server.c | daytimeserver.c | concurrentechoserver.c | client1.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       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]);
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
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
124         //printf("friend number is %d\n", maxFriend);
125         //printf("friend max fd is %d\n", maxfd);
126
127         numReady--;
128         if(numReady <= 0)continue;
129       }
130       //save friend info
131
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         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 | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```
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;
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <unistd.h>

#define MAXLINE 1024
#define backlog 10

int main(int argc, char *argv[]){
  int myPORT, friendPORT;
  char *myIP, *friendIP, *myNAME;
  char sendbuffer[MAXLINE], receivebuffer[MAXLINE], receiver[MAXLINE], sendMessage[
    MAXLINE], message[MAXLINE];
  char *friendNAME[backlog];
  struct sockaddr_in myAddr, friendAddr;
  int mysockfd, maxfd, maxFriend, numReady, friendfd;
  int friend[backlog];
  fd_set allset, rset;
  FILE *fp = stdin;
  int i,j;

  myIP = "127.0.0.1";
  myPORT = atoi(argv[2]);
  myNAME = argv[1];
  mysockfd = socket(AF_INET, SOCK_STREAM, 0);

  bzero(&myAddr, sizeof(myAddr));
  myAddr.sin_family = AF_INET;
  myAddr.sin_port = htons(myPORT);
  inet_pton(AF_INET, myIP, &myAddr.sin_addr);
  // my information

  if(bind(mysockfd, (struct sockaddr *)&myAddr, sizeof(myAddr)) < 0){
    printf("bind error\n");
    exit(1);
  }
  // bind my infor to socket

  listen(mysockfd, backlog);
  maxfd = mysockfd;
  maxFriend = -1;
  for(i=0; i<backlog; i++){
    friend[i] = -1;
  }

  FD_ZERO(&allset);
  FD_SET(mysockfd, &allset);
  FD_SET(fileno(fp), &allset);
  if(fileno(fp) > maxfd) maxfd = fileno(fp);

  for(i=3; i<argc; i++){
    friendPORT = atoi(argv[i]);
    friendIP = "127.0.0.1";
    //printf("try to connect port %d\n", friendPORT);
    friendfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&friendAddr, sizeof(friendAddr));
    friendAddr.sin_family = AF_INET;
    friendAddr.sin_port = htons(friendPORT);
    inet_pton(AF_INET, friendIP, &friendAddr.sin_addr);

```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```
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 | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```
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
66    if(FD_ISSET(listenfd, &rset)){
67      connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
68
69      for(c=0; c<10; c++){
70        if(client[c] < 0){
71          client[c] = connfd;
72          break;
73        }
74      }
75      if(c == 10){
76        printf("Too many client\n");
77        exit(1);
78      }
79      FD_SET(connfd, &allset);
80      if(connfd > maxfd) maxfd = connfd;
81      if(c > maxClient) maxClient = c;
82
83      numReady--;
84      if(numReady <= 0)continue;
85    }
86
87    for(c=0; c <= maxClient; c++){
88      if(client[c] < 0)continue;
89
90      if(FD_ISSET(client[c], &rset)){
91        ssize_t len;
92        len = read(client[c], &buffer, MAXLINE);
93
94        if(len == 0){
95          close(client[c]);
96          FD_CLR(client[c], &allset);
97          client[c] = -1;
98        }else{
99          if(buffer[strlen(buffer) - 1] == '\n'){
100         buffer[strlen(buffer) - 1] = '\0';
101         }
102         //printf("read done\n");
103         char *ptr = strtok(buffer, " ");
104         int i = 0;
105         strcpy(cal, ptr);
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++){
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <limits.h>
#include <sys/select.h>

#define MAXLINE 1024

int main(int argc, char *argv[]){
  int listenfd, connfd;
  struct sockaddr_in server;
  char buffer[MAXLINE];
  char cal[MAXLINE];
  int arg[10];
  unsigned long int ans;
  int serverPORT;
  char send[MAXLINE];
  int client[10];
  int maxfd, maxClient;
  fd_set allset, rset;
  int numReady;
  int c;

  if(argc != 2){
    printf("wrong input\n");
    exit(1);
  }

  serverPORT = atoi(argv[1]);

  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  server.sin_addr.s_addr = htonl(INADDR_ANY);

  if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("bind error\n");
    exit(1);
  }

  listen(listenfd, 10);

  maxfd = listenfd;
  maxClient = -1;

  for(c=0; c<10; c++){
    client[c] = -1;
  }

  FD_ZERO(&allset);
  FD_SET(listenfd, &allset);

  for(;;){
    rset = allset;
    numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
        //printf("read done\n");
        char *ptr = strtok(buffer, " ");
        int i = 0;
        strcpy(cal, ptr);
        //printf("%s\n", cal);

        while(1){
          char temp[MAXLINE];
          ptr = strtok(NULL, " ");
          if(ptr == NULL)break;
          strcpy(temp, ptr);
          //printf("%s\n", temp);
          arg[i] = atoi(temp);
          //printf("%d\n", arg[i]);
          i++;
        }

        //printf("%s\n", cal);
        //for(int k=0; k<i; k++){
        //  printf("%d ",arg[k]);
        //}

        if(strcmp(cal, "ADD") == 0){
          ans = 0;
          for(int j=0; j<i; j++){
            ans += arg[j];
          }
          if(ans > UINT_MAX){
            strcpy(send, "Overflowed\n");
            write(connfd, send, sizeof(send));
            continue;
          }
          sprintf(send, "%lu\n", ans);
          write(connfd, send, sizeof(send));
        }else if(strcmp(cal, "MUL") == 0){
          ans = 1;
          for(int j=0; j<i; j++){
            ans = ans * arg[j];
          }
          if(ans > UINT_MAX){
            strcpy(send, "Overflowed\n");
            write(connfd, send, sizeof(send));
            continue;
          }
          sprintf(send, "%lu\n", ans);
          write(connfd, send, sizeof(send));
        }else if(strcmp(cal, "EXIT") == 0){
          close(connfd);
          break;
        }else{
          sprintf(send, "wrong command\n");
          write(connfd, send, sizeof(send));
        }
      }
      printf("Client has closed the connection\n");
      exit(0);
    }else{
      close(connfd);
    }
  }
}

