;
85 FD_SET(friendfd, &allset);
86
87 //printf("connection done\n");
88 }
89 // connect to everybody
90
91 //printf("friend number is %d\n", maxFriend);
92 //printf("friend max fd is %d\n", maxfd);
93
94 for();{
95     rset = allset;
96     numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
97     //printf("select done once\n");
98
99     if(FD_ISSET(mysockfd, &rset)){
100        //printf("someone try to connect\n");
101        friendfd = accept(mysockfd, (struct sockaddr *)NULL, NULL);
102        write(friendfd, myNAME, sizeof(myNAME));
103
104        for(i=0; i<backlog; i++){
105            if(friend[i] < 0){
106                friend[i] = friendfd;
107                break;
108            }
109        }
110        if(i == backlog){
111            printf("Too many friends\n");
112            exit(1);
113        }
114
115        read(friendfd, &receivebuffer, MAXLINE);
116        //printf("receive name %s\n", receivebuffer);
117        friendNAME[i] = malloc(MAXLINE * sizeof(char));
118        strcpy(friendNAME[i], receivebuffer);
119
120        FD_SET(friendfd, &allset);
121        if(friendfd > maxfd) maxfd = friendfd;
122        if(i > maxFriend) maxFriend = i;
123    }
```

```
server.c      x   daytimeServer.c      x   concurrentechoserver.c      x   client1.c      x   client4.c      x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <sys/select.h>
8 #include <unistd.h>
9
10#define MAXLINE 1024
11#define backlog 10
12
13int main(int argc, char *argv[]){
14    int myPORT, friendPORT;
15    char *myIP, *friendIP, *myNAME;
16    char sendbuffer[MAXLINE], receivebuffer[MAXLINE], receiver[MAXLINE], sendMessage[1024], message[MAXLINE];
17    char *friendNAME[backlog];
18    struct sockaddr_in myAddr, friendAddr;
19    int mysockfd, maxfd, maxFriend, numReady, friendfd;
20    int friend[backlog];
21    fd_set allset, rset;
22    FILE *fp = stdin;
23    int i,j;
24
25    myIP = "127.0.0.1";
26    myPORT = atoi(argv[2]);
27    myNAME = argv[1];
28    mysockfd = socket(AF_INET, SOCK_STREAM, 0);
29
30    bzero(&myAddr, sizeof(myAddr));
31    myAddr.sin_family = AF_INET;
32    myAddr.sin_port = htons(myPORT);
33    inet_pton(AF_INET, myIP, &myAddr.sin_addr);
34    // my information
35
36    if(bind(mysockfd, (struct sockaddr *)&myAddr, sizeof(myAddr)) < 0){
37        printf("bind error\n");
38        exit(1);
39    }
40    // bind my infor to socket
41
42    listen(mysockfd, backlog);
43    maxfd = mysockfd;
44    maxFriend = -1;
45    for(i=0; i<backlog; i++){
46        friend[i] = -1;
47    }
48
49    FD_ZERO(&allset);
50    FD_SET(mysockfd, &allset);
51    FD_SET(fileno(fp), &allset);
52    if(fileno(fp) > maxfd) maxfd = fileno(fp);
53
54    for(i=3; i<argc; i++){
55        friendPORT = atoi(argv[i]);
56        friendIP = "127.0.0.1";
57        //printf("try to connect port %d\n", friendPORT);
58        friendfd = socket(AF_INET, SOCK_STREAM, 0);
59
60        bzero(&friendAddr, sizeof(friendAddr));
61        friendAddr.sin_family = AF_INET;
62        friendAddr.sin_port = htons(friendPORT);
63        inet_pton(AF_INET, friendIP, &friendAddr.sin_addr);
64    }
}
```

```
106 //printf("%s\n", cal);
107
108     while(1){
109         char temp[MAXLINE];
110         ptr = strtok(NULL, " ");
111         if(ptr == NULL)break;
112         strcpy(temp, ptr);
113         //printf("%s\n", temp);
114         arg[i] = atoi(temp);
115         //printf("%d\n", arg[i]);
116         i++;
117     }
118     if(strcmp(cal, "ADD") == 0){
119         ans = 0;
120         for(int j=0; j<i; j++){
121             ans += arg[j];
122         }
123         if(ans > UINT_MAX){
124             strcpy(send, "Overflowed\n");
125             write(client[c], send, sizeof(send));
126             continue;
127         }
128         sprintf(send, "%lu\n", ans);
129         write(client[c], send, sizeof(send));
130     }else if(strcmp(cal, "MUL") == 0){
131         ans = 1;
132         for(int j=0; j<i; j++){
133             ans = ans * arg[j];
134         }
135         if(ans > UINT_MAX){
136             strcpy(send, "Overflowed\n");
137             write(client[c], send, sizeof(send));
138             continue;
139         }
140         sprintf(send, "%lu\n", ans);
141         write(client[c], send, sizeof(send));
142     }else if(strcmp(cal, "EXIT") == 0){
143         close(client[c]);
144         FD_CLR(client[c], &allset);
145         client[c] = -1;
146         break;
147     }else{
148         sprintf(send, "wrong command\n");
149         write(client[c], send, sizeof(send));
150     }
151 }
152 numReady--;
153 if(numReady <= 0)break;
154 }
155 }
156 }
157 }
158 }
```



```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8 #include <limits.h>
9 #include <sys/select.h>
10
11 #define MAXLINE 1024
12
13 int main(int argc, char *argv[]){
14     int listenfd, connfd;
15     struct sockaddr_in server;
16     char buffer[MAXLINE];
17     char cal[MAXLINE];
18     int arg[10];
19     unsigned long int ans;
20     int serverPORT;
21     char send[MAXLINE];
22     int client[10];
23     int maxfd, maxClient;
24     fd_set allset, rset;
25     int numReady;
26     int c;
27
28     if(argc != 2){
29         printf("wrong input\n");
30         exit(1);
31     }
32
33     serverPORT = atoi(argv[1]);
34
35     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
36         printf("socket error\n");
37         exit(1);
38     }
39
40     bzero(&server, sizeof(server));
41     server.sin_family = AF_INET;
42     server.sin_port = htons(serverPORT);
43     server.sin_addr.s_addr = htonl(INADDR_ANY);
44
45     if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
46         printf("bind error\n");
47         exit(1);
48     }
49
50     listen(listenfd, 10);
51
52     maxfd = listenfd;
53     maxClient = -1;
54
55     for(c=0; c<10; c++){
56         client[c] = -1;
57     }
58
59     FD_ZERO(&allset);
60     FD_SET(listenfd, &allset);
61
62     for(;;){
63         rset = allset;
64         numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
65     }
}
```

```
client.c      x    main.cpp      x    daytimeclient.c      x    server1.c      x    server2.c      x    server3.c      x
73 //printf("read done\n");
74 char *ptr = strtok(buffer, " ");
75 int i = 0;
76 strcpy(cal, ptr);
77 //printf("%s\n", cal);
78
79 while(1){
80     char temp[MAXLINE];
81     ptr = strtok(NULL, " ");
82     if(ptr == NULL)break;
83     strcpy(temp, ptr);
84     //printf("%s\n", temp);
85     arg[i] = atoi(temp);
86     //printf("%d\n", arg[i]);
87     i++;
88 }
89
90 //printf("%s\n", cal);
91 //for(int k=0; k<i; k++){
92 //    printf("%d ",arg[k]);
93 //}
94
95 if(strcmp(cal, "ADD") == 0){
96     ans = 0;
97     for(int j=0; j<i; j++){
98         ans += arg[j];
99     }
100    if(ans > UINT_MAX){
101        strcpy(send, "Overflowed\n");
102        write(connfd, send, sizeof(send));
103        continue;
104    }
105    sprintf(send, "%lu\n", ans);
106    write(connfd, send, sizeof(send));
107 }else if(strcmp(cal, "MUL") == 0){
108     ans = 1;
109     for(int j=0; j<i; j++){
110         ans = ans * arg[j];
111     }
112     if(ans > UINT_MAX){
113         strcpy(send, "Overflowed\n");
114         write(connfd, send, sizeof(send));
115         continue;
116     }
117     sprintf(send, "%lu\n", ans);
118     write(connfd, send, sizeof(send));
119 }else if(strcmp(cal, "EXIT") == 0){
120     close(connfd);
121     break;
122 }else{
123     sprintf(send, "wrong command\n");
124     write(connfd, send, sizeof(send));
125 }
126 }
127 printf("Client has closed the connection\n");
128 exit(0);
129 }else{
130     close(connfd);
131 }
132 }
133 }
134 }
```

