CODE: Server Side

```
#include <stdio.h>
#include "oldaapi.h"                //Header file for DT module functions
#include<stdlib.h>
#include <windows.h>
#include <WinSock.h>
#include <olmem.h>
#include <olerrors.h>


typedef struct sp_comm {
      WSADATA wsaData;
      SOCKET cmdrecvsock;
      SOCKET cmdstatusock;
      SOCKET datasock;
      struct sockaddr_in server;
} * sp_comm_t;

typedef struct sp_flags {
      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;


#define ERR_CODE_NONE   0
#define ERR_CODE_SWI    1

#define CMD_LENGTH      5

#define ARG_NONE  1
#define ARG_NUMBER      2

#define NUM_OL_BUFFERS  4

//Variable mode takes the values below
/*-------------------------------------------------------------------
      mode      |               Indicates                          |
--------------------------------------------------------------------
    SYSTEM_OFF  | System OFF                                       |
--------------------------------------------------------------------
 WAIT_FOR_START | Connection established with the client and sampling  |
                | frequency recieved. Now waiting for the "s" command  |
--------------------------------------------------------------------
    SYSTEM_ON   | "s" received and analog switch 1 turned ON       |
                | acquiring data from digital channel 0            |
--------------------------------------------------------------------
  SYSTEM_PAUSED | "p" command received and data acquisition paused     |
--------------------------------------------------------------------
 SYSTEM_STOPPED | "t" command received and data acquisition terminated |
                | Store data in a file                             |
-------------------------------------------------------------------*/
```

```
#define SYSTEM_OFF      0
#define WAIT_FOR_START  1
#define SYSTEM_ON       2
#define SYSTEM_PAUSED   3
#define SYSTEM_STOPPED  4

#define CHECKERROR( ecode )                           \
do                                                    \
{                                                     \
   ECODE olStatus;                                    \
   if( OLSUCCESS != ( olStatus = ( ecode ) ) )        \
   {                                                  \
      printf("OpenLayers Error %d\n", olStatus );  \
      exit(1);                                        \
   }                                                  \
}                                                     \
while(0)



#define STRLEN 80        // string size for general text manipulation
char str[STRLEN];        // global string for general text manipulation

#define SHOW_ERROR(ecode) MessageBox(HWND_DESKTOP,olDaGetErrorString(ecode,\
                  str,STRLEN),"Error", MB_ICONEXCLAMATION | MB_OK);

#define CHECK_ERROR(ecode) if ((board.status = (ecode)) != OLNOERROR)\
                  {\
                  SHOW_ERROR(board.status);\
                  olDaReleaseDASS(board.hdass);\
                  olDaTerminate(board.hdrvr);\
                  exit(0);}

FILE *fp;
float sampling_freq;

int mode;

//flag = 0 indicates waiting on an action
//flag = 1 indicates an action occurred
int flag_system_on = 0;
int flag_wait_for_start = 0;
int flag_a2d = 0;
int flag_inidicate_on = 0;
int flag_system_pause = 0;
int flag_system_stop = 0;
int flag_system_resume = 0;
int x = 1;
int wait_for_processing = 0;
int flag_set_1 = 0;
int temp = 0;
int counter = 0;
ULNG value[2000000];
int index =0 ;
int k = 0;

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

HANDLE hClientThread;
```

```
DWORD dwClientThreadID;
VOID client_iface_thread(LPVOID parameters);


UINT resolution,encoding;
PWORD  pBuffer32 = NULL;
DBL volts;

DBL max,min;

LRESULT WINAPI WndProc( HWND hWnd, UINT msg, WPARAM hAD, LPARAM lParam){
