# 3D3 Computer networks

Assignment 2

HTTP Client and Server

Ion Alin Spiridon - 14305180

# HTTP Client Code

```c
#include    "unp.h"

int main(int argc, char **arg)
{
    int                 sockfd, n;
    char                recvline[MAXLINE + 1];
    struct sockaddr_in  servaddr;
    char                buff[MAXLINE];


    if (argc != 4)
        err_quit("usage: ./httpclient <IPaddress> <Port Number> <Resource Name>"); //to run the program use this command

    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        err_sys("socket error");

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port   = htons(atoi(argv[2])); // port number from the command line
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)  //inet_pton converts the IP address from the command line.
        err_quit("inet_pton error for %s", argv[1]);

    if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
        err_sys("connect error");

    snprintf(buff, sizeof(buff), "GET /%s HTTP/1.1\r\nHost: www.comp. it.ie\r\n\r\n", argv[3]); //create the client request
                            // in the outgoing buffer 'buff'. Note the hostname must be changed to reflect the proper hostname.
                            // Also note that the resource name is passed in from the command line
    Write(sockfd, buff, strlen(buff));  //write contents of send buffer to the socket

    while ( (n = read(sockfd, recvline, MAXLINE)) > 0) //read the data returned by the server into the receive buffer
    {
        recvline[n] = 0;    // null terminate the receive buffer
        if (fputs(recvline, stdout) == EOF) //print the contents of the receive buffer to the screen
            err_sys("fputs error");
    }

    if (n < 0)
        err_sys("read error");

    exit(0);
}
```

# Complex HTTP Server Code with File Open Functionality and Client Address Retrieval

```c
#include    "unp.h"
#include    <string.h>
#include    <stdio.h>

int main(int argc, char **argv)
{
    socklen_t   len;  //this var is used for obtaining the client addressing information.  See Accept call.
    int     n, listenfd, connfd, char_in, count = 0; //declare usual socket variables and extra vars for reading from file
    struct sockaddr_in  servaddr, cliaddr; //declare address structures for both the client and server addressing data
    char    buff[40], wbuff[MAXLINE], rbuff[MAXLINE], cmd[16], path1[64]=".", path[64], vers[16]; //declare read and write buffers;
                                                    //extra buffers for parsing the client request and printing client address

    FILE * hFile; //declare file pointer

    if (argc != 2)
        err_quit("usage: a.out <Port>");

    listenfd = Socket(AF_INET, SOCK_STREAM, 0);  //open local socket

    bzero(&servaddr, sizeof(servaddr));  //Populate server addressing details
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(atoi(argv[1]));

    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));  //bind the socket to the IP interface

    Listen(listenfd, LISTENQ);  //transform the socket to a listening socket

    for ( ; ; ) //infinite loop for dealing with client connections
    {
        len = sizeof(cliaddr); //set len to sizeof of cliaddr struct - to be used to obtain client addressing data

        connfd = Accept(listenfd, (SA *) &cliaddr, &len);  //accept the next client connection request and retrieve client address

        printf("\nConnection from %s, port %d\n", Inet_ntop(AF_INET, &cliaddr.sin_addr, buff, sizeof(buff)), ntohs(cliaddr.sin_port));
                                // printing the client IP address and Port number.  Note use of ntop asnd ntohs functions for conversion
                                // from network byte order to host byte order

        while ( (n = read(connfd, rbuff, MAXLINE)) > 0) //read loop
        {

            rbuff[n] = 0;   // null terminate rbuff prior to screen print

            if (fputs(rbuff, stdout) == EOF)  // screen print contents of read buffer
                err_sys("fputs error");
```

```
47              if (strstr(rbuff,"\r\n\r\n")>0) //looks for a needle in a haystack char *strstr(const
48                  {          //char *haystack, const char *needle); It returns a pointer
49                  break;         //to the location needle within haystack otherwise
50                  }          //if not found it returns zero.  Used to break from Read Loop.
51
52          }// end read loop
53
54          if (n < 0) //error check on read loop
55              err_sys("read error");
56
57          sscanf(rbuff, "%s %s %s", cmd, path, vers); //parsing the incoming client request
58
59          strcat(path1, path); //concatenate the resource name to a full-stop to refer to "this" directory
60
61          if(strcmp(path1, "./") == 0)
62          {
63              strcpy(path1, "./index.html"); //check for empty "slash" and replace with default page
64          }
65
66          hFile = fopen(path1, "r");       //open the requested file
67
68          if (hFile == NULL)           //if requested file does not exist
69                  {
70                  hFile = fopen("error.html", "r");   //open the error file
71                  }
72
73          strcpy(wbuff,"");                //empty the outgoing buffer
74
75          while((char_in = fgetc(hFile))!= EOF)   //reading one char at-a-time from the open file
76              {
77                  wbuff[count] = char_in; //storing the char in the outgoing buffer
78                  count++;               //increment the buffer index ready for next character
79              }
80
81          wbuff[count] = 0;            // null terminate outgoing buffer ready for writing to socket
82
83          Write(connfd, wbuff, strlen(wbuff));        //write contents of wbuff to the socket
84          count = 0;               //reset the buffer index
85          fclose(hFile);           //close the file
86          strcpy(path1, ".");      //reset the path1 buffer back to "."
87
88          Close(connfd);           //close the TCP connection
89
90      } //end infinite for
91
92  } //end main
93
```