```
client.c    x  main.cpp  x  daytimeclient.c  x  server1.c  x  server2.c  x  server3.c  x
52     printf("bind error\n");
53     exit(1);
54 }
55
56 listen(listenfd, 10);
57 signal(SIGCHLD, sig_fork);
58
59 for(;){
60     connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
61     if((pid = fork()) == 0){
62         close(listenfd);
63         for(;{
64             ssize_t len;
65             len = read(connfd, &buffer, MAXLINE);
66             if(len == 0){
67                 break;
68             }
69
70             if(buffer[strlen(buffer) - 1] == '\n'){
71                 buffer[strlen(buffer) - 1] = '\0';
72             }
73             //printf("read done\n");
74             char *ptr = strtok(buffer, " ");
75             int i = 0;
76             strcpy(cal, ptr);
77             //printf("%s\n", cal);
78
79             while(1){
80                 char temp[MAXLINE];
81                 ptr = strtok(NULL, " ");
82                 if(ptr == NULL)break;
83                 strcpy(temp, ptr);
84                 //printf("%s\n", temp);
85                 arg[i] = atoi(temp);
86                 //printf("%d\n", arg[i]);
87                 i++;
88             }
89
90             //printf("%s\n", cal);
91             //for(int k=0; k<i; k++){
92             //    printf("%d ",arg[k]);
93             //}
94
95             if(strcmp(cal, "ADD") == 0){
96                 ans = 0;
97                 for(int j=0; j<i; j++){
98                     ans += arg[j];
99                 }
100                if(ans > UINT_MAX){
101                    strcpy(send, "Overflowed\n");
102                    write(connfd, send, sizeof(send));
103                    continue;
104                }
105                sprintf(send, "%lu\n", ans);
106                write(connfd, send, sizeof(send));
107            }else if(strcmp(cal, "MUL") == 0){
108                ans = 1;
109                for(int j=0; j<i; j++){
110                    ans = ans * arg[j];
111                }
112                if(ans > UINT_MAX){
113                    strcpy(send, "Overflowed\n");
114                    write(connfd, send, sizeof(send));
115                    continue;
116                }
}
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8 #include <limits.h>
9 #include <signal.h>
10 #include <sys/wait.h>
11
12 #define MAXLINE 1024
13
14 void sig_fork(int signo){
15     pid_t pid;
16     int status;
17     while((pid = waitpid(-1, &status, WNOHANG)) > 0){
18         printf("Child server process PID=%d has terminated\n", pid);
19     }
20     return;
21 }
22
23 int main(int argc, char *argv[]){
24     int listenfd, connfd;
25     struct sockaddr_in server;
26     char buffer[MAXLINE];
27     char cal[MAXLINE];
28     int arg[10];
29     unsigned long int ans;
30     int serverPORT;
31     char send[MAXLINE];
32     pid_t pid;
33
34     if(argc != 2){
35         printf("wrong input\n");
36         exit(1);
37     }
38
39     serverPORT = atoi(argv[1]);
40
41     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
42         printf("socket error\n");
43         exit(1);
44     }
45
46     bzero(&server, sizeof(server));
47     server.sin_family = AF_INET;
48     server.sin_port = htons(serverPORT);
49     server.sin_addr.s_addr = htonl(INADDR_ANY);
50
51     if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
52         printf("bind error\n");
53         exit(1);
54     }
55
56     listen(listenfd, 10);
57     signal(SIGCHLD, sig_fork);
58
59     for(;){
60         connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
61         if((pid = fork()) == 0){
62             close(listenfd);
63             for(;){
64                 ssize_t len;
```

```
client.c    x  main.cpp  x  daytimeclient.c  x  server1.c  x  server2.c  x  server3.c  x
58     //printf("read done\n");
59     char *ptr = strtok(buffer, " ");
60     int i = 0;
61     strcpy(cal, ptr);
62     //printf("%s\n", cal);
63
64     while(1){
65         char temp[MAXLINE];
66         ptr = strtok(NULL, " ");
67         if(ptr == NULL)break;
68         strcpy(temp, ptr);
69         //printf("%s\n", temp);
70         arg[i] = atoi(temp);
71         //printf("%d\n", arg[i]);
72         i++;
73     }
74
75     //printf("%s\n", cal);
76     //for(int k=0; k<i; k++){
77     //    printf("%d ",arg[k]);
78     //}
79
80     if(strcmp(cal, "ADD") == 0){
81         ans = 0;
82         for(int j=0; j<i; j++){
83             ans += arg[j];
84         }
85         if(ans > UINT_MAX){
86             strcpy(send, "Overflowed\n");
87             write(connfd, send, sizeof(send));
88             continue;
89         }
90         sprintf(send, "%lu\n", ans);
91         write(connfd, send, sizeof(send));
92     }else if(strcmp(cal, "MUL") == 0){
93         ans = 1;
94         for(int j=0; j<i; j++){
95             ans = ans * arg[j];
96         }
97         if(ans > UINT_MAX){
98             strcpy(send, "Overflowed\n");
99             write(connfd, send, sizeof(send));
100            continue;
101        }
102        sprintf(send, "%lu\n", ans);
103        write(connfd, send, sizeof(send));
104    }else if(strcmp(cal, "EXIT") == 0){
105        close(connfd);
106        break;
107    }else{
108        sprintf(send, "wrong command\n");
109        write(connfd, send, sizeof(send));
110    }
111}
112}
113}
114}
115}
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8 #include <limits.h>
9
10#define MAXLINE 1024
11
12int main(int argc, char *argv[]){
13    int listenfd, connfd;
14    struct sockaddr_in server;
15    char buffer[MAXLINE];
16    char cal[MAXLINE];
17    int arg[10];
18    unsigned long int ans;
19    int serverPORT;
20    char send[MAXLINE];
21
22    if(argc != 2){
23        printf("wrong input\n");
24        exit(1);
25    }
26
27    serverPORT = atoi(argv[1]);
28
29    if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
30        printf("socket error\n");
31        exit(1);
32    }
33
34    bzero(&server, sizeof(server));
35    server.sin_family = AF_INET;
36    server.sin_port = htons(serverPORT);
37    server.sin_addr.s_addr = htonl(INADDR_ANY);
38
39    if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
40        printf("bind error\n");
41        exit(1);
42    }
43
44    listen(listenfd, 1);
45
46    for();{
47        connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
48        for();{
49            ssize_t len;
50            len = read(connfd, &buffer, MAXLINE);
51            if(len == 0){
52                break;
53            }
54
55            if(buffer[strlen(buffer) - 1] == '\n'){
56                buffer[strlen(buffer) - 1] = '\0';
57            }
58            //printf("read done\n");
59            char *ptr = strtok(buffer, " ");
60            int i = 0;
61            strcpy(cal, ptr);
62            //printf("%s\n", cal);
63        }
64    }
65}
```

```
server.c      x   daytimeserver.c    x   concurrentechoserver.c x   client1.c    x   client4.c    x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8
9 #define MAXLINE 1024
10
11 int main(int argc, char *argv[]){
12     char *serverIP;
13     int serverPORT;
14     int sockfd;
15     struct sockaddr_in server;
16     char buffer[MAXLINE], receivebuffer[MAXLINE];
17     FILE *fp = stdin;
18
19     if(argc != 3){
20         printf("wrong input\n");
21         exit(1);
22     }
23
24     serverIP = argv[1];
25     serverPORT = atoi(argv[2]);
26
27     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
28         printf("socket error\n");
29         exit(1);
30     }
31
32     bzero(&server, sizeof(server));
33     server.sin_family = AF_INET;
34     server.sin_port = htons(serverPORT);
35     inet_pton(AF_INET, serverIP, &server.sin_addr);
36
37     if(connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0){
38         printf("connect error\n");
39         exit(1);
40     }
41
42     for(;){
43         fgets(buffer, MAXLINE, fp);
44         write(sockfd, buffer, sizeof(buffer));
45
46         ssize_t len;
47         len = read(sockfd, receivebuffer, MAXLINE);
48         if(len == 0){
49             printf("The server has closed the connection\n");
50             close(sockfd);
51             exit(0);
52         }
53         fputs(receivebuffer, stdout);
54     }
55 }
```

