```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include <math.h>
#include "../Includes/light_task.h"


void test_all_reg_rd_wr(void** state)
{
        int fd = light_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = all_reg_rd_wr(fd);
        assert_int_equal(status, 0);

}


void test_control_reg_int(void **state)
{

        int fd = light_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = control_reg_int_wr(fd, 0x10);
        assert_int_not_equal(status, -1);
        status = control_reg_int_rd(fd);
        assert_int_not_equal(status, -1);
}



void test_get_lux(void **state){
        int fd = light_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = get_lux();
        assert_int_not_equal(status, -1);
}


int main(void)
{
    const struct CMUnitTest tests[] =
    {
        cmocka_unit_test(test_all_reg_rd_wr),
        cmocka_unit_test(test_control_reg_int),
        cmocka_unit_test(test_get_lux)
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdarg.h>
```

```
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include <math.h>
#include "../Includes/temp_task.h"



void test_all_temprg_rd_wr(void** state)
{
        int fd = temp_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = all_temprg_rd_wr();
        assert_int_equal(status, 0);

}


void test_tlowcheck(void **state)
{

        int fd = temp_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = write_tlow_reg(0x02, 45);
        assert_int_not_equal(status, -1);
        status = read_tlow_reg(0x02);
        assert_int_not_equal(status, -1);
}



void test_read_temp_data_reg(void **state)
{
        int fd = temp_init();
        int status;
        assert_int_not_equal(fd, -1);
        status = read_temp_data_reg(0);
        assert_int_not_equal(status, -300);
}


int main(void)
{
    const struct CMUnitTest tests[] =
    {
        cmocka_unit_test(test_all_temprg_rd_wr),
        cmocka_unit_test(test_tlowcheck),
        cmocka_unit_test(test_read_temp_data_reg)
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}
#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>
```

```
#define PORT_ADR    2000

typedef struct
{
  char    buf[20];
  int  buf_len;
  bool    usrLED_OnOff;
}payload_t;


int socket_task()
{
  struct sockaddr_in addr, peer_addr;
  int addr_len = sizeof(peer_addr);
  char rdbuff[1024] = {0};
  int server_socket, accepted_soc, opt = 1;
  int i = 0;
  payload_t *ploadptr;
  int read_b;
  int pload_len = 0;
  char ackbuf[50];
  float temp,lumen;


  /* create socket */
  if((server_socket = socket(AF_INET,SOCK_STREAM,0)) == 0)
  {
    printf("[Server] [ERROR] Socket Creation Error\n");
    return 1;
  }
  else
    printf("[Server] Socket Created Successfully\n");

  /* set socket options */
  if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &(opt), sizeof(opt)))
  {
    printf("[Server] [ERROR] Socket options set error\n");
    return 1;
  }

  /*Set the sockaddr_in structure */
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(PORT_ADR);

  /*bind socket to a address */
  if((bind(server_socket,(struct sockaddr*)&addr, sizeof(addr))) < 0)
  {
    printf("[Server] [ERROR] Bind socket Error\n");
    return 1;
  }
  else
    printf("[Server] Socket binded Successfully\n");



  /* listen for connections*/
  if(listen(server_socket,5) < 0)
  {
    printf("[Server] [ERROR] Can't listen connection\n");
    return 1;
  }
while(1)
 {
  /*accept connection */
  accepted_soc = accept(server_socket, (struct sockaddr*)&peer_addr,
(socklen_t*)&addr_len);
  if(accepted_soc < 0)
  {
    printf("[Server] [ERROR] Can't accept connection\n");
    return 1;
  }

  /* read payload length */
  read_b = read(accepted_soc, &pload_len, sizeof(int));
  if(read_b == sizeof(int))
  {
    printf("[Server] Size of incoming payload: %d\n",pload_len);
  }
  else
  {
    printf("[Server] [ERROR] Invalid data\n");
    return 1;
  }

  /* read payload */
  while((read_b = read(accepted_soc, rdbuff+i, 1024)) < pload_len)
  {
    i+=read_b;
  }
  ploadptr= (payload_t*)rdbuff;
  /* display data */
  printf("[Server] Message Recvd from Client\n{\n Message:%s\n MessageLen:%d\n USRLED:
%d\n}\n",ploadptr->buf, ploadptr->buf_len, ploadptr->usrLED_OnOff);

  if(strcmp(ploadptr->buf,"get_temp_celcius")==0)
  {
        printf("You Want Temperature in Celcius\n");
    temp = read_temp_data_reg(0);
    //printf("Temp in cel %f\n",temp );
    snprintf(ackbuf, 50, "Temp in celcius %f",temp);
    send(accepted_soc , ackbuf , 50, 0);
  }
  else if(strcmp(ploadptr->buf,"get_temp_kelvin")==0)
  {
        printf("You Want Temperature in Kelvin\n");
    temp = read_temp_data_reg(1);
    //printf("Temp in cel %f\n",temp );
    snprintf(ackbuf, 50, "Temp in kelvin  %f",temp);
    send(accepted_soc , ackbuf , 50, 0);
  }
  else if(strcmp(ploadptr->buf,"get_temp_fahrenheit")==0)
  {
        printf("You Want Temperature in Fahrenheit\n");
    temp = read_temp_data_reg(2);
    //printf("Temp in cel %f\n",temp );
    snprintf(ackbuf, 50, "Temp in celcius %f",temp);
    send(accepted_soc , ackbuf , 50, 0);
  }
  else if(strcmp(ploadptr->buf,"isitday")==0)
  {
        printf("Day ? Don't Know!!\n");
    lumen = get_lux();
    if(lumen < 10)
    {
      send(accepted_soc , "No, it is Night" , 50, 0);
    }
    else
    {
      send(accepted_soc , "Yes, it is Day" , 50, 0);
    }
  }
  else if(strcmp(ploadptr->buf,"isitnight")==0)
  {
        printf("Night ? Don't Know!!\n");
    lumen = get_lux();
    if(lumen < 10)
    {
      send(accepted_soc , "Yes, it is Night" , 50, 0);
    }
    else
    {
      send(accepted_soc , "No, it is Day" , 50, 0);
    }
  }
  else if(strcmp(ploadptr->buf,"get_lux")==0)
  {
    printf("You want the lumen value!!\n");
