Khalifa University

Department of Electrical Engineering and Computer Science

ECCE 356 –Computer Networks

Spring 2022

Assignment 1 Manual

Introduction:

This document is intended to complement the Assignment 1 document. It includes the steps to execute the complementary code that requires editing to accomplish the objectives of assignment 1 and explains the different parts of the code part of the given code.

The supplementary code:

A server/client with multi-thread in C++ language: multithreadCPP.zip

Software required:

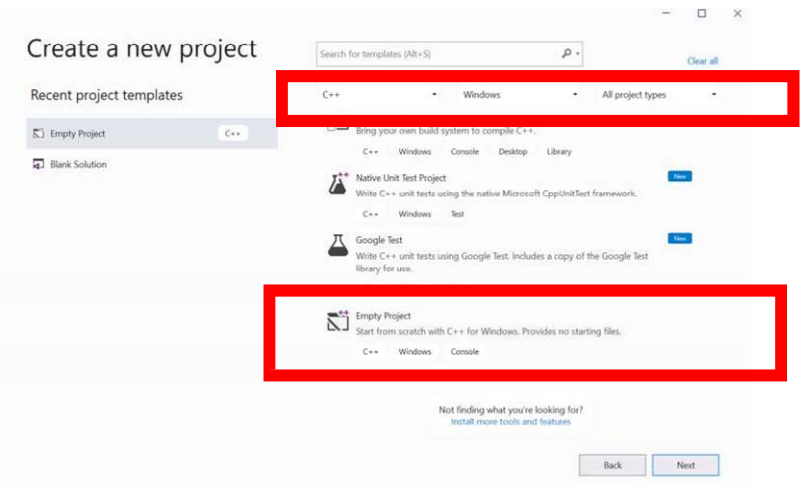
Visual Studio 2019

To download it:https://visualstudio.microsoft.com/downloads/

Once downloaded, login with university email for free access

Creating the VS projects:

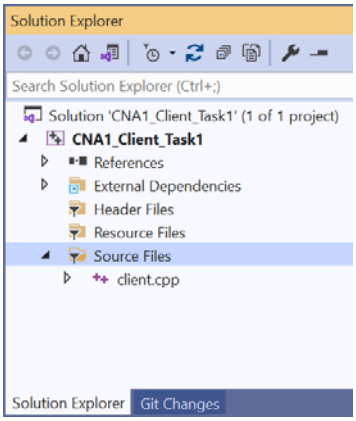
- Open visual studio 2019
- Click file -> new> create new project-> empty project from c++, windows
    - If you don't see windows, then update your software



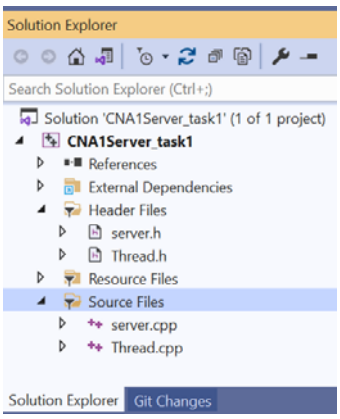
- Name the **Server** project and Import the .cpp files to the source files.
  - Right click on source files>add->new existing items.
- Import the .h files to the header files.
  - Right click on header files->add->new existing items.

For the server add resources for the client to request by placing the document inside the project folder.

- Open a new instance of the Visual studio in a new window
- Create new project for the **Client** and add .cpp files to the source files.



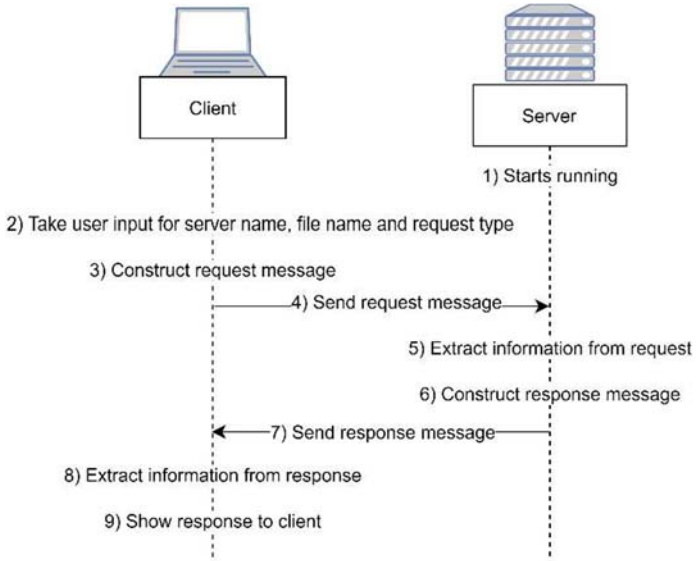
Client Project



Server Project

Codes Execution Process:

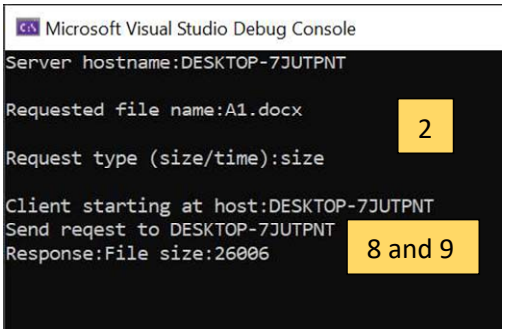
With both projects open, the server code should be first executed. Then, the client code should be run for the input to be entered and the request to be created. The below figure shows how the codes would execute. Given that the server is running, the client can request information about a given file. The codes takes the input of the user for the server's hostname, the file name, and the requested information (size of file or time it was edited). Once the user is entered, the client code encapsulates it into a request message and sends it to the server. The server extracts the information, constructs the response and sends it back. The client code then extracts the information from the response and shows it to the client user.