```
server.c      x    daytimeserver.c   x    concurrentechoserver.c x    client1.c      x    client4.c      x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8 #include <signal.h>
9 #include <sys/wait.h>
10
11 #define backlog 10
12 #define MAXLINE 1024
13
14 void sig_fork(int signo){
15     pid_t pid;
16     int status;
17     while((pid = waitpid(-1, &status, WNOHANG)) > 0){
18         printf("child %d terminated\n", pid);
19     }
20     return;
21 }
22
23 int main(int argc, char *argv[]){
24     int serverPORT;
25     int listenfd, connfd;
26     char buffer[MAXLINE];
27     struct sockaddr_in server;
28     pid_t pid;
29
30     serverPORT = atoi(argv[1]);
31     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
32         printf("socket error\n");
33         exit(1);
34     }
35
36     bzero(&server, sizeof(server));
37     server.sin_family = AF_INET;
38     server.sin_port = htons(serverPORT);
39     server.sin_addr.s_addr = htonl(INADDR_ANY);
40
41     if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
42         printf("bind error\n");
43         exit(1);
44     }
45
46     listen(listenfd, backlog);
47     signal(SIGCHLD, sig_fork);
48     for();{
49         connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
50         if((pid = fork()) == 0){
51             close(listenfd);
52             for();{
53                 read(connfd, &buffer, MAXLINE);
54                 if(strcmp(buffer, "exit\n") == 0){
55                     break;
56                 }
57                 write(connfd, buffer, sizeof(buffer));
58             }
59             close(connfd);
60             exit(0);
61         }else{
62             close(connfd);
63         }
64     }
65 }
```

```
server.c      x  daytimeserver.c      x  concurrentchoserver.c      x  client1.c      x  client4.c      x
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8 #include <time.h>
9
10 #define backlog 10
11 #define MAXLINE 1024
12
13 int main(int argc, char *argv[]){
14     int listenfd, connfd;
15     struct sockaddr_in server;
16     char buffer[MAXLINE];
17     int serverPORT;
18
19     serverPORT = atoi(argv[1]);
20
21     time_t ticks;
22     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
23         printf("socket error\n");
24         exit(1);
25     }
26
27     bzero(&server, sizeof(server));
28     server.sin_family = AF_INET;
29     server.sin_port = htons(serverPORT);
30     server.sin_addr.s_addr = htonl(INADDR_ANY);
31
32     if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
33         printf("bind failed\n");
34         exit(1);
35     }
36
37     listen(listenfd, backlog);
38
39     for(;){
40         connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
41         ticks = time(NULL);
42         sprintf(buffer, "%.24s\n", ctime(&ticks));
43         write(connfd, buffer, sizeof(buffer));
44         close(connfd);
45     }
46 }
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <unistd.h>
8
9 #define MAXLINE 1024
10
11 int main(int argc, char *argv[]){
12     char *serverIP;
13     int serverPORT;
14     int sockfd;
15     struct sockaddr_in server;
16     char sendbuffer[MAXLINE], receivebuffer[MAXLINE];
17     int n;
18     FILE *fp = stdin;
19
20     serverIP = argv[1];
21     serverPORT = atoi(argv[2]);
22
23     if(argc != 3){
24         printf("Wrong input\n");
25         exit(1);
26     }
27
28     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
29         printf("socket error\n");
30         exit(1);
31     }
32
33     bzero(&server, sizeof(server));
34     server.sin_family = AF_INET;
35     server.sin_port = htons(serverPORT);
36     if((inet_pton(AF_INET, serverIP, &server.sin_addr)) <= 0){
37         printf("inet_ntop error\n");
38         exit(1);
39     }
40
41     if(connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0){
42         printf("connect error\n");
43         exit(1);
44     }
45
46     for(;;){
47         fgets(sendbuffer, MAXLINE, fp);
48         write(sockfd, sendbuffer, sizeof(sendbuffer));
49
50         ssize_t len;
51         len = read(sockfd, receivebuffer, MAXLINE);
52         if(len == 0){
53             printf("connection closed\n");
54             exit(1);
55         }
56         fputs(receivebuffer, stdout);
57     }
58 }
59 }
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
19     strcpy(cuts, cut.c_str());
20
21     char *pch = strtok(words, cuts);
22     while(pch != NULL){
23         cout << pch << " ";
24         pch = strtok(NULL, cuts);
25     }
26     cout << endl;
27 }
28
29 int main(int argc, char *argv[]){
30     ifstream inputFile;
31     inputFile.open(argv[1], ifstream::in);
32     string line;
33     string cut = argv[2];
34     int filelen = strlen(argv[1]);
35
36     cout << "-----Input file " << argv[1] << "-----"
37         << endl;
38
39     while(getline(inputFile, line)){
40         string word;
41         string command;
42         istringstream iss(line);
43         iss >> command;
44         istringstream iss2(line);
45         while(iss2 >> word){
46             cout << word << " ";
47         }
48         cout << endl;
49         if(command == "reverse"){
50             reverse(word);
51         }else if(command == "split"){
52             split(word, cut);
53         }
54     }
55
56     cout << "-----End of input file " << argv[1] << "-----"
57         << endl;
58     cout << "*****User input*****";
59     for(int i=0; i<filelen; i++){
60         cout << "*";
61     }
62     cout << endl;
63
64     string input;
65     while(getline(cin, input)){
66         istringstream iss3(input);
67         string userCommand;
68         string userWord;
69         iss3 >> userCommand;
70         if(userCommand == "exit"){
71             return 0;
72         }else if(userCommand == "reverse"){
73             while(iss3 >> userWord){
74                 reverse(userWord);
75             }
76         }else if(userCommand == "split"){
77             while(iss3 >> userWord){
78                 split(userWord, cut);
79             }
80     }
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 void reverse(string &word){
6     string output = "";
7     int len = word.length();
8     for(int i=len-1; i>=0; i--){
9         output = output + word[i];
10    }
11    cout << output << endl;
12 }
13
14 void split(string &word, string &cut){
15     char *words = new char[word.length() + 1];
16     strcpy(words, word.c_str());
17
18     char *cuts = new char[cut.length() + 1];
19     strcpy(cuts, cut.c_str());
20
21     char *pch = strtok(words, cuts);
22     while(pch != NULL){
23         cout << pch << " ";
24         pch = strtok(NULL, cuts);
25     }
26     cout << endl;
27 }
28
29 int main(int argc, char *argv[]){
30     ifstream inputFile;
31     inputFile.open(argv[1], ifstream::in);
32     string line;
33     string cut = argv[2];
34     int filelen = strlen(argv[1]);
35
36     cout << "-----Input file " << argv[1] << "-----"
37         << endl;
38
39     while(getline(inputFile, line)){
40         string word;
41         string command;
42         istringstream iss(line);
43         iss >> command;
44         istringstream iss2(line);
45         while(iss2 >> word){
46             cout << word << " ";
47         }
48         cout << endl;
49         if(command == "reverse"){
50             reverse(word);
51         }else if(command == "split"){
52             split(word, cut);
53         }
54
55         cout << "-----End of input file " << argv[1] << "-----"
56         << endl;
57         cout << "*****User input*****";
58         for(int i=0; i<filelen; i++){
59             cout << "*";
60         }
61         cout << endl;
62 }
```

```
server.c      x  daytimeserver.c   x  concurrentechoserver.c x  client1.c   x  client4.c   x
226
227
228     }
229     for(j=0; j <= maxClient; j++){
230         if(strcmp(receiver, username[j]) == 0){
231             break;
232         }
233     } // user[j] is receiver
234
235     if(j > maxClient){
236         strcpy(message, "[Server] ERROR: The receiver doesn't exist.\n");
237         write(client[i], message, sizeof(message));
238         continue;
239     }else if(ban[j] == 1){
240         sprintf(message, "that user has been banned\n");
241         write(client[i], message, sizeof(message));
242         continue;
243     }else{
244         strcpy(message, "[Server] SUCCESS: Your message has been sent.\n");
245         write(client[i], message, sizeof(message));
246         sprintf(message, "[Server] %s tell you %s\n", username[i], sendbuffer);
247         write(client[j], message, sizeof(message));
248         continue;
249     }
250     }else if(strcmp(command, "yell") == 0){
251         printf("someone input yell\n");
252         for(j=0; j <= maxClient; j++){
253             if(client[j] >= 0 && ban[j] == 0){
254                 sprintf(message, "[Server] %s yell %s\n", username[i], text);
255                 write(client[j], message, sizeof(message));
256             }
257         }
258     }else if(strcmp(command, "ban") == 0){
259         printf("bannnnn\n");
260         for(j=0; j<= maxClient; j++){
261             printf("%s\n", username[j]);
262             if(strcmp(text, username[j]) == 0){
263                 printf("find\n");
264                 ban[j] = 1;
265                 sprintf(message, "you have ban %s\n", username[j]);
266                 write(client[i], message, sizeof(message));
267                 break;
268             }
269         }
270         if(j > maxClient){
271             sprintf(message, "wrong user\n");
272             write(client[i], message, sizeof(message));
273         }
274     }else{
275         strcpy(message, "[Server] ERROR: Error command.\n");
276         write(client[i], message, sizeof(message));
277     }
278
279     numReady--;
280     if(numReady <= 0)break;
281 }
282
283
284
285
286 }
```