    lumen = get_lux();
    printf("Lux value %f\n",lumen );
    snprintf(ackbuf, 50, "Lux Value is %f",lumen);
    send(accepted_soc , ackbuf , 50, 0);

  }else
  {
        printf("I Don't Understand !!");
    send(accepted_soc , "I Don't Understand !!" , 50, 0);
  }

  /* send message from server to client */
//  send(accepted_soc , "ACK" , 4, 0);
//  printf("[Server] Message sent from Server: ACK\n");

  /*close socket */
  close(accepted_soc);
        }
  return 0;
}
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <assert.h>
#include <stdint.h>

int fd;
char *bus = "/dev/i2c-2";     /* Pins P9_19 and P9_20 */
```

```
int addr = 0x48;              /* The I2C address of TMP102 */
char buf[2] = {0},pika=0x00;
int temp;
unsigned char MSB, LSB;
float f,c,k;

typedef enum
{
  TEMP_CEL = 0,
  TEMP_KEL = 1,
  TEMP_FAH = 2
}temp_format;

int default_config_byte_one = 0x50;
int default_config_byte_two = 0xA0;

void write_pointer_reg(uint8_t value)
{
  if(write(fd, &value, 1) != 1)
  {
    perror("Write Pointer Register Error\n");
  }
}

int write_tlow_reg(int reg, uint16_t value )
{
  write_pointer_reg(reg);

  if (write(fd, &value, 2) != 2)
  {
    perror("T-low register write error");
    return -1;
  }
  return 0;
}


int write_config_reg_on_off(uint8_t value )
{
  write_pointer_reg(0b00000001);
  if((value == 0) || (value == 1))
  {
    default_config_byte_one |= value;

    if (write(fd, &default_config_byte_one, 1) != 1)
    {
      perror("Configuration register write error for first byte");
        return -1;
    }

    if (write(fd, &default_config_byte_two, 1) != 1)
    {
      perror("Configuration register write error for second byte");
        return -1;
    }
  }
return 0;
}

int write_config_reg_em(uint8_t value )
{
  write_pointer_reg(0b00000001);
  if((value == 0) || (value == 1))
  {
    default_config_byte_two |= (value << 4);

    if (write(fd, &default_config_byte_one, 1) != 1)
    {
      perror("Configuration register write error for first byte");
        return -1;
    }

    if (write(fd, &default_config_byte_two, 1) != 1)
    {
      perror("Configuration register write error for second byte");
        return -1;
    }
  }
return 0;
}

int write_config_reg_conv_rate(uint8_t value )
{
  write_pointer_reg(0b00000001);
  if((value >= 0) || (value <= 3))
  {
    default_config_byte_two |= (value << 6);
    if (write(fd, &default_config_byte_one, 1) != 1)
    {
      perror("Configuration register write error for first byte");
        return -1;
    }
    if (write(fd, &default_config_byte_two, 1) != 1)
    {
      perror("Configuration register write error for second byte");
        return -1;
    }
  }
return 0;
}

int write_config_register_default( )
{
  write_pointer_reg(0b00000001);
  if (write(fd, &default_config_byte_one, 1) != 1)
  {
    perror("Configuration register write error for first byte");
        return -1;
  }
  if (write(fd, &default_config_byte_two, 1) != 1)
  {
    perror("Configuration register write error for second byte");
        return -1;
  }
return 0;
}



uint16_t read_tlow_reg(int reg)
{
  uint16_t value;
  uint8_t v[1]={0};
  write_pointer_reg(reg);
  if (read(fd, v, 1) != 1)
  {
    perror("T-low register read error");
    return -1;
  }
  value = (v[0]<<4 | (v[1] >> 4 & 0XF));
  printf("T-low register value is: %d \n", value);
  return value;
}

uint16_t read_temp_config_register()
{
  uint16_t value;
  uint8_t v[1]={0};
  write_pointer_reg(0b00000001);
  if (read(fd, v, 1) != 1)
  {
    perror("Temperature configuration register read error");
    return -1;
  }
  value = (v[0]<<8 | v[1]);
  printf("Temperature configuration register value is: %d \n", value);
  return value;
}

int all_temprg_rd_wr()
{
if (write_config_reg_on_off(1) < 0)
{
return -1;
}

if (write_config_reg_em(1) < 0)
{
return -1;
}

if (read_temp_config_register() < 0)
{
return -1;
}

if (write_config_register_default() < 0)
{
return -1;
}

if (write_config_reg_conv_rate(2) < 0)
{
return -1;
}

if (write_tlow_reg(0x02,45) < 0)
{
return -1;
}

if (read_tlow_reg(0x02) < 0)
{
return -1;
}
```

```
return 0;

}

float read_temp_data_reg(int unit)
{

  write_pointer_reg(0b00000000);

  int x = read(fd,&buf,2);

  //printf("number of bytes read = %d\n",x);

  if (x != 2)
  {
    /* ERROR HANDLING: i2c transaction failed */
    perror("Failed to read from the i2c bus.\n");
    printf("ERROR : %s\n", strerror(errno));
    return -300;
  }
  else
  {

    MSB = buf[0];
    LSB = buf[1];

    /* Convert 12bit int using two's compliment */
    temp = ((MSB << 8) | LSB) >> 4;

    c = temp*0.0625;
    f = (1.8 * c) + 32;
    k = c + 273.15;
    //printf("Temp Fahrenheit: %f Celsius: %f\n", f, c);

    if(unit == TEMP_CEL)
      return c;
    else if(unit == TEMP_KEL)
      return k;
    else if(unit == TEMP_FAH)
      return f;
  }
}

int temp_init()
{
  if((fd = open(bus, O_RDWR)) < 0)
  {
    perror("Failed to open the i2c bus");
    /* ERROR HANDLING: you can check errno to see what went wrong */
    return -1;
  }

  if(ioctl(fd, I2C_SLAVE, addr) < 0)
  {
    perror("Failed to open the i2c bus");
    /* ERROR HANDLING: you can check errno to see what went wrong */
    return -1;
  }

  return 0;
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <math.h>
#include <float.h>
#include <complex.h>
#include <stdint.h>
#include "../Includes/light_task.h"
#include <time.h>

int a;

int control_reg_wr ( int fd, int msg)
{
int temp = 0x80;                      //for control register
if(write(fd,&temp,1) != 1)
{
printf("Error in writing to control register\n");
return -1;
}

if(write(fd,&msg,1) != 1)
{
printf("Error in writing message to control register\n");
return -1;
}
return 0;
