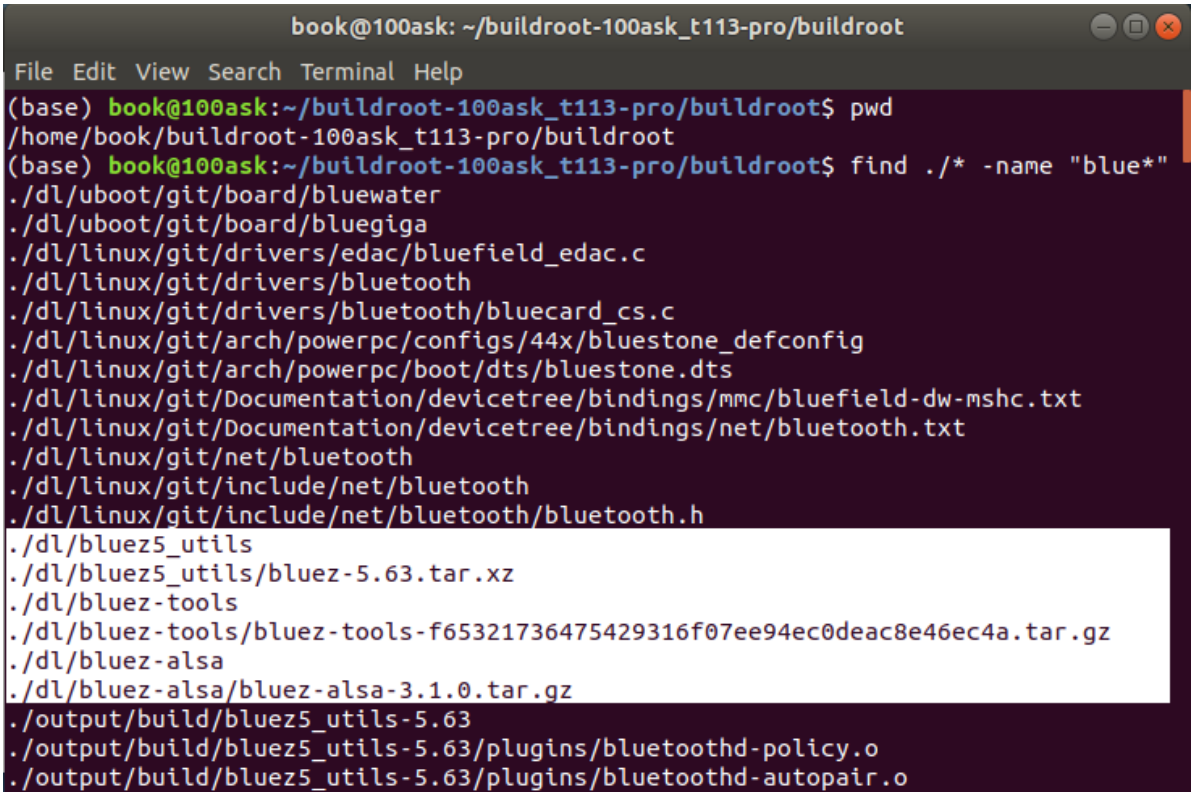
Linux蓝牙通信(基于BlueZ的C语言BLE蓝牙编程)

参考自https://blog.csdn.net/qq_46079439/article/details/126252232?spm=1001.2014.3001.5502