```
server.c      x   daytimeserver.c      x   concurrentechoserver.c      x   client1.c      x   client4.c      x
171     write(client[i], message, sizeof(message));
172     continue;
173 } // no anonymous
174 for(j=0; j <= maxClient; j++){
175     if(strcmp(text, username[j]) == 0 && client[j] != sockfd){
176         sprintf(message, "[Server] ERROR: %s has been used by others.\n",
177                 text);
178         write(client[i], message, sizeof(message));
179         break;
180     }
181     if(j <= maxClient) continue; // no the same
182
183     int allenglish = 1;
184     for(j=0; j < strlen(text); j++){
185         if(text[j] < 'A' || text[j] > 'z'){
186             allenglish = 0;
187             break;
188         }
189     }
190     if(strlen(text) < 2 || strlen(text) > 12 || allenglish == 0){
191         strcpy(message, "[Server] ERROR: Username can only consists of 2~12
192             English letters.\n");
193         write(client[i], message, sizeof(message));
194         continue;
195     }
196     strcpy(oldname, username[i]);
197     strcpy(username[i], text); //update username
198
199     for(j=0; j <= maxClient; j++){
200         if(client[j] == sockfd){
201             sprintf(message, "[Server] You're now known as %s.\n", text);
202             write(client[i], message, sizeof(message));
203         }else if(client[j] >= 0 && ban[j] == 0){
204             sprintf(message, "[Server] %s is now known as %s.\n", oldname, text);
205             write(client[j], message, sizeof(message));
206         }
207     }
208 }else if(strcmp(command, "tell") == 0){
209     printf("someone input tell\n");
210     if(strcmp(username[i], "anonymous") == 0){
211         strcpy(message, "[Server] ERROR: You are anonymous.\n");
212         write(client[i], message, sizeof(message));
213         continue;
214     }
215
216     strcpy(receiver, strtok(text, " "));
217     if(receiver[strlen(receiver) - 1] == '\n') receiver[strlen(receiver) - 1]
218         = '\0'; // the user want to send
219
220     strcpy(sendbuffer, text + strlen(receiver) + 1);
221     if(sendbuffer[strlen(sendbuffer) - 1] == '\n') sendbuffer[strlen(
222         sendbuffer) - 1] = '\0'; // the message want to send
223
224     if(strcmp(receiver, "anonymous") == 0){
225         strcpy(message, "[Server] ERROR: The client to which you sent is
226             anonymous.\n");
227         write(client[i], message, sizeof(message));
228         continue;
229     }
230
231     for(j=0; j <= maxClient; j++){
232         if(strcmp(receiver, username[j]) == 0){
233             break;
234         }
235     }
```

```
server.c      x    daytimeserver.c      x    concurrentechoserver.c      x    client1.c      x    client4.c      x
111 tempbuffer = receivebuffer;
112 for(n=1; n<MAXLINE; n++){
113     if((templen = read(sockfd, &templetter, 1)) == 1){
114         *tempbuffer++ = templetter;
115         if(templetter == '\n') break;
116     }else if(templen == 0){
117         *tempbuffer = '\0';
118         n--;
119         break;
120     }
121 }
122 *tempbuffer = '\0';
123 len = n;
124 printf("receive length %zd\n", len);
125
126 if(len == 0){ //client closed
127     printf("client port %d leave\n", clientPort[i]);
128
129     for(j = 0; j <= maxClient; j++){
130         if(client[j] == sockfd){
131             continue;
132         }else if(client[j] >= 0){
133             sprintf(message, "[Server] %s is offline.\n", username[i]);
134             write(client[j], message, sizeof(message));
135         }
136     }
137
138     close(sockfd);
139     FD_CLR(sockfd, &allset);
140     client[i] = -1;
141     free(clientIP[i]);
142     free(username[i]);
143
144 }else{
145     if(ban[i] == 1){
146         sprintf(message, "you can not send message\n");
147         write(client[i], message, sizeof(message));
148         continue;
149     }
150     printf("line: %s\n", receivebuffer);
151     strcpy(command, strtok(receivebuffer, " "));
152     if(command[strlen(command) - 1] == '\n') command[strlen(command) - 1] = '\0';
153     strcpy(text, receivebuffer + strlen(command) + 1 );
154     if(text[strlen(text) - 1] == '\n') text[strlen(text) - 1] = '\0';
155
156     if(strcmp(command, "who") == 0){
157         printf("someone input who\n");
158         for(j=0; j <= maxClient; j++){
159             if(client[j] == sockfd){
160                 sprintf(message, "[Server] %s %s:%d ->me\n", username[j], clientIP[j]
161                     , clientPort[j]);
162                 write(client[i], message, sizeof(message));
163             }else if(client[j] >= 0){
164                 sprintf(message, "[Server] %s %s:%d\n", username[j], clientIP[j],
165                     clientPort[j]);
166                 write(client[i], message, sizeof(message));
167             }
168         }
169     }else if(strcmp(command, "name") == 0){
170         printf("someone input name\n");
171         if(strcmp(text, "anonymous") == 0){
172             strcpy(message, "[Server] ERROR: Username cannot be anonymous.\n");
173             write(client[i], message, sizeof(message));
174             continue;
175         } // no anonymous
176     }
177 }
```