}

int control_reg_rd ( int fd)
{
int temp = 0x80 ;                     //for control register
if(write(fd,&temp,1) != 1)
{
printf("Error in writing from control register\n");
return -1;
}

if(read(fd,&temp,1) != 1)
{
printf("Error in reading message from control register\n");
return -1;
}
return temp;
}

int timing_reg_wr ( int fd, int msg)
{
int temp = 0x81  ;                    //for timing register
if(write(fd,&temp,1) != 1)
{
printf("Error in writing to control register\n");
return -1;
}

if(write(fd,&msg,1) != 1)
{
printf("Error in writing message to control register\n");
return -1;
}
return 0;
}

int timing_reg_rd(int fd){
int temp =  0x81;                    //for timing register
if( write(fd, &temp, 1) != 1)
{
printf("Error in writing to control register\n");
return -1;
}

if( read(fd, &temp, 1) != 1)
{
printf("Error in writing message to control register\n");
return -1;
}

return temp;
}


int control_reg_int_wr(int fd, int msg)
{
int temp = 0x86;                    //for interrupt control register
if( write(fd, &temp, 1) != 1)
{
printf("Error in writing to interrupt control register\n");
return -1;
}

if( write(fd, &msg, 1) != 1)
{
printf("Error in writing message to interrupt control register\n");
return -1;
}
return 0;
}

/*read from interrupt control register */
int control_reg_int_rd(int fd)
{
int temp =  0x86;
if( write(fd, &temp, 1) != 1)
{
printf("Error in writing to control interrupt register\n");
return -1;
}
if( read(fd, &temp, 1) != 1)
{
printf("Error in reading from control interrupt register\n");
return -1;
}

return temp;
```

```
}

int threshold_int_reg_wr(int fd, int *array)

{
        int temp = 0x82;                                //for threshold low-low to 
threshold high-high
        if( write(fd, &temp, 1) != 1)
         {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        temp = array[0];
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }

        temp = 0x83;
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        temp = array[1];
        if( write(fd, &temp, 1) != 1)
        {
        printf("Unable to write to threshhold interrupt register\n");
        return -1;
        }

        temp = 0x84;
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable write to threshhold interrupt register\n");
                return -1;
        }
        temp = array[2];

        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }

        temp = 0x85;
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        temp = array[3];
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        return 0;

}
```

```
/*Read value from interrupt_threshhold register
 * Returns either read value and fail on failure
 */
int threshold_int_reg_rd(int fd, int *array){

        int temp =  0x82 ;                          //for threshold low-low to
threshold high-high
        if( write(fd, &temp, 1) != 1){
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        if( read(fd, &temp, 1) != 1){
                printf("Unable to read from threshhold interrupt register\n");
                return -1;
        }
        array[0] = temp;

        temp =  0x83;
        if( write(fd, &temp, 1) != 1){
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        if( read(fd, &temp, 1) != 1){
                printf("Unable to read from threshhold interrupt register\n");
                return -1;
        }
        array[1] = temp;

        temp =  0x84 ;
        if( write(fd, &temp, 1) != 1){
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        if( read(fd, &temp, 1) != 1){
                printf("Unable to read from threshhold interrupt register\n");
                return -1;
        }
        array[2] = temp;

        temp =  0x85 ;
        if( write(fd, &temp, 1) != 1){
                printf("Unable to write to threshhold interrupt register\n");
                return -1;
        }
        if( read(fd, &temp, 1) != 1){
                printf("Unable to read from threshhold interrupt register\n");
                return -1;
        }
        array[3] = temp;
        return 0;

}

int id_reg_rd(int fd)
{
        int temp =  0x8A;                             //for id register
        if( write(fd, &temp, 1) != 1){
                printf("Unable to write to id register\n");
                return -1;
        }
        if( read(fd, &temp, 1) != 1){
                printf("Unable to read from id register\n");
                return -1;
        }

        return 0;
}

uint16_t data0_reg_rd(int fd){
        int temp =  0x8C;

        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to data 0 register\n");
                return -1;
        }
        uint8_t dlow0;

        if( read(fd, &dlow0, 1) != 1)
        {
                printf("Unable to read from data 0 register\n");
                return -1;
        }

        temp =  0x8D;
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to data 0 register\n");
                return -1;
        }
        uint16_t dhigh0;

        if( read(fd, &dhigh0, 1) != 1)
        {
                printf("Unable to read from write 0 register\n");
                return -1;
        }


        uint16_t final = dhigh0<<8 | dlow0;
        return final;
}

uint16_t data1_reg_rd(int fd){
        int temp =  0x8E;

        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to data 1 register\n");
                return -1;
        }
        uint8_t dlow1;

        if( read(fd, &dlow1, 1) != 1)
        {
                printf("Unable to read from data 1 register\n");
                return -1;
        }

        temp =  0x8F;
        if( write(fd, &temp, 1) != 1)
        {
                printf("Unable to write to data 1 register\n");