```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
    printf("bind error\n");
    exit(1);
  }

  listen(listenfd, 10);
  signal(SIGCHLD, sig_fork);

  for(;;){
    connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
    if((pid = fork()) == 0){
      close(listenfd);
      for(;;){
        ssize_t len;
        len = read(connfd, &buffer, MAXLINE);
        if(len == 0){
          break;
        }

        if(buffer[strlen(buffer) - 1] == '\n'){
          buffer[strlen(buffer) - 1] = '\0';
        }
        //printf("read done\n");
        char *ptr = strtok(buffer, " ");
        int i = 0;
        strcpy(cal, ptr);
        //printf("%s\n", cal);

        while(1){
          char temp[MAXLINE];
          ptr = strtok(NULL, " ");
          if(ptr == NULL)break;
          strcpy(temp, ptr);
          //printf("%s\n", temp);
          arg[i] = atoi(temp);
          //printf("%d\n", arg[i]);
          i++;
        }

        //printf("%s\n", cal);
        //for(int k=0; k<i; k++){
        //  printf("%d ",arg[k]);
        //}

        if(strcmp(cal, "ADD") == 0){
          ans = 0;
          for(int j=0; j<i; j++){
            ans += arg[j];
          }
          if(ans > UINT_MAX){
            strcpy(send, "Overflowed\n");
            write(connfd, send, sizeof(send));
            continue;
          }
          sprintf(send, "%lu\n", ans);
          write(connfd, send, sizeof(send));
        }else if(strcmp(cal, "MUL") == 0){
          ans = 1;
          for(int j=0; j<i; j++){
            ans = ans * arg[j];
          }
          if(ans > UINT_MAX){
            strcpy(send, "Overflowed\n");
            write(connfd, send, sizeof(send));
            continue;
          }
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <limits.h>
#include <signal.h>
#include <sys/wait.h>

#define MAXLINE 1024

void sig_fork(int signo){
  pid_t pid;
  int status;
  while((pid = waitpid(-1, &status, WNOHANG)) > 0){
    printf("Child server process PID=%d has terminated\n", pid);
  }
  return;
}

int main(int argc, char *argv[]){
  int listenfd, connfd;
  struct sockaddr_in server;
  char buffer[MAXLINE];
  char cal[MAXLINE];
  int arg[10];
  unsigned long int ans;
  int serverPORT;
  char send[MAXLINE];
  pid_t pid;

  if(argc != 2){
    printf("wrong input\n");
    exit(1);
  }

  serverPORT = atoi(argv[1]);