      DWORD samples;
      int y,z;
      int j=0;

      switch( msg ){
            case OLDA_WM_BUFFER_DONE:
                  printf( "\nBuffer Done Count : %ld \r", counter );
                  HBUF hBuf;
                  counter++;
                  olDaGetBuffer( (HDASS)hAD, &hBuf );
                  olDaGetRange(HDASS(hAD),&max,&min);
                  olDaGetEncoding(HDASS(hAD),&encoding);
                  olDaGetResolution((HDASS) hAD,&resolution);
                  olDmGetValidSamples( hBuf, &samples );
                  olDmGetBufferPtr( hBuf,(LPVOID*)&pBuffer32);
                  if(flag_a2d == 0){
                        for(j=0;j<1000;j++){
                              value[index++] = pBuffer32[j];
                        }
                  }
                  else{
                        for(j=0;j<1000;j++){
                              value[index++] = pBuffer32[j];
                        }
                  }

                  for(j=1;j<1000;j++){
                        k++;
                        if(k%2 != 0){
                        volts = ((float)max-(float)min)/(1L<<resolution)*value[k] +
(float)min;
                        if(mode == WAIT_FOR_START && volts > 4.0){
                              printf("\nSwitch on");
                              flag_a2d = 1;
                              olDaPutBuffer( (HDASS)hAD, hBuf );
                              break;
                        }
                        if(mode == SYSTEM_ON && volts == 0.0){
                              printf("\nSwitch off");
                              flag_system_stop = 1;
                              olDaPutBuffer( (HDASS)hAD, hBuf );
                              break;
                        }


                        }
                  }
                  olDaPutBuffer( (HDASS)hAD, hBuf );
                  break;

                  case OLDA_WM_QUEUE_DONE:
                  printf( "\nAcquisition stopped, rate too fast for current options." );
                  PostQuitMessage(0);
                  break;

                  case OLDA_WM_TRIGGER_ERROR:
                  printf( "\nTrigger error: acquisition stopped." );
                  PostQuitMessage(0);
                  break;

                  case OLDA_WM_OVERRUN_ERROR:
                  printf( "\nInput overrun error: acquisition stopped." );
                  PostQuitMessage(0);
                  break;

                  default:
                  return DefWindowProc( hWnd, msg, hAD, lParam );
      }
      return 0;
}



BOOL CALLBACK EnumBrdProc( LPSTR lpszBrdName, LPSTR lpszDriverName, LPARAM lParam){

      // Make sure we can Init Board
      if( OLSUCCESS != ( olDaInitialize( lpszBrdName, (LPHDEV)lParam ) ) ){
                  return TRUE;  // try again
      }

      // Make sure Board has an A/D Subsystem
      UINT uiCap = 0;
      olDaGetDevCaps ( *((LPHDEV)lParam), OLDC_ADELEMENTS, &uiCap );

      if( uiCap < 1 ){
                  return TRUE;  // try again
      }

      printf( "%s succesfully initialized.\n", lpszBrdName );
      return FALSE;    // all set , board handle in lParam
}



//Simple structure used with board

typedef struct tag_board {
   HDEV hdrvr;         // device handle
   HDASS hdass;        // sub system handle
   ECODE status;       // board error status
   HBUF  hbuf;         // sub system buffer handle
   PWORD lpbuf;        // buffer pointer
   char name[MAX_BOARD_NAME_LENGTH];      // string for board name
   char entry[MAX_BOARD_NAME_LENGTH];   // string for board name
} BOARD;

typedef BOARD* LPBOARD;
static BOARD board;

/*
this is a callback function of olDaEnumBoards, it gets the
strings of the Open Layers board and attempts to initialize
the board.  If successful, enumeration is halted.
