(1条消息)嵌入式linux下用C实现MQTT数据（JSON编码）发布 - xillinx的专栏 - CSDN博客

嵌入式Linux硬件很多，在网上可以买到很多款，我采用了一款带4G和SDK开发环境的HJ8300硬件，采用MIPS处理，580Mhz的主频，128M内存，作为MQTT的开发已经足够。

HJ8300已经集成了GCC、GDB和LIB等编译调试工具，用SSH登录到设备就可以编译程序和调试程序。

里面主要文件是main.c和base.h，包含linux项目Makefile文件，打开Makefile，

```
    INCLUDES= base.h
CFLAGS  = -Wall -O2 -I/mmz/mipsel-mt76xx-linux-gnu/include -L/mmz/mipsel-mt76xx-linux-gnu/lib
CC=gcc
all : mqttdemo
%.o : %.c $(INCLUDES)
 $(CC) $(CFLAGS) -c $<
mqttdemo : main.o
 $(RM) mqttdemo
 $(CC) -o $@ $(CFLAGS) -L. -lpthread -lrt -ljson-c -lmosquitto -lssl -lcrypto -lcares $<
clean:
 $(RM) *.o *~ mqttdemo
```

可以看出，编译器安装的路径在/mmz/mipsel-mt76xx-linux-gnu，源文件是main.c，编译后生成mqttdemo可执行文件

main.c的文件较长，下面介绍主要流程。

```
int main(int argc, char *argv[])
{   uint08t etha[0x8];
    int status;

    blue_system(0,"ulimit -c 1024");
    blue_system(0,"ifconfig eth0 up");
    blue_read_net_interface("eth0",etha);


    sprintf(blue_etha_string,"%02X-%02X-%02X-%02X-%02X-%02X",etha[0],etha[1],etha[2],etha[3],etha[4],etha[5]);
    memset(&blue_modbus_block,0x0,sizeof(blue_thread_block_t));
    memset(&blue_mqtt_block,0x0,sizeof(blue_thread_block_t));
    blue_mqtt_init_from_file("./mqttconf.txt");
    blue_timer_initilaize();
    status=blue_mqtt_thread_start(&blue_mqtt_block);
    if( status<0)
    {   blue_printf("start MQTT service Failed\r\n");
        return -1;
    }
    sem_init(&system_close_semphore,0,0);       /* 关闭信号量 */
    blue_sem_wait(&system_close_semphore);      /* 等待关闭 */
    blue_mqtt_thread_stop(&blue_mqtt_block);
    return 0x1;
}
```

从main函数可以看出，系统先读取HJ8300的MAC地址和MQTT的配置参数，启动了一个mqtt线程来处理MQTT的事物，main程序就等
待关闭信号了。

分析一下mqtt线程

```
void * blue_mqtt_thread(void * parameter)
{   blue_thread_block_t * block=(blue_thread_block_t*)parameter;
    char mqtt_dmain[NAME_TXT_MAX];
    char mqtt_topic[NAME_TXT_MAX];
    int status;
    int mid=0;
```

```
sprintf(mqtt_dmain,"%d.%d.
%d.%d",mqtt_server_addr[0],mqtt_server_addr[1],mqtt_server_addr[2],mqtt_server_addr[3]);
mosquitto_lib_init();
while(block->runs==1)
{   block->mqtt=mosquitto_new("COM",TRUE,NULL);
    if( block->mqtt==NULL)
    {   break;
    }
    mosquitto_connect_callback_set(block->mqtt,     blue_mqtt_connect_callback);
    mosquitto_disconnect_callback_set(block->mqtt,  blue_mqtt_disconnect_callback);
    mosquitto_publish_callback_set(block->mqtt,     blue_mqtt_publish_callback);
    mosquitto_message_callback_set(block->mqtt,     blue_mqtt_message_callback);
    status=mosquitto_connect(block->mqtt,mqtt_dmain,mqtt_server_port,600);
    if( status)
    {   mosquitto_destroy(block->mqtt);
        block->mqtt=NULL;
        blue_printf("blue MQTT connect <%s>-<%d> failed\r\n",mqtt_dmain,mqtt_server_port);
        sleep(2);
          continue;
    }
    snprintf(mqtt_topic,NAME_TXT_MAX,"/mqtt-demo/%s/%s/%s
/0",mqtt_user_name,mqtt_user_pass,blue_etha_string);
       status=mosquitto_subscribe(block->mqtt,&mid,mqtt_topic,0);
    if( status!=0)
       {   mosquitto_destroy(block->mqtt);
        block->mqtt=NULL;
        blue_printf("blue MQTT subscribe failed\r\n");
        sleep(2);
          continue;
       }
    status=blue_modbus_thread_start(&blue_modbus_block);
    if( status<0)
       {   mosquitto_destroy(block->mqtt);
          block->mqtt=NULL;
          blue_printf("blue MQTT start CLX failed\r\n");
        sleep(5);
          continue;
       }
       while(block->runs==1)
       {   status=mosquitto_loop(block->mqtt,1,10);
          if( status==0)
          {   ;
          }
          else
          {   blue_printf("blue MQTT %d restart\r\n",status);
             break;
          }
       }
       blue_modbus_thread_stop(&blue_modbus_block);
       mosquitto_destroy(block->mqtt);
       block->mqtt=NULL;
   }
   if( block->mqtt)
{   mosquitto_destroy(block->mqtt);
}
mosquitto_lib_cleanup();
block->mqtt=NULL;
block->runs=-1;
   return NULL;
}
```