  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  server.sin_addr.s_addr = htonl(INADDR_ANY);

  if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("bind error\n");
    exit(1);
  }

  listen(listenfd, 10);
  signal(SIGCHLD, sig_fork);

  for(;;){
    connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
    if((pid = fork()) == 0){
      close(listenfd);
      for(;;){
        ssize_t len;
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```
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 | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <limits.h>

#define MAXLINE 1024

int main(int argc, char *argv[]){
  int listenfd, connfd;
  struct sockaddr_in server;
  char buffer[MAXLINE];
  char cal[MAXLINE];
  int arg[10];
  unsigned long int ans;
  int serverPORT;
  char send[MAXLINE];

  if(argc != 2){
    printf("wrong input\n");
    exit(1);
  }

  serverPORT = atoi(argv[1]);

  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  server.sin_addr.s_addr = htonl(INADDR_ANY);

  if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("bind error\n");
    exit(1);
  }

  listen(listenfd, 1);

  for(;;){
    connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
    for(;;){
      ssize_t len;
      len = read(connfd, &buffer, MAXLINE);
      if(len == 0){
        break;
      }

      if(buffer[strlen(buffer) - 1] == '\n'){
        buffer[strlen(buffer) - 1] = '\0';
      }
      //printf("read done\n");
      char *ptr = strtok(buffer, " ");
      int i = 0;
      strcpy(cal, ptr);
      //printf("%s\n", cal);

```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

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

#define MAXLINE 1024

int main(int argc, char *argv[]){
  char *serverIP;
  int serverPORT;
  int sockfd;
  struct sockaddr_in server;
  char buffer[MAXLINE], receivebuffer[MAXLINE];
  FILE *fp = stdin;

  if(argc != 3){
    printf("wrong input\n");
    exit(1);
  }

  serverIP = argv[1];
  serverPORT = atoi(argv[2]);

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  inet_pton(AF_INET, serverIP, &server.sin_addr);

  if(connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("connect error\n");
    exit(1);
  }

  for(;;){
    fgets(buffer, MAXLINE, fp);
    write(sockfd, buffer, sizeof(buffer));

    ssize_t len;
    len = read(sockfd, receivebuffer, MAXLINE);
    if(len == 0){
      printf("The server has closed the connection\n");
      close(sockfd);
      exit(0);
    }
    fputs(receivebuffer, stdout);
  }
}
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

#define backlog 10
#define MAXLINE 1024

void sig_fork(int signo){
  pid_t pid;
  int status;
  while((pid = waitpid(-1, &status, WNOHANG)) > 0){
    printf("child %d terminated\n", pid);
  }
  return;
}

int main(int argc, char *argv[]){
  int serverPORT;
  int listenfd, connfd;
  char buffer[MAXLINE];
  struct sockaddr_in server;
  pid_t pid;

  serverPORT = atoi(argv[1]);
  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  server.sin_addr.s_addr = htonl(INADDR_ANY);

  if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("bind error\n");
    exit(1);
  }

  listen(listenfd, backlog);
  signal(SIGCHLD, sig_fork);
  for(;;){
    connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
    if((pid = fork()) == 0){
      close(listenfd);
      for(;;){
        read(connfd, &buffer, MAXLINE);
        if(strcmp(buffer, "exit\n") == 0){
          break;
        }
        write(connfd, buffer, sizeof(buffer));
      }
      close(connfd);
      exit(0);
    }else{
      close(connfd);
    }
  }
}
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>

#define backlog 10
#define MAXLINE 1024

int main(int agrc, char *argv[]){
  int listenfd, connfd;
  struct sockaddr_in server;
  char buffer[MAXLINE];
  int serverPORT;

  serverPORT = atoi(argv[1]);

  time_t ticks;
  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  server.sin_addr.s_addr = htonl(INADDR_ANY);

  if(bind(listenfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("bind failed\n");
    exit(1);
  }

  listen(listenfd, backlog);

  for(;;){
    connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);
    ticks = time(NULL);
    sprintf(buffer, "%.24s\n", ctime(&ticks));
    write(connfd, buffer, sizeof(buffer));
    close(connfd);
  }
}
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

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

#define MAXLINE 1024

int main(int argc, char *argv[]){
  char *serverIP;
  int serverPORT;
  int sockfd;
  struct sockaddr_in server;
  char sendbuffer[MAXLINE], receivebuffer[MAXLINE];
  int n;
  FILE *fp = stdin;

  serverIP = argv[1];
  serverPORT = atoi(argv[2]);

  if(argc != 3){
    printf("Wrong input\n");
    exit(1);
  }

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket error\n");
    exit(1);
  }

  bzero(&server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = htons(serverPORT);
  if((inet_pton(AF_INET, serverIP, &server.sin_addr)) <= 0){
    printf("inet_pton error\n");
    exit(1);
  }

  if(connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0){
    printf("connect error\n");
    exit(1);
  }

  for(;;){
    fgets(sendbuffer, MAXLINE, fp);
    write(sockfd, sendbuffer, sizeof(sendbuffer));

    ssize_t len;
    len = read(sockfd, receivebuffer, MAXLINE);
    if(len == 0){
      printf("connection closed\n");
      exit(1);
    }
    fputs(receivebuffer, stdout);
  }
  exit(0);
}
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```
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] << "-----------------"
        << endl;
37
38    while(getline(inputFile, line)){
39      string word;
40      string command;
41      istringstream iss(line);
42      iss >> command;
43      istringstream iss2(line);
44      while(iss2 >> word){
45        cout << word << " ";
46      }
47      cout << endl;
48      if(command == "reverse"){
49        reverse(word);
50      }else if(command == "split"){
51        split(word, cut);
52      }
53    }
54
55    cout << "-------------------End of input file " << argv[1] << "----------"
        << endl;
56    cout << "********************User input*******************";
57    for(int i=0; i<filelen; i++){
58      cout << "*";