*/
```

```
BOOL CALLBACK GetDriver( LPSTR lpszName, LPSTR lpszEntry, LPARAM lParam ){
      LPBOARD lpboard = (LPBOARD)(LPVOID)lParam;

      //fill in board strings

      #ifdef WIN32
      strncpy(lpboard->name,lpszName,MAX_BOARD_NAME_LENGTH-1);
      strncpy(lpboard->entry,lpszEntry,MAX_BOARD_NAME_LENGTH-1);
      #else
      lstrcpyn(lpboard->name,lpszName,MAX_BOARD_NAME_LENGTH-1);
      lstrcpyn(lpboard->entry,lpszEntry,MAX_BOARD_NAME_LENGTH-1);
      #endif

      //try to open board

      lpboard->status = olDaInitialize(lpszName,&lpboard->hdrvr);
      if(lpboard->hdrvr != NULL)
                  return FALSE;           //false to stop enumerating
      else
                  return TRUE;            //true to continue
}

//Function to initialize the DT console board (DT9816- Data acquisition module)
void Initialize_DOUT(int value_dout){

      UINT resolution;
      UINT channel = 0;
      DBL gain = 1.0;

      board.hdrvr = NULL;

      CHECK_ERROR (olDaEnumBoards(GetDriver,(LPARAM)(LPBOARD)&board));

      //check for error within callback function
      CHECK_ERROR (board.status);

      //check for NULL driver handle - means no boards
      if (board.hdrvr == NULL){
                  MessageBox(HWND_DESKTOP, " No Open Layer boards!!!", "Error",
                  MB_ICONEXCLAMATION | MB_OK);
                  exit(0);
      }

      //get handle to DOUT sub system
      CHECK_ERROR(olDaGetDASS(board.hdrvr,OLSS_DOUT,0,&board.hdass));

      //set subsystem for single value operation
      CHECK_ERROR (olDaSetDataFlow(board.hdass,OL_DF_SINGLEVALUE));
      CHECK_ERROR (olDaConfig(board.hdass));

      //put all 0's single value
      CHECK_ERROR (olDaPutSingleValue(board.hdass,value_dout,channel,gain));

}

void main()
{
      struct sp_struct profiler;
      struct sockaddr_in saddr;
      struct hostent *hp;
      int res = 0;
      int i;
      char numberstr[10];
      WNDCLASS wc;
      memset( &wc, 0, sizeof(wc));
      wc.lpfnWndProc = WndProc;
      wc.lpszClassName = "DtConsoleClass";
      RegisterClass( &wc );
      LPHDASS ptr;
      HDASS P;

      //char ParamBuffer[100];
      //Initial state before establishing a connection with the client
      mode = SYSTEM_OFF;
      flag_system_on = 0;

      //Parameters for the data reception thread
      memset(&profiler, 0, sizeof(profiler));
      sp_comm_t comm = &profiler.comm;

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

      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)127;
      hp->h_addr_list[0][1] = (signed char)0;
      hp->h_addr_list[0][2] = (signed char)0;
      hp->h_addr_list[0][3] = (signed char)1;
      hp->h_addr_list[0][4] = 0;


        //Setup a socket and connect with the client
      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(1024);

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

      }

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

      //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(1500);
      if ((comm->cmdrecvsock = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET) {
            fprintf(stderr,"socket(cmdrecvsock) failed: %d\n",WSAGetLastError());
            WSACleanup();
            return(ERR_CODE_NONE);
      }

      if (bind(comm->cmdrecvsock,(struct sockaddr*)&saddr,sizeof(saddr) ) ==
SOCKET_ERROR) {
            fprintf(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);

      //Initialize the DT Console board
      Initialize_DOUT(0);

      //Wait for the sampling frequency from the client
      while(!flag_system_on);
      printf("The sampling frequency is: %.3f",sampling_freq);

      //Wait for the start "s" command form the client
      mode =  WAIT_FOR_START;
      flag_wait_for_start = 0;
      while(!flag_wait_for_start);

      //Begin monitoring channels 0 and 1
      //Setup the analog inputs and the monitor the digital outputs
      //Wait until the user turns ON the analog Switch 1
      //Analog 0 connected to the incoming sine wave (analog data)