1.获取BlueZ源码并编译

在buildroot目录下通过find 命令查找bluez的包在哪里

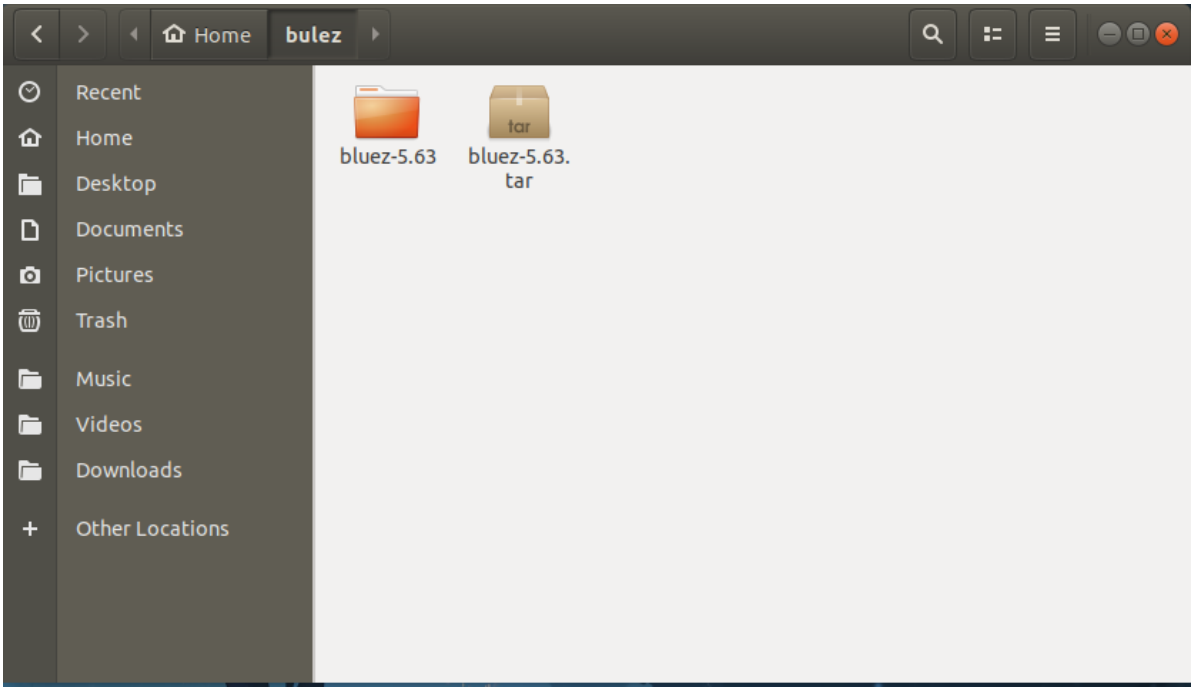
```
cd /buildroot-100ask_t113-pro/buildroot
find ./* -name "blue*"
```



`./dl/bluez5_utils/bluez-5.63.tar.xz` 就是我们想要的软件包，将它复制到某个目录下，然后解压

我这里是在/home/book目录下新建了一个bluez目录，把它复制到其中，并解压

```
cd /home/book
mkdir bluez
cp -rf ./dl/bluez5_utils/bluez-5.63.tar.xz /home/book/bluez
cd /home/book/bluez
xz -d bluez-5.63.tar.xz
tar -xvf bluez-5.63.tar
```