59    }
60    cout << endl;
61
62    string input;
63    while(getline(cin, input)){
64      istringstream iss3(input);
65      string userCommand;
66      string userWord;
67      iss3 >> userCommand;
68      if(userCommand == "exit"){
69        return 0;
70      }else if(userCommand == "reverse"){
71        while(iss3 >> userWord){
72          reverse(userWord);
73        }
74      }else if(userCommand == "split"){
75        while(iss3 >> userWord){
76          split(userWord, cut);
77        }
78      }
79    }
80  }
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```cpp
#include<bits/stdc++.h>

using namespace std;

void reverse(string &word){
  string output = "";
  int len = word.length();
  for(int i=len-1; i>=0; i--){
    output = output + word[i];
  }
  cout << output << endl;
}

void split(string &word, string &cut){
  char *words = new char[word.length() + 1];
  strcpy(words, word.c_str());

  char *cuts = new char[cut.length() + 1];
  strcpy(cuts, cut.c_str());

  char *pch = strtok(words, cuts);
  while(pch != NULL){
    cout << pch << " ";
    pch = strtok(NULL, cuts);
  }
  cout << endl;
}

int main(int argc, char *argv[]){
  ifstream inputFile;
  inputFile.open(argv[1], ifstream::in);
  string line;
  string cut = argv[2];
  int filelen = strlen(argv[1]);

  cout << "-------------------Input file "<< argv[1] << "-----------------"
    << endl;

  while(getline(inputFile, line)){
    string word;
    string command;
    istringstream iss(line);
    iss >> command;
    istringstream iss2(line);
    while(iss2 >> word){
      cout << word << " ";
    }
    cout << endl;
    if(command == "reverse"){
      reverse(word);
    }else if(command == "split"){
      split(word, cut);
    }
  }

  cout << "-------------------End of input file " << argv[1] << "----------"
    << endl;
  cout << "********************User input********************";
  for(int i=0; i<filelen; i++){
    cout << "*";
  }
  cout << endl;
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```c
            }

            for(j=0; j <= maxClient; j++){
              if(strcmp(receiver, username[j]) == 0){
                break;
              }
            } // user[j] is receiver

            if(j > maxClient){
              strcpy(message, "[Server] ERROR: The receiver doesn't exist.\n");
              write(client[i], message, sizeof(message));
              continue;
            }else if(ban[j] == 1){
              sprintf(message, "that user has been banned\n");
              write(client[i],message, sizeof(message));
              continue;
            }else{
              strcpy(message, "[Server] SUCCESS: Your message has been sent.\n");
              write(client[i], message, sizeof(message));
              sprintf(message, "[Server] %s tell you %s\n", username[i], sendbuffer);
              write(client[j], message, sizeof(message));
              continue;
            }
          }else if(strcmp(command, "yell") == 0){
            printf("someone input yell\n");
            for(j=0; j <= maxClient; j++){
              if(client[j] >= 0 && ban[j] == 0){
                sprintf(message, "[Server] %s yell %s\n", username[i], text);
                write(client[j], message, sizeof(message));
              }
            }
          }else if(strcmp(command, "ban") == 0){
            printf("bannnnn\n");
            for(j=0; j<= maxClient; j++){
              printf("%s\n", username[j]);
              if(strcmp(text, username[j]) == 0){
                printf("find\n");
                ban[j] = 1;
                sprintf(message, "you have ban %s\n", username[j]);
                write(client[i], message, sizeof(message));
                break;
              }
            }
            if(j > maxClient){
              sprintf(message, "wrong user\n");
              write(client[i], message, sizeof(message));
            }
          }else{
            strcpy(message, "[Server] ERROR: Error command.\n");
            write(client[i], message, sizeof(message));
          }
        }

        numReady--;
        if(numReady <= 0)break;
      }
    }
  }

  return 0;
}
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```
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",
                    text);
177               write(client[i], message, sizeof(message));
178               break;
179             }
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
                    English letters.\n");
192             write(client[i], message, sizeof(message));
193             continue;
194           }
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]
                = '\0'; // the user want to send
218
219           strcpy(sendbuffer, text + strlen(receiver) + 1);
220           if(sendbuffer[strlen(sendbuffer) - 1] == '\n')sendbuffer[strlen(
                sendbuffer) - 1] = '\0'; // the message want to send
221
222           if(strcmp(receiver, "anonymous") == 0){
223             strcpy(message, "[Server] ERROR: The client to which you sent is
                    anonymous.\n");
224             write(client[i], message, sizeof(message));
225             continue;
226           }
227
228           for(j=0; j <= maxClient; j++){
229             if(strcmp(receiver, username[j]) == 0){
230               break;
231             }
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```
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]
                      , clientPort[j]);
161                 write(client[i], message, sizeof(message));
162               }else if(client[j] >= 0){
163                 sprintf(message, "[Server] %s %s:%d\n", username[j], clientIP[j],
                      clientPort[j]);
164                 write(client[i], message, sizeof(message));
165               }
166             }
167           }else if(strcmp(command, "name") == 0){
168             printf("someone input name\n");
169             if(strcmp(text, "anonymous") == 0){
170               strcpy(message, "[Server] ERROR: Username cannot be anonymous.\n");
171               write(client[i], message, sizeof(message));
172               continue;
173             } // no anonymous
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

