CODE: Client Side

```
#include <string.h>
#include <stdio.h>
#include <windows.h>
#include <WinSock2.h>
#include <stdlib.h>

#define ERR_CODE_NONE   0     //No error
#define ERR_CODE_SWI    1     //Software error

#define CMD_LENGTH      5

#define ARG_NONE  1
#define ARG_NUMBER      2

float fs;               //Sampling frequency (sent to the server)
char filename[10];            //Create a file to save output samples
char A[10];                   //For storing the sampling frequency temporarily
FILE *fp;               //File pointer (temporary pointer for file operations)
char accept[10];        //String to accept commands from the user

//flag = 0 indicates waiting on an action
//flag = 1 indicates an action occurred
int flag_system_paused = 0;   //Flag to indicate system paused
int flag_system_resumed = 0;  //Flag to indicate system resuming normal operation
int flag_wait_for_start = 0;  //Flag to indicate system starts data acquisition
int flag_wait_for_terminate = 0;//Flag to indicate system stops data acquisition

/*-------------------------------------------------------------
  mode    |            Indicates                              |
-------------------------------------------------------------
  0       | Data acquisition not started yet                  |
-------------------------------------------------------------
  1       | Data acquisition in progress                      |
-------------------------------------------------------------
  2       | Data acquisition paused                           |
------------------------------------------------------------- */

int mode;               //Indicator for ensuring thread synchronization

int x = 1;


typedef struct sp_comm {      //Structure for connection establishment
WSADATA wsaData;
SOCKET cmdrecvsock;
SOCKET cmdstatusock;
SOCKET datasock;
struct sockaddr_in server;
} * sp_comm_t;



typedef struct sp_flags {     //Structure required for thread creation
unsigned int start_system:1;
unsigned int pause_system:1;
unsigned int shutdown_system:1;
unsigned int analysis_started:1;
unsigned int restart:1;
unsigned int transmit_data:1;
} * sp_flags_t;
```

```
typedef struct sp_struct{
struct sp_comm    comm;
struct sp_flags   flags;
} * sp_struct_t;

typedef struct {
char cmd[CMD_LENGTH];
int arg;
} cmd_struct_t;

WSADATA wsaData;

//Thread for simultaneously recieveing data from the server
HANDLE hClientThread;                           //Thread parameters
DWORD dwClientThreadID;
VOID client_iface_thread(LPVOID parameters);    //Function(Thread) for parallel data
reception

//Main() provides a user interface. Continuously accepts commands from the user and other
information.
//Main() creates the thread for data reception
//Main() contains the code for establishing a connection using winsock.h, with the server

int main()
{
  struct sp_struct profiler;
  struct sockaddr_in saddr;
  struct hostent *hp;
  int res = 0;
  char ParamBuffer[110]="";
  char inputChar[110] = "";

  memset(&profiler, 0, sizeof(profiler));
  sp_comm_t comm = &profiler.comm;


  //Linking the "ws2_32.lib"
  if((res = WSAStartup(0x202,&wsaData)) != 0){
      printf(stderr,"WSAStartup failed with error %d\n",res);
      WSACleanup();
      return(ERR_CODE_SWI);
  }

  hp = (struct hostent*)malloc(sizeof(struct hostent));
  hp->h_name = (char*)malloc(sizeof(char)*17);
  hp->h_addr_list = (char**)malloc(sizeof(char*)*2);
  hp->h_addr_list[0] = (char*)malloc(sizeof(char)*5);
  strcpy(hp->h_name, "lab_example\0");
  hp->h_addrtype = 2;
  hp->h_length = 4;

  hp->h_addr_list[0][0] = (signed char)192;
  hp->h_addr_list[0][1] = (signed char)168;
  hp->h_addr_list[0][2] = (signed char)0;
  hp->h_addr_list[0][3] = (signed char)140;
  hp->h_addr_list[0][4] = 0;

  //Setup a socket and connect with the server
  memset(&saddr, 0, sizeof(saddr));
  saddr.sin_family = hp->h_addrtype;
  memcpy(&(saddr.sin_addr), hp->h_addr, hp->h_length);
  saddr.sin_port = htons(1500);
```

```
  if((comm->datasock = socket(AF_INET,SOCK_DGRAM, 0)) == INVALID_SOCKET){
      printf(stderr,"socket(datasock) failed: %d\n",WSAGetLastError());
      WSACleanup();
      return(ERR_CODE_NONE);
  }

  if(connect(comm->datasock,(struct sockaddr*)&saddr,sizeof(saddr)) == SOCKET_ERROR) {
      printf(stderr,"connect(datasock) failed: %d\n",WSAGetLastError());
      WSACleanup();
      return(ERR_CODE_SWI);
  }

  //Setup and bind a socket to listen for commands from server
  memset(&saddr, 0, sizeof(struct sockaddr_in));
  saddr.sin_family = AF_INET;
  saddr.sin_addr.s_addr = INADDR_ANY;