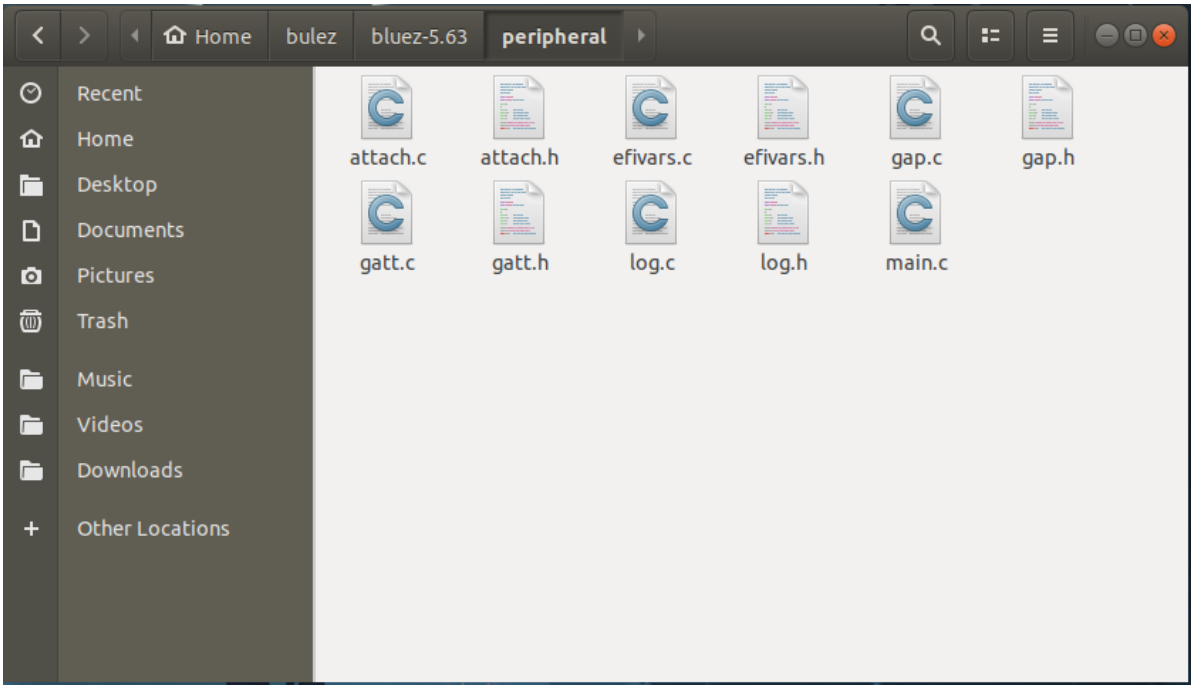


进入到*bluez-5.63*目录

```
cd bluez-5.63
```

使用的是以下这个目录 `peripheral`

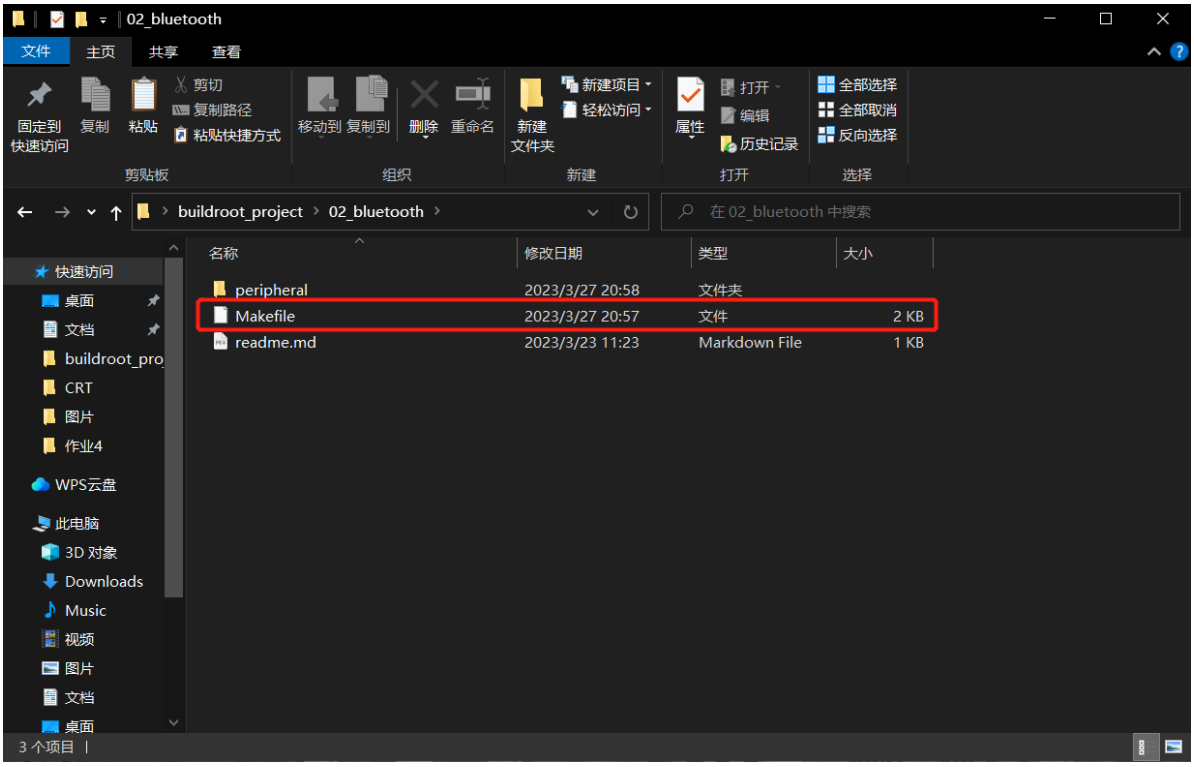