```
server.c      x      daytimeserver.c      x      concurrentechoserver.c      x      client1.c      x      client4.c      x
52     client[i] = -1;
53     ban[i] = 0;
54 } // no client
55 FD_ZERO(&allset); //clear allset
56 FD_SET(listenfd, &allset); // put listen socket in allset
57
58 for(;;){ //server run forever
59     rset = allset; // avoid change allset
60     numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
61
62     if(FD_ISSET(listenfd, &rset)){ //listen socket is readable = have client
63         clientlen = sizeof(clientAddress);
64         clientfd = accept(listenfd, (struct sockaddr *)&clientAddress, &clientlen);
65         // accept a new client and give it a new socket
66
67         for(i=0; i<backlog; i++){
68             if(client[i] < 0){
69                 client[i] = clientfd;
70                 break;
71             }
72         } // record i is what fd
73         if(i == backlog){
74             printf("To many client\n");
75             exit(1);
76         }
77         //printf("client is fd %d\n", i);
78
79         clientIP[i] = malloc(INET_ADDRSTRLEN);
80         inet_ntop(AF_INET, &clientAddress.sin_addr, clientIP[i], INET_ADDRSTRLEN);
81         clientPort[i] = ntohs(clientAddress.sin_port);
82         username[i] = malloc(MAXLINE * sizeof(char));
83         strcpy(username[i], "anonymous"); //client information finished
84
85         FD_SET(clientfd, &allset);
86         if(clientfd > maxfd) maxfd = clientfd;
87         if(i > maxClient) maxClient = i;
88
89         for(j=0; j <= maxClient; j++){
90             if(clientfd == client[j]){
91                 sprintf(message, "[Server] Hello, anonymous! From: %s:%d\n", clientIP[j],
92                             clientPort[j]);
93                 write(client[j], message, sizeof(message));
94             }else if(client[j] >= 0){
95                 strcpy(message, "[Server] Someone is coming!\n");
96                 write(client[j], message, sizeof(message));
97             }
98         }
99         numReady--;
100        if(numReady <= 0) continue;
101    } //connect handle finish
102
103    for(i=0; i <= maxClient; i++){
104        sockfd = client[i];
105        if(client[i] < 0) continue;
106
107        if(FD_ISSET(sockfd, &rset)){ //someone is readable
108            ssize_t n, templen;
109            char templetter;
110            char *tempbuffer;
111            tempbuffer = receivebuffer;
112            for(n=1; n<MAXLINE; n++){
113                if((templen = read(sockfd, &templetter, 1)) == 1){
114                    *tempbuffer++ = templetter;
115                    if(templetter == '\n') break;
116                }
117            }
118            if(templetter == '\n') break;
119        }
120    }
121
122    if(i == maxClient) break;
123
124    if(numReady <= 0) continue;
125
126    for(i=0; i <= maxClient; i++){
127        sockfd = client[i];
128        if(client[i] < 0) continue;
129
130        if(FD_ISSET(sockfd, &rset)){ //someone is readable
131            ssize_t n, templen;
132            char templetter;
133            char *tempbuffer;
134            tempbuffer = receivebuffer;
135            for(n=1; n<MAXLINE; n++){
136                if((templen = read(sockfd, &templetter, 1)) == 1){
137                    *tempbuffer++ = templetter;
138                    if(templetter == '\n') break;
139                }
140            }
141            if(templetter == '\n') break;
142        }
143    }
144
145    if(i == maxClient) break;
146
147    if(numReady <= 0) continue;
148
149    for(i=0; i <= maxClient; i++){
150        sockfd = client[i];
151        if(client[i] < 0) continue;
152
153        if(FD_ISSET(sockfd, &rset)){ //someone is readable
154            ssize_t n, templen;
155            char templetter;
156            char *tempbuffer;
157            tempbuffer = receivebuffer;
158            for(n=1; n<MAXLINE; n++){
159                if((templen = read(sockfd, &templetter, 1)) == 1){
160                    *tempbuffer++ = templetter;
161                    if(templetter == '\n') break;
162                }
163            }
164            if(templetter == '\n') break;
165        }
166    }
167
168    if(i == maxClient) break;
169
170    if(numReady <= 0) continue;
171
172    for(i=0; i <= maxClient; i++){
173        sockfd = client[i];
174        if(client[i] < 0) continue;
175
176        if(FD_ISSET(sockfd, &rset)){ //someone is readable
177            ssize_t n, templen;
178            char templetter;
179            char *tempbuffer;
180            tempbuffer = receivebuffer;
181            for(n=1; n<MAXLINE; n++){
182                if((templen = read(sockfd, &templetter, 1)) == 1){
183                    *tempbuffer++ = templetter;
184                    if(templetter == '\n') break;
185                }
186            }
187            if(templetter == '\n') break;
188        }
189    }
190
191    if(i == maxClient) break;
192
193    if(numReady <= 0) continue;
194
195    for(i=0; i <= maxClient; i++){
196        sockfd = client[i];
197        if(client[i] < 0) continue;
198
199        if(FD_ISSET(sockfd, &rset)){ //someone is readable
200            ssize_t n, templen;
201            char templetter;
202            char *tempbuffer;
203            tempbuffer = receivebuffer;
204            for(n=1; n<MAXLINE; n++){
205                if((templen = read(sockfd, &templetter, 1)) == 1){
206                    *tempbuffer++ = templetter;
207                    if(templetter == '\n') break;
208                }
209            }
210            if(templetter == '\n') break;
211        }
212    }
213
214    if(i == maxClient) break;
215
216    if(numReady <= 0) continue;
217
218    for(i=0; i <= maxClient; i++){
219        sockfd = client[i];
220        if(client[i] < 0) continue;
221
222        if(FD_ISSET(sockfd, &rset)){ //someone is readable
223            ssize_t n, templen;
224            char templetter;
225            char *tempbuffer;
226            tempbuffer = receivebuffer;
227            for(n=1; n<MAXLINE; n++){
228                if((templen = read(sockfd, &templetter, 1)) == 1){
229                    *tempbuffer++ = templetter;
230                    if(templetter == '\n') break;
231                }
232            }
233            if(templetter == '\n') break;
234        }
235    }
236
237    if(i == maxClient) break;
238
239    if(numReady <= 0) continue;
240
241    for(i=0; i <= maxClient; i++){
242        sockfd = client[i];
243        if(client[i] < 0) continue;
244
245        if(FD_ISSET(sockfd, &rset)){ //someone is readable
246            ssize_t n, templen;
247            char templetter;
248            char *tempbuffer;
249            tempbuffer = receivebuffer;
250            for(n=1; n<MAXLINE; n++){
251                if((templen = read(sockfd, &templetter, 1)) == 1){
252                    *tempbuffer++ = templetter;
253                    if(templetter == '\n') break;
254                }
255            }
256            if(templetter == '\n') break;
257        }
258    }
259
260    if(i == maxClient) break;
261
262    if(numReady <= 0) continue;
263
264    for(i=0; i <= maxClient; i++){
265        sockfd = client[i];
266        if(client[i] < 0) continue;
267
268        if(FD_ISSET(sockfd, &rset)){ //someone is readable
269            ssize_t n, templen;
270            char templetter;
271            char *tempbuffer;
272            tempbuffer = receivebuffer;
273            for(n=1; n<MAXLINE; n++){
274                if((templen = read(sockfd, &templetter, 1)) == 1){
275                    *tempbuffer++ = templetter;
276                    if(templetter == '\n') break;
277                }
278            }
279            if(templetter == '\n') break;
280        }
281    }
282
283    if(i == maxClient) break;
284
285    if(numReady <= 0) continue;
286
287    for(i=0; i <= maxClient; i++){
288        sockfd = client[i];
289        if(client[i] < 0) continue;
290
291        if(FD_ISSET(sockfd, &rset)){ //someone is readable
292            ssize_t n, templen;
293            char templetter;
294            char *tempbuffer;
295            tempbuffer = receivebuffer;
296            for(n=1; n<MAXLINE; n++){
297                if((templen = read(sockfd, &templetter, 1)) == 1){
298                    *tempbuffer++ = templetter;
299                    if(templetter == '\n') break;
300                }
301            }
302            if(templetter == '\n') break;
303        }
304    }
305
306    if(i == maxClient) break;
307
308    if(numReady <= 0) continue;
309
310    for(i=0; i <= maxClient; i++){
311        sockfd = client[i];
312        if(client[i] < 0) continue;
313
314        if(FD_ISSET(sockfd, &rset)){ //someone is readable
315            ssize_t n, templen;
316            char templetter;
317            char *tempbuffer;
318            tempbuffer = receivebuffer;
319            for(n=1; n<MAXLINE; n++){
320                if((templen = read(sockfd, &templetter, 1)) == 1){
321                    *tempbuffer++ = templetter;
322                    if(templetter == '\n') break;
323                }
324            }
325            if(templetter == '\n') break;
326        }
327    }
328
329    if(i == maxClient) break;
330
331    if(numReady <= 0) continue;
332
333    for(i=0; i <= maxClient; i++){
334        sockfd = client[i];
335        if(client[i] < 0) continue;
336
337        if(FD_ISSET(sockfd, &rset)){ //someone is readable
338            ssize_t n, templen;
339            char templetter;
340            char *tempbuffer;
341            tempbuffer = receivebuffer;
342            for(n=1; n<MAXLINE; n++){
343                if((templen = read(sockfd, &templetter, 1)) == 1){
344                    *tempbuffer++ = templetter;
345                    if(templetter == '\n') break;
346                }
347            }
348            if(templetter == '\n') break;
349        }
350    }
351
352    if(i == maxClient) break;
353
354    if(numReady <= 0) continue;
355
356    for(i=0; i <= maxClient; i++){
357        sockfd = client[i];
358        if(client[i] < 0) continue;
359
360        if(FD_ISSET(sockfd, &rset)){ //someone is readable
361            ssize_t n, templen;
362            char templetter;
363            char *tempbuffer;
364            tempbuffer = receivebuffer;
365            for(n=1; n<MAXLINE; n++){
366                if((templen = read(sockfd, &templetter, 1)) == 1){
367                    *tempbuffer++ = templetter;
368                    if(templetter == '\n') break;
369                }
370            }
371            if(templetter == '\n') break;
372        }
373    }
374
375    if(i == maxClient) break;
376
377    if(numReady <= 0) continue;
378
379    for(i=0; i <= maxClient; i++){
380        sockfd = client[i];
381        if(client[i] < 0) continue;
382
383        if(FD_ISSET(sockfd, &rset)){ //someone is readable
384            ssize_t n, templen;
385            char templetter;
386            char *tempbuffer;
387            tempbuffer = receivebuffer;
388            for(n=1; n<MAXLINE; n++){
389                if((templen = read(sockfd, &templetter, 1)) == 1){
390                    *tempbuffer++ = templetter;
391                    if(templetter == '\n') break;
392                }
393            }
394            if(templetter == '\n') break;
395        }
396    }
397
398    if(i == maxClient) break;
399
400    if(numReady <= 0) continue;
401
402    for(i=0; i <= maxClient; i++){
403        sockfd = client[i];
404        if(client[i] < 0) continue;
405
406        if(FD_ISSET(sockfd, &rset)){ //someone is readable
407            ssize_t n, templen;
408            char templetter;
409            char *tempbuffer;
410            tempbuffer = receivebuffer;
411            for(n=1; n<MAXLINE; n++){
412                if((templen = read(sockfd, &templetter, 1)) == 1){
413                    *tempbuffer++ = templetter;
414                    if(templetter == '\n') break;
415                }
416            }
417            if(templetter == '\n') break;
418        }
419    }
420
421    if(i == maxClient) break;
422
423    if(numReady <= 0) continue;
424
425    for(i=0; i <= maxClient; i++){
426        sockfd = client[i];
427        if(client[i] < 0) continue;
428
429        if(FD_ISSET(sockfd, &rset)){ //someone is readable
430            ssize_t n, templen;
431            char templetter;
432            char *tempbuffer;
433            tempbuffer = receivebuffer;
434            for(n=1; n<MAXLINE; n++){
435                if((templen = read(sockfd, &templetter, 1)) == 1){
436                    *tempbuffer++ = templetter;
437                    if(templetter == '\n') break;
438                }
439            }
440            if(templetter == '\n') break;
441        }
442    }
443
444    if(i == maxClient) break;
445
446    if(numReady <= 0) continue;
447
448    for(i=0; i <= maxClient; i++){
449        sockfd = client[i];
450        if(client[i] < 0) continue;
451
452        if(FD_ISSET(sockfd, &rset)){ //someone is readable
453            ssize_t n, templen;
454            char templetter;
455            char *tempbuffer;
456            tempbuffer = receivebuffer;
457            for(n=1; n<MAXLINE; n++){
458                if((templen = read(sockfd, &templetter, 1)) == 1){
459                    *tempbuffer++ = templetter;
460                    if(templetter == '\n') break;
461                }
462            }
463            if(templetter == '\n') break;
464        }
465    }
466
467    if(i == maxClient) break;
468
469    if(numReady <= 0) continue;
470
471    for(i=0; i <= maxClient; i++){
472        sockfd = client[i];
473        if(client[i] < 0) continue;
474
475        if(FD_ISSET(sockfd, &rset)){ //someone is readable
476            ssize_t n, templen;
477            char templetter;
478            char *tempbuffer;
479            tempbuffer = receivebuffer;
480            for(n=1; n<MAXLINE; n++){
481                if((templen = read(sockfd, &templetter, 1)) == 1){
482                    *tempbuffer++ = templetter;
483                    if(templetter == '\n') break;
484                }
485            }
486            if(templetter == '\n') break;
487        }
488    }
489
490    if(i == maxClient) break;
491
492    if(numReady <= 0) continue;
493
494    for(i=0; i <= maxClient; i++){
495        sockfd = client[i];
496        if(client[i] < 0) continue;
497
498        if(FD_ISSET(sockfd, &rset)){ //someone is readable
499            ssize_t n, templen;
500            char templetter;
501            char *tempbuffer;
502            tempbuffer = receivebuffer;
503            for(n=1; n<MAXLINE; n++){
504                if((templen = read(sockfd, &templetter, 1)) == 1){
505                    *tempbuffer++ = templetter;
506                    if(templetter == '\n') break;
507                }
508            }
509            if(templetter == '\n') break;
510        }
511    }
512
513    if(i == maxClient) break;
514
515    if(numReady <= 0) continue;
516
517    for(i=0; i <= maxClient; i++){
518        sockfd = client[i];
519        if(client[i] < 0) continue;
520
521        if(FD_ISSET(sockfd, &rset)){ //someone is readable
522            ssize_t n, templen;
523            char templetter;
524            char *tempbuffer;
525            tempbuffer = receivebuffer;
526            for(n=1; n<MAXLINE; n++){
527                if((templen = read(sockfd, &templetter, 1)) == 1){
528                    *tempbuffer++ = templetter;
529                    if(templetter == '\n') break;
530                }
531            }
532            if(templetter == '\n') break;
533        }
534    }
535
536    if(i == maxClient) break;
537
538    if(numReady <= 0) continue;
539
540    for(i=0; i <= maxClient; i++){
541        sockfd = client[i];
542        if(client[i] < 0) continue;
543
544        if(FD_ISSET(sockfd, &rset)){ //someone is readable
545            ssize_t n, templen;
546            char templetter;
547            char *tempbuffer;
548            tempbuffer = receivebuffer;
549            for(n=1; n<MAXLINE; n++){
550                if((templen = read(sockfd, &templetter, 1)) == 1){
551                    *tempbuffer++ = templetter;
552                    if(templetter == '\n') break;
553                }
554            }
555            if(templetter == '\n') break;
556        }
557    }
558
559    if(i == maxClient) break;
560
561    if(numReady <= 0) continue;
562
563    for(i=0; i <= maxClient; i++){
564        sockfd = client[i];
565        if(client[i] < 0) continue;
566
567        if(FD_ISSET(sockfd, &rset)){ //someone is readable
568            ssize_t n, templen;
569            char templetter;
570            char *tempbuffer;
571            tempbuffer = receivebuffer;
572            for(n=1; n<MAXLINE; n++){
573                if((templen = read(sockfd, &templetter, 1)) == 1){
574                    *tempbuffer++ = templetter;
575                    if(templetter == '\n') break;
576                }
577            }
578            if(templetter == '\n') break;
579        }
580    }
581
582    if(i == maxClient) break;
583
584    if(numReady <= 0) continue;
585
586    for(i=0; i <= maxClient; i++){
587        sockfd = client[i];
588        if(client[i] < 0) continue;
589
590        if(FD_ISSET(sockfd, &rset)){ //someone is readable
591            ssize_t n, templen;
592            char templetter;
593            char *tempbuffer;
594            tempbuffer = receivebuffer;
595            for(n=1; n<MAXLINE; n++){
596                if((templen = read(sockfd, &templetter, 1)) == 1){
597                    *tempbuffer++ = templetter;
598                    if(templetter == '\n') break;
599                }
600            }
601            if(templetter == '\n') break;
602        }
603    }
604
605    if(i == maxClient) break;
606
607    if(numReady <= 0) continue;
608
609    for(i=0; i <= maxClient; i++){
610        sockfd = client[i];
611        if(client[i] < 0) continue;
612
613        if(FD_ISSET(sockfd, &rset)){ //someone is readable
614            ssize_t n, templen;
615            char templetter;
616            char *tempbuffer;
617            tempbuffer = receivebuffer;
618            for(n=1; n<MAXLINE; n++){
619                if((templen = read(sockfd, &templetter, 1)) == 1){
620                    *tempbuffer++ = templetter;
621                    if(templetter == '\n') break;
622                }
623            }
624            if(templetter == '\n') break;
625        }
626    }
627
628    if(i == maxClient) break;
629
630    if(numReady <= 0) continue;
631
632    for(i=0; i <= maxClient; i++){
633        sockfd = client[i];
634        if(client[i] < 0) continue;
635
636        if(FD_ISSET(sockfd, &rset)){ //someone is readable
637            ssize_t n, templen;
638            char templetter;
639            char *tempbuffer;
640            tempbuffer = receivebuffer;
641            for(n=1; n<MAXLINE; n++){
642                if((templen = read(sockfd, &templetter, 1)) == 1){
643                    *tempbuffer++ = templetter;
644                    if(templetter == '\n') break;
645                }
646            }
647            if(templetter == '\n') break;
648        }
649    }
650
651    if(i == maxClient) break;
652
653    if(numReady <= 0) continue;
654
655    for(i=0; i <= maxClient; i++){
656        sockfd = client[i];
657        if(client[i] < 0) continue;
658
659        if(FD_ISSET(sockfd, &rset)){ //someone is readable
660            ssize_t n, templen;
661            char templetter;
662            char *tempbuffer;
663            tempbuffer = receivebuffer;
664            for(n=1; n<MAXLINE; n++){
665                if((templen = read(sockfd, &templetter, 1)) == 1){
666                    *tempbuffer++ = templetter;
667                    if(templetter == '\n') break;
668                }
669            }
670            if(templetter == '\n') break;
671        }
672    }
673
674    if(i == maxClient) break;
675
676    if(numReady <= 0) continue;
677
678    for(i=0; i <= maxClient; i++){
679        sockfd = client[i];
680        if(client[i] < 0) continue;
681
682        if(FD_ISSET(sockfd, &rset)){ //someone is readable
683            ssize_t n, templen;
684            char templetter;
685            char *tempbuffer;
686            tempbuffer = receivebuffer;
687            for(n=1; n<MAXLINE; n++){
688                if((templen = read(sockfd, &templetter, 1)) == 1){
689                    *tempbuffer++ = templetter;
690                    if(templetter == '\n') break;
691                }
692            }
693            if(templetter == '\n') break;
694        }
695    }
696
697    if(i == maxClient) break;
698
699    if(numReady <= 0) continue;
700
701    for(i=0; i <= maxClient; i++){
702        sockfd = client[i];
703        if(client[i] < 0) continue;
704
705        if(FD_ISSET(sockfd, &rset)){ //someone is readable
706            ssize_t n, templen;
707            char templetter;
708            char *tempbuffer;
709            tempbuffer = receivebuffer;
710            for(n=1; n<MAXLINE; n++){
711                if((templen = read(sockfd, &templetter, 1)) == 1){
712                    *tempbuffer++ = templetter;
713                    if(templetter == '\n') break;
714                }
715            }
716            if(templetter == '\n') break;
717        }
718    }
719
720    if(i == maxClient) break;
721
722    if(numReady <= 0) continue;
723
724    for(i=0; i <= maxClient; i++){
725        sockfd = client[i];
726        if(client[i] < 0) continue;
727
728        if(FD_ISSET(sockfd, &rset)){ //someone is readable
729            ssize_t n, templen;
730            char templetter;
731            char *tempbuffer;
732            tempbuffer = receivebuffer;
733            for(n=1; n<MAXLINE; n++){
734                if((templen = read(sockfd, &templetter, 1)) == 1){
735                    *tempbuffer++ = templetter;
736                    if(templetter == '\n') break;
737                }
738            }
739            if(templetter == '\n') break;
740        }
741    }
742
743    if(i == maxClient) break;
744
745    if(numReady <= 0) continue;
746
747    for(i=0; i <= maxClient; i++){
748        sockfd = client[i];
749        if(client[i] < 0) continue;
750
751        if(FD_ISSET(sockfd, &rset)){ //someone is readable
752            ssize_t n, templen;
753            char templetter;
754            char *tempbuffer;
755            tempbuffer = receivebuffer;
756            for(n=1; n<MAXLINE; n++){
757                if((templen = read(sockfd, &templetter, 1)) == 1){
758                    *tempbuffer++ = templetter;
759                    if(templetter == '\n') break;
760                }
761            }
762            if(templetter == '\n') break;
763        }
764    }
765
766    if(i == maxClient) break;
767
768    if(numReady <= 0) continue;
769
770    for(i=0; i <= maxClient; i++){
771        sockfd = client[i];
772        if(client[i] < 0) continue;
773
774        if(FD_ISSET(sockfd, &rset)){ //someone is readable
775            ssize_t n, templen;
776            char templetter;
777            char *tempbuffer;
778            tempbuffer = receivebuffer;
779            for(n=1; n<MAXLINE; n++){
780                if((templen = read(sockfd, &templetter, 1)) == 1){
781                    *tempbuffer++ = templetter;
782                    if(templetter == '\n') break;
783                }
784            }
785            if(templetter == '\n') break;
786        }
787    }
788
789    if(i == maxClient) break;
790
791    if(numReady <= 0) continue;
792
793    for(i=0; i <= maxClient; i++){
794        sockfd = client[i];
795        if(client[i] < 0) continue;
796
797        if(FD_ISSET(sockfd, &rset)){ //someone is readable
798            ssize_t n, templen;
799            char templetter;
800            char *tempbuffer;
801            tempbuffer = receivebuffer;
802            for(n=1; n<MAXLINE; n++){
803                if((templen = read(sockfd, &templetter, 1)) == 1){
804                    *tempbuffer++ = templetter;
805                    if(templetter == '\n') break;
806                }
807            }
808            if(templetter == '\n') break;
809        }
810    }
811
812    if(i == maxClient) break;
813
814    if(numReady <= 0) continue;
815
816    for(i=0; i <= maxClient; i++){
817        sockfd = client[i];
818        if(client[i] < 0) continue;
819
820        if(FD_ISSET(sockfd, &rset)){ //someone is readable
821            ssize_t n, templen;
822            char templetter;
823            char *tempbuffer;
824            tempbuffer = receivebuffer;
825            for(n=1; n<MAXLINE; n++){
826                if((templen = read(sockfd, &templetter, 1)) == 1){
827                    *tempbuffer++ = templetter;
828                    if(templetter == '\n') break;
829                }
830            }
831            if(templetter == '\n') break;
832        }
833    }
834
835    if(i == maxClient) break;
836
837    if(numReady <= 0) continue;
838
839    for(i=0; i <= maxClient; i++){
840        sockfd = client[i];
841        if(client[i] < 0) continue;
842
843        if(FD_ISSET(sockfd, &rset)){ //someone is readable
844            ssize_t n, templen;
845            char templetter;
846            char *tempbuffer;
847            tempbuffer = receivebuffer;
848            for(n=1; n<MAXLINE; n++){
849                if((templen = read(sockfd, &templetter, 1)) == 1){
850                    *tempbuffer++ = templetter;
851                    if(templetter == '\n') break;
852                }
853            }
854            if(templetter == '\n') break;
855        }
856    }
857
858    if(i == maxClient) break;
859
860    if(numReady <= 0) continue;
861
862    for(i=0; i <= maxClient; i++){
863        sockfd = client[i];
864        if(client[i] < 0) continue;
865
866        if(FD_ISSET(sockfd, &rset)){ //someone is readable
867            ssize_t n, templen;
868            char templetter;
869            char *tempbuffer;
870            tempbuffer = receivebuffer;
871            for(n=1; n<MAXLINE; n++){
872                if((templen = read(sockfd, &templetter, 1)) == 1){
873                    *tempbuffer++ = templetter;
874                    if(templetter == '\n') break;
875                }
876            }
877            if(templetter == '\n') break;
878        }
879    }
880
881    if(i == maxClient) break;
882
883    if(numReady <= 0) continue;
884
885    for(i=0; i <= maxClient; i++){
886        sockfd = client[i];
887        if(client[i] < 0) continue;
888
889        if(FD_ISSET(sockfd, &rset)){ //someone is readable
890            ssize_t n, templen;
891            char templetter;
892            char *tempbuffer;
893            tempbuffer = receivebuffer;
894            for(n=1; n<MAXLINE; n++){
895                if((templen = read(sockfd, &templetter, 1)) == 1){
896                    *tempbuffer++ = templetter;
897                    if(templetter == '\n') break;
898                }
899            }
900            if(templetter == '\n') break;
901        }
902    }
903
904    if(i == maxClient) break;
905
906    if(numReady <= 0) continue;
907
908    for(i=0; i <= maxClient; i++){
909        sockfd = client[i];
910        if(client[i] < 0) continue;
911
912        if(FD_ISSET(sockfd, &rset)){ //someone is readable
913            ssize_t n, templen;
914            char templetter;
915            char *tempbuffer;
916            tempbuffer = receivebuffer;
917            for(n=1; n<MAXLINE; n++){
918                if((templen = read(sockfd, &templetter, 1)) == 1){
919                    *tempbuffer++ = templetter;
920                    if(templetter == '\n') break;
921                }
922            }
923            if(templetter == '\n') break;
924        }
925    }
926
927    if(i == maxClient) break;
928
929    if(numReady <= 0) continue;
930
931    for(i=0; i <= maxClient; i++){
932        sockfd = client[i];
933        if(client[i] < 0) continue;
934
935        if(FD_ISSET(sockfd, &rset)){ //someone is readable
936            ssize_t n, templen;
937            char templetter;
938            char *tempbuffer;
939            tempbuffer = receivebuffer;
940            for(n=1; n<MAXLINE; n++){
941                if((templen = read(sockfd, &templetter, 1)) == 1){
942                    *tempbuffer++ = templetter;
943                    if(templetter == '\n') break;
944                }
945            }
946            if(templetter == '\n') break;
947        }
948    }
949
950    if(i == maxClient) break;
951
952    if(numReady <= 0) continue;
953
954    for(i=0; i <= maxClient; i++){
955        sockfd = client[i];
956        if(client[i] < 0) continue;
957
958        if(FD_ISSET(sockfd, &rset)){ //someone is readable
959            ssize_t n, templen;
960            char templetter;
961            char *tempbuffer;
962            tempbuffer = receivebuffer;
963            for(n=1; n<MAXLINE; n++){
964                if((templen = read(sockfd, &templetter, 1)) == 1){
965                    *tempbuffer++ = templetter;
966                    if(templetter == '\n') break;
967                }
968            }
969            if(templetter == '\n') break;
970        }
971    }
972
973    if(i == maxClient) break;
974
975    if(numReady <= 0) continue;
976
977    for(i=0; i <= maxClient; i++){
978        sockfd = client[i];
979        if(client[i] < 0) continue;
980
981        if(FD_ISSET(sockfd, &rset)){ //someone is readable
982            ssize_t n, templen;
983            char templetter;
984            char *tempbuffer;
985            tempbuffer = receivebuffer;
986            for(n=1; n<MAXLINE; n++){
987                if((templen = read(sockfd, &templetter, 1)) == 1){
988                    *tempbuffer++ = templetter;
989                    if(templetter == '\n') break;
990                }
991            }
992            if(templetter == '\n') break;
993        }
994    }
995
996    if(i == maxClient) break;
997
998    if(numReady <= 0) continue;
999
1000   for(i=0; i <= maxClient; i++){
1001      sockfd = client[i];
1002      if(client[i] < 0) continue;
1003
1004      if(FD_ISSET(sockfd, &rset)){ //someone is readable
1005          ssize_t n, templen;
1006          char templetter;
1007          char *tempbuffer;
1008          tempbuffer = receivebuffer;
1009          for(n=1; n<MAXLINE; n++){
1010              if((templen = read(sockfd, &templetter, 1)) == 1){
1011                  *tempbuffer++ = templetter;
1012                  if(templetter == '\n') break;
1013              }
1014          }
1015          if(templetter == '\n') break;
1016      }
1017  }
1018
1019  if(i == maxClient) break;
1020
1021  if(numReady <= 0) continue;
1022
1023  for(i=0; i <= maxClient; i++){
1024      sockfd = client[i];
1025      if(client[i] < 0) continue;
1026
1027      if(FD_ISSET(sockfd, &rset)){ //someone is readable
1028          ssize_t n, templen;
1029          char templetter;
1030          char *tempbuffer;
1031          tempbuffer = receivebuffer;
1032          for(n=1; n<MAXLINE; n++){
1033              if((templen = read(sockfd, &templetter, 1)) == 1){
1034                  *tempbuffer++ = templetter;
1035                  if(templetter == '\n') break;
1036              }
1037          }
1038          if(templetter == '\n') break;
1039      }
1040  }
1041
1042  if(i == maxClient) break;
1043
1044  if(numReady <= 0) continue;
1045
1046  for(i=0; i <= maxClient; i++){
1047      sockfd = client[i];
1048      if(client[i] < 0) continue;
1049
1050      if(F
```

