1)

Client

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
//include <netinet/in.h>
//include <arpa/inet.h>
#define MAXLINE 8192 /* Max text line length */
int open_clientfd(char *hostname, char *port) {
        int clientfd;
        struct addrinfo hints, *listp, *p;
        char host[MAXLINE],service[MAXLINE];
        int flags;
        /* Get a list of potential server addresses */
        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_socktype = SOCK_STREAM; /* Open a connection */
        hints.ai_flags = AI_NUMERICSERV; /* ... using a numeric port arg.
        */
        hints.ai_flags |= AI_ADDRCONFIG; /* Recommended for connections
        where we get IPv4 or IPv6 addresses */
        getaddrinfo(hostname, port, &hints, &listp);
        /* Walk the list for one that we can successfully connect to */
        for (p = listp; p; p = p->ai_next) {
                /* Create a socket descriptor */
                if ((clientfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) < 0)
                        continue; /* Socket failed, try the next */
                flags = NI_NUMERICHOST | NI_NUMERICSERV; /* Display
                address string instead of domain name and port number instead of
                service name */
                getnameinfo(p->ai_addr, p->ai_addrlen, host, MAXLINE,
                service, MAXLINE, flags);
                printf("host:%s, service:%s\n", host, service);
                /* Connect to the server */
                if (connect(clientfd, p->ai_addr, p->ai_addrlen) != -1)
                {
                        printf("Connected to server %s at port %s\n",
                        host, service);
                        break; /* Success */
                }
                close(clientfd); /* Connect failed, try another */
```

```
		//line:netp:openclientfd:closefd
	}
	/* Clean up */
	freeaddrinfo(listp);
	if (!p) /* All connects failed */
		return -1;
		else /* The last connect succeeded */
	return clientfd;
}
int main(int argc, char **argv)
{
	int clientfd;
	char* questions[] = {"You study in which university?", "which course are you
studying?", "what is your area of interest?", "Which degree  have you registered for?"};
	char *host, *port, buf[MAXLINE];
	host = argv[1];
	port = argv[2];
	clientfd = open_clientfd(host, port);
	int sw=0;
	do{
		printf("\nEnter Question num\n");
		scanf("%d",&sw);
		switch(sw){
			case 1:
				write(clientfd, questions[0], strlen(questions[0]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 2:
				write(clientfd, questions[1], strlen(questions[1]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 3:
				write(clientfd, questions[2], strlen(questions[2]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 4:
				write(clientfd, questions[3], strlen(questions[3]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
```

```
                        case 0:
                                printf("Thank you\n");
                                break;
                        default:
                                printf("Invalid option\n");
                                break;
                }
        }while(sw!=0);
        close(clientfd);
        while (fgets(buf, MAXLINE, stdin) != NULL) {
        write(clientfd, buf, strlen(buf));
        read(clientfd, buf, MAXLINE);
        fputs(buf, stdout);
        if (buf[0] == '\n')
        break;
        }
        close(clientfd);
        exit(0);
}
```

Server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <pthread.h>
//#include <netinet/in.h>
//#include <arpa/inet.h>
/* Misc constants */
#define MAXLINE 8192 /* Max text line length */
#define LISTENQ 1024 /* Second argument to listen() */
int open_listenfd(char *port)
{
        struct addrinfo hints, *listp, *p;
        int listenfd, optval=1;
        char host[MAXLINE],service[MAXLINE];
        int flags;
        /* Get a list of potential server addresses */
        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_socktype = SOCK_STREAM; /* Accept connections
        */
        hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG; /* ... on any IP
        address AI_PASSIVE - used on server for TCP passive connection,
        AI_ADDRCONFIG - to use both IPv4 and IPv6 addresses */
        hints.ai_flags |= AI_NUMERICSERV; /* ... using port
        number instead of service name*/
        getaddrinfo(NULL, port, &hints, &listp);
        /* Walk the list for one that we can bind to */
        for (p = listp; p; p = p->ai_next) {
        /* Create a socket descriptor */
                if ((listenfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) < 0)
                        continue; /* Socket failed, try the next */
                /* Eliminates "Address already in use" error from bind */
                        setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                //line:netp:csapp:setsockopt
                (const void *)&optval , sizeof(int));
                flags = NI_NUMERICHOST | NI_NUMERICSERV; /* Display
                address string instead of domain name and port number instead of
                service name */
                getnameinfo(p->ai_addr, p->ai_addrlen, host, MAXLINE,
                service, MAXLINE, flags);
                printf("host:%s, service:%s\n", host, service);
```

```
                /* Bind the descriptor to the address */
                if (bind(listenfd, p->ai_addr, p->ai_addrlen) == 0)
                        break; /* Success */
                close(listenfd); /* Bind failed, try the next */
        }
        /* Clean up */
        freeaddrinfo(listp);
        if (!p) /* No address worked */
                return -1;
        /* Make it a listening socket ready to accept connection requests
        */
        if (listen(listenfd, LISTENQ) < 0) {
                close(listenfd);
                return -1;
        }
        return listenfd;
}
void echo(int connfd)
{
        char* questions[] = {"You study in which university?", "which course are you
studying?", "what is your area of interest?", "Which degree  have you registered for?"};
        char* answers[] = {"DAIICT", "Systems Programming", "Embedded Systems", "MSc
IT"};
        size_t n;
        char buf[MAXLINE];
        while((n = read(connfd, buf, MAXLINE)) != 0) {
                //printf("server received %d bytes\n", (int)n);
                buf[n] = '\0';
                int index;
                for(index=0;index<4;index++){
                        if( strcmp(buf,questions[index]) == 0 )
                                break;
                }
                printf("server received questions : %s\nAnswer \" %s \" is send\n",
buf,answers[index]);
                write(connfd, answers[index], strlen(answers[index]));
        }
}
int main(int argc, char **argv)
{
        int listenfd, connfd;
        socklen_t clientlen;
        struct sockaddr_storage clientaddr; /* Enough room for any addr */
```

```
        char client_hostname[MAXLINE], client_port[MAXLINE];
        listenfd = open_listenfd(argv[1]);
        while (1) {
                printf("Waiting for a new Client to connect\n");
                clientlen = sizeof(struct sockaddr_storage); /* Important! */
                connfd = accept(listenfd, (struct sockaddr *)&clientaddr,
                &clientlen);
                getnameinfo((struct sockaddr *) &clientaddr, clientlen,
                client_hostname, MAXLINE, client_port, MAXLINE, 0);
                printf("Connected to (%s, %s)\n", client_hostname,
                client_port);
                printf("Start Communication with Client\n");
                echo(connfd);
                printf("End Communication with Client\n");
                close(connfd);
        }
        exit(0);
}
```

