```
//    Embedded Systems
//    BeagleBoard-XM

//    Author : Jose Goncalves
//    jose.braga.pt@gmail.com

//    Teacher : Nuno Peixoto
//    University : IPCA
//    17.03.2014



#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <strings.h>
#include <fcntl.h>
#include <errno.h>


//------------------------------------------------- improvments to do  -------------------------------------------------------
// using strings instead of sequential characteres : 'T' + 'E' + 'M' + 'P', ......



using namespace std ;

// ---------------------------------------------------------------------------------------------------------------------------
// --------------------------------------------------- Prototypes ------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------


#include "Gpio.h"
// Class Gpio

#include "I2C_sensor.c"
//char* Temp()  return Temperature value
//char* HR()    return relative humidity value

// Threads functions:
void *connection_handler(void *) ;
void *GPIO_Input(void *inPort);

void respond (int sock, Gpio Led1,Gpio Led2) ;
void error(char *msg) ;


// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------- main () --------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------


int main()
{


     // Código do daemon -------------------------------------------------------

     pid_t pid, sid;
     pid = fork();

     if(pid < 0)
     {
     fprintf(stderr, "Fatal error. Cannot initiate the daemon!\n");
     exit(EXIT_FAILURE);
     }

     if(pid > 0) exit(EXIT_SUCCESS) ;
     // Change the file mode mask
     umask(0);
     // Open any logs here
     //...

     // Create a new SID for the child process

     sid = setsid();
     if(sid < 0)
     {
     // Log any failure
     exit(EXIT_FAILURE);
     }


     // Close out the standard file descriptors

     close(STDIN_FILENO);      // inibe stdin, stdout, stderr.  por exemplo inibe os printf's()
     close(STDOUT_FILENO);
     close(STDERR_FILENO);

     // The big loop

     while(1)
     {
     // O nosso código vem aqui…
```

```c
    // create thread GPIO_Input ---------------------------------------------------------

    pthread_t thread_input;
    int *p ;

    if( pthread_create( &thread_input , NULL ,  GPIO_Input , (void*) &p) < 0)
    {
        perror("could not create thread");
        return 1;
    }

    // Create socket --------------------------------------------------------------------

    int socket_desc , client_sock , c;

    // server is a structure of type struct sockaddr_in. This structure has 4 fields.
    struct sockaddr_in server , client;

    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
        printf("Could not create socket");
    puts("Socket created");

    // Prepare the sockaddr_in structure ------------------------------------------------

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;  // INADDR_ANY : symbolic constant which gets the IP address
    server.sin_port = htons( 5000 );

    // Bind -----------------------------------------------------------------------------
    // Assign socket to the IP address + port number of the host

    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
        {
        perror("bind failed. Error");
        return 1;
        }
    puts("bind done");

    // Listen ----------------------------------------------------------------------------
    // initialize a wait queue of connections for socket socket_desc
    // maximum of 5 clients in the queue of pending connections

    listen(socket_desc , 5);


    // Accept and incoming connection
    // It extracts the first connection request on the queue of pending connections for the listening socket, sockfd,
    // creates a new connected socket, and returns a new file descriptor referring to that socket.
    // The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.


    pthread_t thread_id;
    puts("Waiting for first incoming connections...");
    c = sizeof(struct sockaddr_in);

    while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c)) )
    {

        puts("Connection accepted");
        printf("Socket client ID : %d \n",client_sock);

        // create thread connection_handler(void *socket_desc) --------------------------

        if( pthread_create( &thread_id , NULL ,  connection_handler , (void*) &client_sock) < 0)
        {
            perror("could not create thread");
            return 1;
        }

        // Now join the thread , so that we don't terminate before the thread
        // pthread_join( thread_id , NULL);
        puts("Handler assigned");
    }

    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }


     }

         exit(EXIT_SUCCESS);



    return 0;
}





//---------------------------------------------------------------------------------------------------------------------
//---------------------------------------------------------------------------------------------------------------------
```

```
//---------------------------------------  This will handle connection for each client --------------------------------


void *connection_handler(void *SocketCliente)
{
    //Get the socket descriptor
    int sock = *(int*)SocketCliente;
    int read_size;
    char *message , client_message[20];

    // Send some messages to the client
    message = "Conexao bem sucedida ao servidor : 192.168.1.63:5000 \n";
    write(sock , message , strlen(message));

    // Receive a message from client
    // MSG_PEEK : the data is treated as unread, look at the data but don't remove it from the input queue.