```
server.c      x  daytimeserver.c      x  concurrentechoserver.c  x  client1.c      x  client4.c      x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <sys/select.h>
8 #include <unistd.h>
9 #include <sys/types.h>
10
11 #define backlog 10
12 #define MAXLINE 1024
13
14 int main(int argc, char *argv[]){
15     int serverPORT;
16     int sockfd, listenfd, clientfd;
17     char message[MAXLINE];
18     struct sockaddr_in serverAddress, clientAddress;
19     char *clientIP[backlog], *username[backlog];
20     int maxfd, maxClient, numReady, i, j;
21     socklen_t clientlen;
22     int client[backlog], clientPort[backlog], ban[backlog];//save all client's sockfd,
23     and port number
24     fd_set allset, rset; //for select
25     char sendbuffer[MAXLINE], receivebuffer[MAXLINE], receiver[MAXLINE];
26     char command[MAXLINE], text[MAXLINE];
27     char oldname[MAXLINE];
28     ssize_t len;
29
30     serverPORT = atoi(argv[1]);
31
32     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
33         printf("socket errore\n");
34         exit(1);
35     } //listen socket
36     bzero(&serverAddress, sizeof(serverAddress));
37     serverAddress.sin_family = AF_INET;
38     serverAddress.sin_port = htons(serverPORT); // port is unsigned short int, set
39     server is on port 8080
40     serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
41     //every ip to this server, s_addr is 32 bytes => long
42
43     if(bind(listenfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0){
44         printf("bind failed\n");
45         exit(1);
46     }
47     // bind server infomation to listen socket
48
49     listen(listenfd, backlog); //let listen be the socket to wait for connect, and up
50     to backlog
51
52     maxfd = listenfd; // current max fd is listen
53     maxClient = -1; // no client is recorded
54     for(int i=0; i<backlog; i++){
55         client[i] = -1;
56         ban[i] = 0;
57     } // no client
58     FD_ZERO(&allset); //clear allset
59     FD_SET(listenfd, &allset); // put listen socket in allset
60
61     for(;;){ //server run forever
62         rset = allset; // avoid change allset
63         numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
64 }
```