2)

Client

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
//include <netinet/in.h>
//include <arpa/inet.h>
#define MAXLINE 8192 /* Max text line length */
int open_clientfd(char *hostname, char *port) {
        int clientfd;
        struct addrinfo hints, *listp, *p;
        char host[MAXLINE],service[MAXLINE];
        int flags;
        /* Get a list of potential server addresses */
        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_socktype = SOCK_STREAM; /* Open a connection */
        hints.ai_flags = AI_NUMERICSERV; /* ... using a numeric port arg.
        */
        hints.ai_flags |= AI_ADDRCONFIG; /* Recommended for connections
        where we get IPv4 or IPv6 addresses */
        getaddrinfo(hostname, port, &hints, &listp);
        /* Walk the list for one that we can successfully connect to */
        for (p = listp; p; p = p->ai_next) {
                /* Create a socket descriptor */
                if ((clientfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) < 0)
                        continue; /* Socket failed, try the next */
                flags = NI_NUMERICHOST | NI_NUMERICSERV; /* Display
                address string instead of domain name and port number instead of
                service name */
                getnameinfo(p->ai_addr, p->ai_addrlen, host, MAXLINE,
                service, MAXLINE, flags);
                printf("host:%s, service:%s\n", host, service);
                /* Connect to the server */
                if (connect(clientfd, p->ai_addr, p->ai_addrlen) != -1)
                {
                        printf("Connected to server %s at port %s\n",
                        host, service);
                        break; /* Success */
                }
                close(clientfd); /* Connect failed, try another */
```

```
		//line:netp:openclientfd:closefd
	}
	/* Clean up */
	freeaddrinfo(listp);
	if (!p) /* All connects failed */
		return -1;
		else /* The last connect succeeded */
	return clientfd;
}
int main(int argc, char **argv)
{
	int clientfd;
	char* questions[] = {"You study in which university?", "which course are you
studying?", "what is your area of interest?", "Which degree  have you registered for?"};
	char *host, *port, buf[MAXLINE];
	host = argv[1];
	port = argv[2];
	clientfd = open_clientfd(host, port);
	int sw=0;
	do{
		printf("\nEnter Question num\n");
		scanf("%d",&sw);
		switch(sw){
			case 1:
				write(clientfd, questions[0], strlen(questions[0]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 2:
				write(clientfd, questions[1], strlen(questions[1]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 3:
				write(clientfd, questions[2], strlen(questions[2]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
			case 4:
				write(clientfd, questions[3], strlen(questions[3]));
				read(clientfd, buf, MAXLINE);
				fputs(buf, stdout);
				break;
```