```
cd peripheral
```



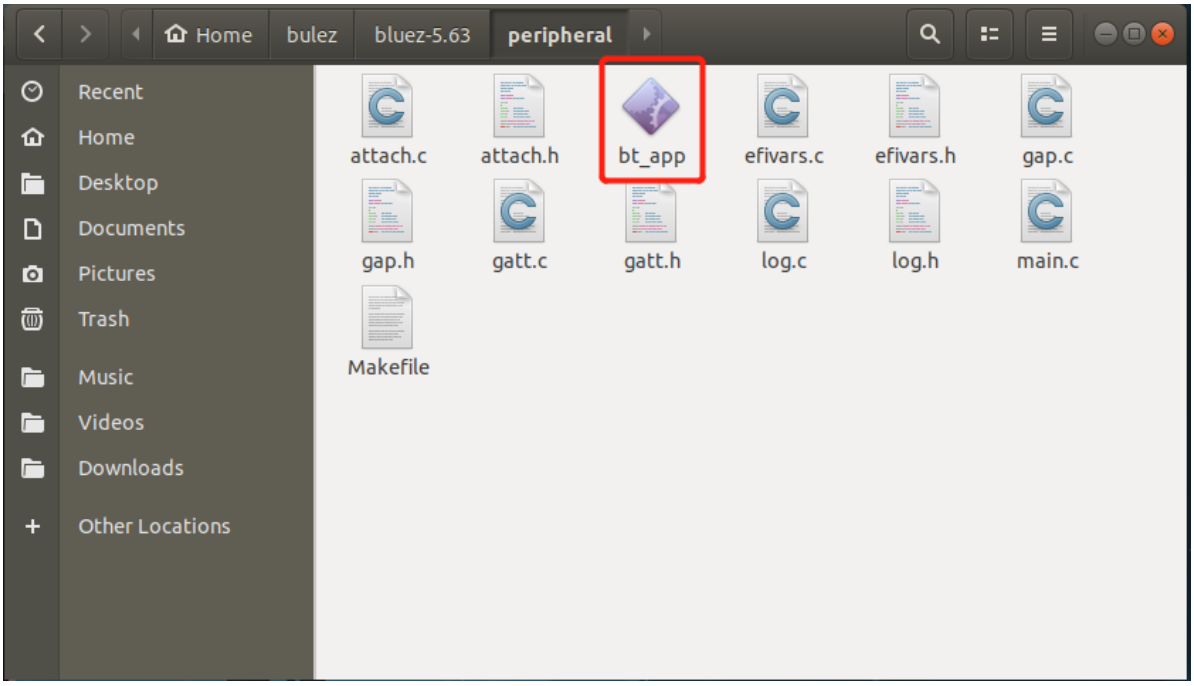
可以把它当作一个装应用程序的文件夹，虽然可以直接编译，但还是写一个makefile文件比较好

```
sudo gedit Makefile
```