```
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],
                clientPort[j]);
92            write(client[j], message, sizeof(message));
93          }else if(client[j] >= 0){
94            strcpy(message, "[Server] Someone is coming!\n");
95            write(client[j], message, sizeof(message));
96          }
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;
```

server.c | daytimeserver.c | concurrentechoserver.c | client1.c | client4.c

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

#define backlog 10
#define MAXLINE 1024

int main(int argc, char *argv[]){
  int serverPORT;
  int sockfd, listenfd, clientfd;
  char message[MAXLINE];
  struct sockaddr_in serverAddress, clientAddress;
  char *clientIP[backlog], *username[backlog];
  int maxfd, maxClient, numReady, i, j;
  socklen_t clientlen;
  int client[backlog], clientPort[backlog], ban[backlog];//save all client's sockfd, and port number
  fd_set allset, rset; //for select
  char sendbuffer[MAXLINE], receivebuffer[MAXLINE], receiver[MAXLINE];
  char command[MAXLINE], text[MAXLINE];
  char oldname[MAXLINE];
  ssize_t len;

  serverPORT = atoi(argv[1]);

  if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket erroe\n");
    exit(1);
  } //listen socket
  bzero(&serverAddress, sizeof(serverAddress));
  serverAddress.sin_family = AF_INET;
  serverAddress.sin_port = htons(serverPORT); // port is unsigned short int, set server is on port 8080
  serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
  //every ip to this server, s_addr is 32 bytes => long

  if(bind(listenfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0){
    printf("bind failed\n");
    exit(1);
  }
  // bind server infomation to listen socket

  listen(listenfd, backlog); //let listen be the socket to wait for connect, and up to backlog

  maxfd = listenfd; // current max fd is listen
  maxClient = -1; // no client is recorded
  for(int i=0; i<backlog; i++){
    client[i] = -1;
    ban[i] = 0;
  } // no client
  FD_ZERO(&allset); //clear allset
  FD_SET(listenfd, &allset); // put listen socket in allset

  for(;;){ //server run forever
    rset = allset; // avoid change allset
    numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
  serverIP = argv[1];
  serverPORT = atoi(argv[2]);

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("Socket error\n");
    exit(1);
  } //build socket

  bzero(&serverAddress, sizeof(serverAddress));
  serverAddress.sin_family = AF_INET; //ipv4
  serverAddress.sin_port = htons(serverPORT); //sin_port is network short int
  if((inet_pton(AF_INET, serverIP, &serverAddress.sin_addr)) <= 0){
    printf("inet_pton error\n");
    exit(1);
  }
  //sin_addr is network type, inet_pton return in_addr type

  if((connect(sockfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress))) < 0){
    printf("Connect error\n");
    exit(1);
  }
  // connet to server, need to be careful of sockaddr and sockaddr_in type
  FD_ZERO(&rset);
  stdineof = 0;

  for(;;){
    FD_SET(0, &rset);
    FD_SET(sockfd, &rset); //set two fd_set

    if(0 > sockfd) maxfdp1 = 0 + 1;
    else maxfdp1 = sockfd + 1; //update maxfd

    select(maxfdp1, &rset, NULL, NULL, NULL); // listen socket and stdin is readable or not

    if(FD_ISSET(0, &rset)){
      fgets(sendbuffer, MAXLINE, fp);
      if(strcmp(sendbuffer, "exit\n") == 0){
        stdineof = 1;
        shutdown(sockfd, SHUT_WR);
        FD_CLR(0, &rset);
        continue;
      }//exit
      write(sockfd, sendbuffer, strlen(sendbuffer));
    }//something input

    if(FD_ISSET(sockfd, &rset)){
      ssize_t len;
      len = read(sockfd, receivebuffer, MAXLINE);
      if(len == 0){
        if(stdineof == 1){
          exit(0);
        }else{
          printf("Server terminate\n");
          exit(1);
        }
      }
      fputs(receivebuffer, stdout);
    }//server send something
  }
  return 0;
}
```