```
                        case 0:
                                printf("Thank you\n");
                                break;
                        default:
                                printf("Invalid option\n");
                                break;
                }
        }while(sw!=0);
        close(clientfd);
        while (fgets(buf, MAXLINE, stdin) != NULL) {
        write(clientfd, buf, strlen(buf));
        read(clientfd, buf, MAXLINE);
        fputs(buf, stdout);
        if (buf[0] == '\n')
        break;
        }
        close(clientfd);
        exit(0);
}
```

Server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <pthread.h>
//#include <netinet/in.h>
//#include <arpa/inet.h>
/* Misc constants */
#define MAXLINE 8192 /* Max text line length */
#define LISTENQ 1024 /* Second argument to listen() */
struct req
{
        socklen_t clientlen;
        struct sockaddr_storage clientaddr;
        char client_hostname[MAXLINE], client_port[MAXLINE];
        int listenfd, connfd;
};
int open_listenfd(char *port)
{
        struct addrinfo hints, *listp, *p;
        int listenfd, optval=1;
        char host[MAXLINE],service[MAXLINE];
        int flags;
        /* Get a list of potential server addresses */
        memset(&hints, 0, sizeof(struct addrinfo));
        hints.ai_socktype = SOCK_STREAM; /* Accept connections
        */
        hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG; /* ... on any IP
        address AI_PASSIVE - used on server for TCP passive connection,
        AI_ADDRCONFIG - to use both IPv4 and IPv6 addresses */
        hints.ai_flags |= AI_NUMERICSERV; /* ... using port
        number instead of service name*/
        getaddrinfo(NULL, port, &hints, &listp);
        /* Walk the list for one that we can bind to */
        for (p = listp; p; p = p->ai_next) {
        /* Create a socket descriptor */
                if ((listenfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) < 0)
                        continue; /* Socket failed, try the next */
                /* Eliminates "Address already in use" error from bind */
                        setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
```

```
                //line:netp:csapp:setsockopt
                (const void *)&optval , sizeof(int));
                flags = NI_NUMERICHOST | NI_NUMERICSERV; /* Display
                address string instead of domain name and port number instead of
                service name */
                getnameinfo(p->ai_addr, p->ai_addrlen, host, MAXLINE,
                service, MAXLINE, flags);
                printf("host:%s, service:%s\n", host, service);
                /* Bind the descriptor to the address */
                if (bind(listenfd, p->ai_addr, p->ai_addrlen) == 0)
                        break; /* Success */
                close(listenfd); /* Bind failed, try the next */
        }
        /* Clean up */
        freeaddrinfo(listp);
        if (!p) /* No address worked */
                return -1;
        /* Make it a listening socket ready to accept connection requests
        */
        if (listen(listenfd, LISTENQ) < 0) {
                close(listenfd);
                return -1;
        }
        return listenfd;
}
void echo(int connfd)
{
        char* questions[] = {"You study in which university?", "which course are you
studying?", "what is your area of interest?", "Which degree  have you registered for?"};
        char* answers[] = {"DAIICT", "Systems Programming", "Embedded Systems", "MSc
IT"};
        size_t n;
        char buf[MAXLINE];
        while((n = read(connfd, buf, MAXLINE)) != 0) {
                //printf("server received %d bytes\n", (int)n);
                buf[n] = '\0';
                int index;
                for(index=0;index<4;index++){
                        if( strcmp(buf,questions[index]) == 0 )
                                break;
                }
                printf("server received questions : %s\nAnswer \" %s \" is send\n",
buf,answers[index]);
```

```
                write(connfd, answers[index], strlen(answers[index]));
        }
}
void * do_work(void * object){
        struct req *obj=object;
        getnameinfo((struct sockaddr *) &obj->clientaddr, obj->clientlen,
                        obj->client_hostname, MAXLINE, obj->client_port, MAXLINE, 0);
                        printf("Connected to (%s, %s)\n", obj->client_hostname,
                        obj->client_port);
                        printf("Start Communication with Client\n");
                        echo(obj->connfd);
                        printf("End Communication with Client\n");
                        close(obj->connfd);
}
int main(int argc, char **argv)
{
        pthread_t threadID;
        struct req obj;
        obj.listenfd = open_listenfd(argv[1]);
        while (1) {
                printf("Waiting for a new Client to connect\n");
                obj.clientlen = sizeof(struct sockaddr_storage); /* Important! */
                obj.connfd = accept(obj.listenfd, (struct sockaddr *)&obj.clientaddr,
                &obj.clientlen);
                pthread_create(&threadID, NULL, do_work, (void *)&obj);

        }
        exit(0);
}
```