                return -1;
        }
        uint16_t dhigh1;

        if( read(fd, &dhigh1, 1) != 1)
        {
                printf("Unable to read from write 1 register\n");
                return -1;
        }


        uint16_t final = dhigh1<<8 | dlow1;
        return final;
}

int all_reg_rd_wr(int fd)

        {
        int out;
        out = control_reg_wr(fd, 0x03);
        if( out ==  -1 )
                return -1;

        out = control_reg_rd(fd);
        if( out ==  -1 )
                return -1;
        out = timing_reg_wr(fd, 0x12);
        if( out ==  -1 )
                return -1;
        out = timing_reg_rd(fd);
        if( out ==  -1 )
                return -1;
        /*Interrupt threshhold register reads 4 bytes*/

        int array[4] = {0, 0, 0, 0};

        out = threshold_int_reg_rd(fd, array);
        if( out ==  -1 )
        return -1;

        int arr[1] = {0x0F};
        out = threshold_int_reg_wr(fd, arr);
        if( out ==  -1 )
                return -1;

        out = id_reg_rd(fd);
        if( out ==  -1 )
                return -1;

        out = data0_reg_rd(fd);
        if( out ==  -1 )
                return -1;

        out = data1_reg_rd(fd);
        if( out ==  -1 )
                return -1;

        return 0;

}

int light_init(void)
{
        int file;
//      char myfile[20];

        char *myfile = "/dev/i2c-2";
        file = open(myfile, O_RDWR);
        if (file < 0)
         {
                perror("Unable to open the i2c file.\n");
                return -1;
        }
        int addr = 0x39; //The I2C slave address

        if (ioctl(file, I2C_SLAVE, addr) < 0)
         {
                perror("Unable to use ioctl call.\n");
                return -1;
        }
        a=file;
        return file;
}

float get_lux()
{
        float ch_0 = 0, ch_1 = 0;
        float adc,lux;

        if(control_reg_wr(a, 0x03) < 0) //to power up the sensor
        {
          return -1;
        }
        if(timing_reg_wr(a, time_high|gain) < 0)
        {
         return -1;
        }

        //usleep(5000);

        ch_0 = (float)data0_reg_rd(a);
        ch_1 = (float)data1_reg_rd(a);

        adc = ch_1/ch_0;

        /*As per datasheet*/
        if(adc>0 && adc <= 0.5)
        return lux = (0.0304 * ch_0) - (0.062 * ch_0 * pow(adc, 1.4));

        else if((adc>0.5) && (adc<=0.61))
        return lux = (0.0224 * ch_0) - (0.031 * ch_1);

        else if((adc>0.61)&&(adc<=0.80))
        return lux = (0.0128 * ch_0) - (0.0153 * ch_1);

        else if((adc>0.80) && (adc<=1.30))
        return lux = (0.00146 * ch_0) - (0.00112 * ch_1);

        else if(adc > 1.30)
        return lux = 0;

return -1;
}
/*
int main()
```

```
{int file;
file = light_init(2);
time_t curtime;
time(&curtime);
while(1)
{
float lumen = get_lux(file);
printf("Time stamp: %s",ctime(&curtime));
printf("Length of time stamp : %ld",strlen(ctime(&curtime)));
printf("The current lux is %f\n", lumen);
}
return 0;
}
*/
#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>


#define PORT_ADR    2000
#define IP_ADR      "127.0.0.1" /* Loppback IP Address*/

typedef struct
{
  char    buf[20];
  size_t  buf_len;
  bool    usrLED_OnOff;
}payload_t;


int main()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  char msg[20] = "Message from Client";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;*/

while(1)
{
  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
  else
    printf("[Client] Socket Created Successfully\n");

  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(PORT_ADR);




  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }



  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }


    printf("\n\n Enter The API Message :");
  gets(&msg);



  printf("\n You entered : %s\n",msg);
  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n
USRLED: %d\n}\n", \
                           ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  read(client_socket, r_data, 25);
  printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
  close(client_socket);
  }
  return 0;
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <math.h>
#include <float.h>
#include <complex.h>
#include <stdint.h>
#include <time.h>
#include <mqueue.h>
#include "../Includes/light_task.h"
#include "light_task.c"
#include "../Includes/temp_task.h"
#include "temp_task.c"
#include "socket_task.c"

#define HB_PORT_ADR 5000
#define IP_ADR      "127.0.0.1"

pthread_t logger_id, light_id, temp_id, socket_id;

char file_name[50];

typedef struct              //structure to be sent
{
char timestamp[50];
int source_id;
int log_level;
int data;
float value;
char random_string[50];
}mystruct;

struct threadParam
{
char *filename;
};

void func_led_off()
{
        FILE *LED1 = NULL;
        char *LED2 = "/sys/class/leds/beaglebone:green:usr2/brightness";

        LED1 = fopen(LED2, "r+");
        fwrite("0", sizeof(char), 1, LED1);
        fclose(LED1);
}

void func_led_on()
{
        FILE *LED1 = NULL;
        char *LED2 = "/sys/class/leds/beaglebone:green:usr2/brightness";
        LED1 = fopen(LED2, "r+");
        fwrite("1", sizeof(char), 1, LED1);
        fclose(LED1);
}

int light_client()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  const char* msg = "Light Alive";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;

  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    //printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
  else
    //printf("[Client] Socket Created Successfully\n");

  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(HB_PORT_ADR);

  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    //printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }