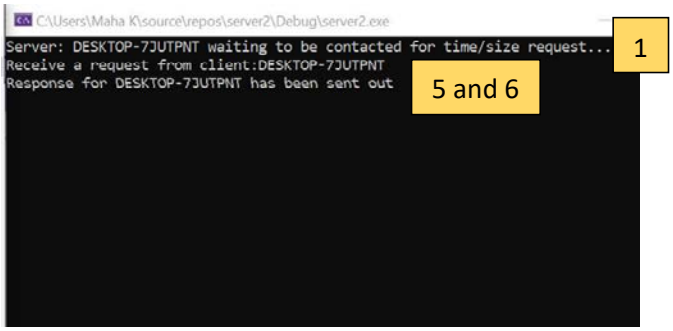
Execution Snapshots

The following snapshots show the codes during execution where the number correspond to the number in the process.

Client



Server



Understanding the codes:

Data structures

The client and server codes have a list of structs that allow them to operate described below.

The Request *Req struct*

Request	Interpretation

`typedef struct`
`{`
`  char hostname[HOSTNAME_LENGTH];`
`  char filename[FILENAME_LENGTH];`
`} Req;  //request`
	Hostname of the destination PC
	Name of the requested file

The response *Resp* struct.

Response	Interpretation

`typedef struct`
`{`
`  char response[RESP_LENGTH];`
`} Resp; //response`
	String holding the response to a request

The message *Msg* structure:

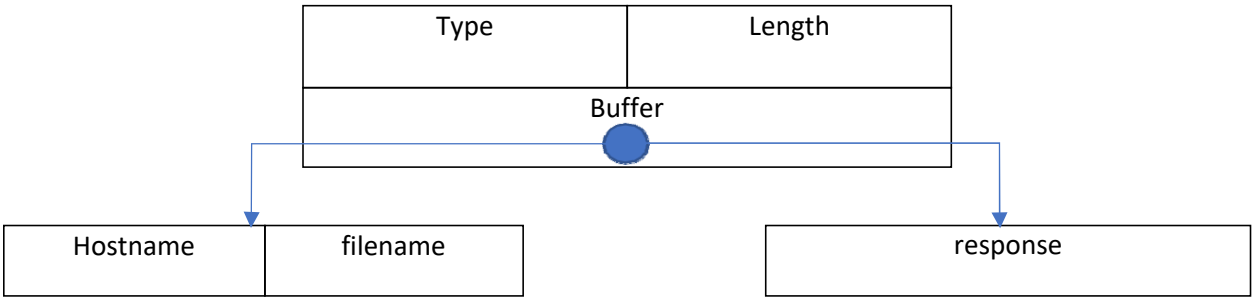
```
typedef struct
{
    Type type;
    int  length; //length of effective bytes in the buffer
    char buffer[BUFFER_LENGTH];
} Msg; //message format used for sending and receiving
```

Type implies the type of the message and is set according to the below enum. Hint: *enum* consists of integral constants that start from a default set value.

```
typedef enum{
    REQ_SIZE=1,REQ_TIME,RESP //Message type
} Type;
```

Length is the length of the message in the buffer.

Buffer holds the message content and is casted to either a request or a response struct based on the message type.



Variables and constants:

```
int sock;                   /* Socket descriptor */
struct sockaddr_in ServAddr; /* server socket address */
unsigned short ServPort;     /* server port */
Req * reqp;                /* pointer to request */
Resp * respp;           /* pointer to response*/
Msg smsg,rmsg;              /* receive_message and send_message
WSADATA wsadata;
```

```
#define HOSTNAME_LENGTH 20
#define RESP_LENGTH 40
#define FILENAME_LENGTH 20
#define REQUEST_PORT 5001
#define BUFFER_LENGTH 1024
#define TRACE 0
#define MSGHDRSIZE 8 //Message Header Size
```

Client main code:

First the user's input is take using the below	`// take input from user`
	`cout<<"Enter the following information:"<<endl;`
	`cout<<"Server hostname:";`
	`cin>> inputserverhostname;`
	`cout<<endl;`
	`cout<<"Requested filename:";`
	`cin>>inputfilename;`
	`cout<<endl;`
	`cout<<"Request type (size/time):";`
	`cin>>inputrequesttype;`
	`cout<<endl;`

Cast the buffer field of smsg to a request struct pointer	`reqp=(Req *)smsg.buffer;`