      printf("\nMonitoring channels 0 and 1!");

      HDEV hDev = NULL;
      CHECKERROR( olDaEnumBoards( EnumBrdProc, (LPARAM)&hDev ) );

      HDASS hAD = NULL;
      olDaGetDASS( hDev, OLSS_AD, 0,(PHDASS)&hAD );
      CHECKERROR( olDaSetWndHandle( HDASS(hAD), hWnd, 0 ) );
      CHECKERROR( olDaSetDataFlow( HDASS(hAD), OL_DF_CONTINUOUS ) );

      //Setup the order of reception and storing the digital data in the buffer
        CHECKERROR( olDaSetChannelListEntry( HDASS(hAD), 0, 0 ) );
      CHECKERROR( olDaSetChannelListEntry( HDASS(hAD), 1, 1 ) );

      //Set up the channel gains
      CHECKERROR( olDaSetGainListEntry( HDASS(hAD), 0, 1 ) );
      CHECKERROR( olDaSetGainListEntry( HDASS(hAD), 1, 1 ) );

      //Channel list = 2 because using 2 channels
      CHECKERROR( olDaSetChannelListSize( HDASS(hAD), 2 ) );

      //Other details
      CHECKERROR( olDaSetTrigger( HDASS(hAD), OL_TRG_SOFT ) );
      CHECKERROR( olDaSetClockSource( HDASS(hAD), OL_CLK_INTERNAL ) );

      //Set the sampling frequency as accepted from the client
      CHECKERROR( olDaSetClockFrequency( HDASS(hAD), sampling_freq ) );
      CHECKERROR( olDaSetWrapMode( HDASS(hAD), OL_WRP_NONE ) );
      CHECKERROR( olDaConfig( HDASS(hAD) ) );

      HBUF hBufs[NUM_OL_BUFFERS];
      for( int i=0; i < NUM_OL_BUFFERS; i++ ){
            if( OLSUCCESS != olDmAllocBuffer( GHND, 1000, &hBufs[i] ) ){
                  for ( i--; i>=0; i-- ){
                              olDmFreeBuffer( hBufs[i] );
                  }
                  exit( 1 );
            }
            olDaPutBuffer( (HDASS)hAD,hBufs[i] );
       }

      if( OLSUCCESS != ( olDaStart( (HDASS)hAD ) ) ){
                  printf( "A/D Operation Start Failed...exiting!\n" );
            exit(1);
      }
      else{
                  printf( "A/D Operation Started....\n\n" );
                  printf( "Buffer Done Count : %ld \r", counter );
      }

      SetMessageQueue(50);

      //flag_a2d will be set by GetMessage() on performing the necessary checks
      flag_a2d = 0;
      MSG msg;

      //GetMessage() for data reception from the DT console board
      while( (GetMessage( (LPMSG)&msg,          // message structure
                        hWnd,        // handle of window receiving the message
                        0,           // lowest message to examine
                        0 ))  )        // highest message to examine
           {
                  TranslateMessage( (LPMSG)&msg );      // Translates virtual key codes
                  DispatchMessage( (LPMSG)&msg );       // Dispatches message to window

            //mode SYSTEM_NOT_ON: Waiting for the user to turn on channel 1 switch
            if(flag_a2d){
                  //Start acquiring data from channel 0
                  mode = SYSTEM_ON;

                  //Write 01 to LEDs 0,1 indicating data acquisition state
                  CHECK_ERROR (olDaPutSingleValue(board.hdass,1,0,1));

                  //Send an acknowledgement to the client regarding the beginning of data
acquisition
                  strcpy(numberstr,"S");
                  send(comm->datasock,numberstr,sizeof(numberstr) , 0);

                  index = 0;
                  k = 0;
                  flag_a2d = 0;
            }

            //mode SYSTEM_ON: Waiting for the analog switch to go off or receive a stop
command from the client
            if(flag_system_stop){
                  printf("System stopped !");

                  //mode SYSTEM_STOPPED: Stop acquiring data
                  mode = SYSTEM_STOPPED;
                  //Send an acknowledgement to the client regarding the termination
                        strcpy(numberstr,"T");
                  send(comm->datasock,numberstr,sizeof(numberstr) , 0);

                  //Write 11 to LEDs 0,1 indicating data acquisition state
                  CHECK_ERROR (olDaPutSingleValue(board.hdass,3,0,1));
                  olDaAbort((HDASS)hAD );
                  printf( "\nA/D Operation Terminated \n" );
                  break;            //Exit the loop for data processing