    // if flag argument was zero, you could as well use "read" instead "recv"

    Gpio Led1,Led2 ;

    while( (read_size = recv(sock , client_message , 20 , MSG_PEEK)) > 0 )
    {
         respond(sock,Led1,Led2) ;
    }

    if(read_size == 0)
        {
        puts("Client disconnected");
        fflush(stdout);
        }

    else if(read_size == -1)
        {
        perror("recv failed");
        close(sock);
        }

    return 0;
}



// ----------------------------------------------------------------------------------------------------------------------------
// ----------------------------------------------------------- respond()--------------------------------------------------------
// ----------------------------------------------------------------------------------------------------------------------------



void respond (int sock,Gpio Led1,Gpio Led2)
{
   int n;
   char buffer[256];
   int b = sock ;

   printf("respond sock %d \n", b);

   // bzero() sets all values in a buffer to zero.
   // arguments : ( pointer to buffer, size of buffer )
   bzero(buffer,256);

   // read buffer
   n = read(sock,buffer,255);

   //if (n < 0) error("ERRO na leitura de socket\n");
   //printf("mensagem recebida: %s",buffer);

   // LED:1:0 command received

   //string str = buffer;
   //sprintf(buffer, str.c_str());;
   //str.compare("LED:1:0");

   if ( (buffer[0]== 'L') && (buffer[1]== 'E') && (buffer[2]== 'D') && (buffer[3]== ':') && (buffer[4]== '1') && (buffer[5]== ':') && (buffer[6])=='0' )
           {
           printf ("desligar LED 1 \n\n");

           Led1.Gpo(138,0);

           //
           write(sock,"LED 1 DESLIGADO",15);
           if (n < 0) error("ERRO na escrita de socket");
           return ;
           }


   // LED:1:1 command received

   if ( (buffer[0]== 'L') && (buffer[1]== 'E') && (buffer[2]== 'D') && (buffer[3]== ':') && (buffer[4]== '1') && (buffer[5]== ':') && (buffer[6])=='1' )
           {
           printf ("ligar LED 1 \n\n");

           Led1.Gpo(138,1);

           write(sock,"LED 1 LIGADO",12);
           if (n < 0) error("ERRO na escrita de socket");
           return ;
           }



   // LED:2:0 command received
   if ( (buffer[0]== 'L') && (buffer[1]== 'E') && (buffer[2]== 'D') && (buffer[3]== ':') && (buffer[4]== '2') && (buffer[5]== ':') && (buffer[6])=='0' )
           {
           printf ("desligar LED 2 \n\n");

           Led2.Gpo(139,0);

           write(sock,"LED 2 DESLIGADO",15);
           if (n < 0) error("ERRO na escrita de socket");
           return ;
           }


   // LED:2:1 command received

   if ( (buffer[0]== 'L') && (buffer[1]== 'E') && (buffer[2]== 'D') && (buffer[3]== ':') && (buffer[4]== '2') && (buffer[5]== ':') && (buffer[6])=='1' )
           {
           printf ("ligar LED 2 \n\n");

           Led2.Gpo(139,1);

           write(sock,"LED 2 LIGADO",12);
           if (n < 0) error("ERRO na escrita de socket");
           return ;
           }


   // TEMP command received

   if ( (buffer[0]=='T') &&  (buffer[1]=='E') && (buffer[2]=='M') && (buffer[3]=='P') )

          {

            // float temp = Temp() ;
            // sprintf(ch,"%.2f",temp);
            // printf ("Temperatura %.3f \n\n", temp );

            n = write(sock,Temp(),10);

            //n = write(sock,"mensagem recebida\n",18);
            if (n < 0) error("ERRO na escrita de socket");
            return;

          }


   // HR command received

   if ( (buffer[0]=='H') &&  (buffer[1]=='R') )
          {
            //float hr = HR() ;
            //sprintf(ch,"%.2f",hr);
            //printf ("Humidade relativa %.3f \n\n",hr);

            n = write(sock,HR(),8);
            //n = write(sock,"mensagem recebida\n",18);
            if (n < 0) error("ERRO na escrita de socket");
            return;
          }

   n = write(sock,"mensagem errada",15);


}






//----------------------------------------------------------------------------------------------------------------------
//----------------------------------------------------------------------------------------------------------------------
//----------------------------------------------------------------------------------------------------------------------



void *GPIO_Input(void *inPort)
  {

  // Class Gpio teve de ser alterada : memory map teve de ser declarada no construtor da class
  // estava inicialmente definido nos modos Gpi() e Gpo()
  // caso contrario abria a cada leitura da porta um novo file descriptor
  // e quando chegasse a 1023 fd abertos , dava erro de "Segmentation Fault".

  int in = *(int*)inPort;
  int status = 0, old_status = 0 ;
  Gpio Input ;
  Gpio Led ;
  while (1)
       {
       status = Input.Gpi(137) ;

       if (status != old_status)
           {
           Led.Gpo(138,0);
           usleep(300000);
           Led.Gpo(138,1);
           usleep(300000);
           Led.Gpo(138,0);
           }

       old_status = status;
       usleep(1000);
       }
  exit(0);
  }



//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------



void error(char *msg)
   {
       perror(msg);
       exit(1);
   }
```