  saddr.sin_port = htons(1024);
  if((comm->cmdrecvsock = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET) {
      printf(stderr,"socket(cmdrecvsock) failed: %d\n",WSAGetLastError());
      WSACleanup();
      return(ERR_CODE_NONE);
  }

  if(bind(comm->cmdrecvsock,(struct sockaddr*)&saddr,sizeof(saddr) ) == SOCKET_ERROR) {
      printf(stderr,"bind() failed: %d\n",WSAGetLastError());
      WSACleanup();
      return(ERR_CODE_NONE);
  }

  //At this point UDP connection complete
  //Create thread for data reception from the server
  hClientThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) client_iface_thread,
(LPVOID)&profiler, 0, &dwClientThreadID);
  SetThreadPriority(hClientThread, THREAD_PRIORITY_LOWEST);


  printf("Enter the filtering sample rate:\n");
  scanf("%f",&fs);

  sprintf(A,"%f",fs);
  send(comm->datasock, A, sizeof(A), 0);
  printf("\nSampling frequency sent!");
  printf("\nEnter the name of your file:");
  scanf("%s",&filename);
  char string[10];

  printf("\n\nEnter 's' to Start Server Processing\n 'p' to Pause\n 'r' to   Resume\n 't'
to Terminate\n");
  while(1){
      if(_kbhit()){
            scanf("%s",&string);
            send(comm->datasock, string, sizeof(string), 0);
            break;
      }
  }

  mode=0;
  while(!flag_wait_for_start);
  mode=1;

  printf("\nYou may now pause or stop the system\nEnter the command:");
  flag_system_resumed = 1;
```

```
  flag_wait_for_terminate = 0;

  while(1){
      if( _kbhit()){
              scanf("%s",accept);
            if(strcmp(accept,"p") == 0 ){
                  flag_system_resumed = 0;
                  send(comm->datasock,accept, sizeof(accept), 0);
                  mode= 2;
                  while(!flag_system_paused);
                  printf("\nEnter the resume command:");
            }
            else{
                  if((strcmp(accept,"r") == 0) && (flag_system_paused == 1)){
                        flag_system_paused = 0;
                        send(comm->datasock,accept, sizeof(accept), 0);
                        mode = 1;
                        while(!flag_system_resumed);
                        printf("\nYou may pause or stop the system now !\nEnter the
command:");
                  }
                  else{
                        if((strcmp(accept,"t") == 0)){
                              mode = 1;
                              send(comm->datasock,accept, sizeof(accept), 0);
                              while(!flag_wait_for_terminate);
                        }
                        else
                        printf("\nError: Unexpected command");
                  }
            }
      }
      if(flag_wait_for_terminate == 1)
      break;
  }

  while(1);
  return 0 ;
}


VOID client_iface_thread(LPVOID parameters) //LPVOID parameters)
{
  sp_struct_t profiler = (sp_struct_t)parameters;
  sp_comm_t comm = &profiler->comm;
  INT retval;
  struct sockaddr_in saddr;
  int saddr_len;
  char ParamBuffer[110]="" ;


  printf("Executing Thread\n");
  printf("Checking for Data\n");
  while(ParamBuffer[0] != '!'){
      memset(ParamBuffer, 0, sizeof(ParamBuffer));
      saddr_len =sizeof(saddr);
      retval = recvfrom(comm->cmdrecvsock, ParamBuffer, sizeof(ParamBuffer), 0, (struct
sockaddr *)&saddr, &saddr_len);
      switch(mode){
            case 0:
            if((strcmp(ParamBuffer,"S")==0)){
                  printf("\nACKNOWLEDGEMENT received, system operation started!");
                  mode  = 1;
```

```
                  flag_wait_for_start = 1;
            }
            else
            printf("\nError:Expected command -'S' for start");
            break;

            case 1:
            if((strcmp(ParamBuffer,"T")==0)){
                  printf("\nACKNOWLEDGEMENT received, system operation stopped!\nWaiting
for filtered data!");
                  flag_wait_for_terminate = 1;
            }
            else{
                  if((strcmp(ParamBuffer,"R")==0)){
                        printf("\nACKNOWLEDGEMENT received, system succesfully
resumed!");
                        flag_system_resumed = 1;
                        }
                  else
                  printf("\nError:Expected command -'T' or 'R'");
            }
            break;

            case 2:
            if((strcmp(ParamBuffer,"P")==0)){
                  printf("\nACKNOWLEDGEMENT received, system succesfully paused!");
                  flag_system_paused = 1;
            }
            else
            printf("\nError:Expected command -'P'");
            break;

      }
  }
  //x = 0;
}
```