  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    //printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  //printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    //printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  //printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n
USRLED: %d\n}\n", \
                           ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  //read(client_socket, r_data, 4);
  //printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
  close(client_socket);

  //return 0;

}

void *func_light()
{       mqd_t mq1;
        printf("[Light Thread] Light Thread Started\n");
        mystruct lightmsg;
        time_t curtime;
        int x = light_init();
        if(x != -1)
        {
          mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
          time(&curtime);
          memcpy(lightmsg.timestamp,ctime(&curtime), strlen(ctime(&curtime)));
          memcpy(lightmsg.random_string,"Light task initiated",20);
          lightmsg.log_level = 0;
          lightmsg.source_id = 1;
          lightmsg.data =1;
          mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1);
                while(1)
          {
            //time_t curtime;
            time(&curtime);
            float lumen = get_lux();

            if(lumen < 0 )
                        {

                        time(&curtime);
                memcpy(lightmsg.random_string,"Error in getting data from light
task",strlen("Error in getting data from light task"));
                memcpy(lightmsg.timestamp,ctime(&curtime),24);
                lightmsg.source_id = 1;
                lightmsg.log_level = 2;
                lightmsg.data = 1;
                          mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1);
              sleep(1);
                        }
            else
            {
             memcpy(lightmsg.random_string,"Light data obtained",19);
                          memcpy(lightmsg.timestamp,ctime(&curtime),24);
              lightmsg.source_id = 1;
              lightmsg.data = 1;
              lightmsg.log_level = 1;
              mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1);

              //printf("The current lux is %f\n", lumen);
              memcpy(lightmsg.timestamp,ctime(&curtime), strlen(ctime(&curtime)));
              lightmsg.data = 0;
              lightmsg.value = lumen;
              if( mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1)== -1)
              {
                printf("Sending failed\n");
              }
              sleep(1);
              light_client();
            //exit(0);
            }
          }
        }
        else
        {
          printf("[Light Thread] Error initialising light task\n");
          time(&curtime);
          memcpy(lightmsg.random_string,"Error in initialising light task",strlen
("Error in initialising light task"));
          memcpy(lightmsg.timestamp,ctime(&curtime),24);
          lightmsg.source_id = 2;
          lightmsg.data = 1;
          lightmsg.log_level=2;
                mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1);
        }


        printf("[Light Thread] Light Thread Finished\n");
              time(&curtime);
        memcpy(lightmsg.random_string,"Light task finished",19);
        memcpy(lightmsg.timestamp,ctime(&curtime),24);
        lightmsg.source_id = 2;
        lightmsg.data = 1;
        lightmsg.log_level= 1;
        mq_send(mq1,(char *)&lightmsg,sizeof(lightmsg),1);

}



int temp_client()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  const char* msg = "Temp Alive";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;

  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    //printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
  else
    //printf("[Client] Socket Created Successfully\n");
  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(HB_PORT_ADR);

  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    //printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }

  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    //printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  //printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    //printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  //printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n 
USRLED: %d\n}\n", \
                           ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  //read(client_socket, r_data, 4);
  //printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
  close(client_socket);

  //return 0;

}

void *func_temp()
{
        mqd_t mq1;
        printf("[Temperature Thread] Temperature Thread Started\n");
        mystruct tempmsg;
        time_t curtime;
        char buffer[50] = {0};
        if(temp_init() != -1)
                    {
                          mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
                          memcpy(tempmsg.random_string,"Temperature task 
initiated",26);
                          time(&curtime);
                          memcpy(tempmsg.timestamp,ctime(&curtime),24);
                          tempmsg.source_id = 2;
                          tempmsg.log_level = 0;
                          tempmsg.data = 1;
                             mq_send(mq1,(char *)&tempmsg,sizeof(tempmsg),1);
                        //memcpy(,buffer, strlen(buffer));
                      while(1)
                {
                        time(&curtime);
                        float temp = read_temp_data_reg(0);
                        if(temp == -300)
                        {
                          time(&curtime);
