< client.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

		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;
}
```

< server.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);

		if(FD_ISSET(listenfd, &rset)){ //listen socket is readable = have client
			clientlen = sizeof(clientAddress);
			clientfd = accept(listenfd, (struct sockaddr *)&clientAddress, &clientlen);
			// accept a new client and give it a new socket

			for(i=0; i<backlog; i++){
				if(client[i] < 0){
					client[i] = clientfd;
					break;
				}
			} // record i is what fd
			if(i == backlog){
				printf("To many client\n");
				exit(1);
			}
			//printf("client is fd %d\n", i);

			clientIP[i] = malloc(INET_ADDRSTRLEN);
			inet_ntop(AF_INET, &clientAddress.sin_addr, clientIP[i], INET_ADDRSTRLEN);
			clientPort[i] = ntohs(clientAddress.sin_port);
			username[i] = malloc(MAXLINE * sizeof(char));
			strcpy(username[i], "anonymous"); //client information finished

			FD_SET(clientfd, &allset);
			if(clientfd > maxfd) maxfd = clientfd;
			if(i > maxClient) maxClient = i;

			for(j=0; j <= maxClient; j++){
				if(clientfd == client[j]){
					sprintf(message, "[Server] Hello, anonymous! From: %s:%d\n", clientIP[j],
clientPort[j]);
					write(client[j], message, sizeof(message));
				}else if(client[j] >= 0){
					strcpy(message, "[Server] Someone is coming!\n");
					write(client[j], message, sizeof(message));
				}
			}

			numReady--;
			if(numReady <= 0) continue;
		} //connect handle finish

		for(i=0; i <= maxClient; i++){
			sockfd = client[i];
			if(client[i] < 0)continue;

			if(FD_ISSET(sockfd, &rset)){ //someone is readable
				ssize_t n, templen;
				char templetter;
				char *tempbuffer;
				tempbuffer = receivebuffer;
				for(n=1; n<MAXLINE; n++){
					if((templen = read(sockfd, &templetter, 1)) == 1){
						*tempbuffer++ = templetter;
						if(templetter == '\n') break;
					}else if(templen == 0){
						*tempbuffer = '\0';
						n--;
```

```
                                        break;
                                }
                        }
                        *tempbuffer = '\0';
                        len = n;
                        printf("receive length %zd\n", len);

                        if(len == 0){ //client closed
                                printf("client port %d leave\n", clientPort[i]);

                                for(j = 0; j <= maxClient; j++){
                                        if(client[j] == sockfd){
                                                continue;
                                        }else if(client[j] >= 0){
                                                sprintf(message, "[Server] %s is offline.\n", username[i]);
                                                write(client[j], message, sizeof(message));
                                        }
                                }

                                close(sockfd);
                                FD_CLR(sockfd, &allset);
                                client[i] = -1;
                                free(clientIP[i]);
                                free(username[i]);

                        }else{
                                if(ban[i] == 1){
                                        sprintf(message, "you can not send message\n");
                                        write(client[i], message, sizeof(message));
                                        continue;
                                }
                                printf("line: %s\n", receivebuffer);
                                strcpy(command, strtok(receivebuffer, " "));
                                if(command[strlen(command) - 1] == '\n') command[strlen(command) - 1] = '\0';
                                strcpy(text, receivebuffer + strlen(command) +1 );
                                if(text[strlen(text) - 1] == '\n') text[strlen(text) - 1] = '\0';

                                if(strcmp(command, "who") == 0){
                                        printf("someone input who\n");
                                        for(j=0; j <= maxClient; j++){
                                                if(client[j] == sockfd){
                                                        sprintf(message, "[Server] %s %s:%d ->me\n",
username[j], clientIP[j], clientPort[j]);
                                                        write(client[i], message, sizeof(message));
                                                }else if(client[j] >= 0){
                                                        sprintf(message, "[Server] %s %s:%d\n", username[j],
clientIP[j], clientPort[j]);
                                                        write(client[i], message, sizeof(message));
                                                }
                                        }
                                }else if(strcmp(command, "name") == 0){
                                        printf("someone input name\n");
                                        if(strcmp(text, "anonymous") == 0){
                                                strcpy(message, "[Server] ERROR: Username cannot be
anonymous.\n");
                                                write(client[i], message, sizeof(message));
                                                continue;
                                        } // no anonymous
                                        for(j=0; j <= maxClient; j++){
                                                if(strcmp(text, username[j]) == 0 && client[j] != sockfd){
                                                        sprintf(message, "[Server] ERROR: %s has been used
by others.\n", text);
                                                        write(client[i], message, sizeof(message));
                                                        break;
                                                }
                                        }
                                        if(j <= maxClient) continue; // no the same

                                        int allenglish = 1;
                                        for(j=0; j < strlen(text); j++){
```

```
                                        if(text[j] < 'A' || text[j] > 'z'){
                                                allenglish = 0;
                                                break;
                                        }
                                }
                                if(strlen(text) < 2 || strlen(text) > 12 || allenglish == 0){
                                        strcpy(message, "[Server] ERROR: Username can only consists
of 2~12 English letters.\n");
                                        write(client[i], message, sizeof(message));
                                        continue;
                                }

                                strcpy(oldname, username[i]);
                                strcpy(username[i], text); //update username

                                for(j=0; j <= maxClient; j++){
                                        if(client[j] == sockfd){
                                                sprintf(message, "[Server] You're now known as
%s.\n", text);
                                                write(client[i], message, sizeof(message));
                                        }else if(client[j] >= 0 && ban[j] == 0){
                                                sprintf(message, "[Server] %s is now known as
%s.\n", oldname, text);
                                                write(client[j], message, sizeof(message));
                                        }
                                }
                        }else if(strcmp(command, "tell") == 0){
                                printf("someone input tell\n");
                                if(strcmp(username[i], "anonymous") == 0){
                                        strcpy(message, "[Server] ERROR: You are anonymous.\n");
                                        write(client[i], message, sizeof(message));
                                        continue;
                                }

                                strcpy(receiver, strtok(text, " "));
                                if(receiver[strlen(receiver) - 1] == '\n')receiver[strlen(receiver) - 1] =
'\0'; // the user want to send

                                strcpy(sendbuffer, text + strlen(receiver) + 1);
                                if(sendbuffer[strlen(sendbuffer) - 1] == '\n')sendbuffer[strlen(sendbuffer) -
1] = '\0'; // the message want to send

                                if(strcmp(receiver, "anonymous") == 0){
                                        strcpy(message, "[Server] ERROR: The client to which you sent
is anonymous.\n");
                                        write(client[i], message, sizeof(message));
                                        continue;
                                }

                                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;
}
```