调用mosquitto_lib_init系统函数直接初始化库，通过mosquitto_connect_callback_set设置回调函数，mosquitto_connect调用main函数获取的配置参数连接到MQTT的服务器，mosquitto_subscribe函数发布MQTT主题，到处，MQTT的任务处理完成。

mqtt线程接着调用blue_modbus_thread_start启动了一个MODBUS线程，完成从RS485端口读取数据，把

读取的数据通过MQTT发布出去。现在分析一下MODBUS线程：

```
void * blue_modbus_thread(blue_thread_block_t * block)
{   struct timeval timeout;
    fd_set readset;
    int status;
    status=blue_uart_connect(block);
    if( status<0)
    {   block->runs=-1;
        return NULL;
    }
    block->ua_status=UART_STAT_IDLE;
    while(block->runs==0x1)
    {   timeout.tv_sec =1;
        timeout.tv_usec=0;
        FD_ZERO(&readset);
        FD_SET(block->sock,&readset);
        status=select(block->sock+1,&readset,NULL,NULL,&timeout);
        if( status<0)
        {    blue_printf("Modbus select failed\r\n");
            continue;
        }
        if( FD_ISSET(block->sock,&readset))
        {   status=blue_modbus_recv_uart_data(block);
            if( status<0x0)
            {   blue_printf("Modbus recv failed\r\n");
            }
            else
            {   blue_printf("Modbus recv OK\r\n");
            }
        }
        blue_modbus_thread_polling(block);
    }
    blue_uart_close(block);
    block->runs=-1;
    return NULL;
}
```

函数blue_uart_connect连接到RS485串口，函数blue_modbus_recv_uart_data从串口接收数据并发布数据，现在分析这个函数：

```
int blue_modbus_recv_uart_data(blue_thread_block_t * block)
{   modbus_command_t * command;
    char buffer[NAME_TXT_MAX];
    uint16t crcchk;
    uint16t crcorg;
    int status;
    int modlen;
    command=block->curcmd;
    if( command==NULL)
    {   recv(block->sock,buffer,BUFF_LEN_MAX,MSG_NOSIGNAL);
        blue_printf("Modbus command null\r\n");
        return -1;
    }
    if( command->rsp_len>=BUFF_LEN_MAX)
    {   command->rsp_len=0;
    }
    status=recv(block->sock,command->rsp+command->rsp_len,BUFF_LEN_MAX-
command->rsp_len,MSG_NOSIGNAL);
    if( status<=0x0)
    {   blue_printf("Modbus command recv null\r\n");
        return -1;
    }
    command->rsp_len+=status;
    if( command->rsp_len<=3)
    {   blue_printf("Modbus command recv len=%d wait\r\n",command->rsp_len);
        return 0x1;
```

```
    }
    modlen =command->rsp[2];
    modlen+=5;              /* add the address/command/length/+...+/CRC = 5 bytes */
    if( command->rsp_len<modlen)
    {   blue_printf("Modbus command recv len=%d need=%d wait\r\n",command->rsp_len,modlen);
        return 0x1;
    }
    if( command->rsp[1]==(command->cmd[1]|0x80))
    {   blue_printf("Modbus command recv response <%d> failed\r\n",command->rsp[1]);
        return 0x1;
    }
    crcchk  =blue_crc16(command->rsp,modlen-0x2);
    crcorg  =command->rsp[modlen-0x2];
    crcorg<<=0x8;
    crcorg +=command->rsp[modlen-0x1];
    if( crcorg!=crcchk)
    {   blue_printf("Modbus command recv CRC <%04X--%04X> failed\r\n",crcchk,crcorg);
        return 0x1;
    }
    blue_mqtt_publish_modbus_data(block,command);
    block->ua_status=UART_STAT_IDLE;
    block->curcmd=NULL;
    blue_printf("Modbus data publish OK\r\n");
    return 0x1;
}
```