                          memcpy(tempmsg.random_string,"Error in getting data from 
Temperature task",strlen("Error in getting data from Temperature task"));
                          memcpy(tempmsg.timestamp,ctime(&curtime),24);
                          tempmsg.source_id = 2;
                          tempmsg.data = 1;
                          tempmsg.log_level = 2;
                          mq_send(mq1,(char *)&tempmsg,sizeof(tempmsg),1);

                        }
                        else
                        {
                                           memcpy(tempmsg.random_string,"Temperature 
data obtained",25);
                                           memcpy(tempmsg.timestamp,ctime
(&curtime),24);
                         tempmsg.source_id = 2;
                         tempmsg.log_level = 1;
                         tempmsg.data = 1;
                                           mq_send(mq1,(char *)&tempmsg,sizeof
(tempmsg),1);

                          //printf("The current temp is %f\n", temp);
                          //tempmsg.source_id = 2;
                          tempmsg.data = 0;
                          tempmsg.value = temp;
                          time(&curtime);
                          memcpy(tempmsg.timestamp,ctime(&curtime),24);
                          if( mq_send(mq1,(char *)&tempmsg,sizeof(tempmsg),1)== -1)
                          {
                            printf("Sending failed\n");
                          }
                          temp_client();
                        }
                        sleep(1);

                }
                  }
      else
           {
        printf("[Temperature Thread] Error initialising temperature task\n");
        time(&curtime);
        memcpy(tempmsg.random_string,"Error in initialising temperature task",strlen
("Error in initialising temperature task"));
        memcpy(tempmsg.timestamp,ctime(&curtime),24);
        tempmsg.source_id = 2;
        tempmsg.data = 1;
        tempmsg.log_level = 2;
               mq_send(mq1,(char *)&tempmsg,sizeof(tempmsg),1);
         exit(0);
           }
        time(&curtime);
        memcpy(tempmsg.random_string,"Temperature task finished\n",25);
        memcpy(tempmsg.timestamp,ctime(&curtime),24);
        tempmsg.source_id = 2;
        tempmsg.log_level = 1;
        tempmsg.data = 1;
               mq_send(mq1,(char *)&tempmsg,sizeof(tempmsg),1);

        printf("[Temperature Thread] Temperature Thread Finished\n");

}

void* logger_task()
{
        FILE *fptr;
        mqd_t my_queue;

        fptr = fopen(file_name,"w");   //use logger_thread -> filename
        my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
        struct mq_attr *pact;
        pact = malloc(sizeof(struct mq_attr));
        mq_getattr(my_queue,pact);
        //fprintf(fptr,"Message queue initialised\n");
        printf("[Logger Thread] Message queue initialised.\n");
        fclose(fptr);
while(1)
{
        fptr = fopen(file_name,"a");
        //my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
        mystruct given;
        //struct mq_attr *pact;
        //pact = malloc(sizeof(struct mq_attr));
        //mq_getattr(my_queue,pact);

        mq_receive(my_queue,(char *)&given,pact->mq_msgsize,NULL);

        char buffer[25]={0}, stringbuffer[50] = {0} ;
        memcpy(buffer, given.timestamp, 24);
        memcpy(stringbuffer, given.random_string, strlen(given.random_string));

        if(given.source_id == 0)  //for main task
        {

        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d, Message from main task: %
s\n",buffer,given.source_id,given.log_level,stringbuffer);


        }

        else if(given.source_id == 1) //for light task
        {
        if(given.data == 0)
        {

        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d,Lux value: %f
\n",buffer,given.source_id,given.log_level,given.value);

        }
        else if(given.data == 1)
        {
```

```
        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d, Message from light task:
%s\n",buffer,given.source_id,given.log_level,stringbuffer);

        }
        }

        else if(given.source_id == 2)  //for temp task
        {
        if(given.data == 0)
        {

        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d, Temperature value: %f
\n",buffer,given.source_id,given.log_level,given.value);

        }
        else if(given.data == 1)
        {

        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d, Message from temp task: %
s\n",buffer,given.source_id,given.log_level,stringbuffer);

        }
        }

        else if(given.source_id == 3)  //for socket task
        {

        fprintf(fptr,"Timestamp:%s, Source ID:%d, Log ID:%d, Message from socket
task: %s\n",buffer,given.source_id,given.log_level,stringbuffer);


        }
        fclose(fptr);

}
        printf("[Logger Thread] Terminating message queue\n");
        return fptr;
}


void* func_socket()
{
  printf("[Socket Thread] Socket Task Started\n");
   time_t curtime;
   time(&curtime);
   mqd_t mq1;
  mystruct sample;
   //char buffer1[50] = {0};
   char buffer[50] = {0};
   //char my_stamp[25];

   //strncpy(buffer1,,27);
   strncpy(sample.random_string,"Socket task initiated", strlen("Socket task
initiated"));
   sample.source_id = 3;
   sample.log_level = 0;
   //my_stamp = ctime(&curtime);
   memcpy(buffer,ctime(&curtime),24);
   memcpy(sample.timestamp,buffer, strlen(buffer));

   mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
   if( mq_send(mq1,(char *)&sample,sizeof(sample),1)== -1)
        {
          printf("Sending failed\n");
        }

  socket_task();
  printf("[Socket Thread] Socket Task Finished\n");