Makefile内容：



这里就不贴了，在make时会报一个错，代码改好了也放在同级目录下了，编译完出来一个可执行文件如下：



2.可执行文件测试

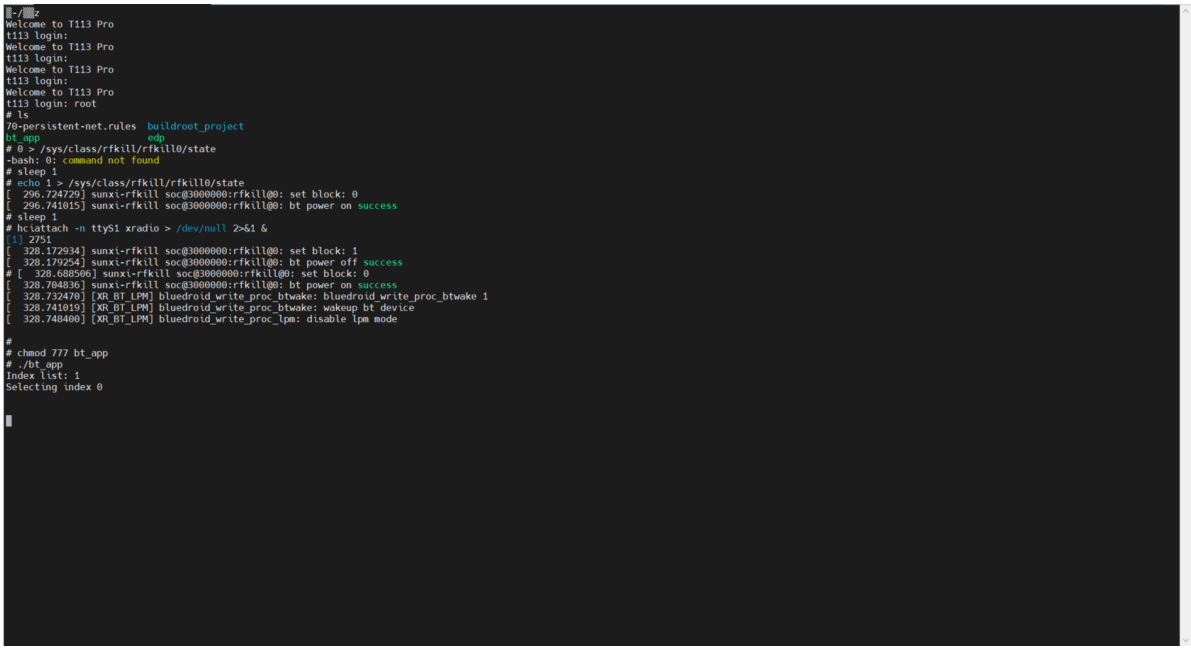
2.1.开发板操作

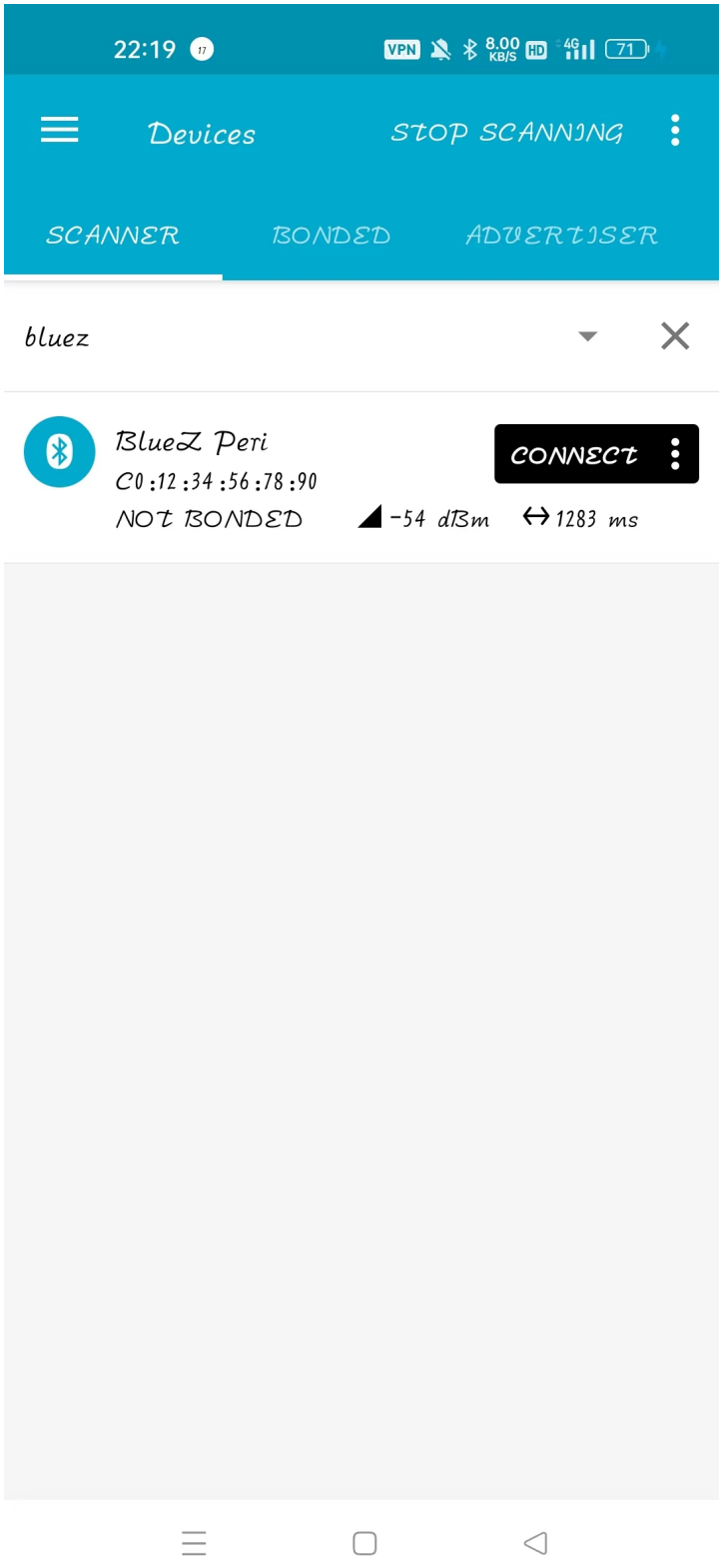
```
echo 0 > /sys/class/rfkill/rfkill0/state
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state
sleep 1

#绑定设备
hciattach -n ttyS1 xradio > /dev/null 2>&1 &
```