```
client.c  x  main.cpp  x  daytimeclient.c  x  server1.c  x  server2.c  x  server3.c  x
28
29 serverIP = argv[1];
30 serverPORT = atoi(argv[2]);
31
32 if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
33     printf("Socket error\n");
34     exit(1);
35 } //build socket
36
37 bzero(&serverAddress, sizeof(serverAddress));
38 serverAddress.sin_family = AF_INET; //ipv4
39 serverAddress.sin_port = htons(serverPORT); //sin_port is network short
40 int
41 if((inet_pton(AF_INET, serverIP, &serverAddress.sin_addr)) <= 0){
42     printf("inet_pton error\n");
43     exit(1);
44 }
45 //sin_addr is network type, inet_pton return in_addr type
46 if((connect(sockfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress
47 )))) < 0){
48     printf("Connect error\n");
49     exit(1);
50 }
51 // connet to server, need to be careful of sockaddr and sockaddr_in type
52 FD_ZERO(&rset);
53 stdineof = 0;
54
55 for(;;){
56     FD_SET(0, &rset);
57     FD_SET(sockfd, &rset); //set two fd_set
58
59     if(0 > sockfd) maxfdp1 = 0 + i;
60     else maxfdp1 = sockfd + 1; //update maxfd
61
62     select(maxfdp1, &rset, NULL, NULL, NULL); // listen socket and stdin is
63     readable or not
64
65     if(FD_ISSET(0, &rset)){
66         fgets(sendbuffer, MAXLINE, fp);
67         if(strcmp(sendbuffer, "exit\n") == 0){
68             stdineof = 1;
69             shutdown(sockfd, SHUT_WR);
70             FD_CLR(0, &rset);
71             continue;
72         }
73         write(sockfd, sendbuffer, strlen(sendbuffer));
74     }//something input
75
76     if(FD_ISSET(sockfd, &rset)){
77         ssize_t len;
78         len = read(sockfd, receivebuffer, MAXLINE);
79         if(len == 0){
80             if(stdineof == 1){
81                 exit(0);
82             }else{
83                 printf("Server terminate\n");
84                 exit(1);
85             }
86         }
87         fputs(receivebuffer, stdout);
88     }//server send something
89 }
90
91 return 0;
92 }
```