             }

             //mode SYSTEM_ON: Waiting for the client to send a pause command
             if(flag_system_pause){
                  printf("\nSystem paused!");
                        flag_system_resume = 0;
                  //mode SYSTEM_PAUSED: Waiting for the client to send a resume command
                        mode = SYSTEM_PAUSED;

                  //Write 10 to LEDs 0,1 indicating data acquisition state
                        CHECK_ERROR (olDaPutSingleValue(board.hdass,2,0,1));
                  olDaStop(HDASS(hAD));

                  //Send an acknowledgement to the client regarding the system pause
                  strcpy(numberstr,"P");
                  send(comm->datasock,numberstr,sizeof(numberstr) , 0);

                  //Wait until a resume "r" command received
                  while(!flag_system_resume);

                  //continue acquiring data, resume normal data acquisition
                  mode = SYSTEM_ON;

                  //Write 01 to LEDs 0,1 indicating data acquisition state
                        CHECK_ERROR (olDaPutSingleValue(board.hdass,1,0,1));

                  flag_system_pause = 0;
                        printf("Resuming processing!");
                  olDaStart(HDASS(hAD));

                  //Send an acknowledgement to the client regarding the system resuming
normal operation again
                  strcpy(numberstr,"R");
                  send(comm->datasock,numberstr,sizeof(numberstr) , 0);
            }
      }

      PostQuitMessage(0);

        int ii =0;
        for( ii=0; ii<NUM_OL_BUFFERS; ii++ ){
                  olDmFreeBuffer( hBufs[ii] );
        }

        olDaTerminate( hDev );
      printf("\nAD system terminated");

      //Convert the data acquired at channel 0 into voltage format
      //Store it in a text file
      FILE *fp;
      fp = fopen("samples.txt","w");
      for(ii=0;ii<index;ii++){
            if(ii%2 == 0){
            volts = ((float)max-(float)min)/(1L<<resolution)*value[ii] + (float)min;
            fprintf(fp,"%.3f\n",volts);
            }
      }
      fclose(fp);

      //Release the DT module
      CHECK_ERROR (olDaReleaseDASS(board.hdass));
        CHECK_ERROR (olDaTerminate(board.hdrvr));

      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,a;
      char ParamBuffer[100] = "" ;
      int f = 0, i = 0;
      static int r = 0;

      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 SYSTEM_OFF:
                  sampling_freq = atof(ParamBuffer);
                  flag_system_on = 1;
                  break;
            case WAIT_FOR_START:
                  if((strcmp(ParamBuffer,"s")==0)){
                        printf("\nStart command received!");
                        flag_wait_for_start = 1;
                  }
                  else
                        printf("\nError:Expected command - start");
                  break;

            case SYSTEM_ON:
                  if(strcmp(ParamBuffer,"p")==0){
                        printf("\nPause command received!");
                        flag_system_pause = 1;
                  }
                  else{
                        if(strcmp(ParamBuffer,"t")==0){
                              printf("\nStopcommand received!");
                              flag_system_stop = 1;
                        }
                        else
                              printf("Error:Expected command - pause or
stop\nWarning:Enter a valid command");}
                  break;

            case SYSTEM_PAUSED:
                  if(strcmp(ParamBuffer,"r")==0){
                        printf("\nResume command received!");
                        flag_system_resume = 1;
                  }
                  else{
                        printf("The received string is:%s",ParamBuffer);
                        printf("\nError: Expected command - resume\nWarning: Enter a
valid command");
                  }
                  break;

            }
      x = 0;
      }

}
```