Initializing the windows sockets	`if (WSAStartup(0x0202,&wsadata)!=0)`
	`{`
	`  WSACleanup();`
	`  err_sys("Error in starting WSAStartup()\n");`
	`}`

Copy the local host name to the reqp hostname field
 	
Display name of local host
	`if(gethostname(reqp->hostname,HOSTNAME_LENGTH)!=0)`
	`  err_sys("can not get the host name,program exit");`
	
	`printf("%s%s\n","Client starting at host:",reqp->hostname);`

Copy the file user to the request file name	`strcpy(reqp->filename,inputfilename);`

Check the request type and set the smsg type accordingly	`if(strcmp(inputrequesttype,"time")==0)`
	`  smsg.type=REQ_TIME;`
	`else if (strcmp(inputrequesttype,"size")==0)`
	`  smsg.type=REQ_SIZE;`
	`else err_sys("Wrong request type\n");`

```
//Create the socket
if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) //create the socket
    err_sys("Socket Creating Error");

//connect to the server
ServPort=REQUEST_PORT;
memset(&ServAddr, 0, sizeof(ServAddr));     /* Zero out structure */
ServAddr.sin_family      = AF_INET;             /* Internet address family */
ServAddr.sin_addr.s_addr = ResolveName(inputserverhostname);   /* Server IP address */
ServAddr.sin_port        = htons(ServPort); /* Server port */
if (connect(sock, (struct sockaddr *) &ServAddr, sizeof(ServAddr)) < 0)
        err_sys("Socket Creating Error");
```

Set the length of the send message	`smsg.length = sizeof(Req);`

Display status to user and send message using *msg_send* function	`fprintf(stdout, "Send request to %s\n", inputserverhostname);`
	`if (msg_send(sock, &smsg) != sizeof(Req))`
	`  err_sys("Sending req packet error.,exit");`

wait for a message to be received to the rmsg variable using *msg_recv* function	`//receive the response`
	`if (msg_recv(sock, &rmsg) != rmsg.length)`
	`  err_sys("recv response error,exit");`

Cast the *rmsg* buffer to *respp* and show the response content	`respp = (Resp*)rmsg.buffer;` `printf("Response:%s\n\n\n", respp->response);`

Server main code:

Constructor	

Initializing the windows sockets	`if (WSAStartup(0x0202,&wsadata)!=0)`
	`{`
	`  WSACleanup();`
	`  err_sys("Error in starting WSAStartup()\n");`
	`}`

Display name of local host	`if(gethostname(servername,HOSTNAME_LENGTH)!=0) //get the hostname`
	`  TcpThread::err_sys("Get the host name error,exit");`
	
	`printf("Server: %s waiting to be contacted for time/size request...\n",servername);`

```
//Create the server socket
if ((serverSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    TcpThread::err_sys("Create socket error,exit");

//Fill-in Server Port and Address info.
ServerPort=REQUEST_PORT;
memset(&ServerAddr, 0, sizeof(ServerAddr));      /* Zero out structure */
ServerAddr.sin_family = AF_INET;                 /* Internet address family */
ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);  /* Any incoming interface */
ServerAddr.sin_port = htons(ServerPort);         /* Local port */

//Bind the server socket
if (bind(serverSock, (struct sockaddr *) &ServerAddr, sizeof(ServerAddr)) < 0)
    TcpThread::err_sys("Bind socket error,exit");

//Successfull bind, now listen for Server requests.
if (listen(serverSock, MAXPENDING) < 0)
    TcpThread::err_sys("Listen socket error,exit");
```

Start	

Forever loop to keep creating new threads and accept messages	`for (;;) /* Run forever */`
	`{`
	`  /* Set the size of the result-value parameter */`
	`  clientLen = sizeof(ClientAddr);`
	
	`  /* Wait for a Server to connect */`
	`  if ((clientSock = accept(serverSock, (struct sockaddr *) &ClientAddr,`
	`  &clientLen)) < 0)`
	`  TcpThread::err_sys("Accept Failed ,exit");`
	
	`  /* Create a Thread for this new connection and run*/`
	`  TcpThread * pt=new TcpThread(clientSock);`
	`  pt->start();`
	`}`

Run	
wait for a message to be received to the rmsg variable using *msg_recv* function	`if(msg_recv(cs,&rmsg)!=rmsg.length)` `  err_sys("Receive Req error,exit");`
Cast the *rmsg* buffer to *reqp* and show the request content	`//cast it to the request packet structure` `reqp=(Req *)rmsg.buffer;`

Check the request type and set the rmsg response accordingly	`if((result = _stat(reqp->filename,&stat_buf))!=0)`
	`  sprintf(respp->response,"No such a file");`
	`else {`
	`  memset(respp->response,0,sizeof(Resp));`
	`  if(rmsg.type==REQ_TIME)`
	`  sprintf(respp->response,"%s", ctime(&stat_buf.st_ctime ));`
	`  else if(rmsg.type==REQ_SIZE)`
	`  sprintf(respp->response,"File size:%ld",stat_buf.st_size );`
	`}`

Send smsg to the client	`if(msg_send(cs,&smsg)!=smsg.length)` `  err_sys("send Respose failed,exit");`