client.c | main.cpp | daytimeclient.c | server1.c | server2.c | server3.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket,connect
#include <netinet/in.h> // hton
#include <arpa/inet.h> //inet_pton
#include <sys/select.h>
#include <unistd.h> //read, write
#include <sys/types.h>

#define MAXLINE 1024

int main(int argc, char *argv[]){
  char *serverIP;
  int serverPORT;
  int sockfd, maxfdp1;
  struct sockaddr_in serverAddress;
  fd_set rset;
  char sendbuffer[MAXLINE], receivebuffer[MAXLINE];
  FILE *fp = stdin;
  int stdineof;

  if(argc != 3){
    printf("Incorrect input\n");
    exit(1);
  }

  serverIP = argv[1];
  serverPORT = atoi(argv[2]);

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    printf("Socket error\n");
    exit(1);
  } //build socket

  bzero(&serverAddress, sizeof(serverAddress));
  serverAddress.sin_family = AF_INET; //ipv4
  serverAddress.sin_port = htons(serverPORT); //sin_port is network short
  int
  if((inet_pton(AF_INET, serverIP, &serverAddress.sin_addr)) <= 0){
    printf("inet_pton error\n");
    exit(1);
  }
  //sin_addr is network type, inet_pton return in_addr type

  if((connect(sockfd, (struct sockaddr *)&serverAddress, sizeof(serverAddress
    ))) < 0){
    printf("Connect error\n");
    exit(1);
  }
  // connet to server, need to be careful of sockaddr and sockaddr_in type
  FD_ZERO(&rset);
  stdineof = 0;

  for(;;){
    FD_SET(0, &rset);
    FD_SET(sockfd, &rset); //set two fd_set

    if(0 > sockfd) maxfdp1 = 0 + 1;
    else maxfdp1 = sockfd + 1; //update maxfd

    select(maxfdp1, &rset, NULL, NULL, NULL); // listen socket and stdin is
    readable or not
```