}

int check_status()
{
  struct sockaddr_in addr, peer_addr;
  int addr_len = sizeof(peer_addr);
  char rdbuff[1024] = {0};
  int server_socket, accepted_soc, opt = 1;
  int i = 0;
  payload_t *ploadptr;
  int read_b;
  size_t pload_len = 0;

  /* create socket */
  if((server_socket = socket(AF_INET,SOCK_STREAM,0)) == 0)
  {
    printf("[HBServer] [ERROR] Socket Creation Error\n");
    return 1;
  }
  else
    printf("[HBServer] Socket Created Successfully\n");

  /* set socket options */
  if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &(opt), sizeof(opt)))
  {
    printf("[HBServer] [ERROR] Socket options set error\n");
    return 1;
  }

  /*Set the sockaddr_in structure */
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(HB_PORT_ADR);

  /*bind socket to a address */
  if((bind(server_socket,(struct sockaddr*)&addr, sizeof(addr))) < 0)
  {
    printf("[HBServer] [ERROR] Bind socket Error\n");
    return 1;
  }
  else
    printf("[HBServer] Socket binded Successfully\n");

  /* listen for connections*/
  if(listen(server_socket,5) < 0)
  {
    printf("[HBServer] [ERROR] Can't listen connection\n");
    return 1;
  }
while(1)
{
  /*accept connection */
  accepted_soc = accept(server_socket, (struct sockaddr*)&peer_addr,
(socklen_t*)&addr_len);
  if(accepted_soc < 0)
  {
    printf("[HBServer] [ERROR] Can't accept connection\n");
    return 1;
  }

  // read payload length
  read_b = read(accepted_soc, &pload_len, sizeof(size_t));
  if(read_b == sizeof(size_t))
  {
    //printf("[HBServer] Size of incoming payload: %d\n",pload_len);
  }
  else
  {
    //printf("[HBServer] [ERROR] Invalid data\n");
    return 1;
  }

  // read payload
  while((read_b = read(accepted_soc, rdbuff+i, 1024)) < pload_len)
  {
    i+=read_b;
  }
  ploadptr= (payload_t*)rdbuff;
  /* display data */
  printf("[HBServer]  Message: %s\n",ploadptr->buf);

  // send message from server to client
  //send(accepted_soc , "ACK" , 4, 0);
  //printf("[HBServer] Message sent from Server: ACK\n");
}
  /*close socket */
  close(accepted_soc);

  return 0;
}


int startup_test()
{
        int x=1;

        if(temp_init() == -1)
                x=0;

        if(light_init() == -1)
                x=0;

        if(pthread_create(&light_id, NULL,func_light,NULL) != 0)
                x=0;
        if(pthread_create(&temp_id, NULL,func_temp,NULL) != 0)
                x=0;

        if(pthread_create(&socket_id, NULL, func_socket, NULL ) !=0)
                x=0;

        return x;
}

int main(int argc, char *argv[])
{

        memset(file_name, '\0', sizeof(file_name));
        strncpy(file_name, argv[1], strlen(argv[1]));

   time_t curtime;
   time(&curtime);
   mqd_t mq1;
   mystruct sample;
   char buffer[50] = {0};

   pthread_create(&logger_id, NULL,logger_task,NULL);
   //pthread_create(&light_id, NULL,func_light,NULL);
   //pthread_create(&temp_id, NULL,func_temp,NULL);
   //pthread_create(&socket_id, NULL, func_socket, NULL );
   //thread1 -> filename = "log.txt";

        int startup_check = startup_test();

        if(startup_check == 0)
        {
                strncpy(sample.random_string,"Startup test failed", strlen("Startup
test failed"));
   sample.source_id = 0;
   sample.log_level = 2;
   //my_stamp = ctime(&curtime);
   memcpy(buffer,ctime(&curtime),24);
   memcpy(sample.timestamp,buffer, strlen(buffer));
   mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
   mq_send(mq1,(char *)&sample,sizeof(sample),1);

                func_led_off();
                printf("\n<<<Startup Test Failed>>>\n\n");
                printf("[Main Task] Killing All Tasks\n");
                pthread_cancel(logger_id);
                pthread_cancel(temp_id);
                pthread_cancel(light_id);
                pthread_cancel(socket_id);
                mq_close(mq1);
                mq_unlink("/my_queue");
                return -1;
        }


   //char buffer1[50] = {0};
   //char buffer[50] = {0};
   //char my_stamp[25];

   //strncpy(buffer1,,27);
   strncpy(sample.random_string,"Initiated child threads", strlen("Initiated child
threads"));
   sample.source_id = 0;
   sample.log_level = 0;
   //my_stamp = ctime(&curtime);
   memcpy(buffer,ctime(&curtime),24);
   memcpy(sample.timestamp,buffer, strlen(buffer));

   mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
   if(mq1 == -1)
   {
      printf("Can't open\n"); //opening queue 1
      return -1;
    }

   if( mq_send(mq1,(char *)&sample,sizeof(sample),1)== -1)
```

```
        {
          printf("Sending failed\n");
        }
    //mq_close(mq1);
   //mq_unlink("/my_queue");






   check_status();

   pthread_join(logger_id,NULL);
   pthread_join(light_id,NULL);
   pthread_join(temp_id,NULL);
   pthread_join(socket_id,NULL);
   pthread_exit(NULL);

   printf("Main Process Terminated\n");
  return 0;
}





/*
*@Filename:light_task.h
*@Description:This is a header for library for the light sensor apds9301
*@Author:Anay Gondhalekar and Sharanjeet Singh Mago
*@Date: 03/15/2018
*@compiler:gcc
*@Usage : Connect light sensor to I2C and use any of the library function to read and
write registers
 */


#ifndef LIGHT_TASK_H_
#define LIGHT_TASK_H_

#define time_high 0x02  //for 402ms
#define time_med 0x01  //for 101ms
#define time_low 0x00  //for 13ms

#define gain 0x10  //for maximum gain

int control_reg_wr ( int fd, int msg);
/**
* @brief Write to Control register of light sensor
*
* @return Error Condition
*/
int control_reg_rd ( int fd);
/**
* @brief Read from Control register of light sensor
*
* @return Error Condition
*/
int timing_reg_wr ( int fd, int msg);
/**
* @brief Write to timing register of light sensor
*
* @return Error Condition