1. `0 > /sys/class/rfkill/rfkill0/state`：将数字0重定向到文件`/sys/class/rfkill/rfkill0/state`，表示关闭rfkill设备号为0的射频开关。
2. `echo 1 > /sys/class/rfkill/rfkill0/state`：将数字1写入文件`/sys/class/rfkill/rfkill0/state`，表示打开rfkill设备号为0的射频开关。
3. `hciattach -n ttyS1 xradio > /dev/null 2>&1 &`：绑定设备，将HCI设备连接到ttyS1端口，使用xradio驱动程序，同时将输出重定向到/dev/null（丢弃输出），并在后台运行。
4. `sleep 1`：让程序暂停1秒钟。

2.2.将可执行文件上传至开发板并运行

```
cd /home/book/bulez/bluez-5.63/peripheral
adb push ./bt_app /root
chmod 777 bt_app
./bt_app
```

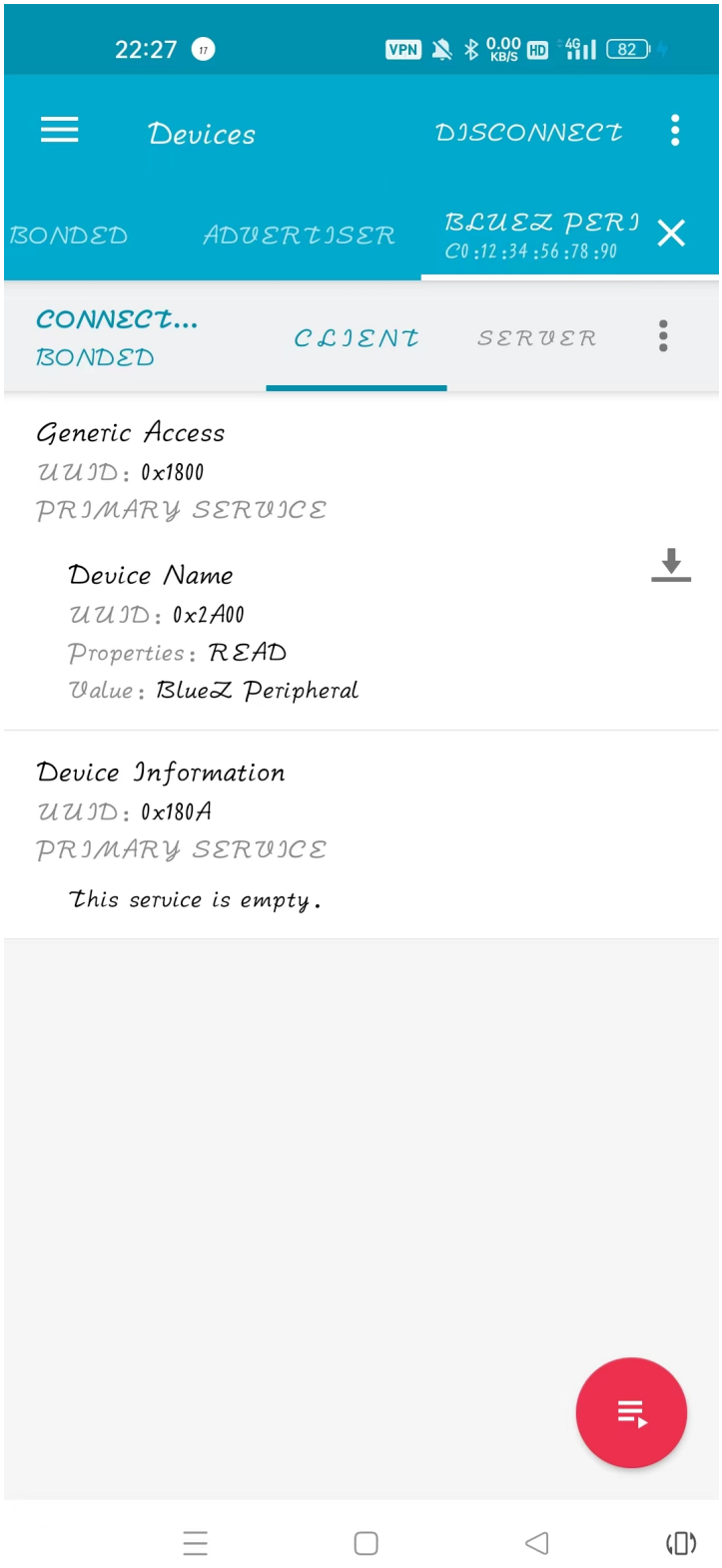




手机用蓝牙助手搜索bluez设备，可以看到这么一个蓝牙设备，点击connect连接

```
~/[redacted]
Welcome to T113 Pro
t113 login:
Welcome to T113 Pro
t113 login:
Welcome to T113 Pro
t113 login:
Welcome to T113 Pro
t113 login: root
# ls
70-persistent-net.rules  buildroot_project
bt_app                   edp
# 0 > /sys/class/rfkill/rfkill0/state
-bash: 0: command not found
# sleep 1
# echo 1 > /sys/class/rfkill/rfkill0/state
[  296.724729] sunxi-rfkill soc@3000000:rfkill@0: set block: 0
[  296.741015] sunxi-rfkill soc@3000000:rfkill@0: bt power on success
# sleep 1
# hciattach -n ttyS1 xradio > /dev/null 2>&1 &
[1] 2751
[  328.172934] sunxi-rfkill soc@3000000:rfkill@0: set block: 1
[  328.179254] sunxi-rfkill soc@3000000:rfkill@0: bt power off success
# [  328.688506] sunxi-rfkill soc@3000000:rfkill@0: set block: 0
[  328.704836] sunxi-rfkill soc@3000000:rfkill@0: bt power on success
[  328.732470] [XR_BT_LPM] bluedroid_write_proc_btwake: bluedroid_write_proc_btwake 1
[  328.741019] [XR_BT_LPM] bluedroid_write_proc_btwake: wakeup bt device
[  328.748400] [XR_BT_LPM] bluedroid_write_proc_lpm: disable lpm mode

#
# chmod 777 bt_app
# ./bt_app
Index list: 1
Selecting index 0


Device connected
New device connected
GATT client discovery complete
New identify resolving key
New connection signature resolving key