这个函数就是从串口的socket里面读取数据，对MODBUS数据进行校验（CRC16），如果数据正确，调用
blue_mqtt_publish_modbus_data这个函数发布MQTT数据。

下面分析这个函数：

```
int blue_mqtt_publish_modbus_data(blue_thread_block_t * modbus, modbus_command_t * command)
{   blue_thread_block_t * mqtt=&blue_mqtt_block;
    char topic[NAME_TXT_MAX];
    char stmr[NAME_TXT_MAX];
    char msg[MSG_TXT_MAX];
    time_t ptm=time(NULL);
    struct tm rtc;
    json_object * head=NULL;
    json_object * body=NULL;
    json_object * json=NULL;
    json_object * jarr=NULL;
    int status;
    int len=0;
    int i;
    if( localtime_r(&ptm,&rtc)==NULL)
    {  return -1;
    }
    else
    {   snprintf(stmr,NAME_TXT_MAX,"%4d-%02d-%02dT
%d:%02d:%02d.000+0800",rtc.tm_year+1900,rtc.tm_mon+0x1,rtc.tm_mday,rtc.tm_hour,rtc.tm_min,rtc.tm_sec);/*yyyy-
MM-dd'T'HH:mm:ss.SSS[+-]HH:ss*/
    }
    status=snprintf(topic,NAME_TXT_MAX,"/modbus/%s/%s
/%d",(char*)mqtt_user_name,blue_etha_string,command->cmd[3]);
    if( status<0)
    {   return -1;
    }
    modbus->sequnce++;
    head=json_object_new_object();
    json_object_object_add(head,"Type",         json_object_new_int(5));              /* message type */
    json_object_object_add(head,"Sequnce",      json_object_new_int(modbus->sequnce));  /* message sequnce */

        body=json_object_new_object();

    json_object_object_add(body,"Reference",    json_object_new_int(5));
    json_object_object_add(body,"SamplingTime", json_object_new_string(stmr));
```

```
    json_object_object_add(body,"Description",  json_object_new_string("Modbus data"));

        len=(int)command->rsp[2];
    json_object_object_add(body,"Bytes",        json_object_new_int(len));
    jarr=json_object_new_array();
    for(i=0;i<len;i++)
    {   json_object_array_add(jarr,json_object_new_int((int)command->rsp[3+i]));
    }
    json_object_object_add(body,"Data",jarr);
    json=json_object_new_object();
    json_object_object_add(json,"h",head);
    json_object_object_add(json,"b",body);
    len=snprintf(msg,MSG_TXT_MAX,"J%s\n",json_object_to_json_string_ext(json,JSON_C_TO_STRING_PRETTY));
    json_object_put(head);
    json_object_put(body);
    json_object_put(json);
    blue_printf("MQTT toptic:%s\r\n%s\r\n",topic,msg);

        if( mqtt->mqtt==NULL)
    {   blue_printf("MQTT publish null failed\r\n");
        return -1;
    }
    status=mosquitto_publish(mqtt->mqtt,NULL,topic,strlen(msg),msg,0,0);
    if( status!=MOSQ_ERR_SUCCESS)
    {   blue_printf("MQTT publish failed\r\n");
        return -1;
    }
    else
    {   blue_printf("MQTT publish OK\r\n");
        return 0x1;
    }
}
```

从函数可以看出，采用JSON对MODBUS数据编码，调用mosquitto_publish发布数据，流程还是很清晰的。

通过HJ8300编译这个程序后，可以直接运行和调试。

通过DEMO程序这样可以节约大量的时间处理流程，把重点放到数据处理。