```

```
*/
int timing_reg_rd(int fd);
/**
* @brief Write from timing register of light sensor
*
* @return Error Condition
*/
int control_reg_int_wr(int fd, int msg);
/**
* @brief Write to Control register Interupt of light sensor
*
* @return Error Condition
*/
int control_reg_int_rd(int fd);
/**
* @brief Read from Control register Interupt of light sensor
*
* @return Error Condition
*/
int threshold_int_reg_wr(int fd, int *array);
/**
* @brief Write to Threshold register Interupt of light sensor
*
* @return Error Condition
*/
int threshold_int_reg_rd(int fd, int *array);
/**
* @brief Read from Threshold register Interupt of light sensor
*
* @return Error Condition
*/
int id_reg_rd(int fd);
/**
* @brief Read Light sensor id register
*
* @return Error Condition
*/
uint16_t data0_reg_rd(int fd);
/**
* @brief Read Light sensor configuration register
*
* @return Config register value
*/
uint16_t data1_reg_rd(int fd);
/**
* @brief Function to read lux from light sensor
*
* @return lux value
*/
float get_lux(void);

/**
* @brief Function to initialize light sensor
*
* @return Error condition if init fails
*/
int light_init(void);
/**
* @brief Function to read and write all functions
**
* @return Error condition
*/
```

```
int all_reg_rd_wr(int fd);

#endif
/**
* @file socket_task.h
* @brief Header File for Server of the socket task
*
*
* @author Sharanjeet Singh Mago
* @date 16 March 2018
*
*/


#ifndef __SOCKET_TASK_H__
#define __SOCKET_TASK_H__

#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>

#define PORT_ADR    2000

typedef struct
{
  char    buf[20];
  int  buf_len;
  bool    usrLED_OnOff;
}payload_t;

/**
* @brief Creates a server for and external client to connect and get data
*
* @return Error siganl is the socket fails
*/
int socket_task(void);

#endif /* __SOCKET_TASK_H__ *//*
*@Filename:temp_task.h
*@Description:This is a header for library for the temp sensor tmp102
*@Author:Anay Gondhalekar and Sharanjeet Singh Mago
*@Date: 03/15/2018
*@compiler:gcc
*@Usage : Connect temperature sensor to I2C and use any of the library function to
read and write registers
 */

#ifndef _TEMP_TASK_H_
#define _TEMP_TASK_H_

/**
* @brief Function to initialize temperature sensor
*
* @return Error condition if init fails
*/
int temp_init();
```

```
/**
* @brief Function to read from temperature data register
*
* @param Unit to get the data (0=Celcius,1=Kelvin,2=Fahrenheit)
*
* @return Converted data or error if conversion fails
*/
float read_temp_data_reg(int unit);

/**
* @brief Function to read and write all functions
**
* @return Error condition
*/
int all_temprg_rd_wr();

/**
* @brief Read Temperature sensor configuration register
*
* @return Config register value
*/
uint16_t read_temp_config_register();

/**
* @brief Read Tlow and Thigh register
*
*
* @param Address of register you want to read
*
* @return Value of the register
*/
uint16_t read_tlow_reg(int reg);

/**
* @brief Function to write config register
*
* @return Error condition
*/
int write_config_register_default();

/**
* @brief Function to set conversion rate from config register
*
* @param value to write
*
* @return Error condition
*/
int write_config_reg_conv_rate(uint8_t value );

/**
* @brief Function to write to config register
*
* @param value to write to config register
*
* @return Error condition
*/
int write_config_reg_em(uint8_t value );

/**
* @brief Function to write On or Off to config register
*
```

```
* @param value to write to config register (0 0r 1)

* @return Error condition
*/
int write_config_reg_on_off(uint8_t value );

/**
* @brief Function to write Tlow or Thigh register
*
* @param reg Address of Thigh or Tlow register
* @param calue Value to write to register
*
* @return Error condition
*/
int write_tlow_reg(int reg, uint16_t value );

/**
* @brief Function to write pointer register
*
* @param Value to write to pointer register
*
* @return Error condition
*/
void write_pointer_reg(uint8_t value);

#endif
/**
* @file main_task.h
* @brief Prototype functions to drive the Projct 1
*
* This header file provides the prototypes of the functions
* to drive the multithreading project for interfacing two sensors
*
* @author Sharanjeet Singh Mago
* @date 17 March 2018
*
*/

#ifndef __MAIN_TASK_H__
#define __MAIN_TASK_H__
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <math.h>
#include <float.h>
#include <complex.h>
#include <stdint.h>
#include <time.h>
#include <mqueue.h>
#include "light_task.h"
//#include "light_task.c"
#include "temp_task.h"
//#include "temp_task.c"
#include "socket_task.h"
```

```
#define HB_PORT_ADR 5000
#define IP_ADR      "127.0.0.1"

pthread_t logger_id, light_id, temp_id, socket_id;

char file_name[50];

typedef struct              //structure to be sent
{
char timestamp[50];
int source_id;
int log_level;
int data;
float value;
char random_string[50];
}mystruct;

struct threadParam
{
char *filename;
};

/**
* @brief Client to send HB data for light sensor
*
*
* @return Error condition
*/
int light_client(void);

/**
* @brief Thread Function for Light Task
*
*/
void *func_light(void);

/**
* @brief Client to send HB data for temperature sensor
*
*
* @return Error condition
*/
int temp_client(void);

/**
* @brief Thread Function for Temperature Task
*
*/
void *func_temp(void);

/**
* @brief Thread Function for Logger Task
*
*/
void* logger_task(void);

/**
* @brief Thread Function for Socket Task
*
*/
void* func_socket(void);
```

```
/**
* @brief Server to recieve data from tasks
*
* @return Error condition if status fails
*/
int check_status(void);

/**
* @brief Function to test all the startup tests
*
* @return Error condition if tests fails
*/
int startup_test(void);

#endif
```