< main.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;

	string input;
	while(getline(cin, input)){
		istringstream iss3(input);
		string userCommand;
		string userWord;
		iss3 >> userCommand;
		if(userCommand == "exit"){
			return 0;
		}else if(userCommand == "reverse"){
			while(iss3 >> userWord){
				reverse(userWord);
			}
		}else if(userCommand == "split"){
			while(iss3 >> userWord){
				split(userWord, cut);
			}
		}
	}
}
```

< daytime client .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);
}
```

< daytime server.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);
	}
}
```

< concurrentserver.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 <time.h>

#define backlog 10
#define MAXLINE 1024

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

	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 failed\n");
		exit(1);
	}

	listen(listenfd, backlog);

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

```
                        sprintf(buffer, "%.24s\n", ctime(&ticks));
                        write(connfd, buffer, sizeof(buffer));
                        close(connfd);
                        exit(0);
                }else{
                        close(connfd);
                }

        }
}
```

< concurrent echo server .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);
                }
        }
}
```

< client1 .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 1 .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);

			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++;
			}
```

```
                        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));
                        }

                }
        }
}
```

< server2.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;
			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++;
			}


			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);
                        }
                }
}
```

< server3.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);

	if(FD_ISSET(listenfd, &rset)){
		connfd = accept(listenfd, (struct sockaddr *)NULL, NULL);

		for(c=0; c<10; c++){
			if(client[c] < 0){
				client[c] = connfd;
				break;
			}
		}
		if(c == 10){
			printf("Too many client\n");
			exit(1);
		}
		FD_SET(connfd, &allset);
		if(connfd > maxfd) maxfd = connfd;
		if(c > maxClient) maxClient = c;

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

	for(c=0; c <= maxClient; c++){
		if(client[c] < 0)continue;

		if(FD_ISSET(client[c], &rset)){
			ssize_t len;
			len = read(client[c], &buffer, MAXLINE);