```
client.c x main.cpp x daytimeclient.c x server1.c x server2.c x server3.c x
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h> //socket,connect
5 #include <netinet/in.h> // htonl
6 #include <arpa/inet.h> //inet_nton
7 #include <sys/select.h>
8 #include <unistd.h> //read, write
9 #include <sys/types.h>
10
11 #define MAXLINE 1024
12
13 int main(int argc, char *argv[]){
14     char *serverIP;
15     int serverPORT;
16     int sockfd, maxfdpl;
17     struct sockaddr_in serverAddress;
18     fd_set rset;
19     char sendbuffer[MAXLINE], receivebuffer[MAXLINE];
20     FILE *fp = stdin;
21     int stdineof;
22
23     if(argc != 3){
24         printf("Incorrect input\n");
25         exit(1);
26     }
27
28     serverIP = argv[1];
29     serverPORT = atoi(argv[2]);
30
31     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
32         printf("Socket error\n");
33         exit(1);
34     } //build socket
35
36     bzero(&serverAddress, sizeof(serverAddress));
37     serverAddress.sin_family = AF_INET; //ipv4
38     serverAddress.sin_port = htons(serverPORT); //sin port is network short
39     int
40     if((inet_nton(AF_INET, serverIP, &serverAddress.sin_addr)) <= 0){
41         printf("inet_nton error\n");
42         exit(1);
43     }
44     //sin_addr is network type, inet_nton return in_addr type
45     if((connect(sockfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress
46     ))) < 0){
47         printf("Connect error\n");
48         exit(1);
49     }
50     // connect to server, need to be careful of sockaddr and sockaddr_in type
51     FD_ZERO(&rset);
52     stdineof = 0;
53
54     for(;){
55         FD_SET(0, &rset);
56         FD_SET(sockfd, &rset); //set two fd_set
57
58         if(0 > sockfd) maxfdpl = 0 + 1;
59         else maxfdpl = sockfd + 1; //update maxfd
60
61         select(maxfdpl, &rset, NULL, NULL, NULL); // listen socket and stdin is
62         readable or not
```