			if(len == 0){
				close(client[c]);
				FD_CLR(client[c], &allset);
				client[c] = -1;
			}else{
				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++;
                                }
                                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(client[c], send, sizeof(send));
                                                continue;
                                        }
                                        sprintf(send, "%lu\n", ans);
                                        write(client[c], 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(client[c], send, sizeof(send));
                                                continue;
                                        }
                                        sprintf(send, "%lu\n", ans);
                                        write(client[c], send, sizeof(send));
                                }else if(strcmp(cal, "EXIT") == 0){
                                        close(client[c]);
                                        FD_CLR(client[c], &allset);
                                        client[c] = -1;
                                        break;
                                }else{
                                        sprintf(send, "wrong command\n");
                                        write(client[c], send, sizeof(send));
                                }
                        }
                        numReady--;
                        if(numReady <= 0)break;
                }
        }
}
```

< client4.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);

        if(connect(friendfd, (struct sockaddr *)&friendAddr, sizeof(friendAddr)) < 0){
                printf("Connect error\n");
                exit(1);
        }
        for(j=0; j<backlog; j++){
                if(friend[j] < 0){
                        friend[j] = friendfd;
                        break;
                }
        }

        //printf("connect success\n");
        write(friendfd, myNAME, sizeof(myNAME));
        read(friendfd, &receivebuffer, MAXLINE);
        //printf("port %d's name is %s\n", friendPORT, receivebuffer);
        friendNAME[j] = malloc(MAXLINE * sizeof(char));
        strcpy(friendNAME[j], receivebuffer);
        //printf("copy name done\n");
        if(friendfd > maxfd) maxfd = friendfd;
        if(j > maxFriend) maxFriend = j;
        FD_SET(friendfd, &allset);

        //printf("connection done\n");
}
// connect to everybody

//printf("friend number is %d\n", maxFriend);
```

```
//printf("friend max fd is %d\n", maxfd);

for(;;){
	rset = allset;
	numReady = select(maxfd + 1, &rset, NULL, NULL, NULL);
	//printf("select done once\n");

	if(FD_ISSET(mysockfd, &rset)){
		//printf("someone try to connect\n");
		friendfd = accept(mysockfd, (struct sockaddr *)NULL, NULL);
		write(friendfd, myNAME, sizeof(myNAME));

		for(i=0; i<backlog; i++){
			if(friend[i] < 0){
				friend[i] = friendfd;
				break;
			}
		}
		if(i == backlog){
			printf("Too many friends\n");
			exit(1);
		}

		read(friendfd, &receivebuffer, MAXLINE);
		//printf("receive name %s\n", receivebuffer);
		friendNAME[i] = malloc(MAXLINE * sizeof(char));
		strcpy(friendNAME[i], receivebuffer);

		FD_SET(friendfd, &allset);
		if(friendfd > maxfd) maxfd = friendfd;
		if(i > maxFriend) maxFriend = i;

		//printf("friend number is %d\n", maxFriend);
		//printf("friend max fd is %d\n", maxfd);

		numReady--;
		if(numReady <= 0)continue;
	}
	//save friend info

	if(FD_ISSET(fileno(fp), &rset)){
		fgets(sendbuffer, MAXLINE, fp);
		if(strcmp(sendbuffer, "EXIT\n") == 0){
			for(i=0; i<backlog; i++){
				if(friend[i] >= 0){
					close(friend[i]);
				}
			}
			exit(0);
		}
		strcpy(receiver, strtok(sendbuffer, " "));
		if(receiver[strlen(receiver) - 1] == '\n')receiver[strlen(receiver) - 1] = '\0';
		strcpy(sendMessage, sendbuffer + strlen(receiver) + 1);
		if(sendMessage[strlen(sendMessage) - 1] == '\n')sendMessage[strlen(sendMessage) - 1] = '\0';

		if(strcmp(receiver, "ALL") == 0){
			for(i=0; i<backlog; i++){
				if(friend[i] >= 0){
					sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
					write(friend[i], message, sizeof(message));
				}
			}
		}else{
			for(i=0; i<backlog; i++){
				if(friend[i] >= 0 && strcmp(friendNAME[i], receiver) == 0){
					sprintf(message, "%s SAID %s\n", myNAME, sendMessage);
					write(friend[i], message, sizeof(message));
					break;
				}
			}
```

```
                                        if(i >= backlog){
                                                sprintf(message, "No user\n");
                                                fputs(message, stdout);
                                        }
                                }
                        }
                        //user input

                        for(i=0; i<=maxFriend; i++){
                                //printf("some has thing to say\n");
                                if(friend[i] < 0) continue;
                                if(FD_ISSET(friend[i], &rset)){
                                        ssize_t len;
                                        len = read(friend[i], receivebuffer, MAXLINE);
                                        if(len == 0){
                                                printf("%s has left the chat room\n", friendNAME[i]);
                                                close(friend[i]);
                                                FD_CLR(friend[i], &allset);
                                                friend[i] = -1;
                                                free(friendNAME[i]);
                                                continue;
                                        }
                                        fputs(receivebuffer, stdout);
                                        numReady--;
                                        if(numReady <= 0)break;
                                }
                        }

                }
        }
```