New connection signature resolving key
New long term key
New long term key
```

连接后T113开发板打印信息如上



可以看到两个BLE服务，第一个服务是接收数据的服务，数据为：BlueZ Peripheral，第二个服务为空。

3.源码分析

3.1.BLE名称更改

gap.c (D:\桌面\...\peripheral) | main.c (D:\桌面\...\peripheral)

gap.c

Symbol Name (Alt+L)

- new_long_term_key_event
- device_connected_event
- device_disconnected_event
- user_confirm_request_event
- user_passkey_request_event
- auth_failed_event
- device_unpaired_event
- passkey_notify_event
- new_irk_event
- new_csrk_event
- new_conn_param_event
- advertising_added_event
- advertising_removed_event
- read_adv_features_complete
- **read_info_complete**
- read_index_list_complete
- index_added_event
- index_removed_event
- read_ext_index_list_complete
- ext_index_added_event
- ext_index_removed_event
- **read_commands_complete**
- gap_start
- gap_stop

```
268:                    user_confirm_request_event, NULL, NULL);
269:    mgmt_register(mgmt, MGMT_EV_USER_PASSKEY_REQUEST, index,
270:                    user_passkey_request_event, NULL, NULL);
271:    mgmt_register(mgmt, MGMT_EV_AUTH_FAILED, index,
272:                    auth_failed_event, NULL, NULL);
273:    mgmt_register(mgmt, MGMT_EV_DEVICE_UNPAIRED, index,
274:                    device_unpaired_event, NULL, NULL);
275:    mgmt_register(mgmt, MGMT_EV_PASSKEY_NOTIFY, index,
276:                    passkey_notify_event, NULL, NULL);
277:    mgmt_register(mgmt, MGMT_EV_NEW_IRK, index,
278:                    new_irk_event, NULL, NULL);
279:    mgmt_register(mgmt, MGMT_EV_NEW_CSRK, index,
280:                    new_csrk_event, NULL, NULL);
281:    mgmt_register(mgmt, MGMT_EV_NEW_CONN_PARAM, index,
282:                    new_conn_param_event, NULL, NULL);
283:    mgmt_register(mgmt, MGMT_EV_ADVERTISING_ADDED, index,
284:                    advertising_added_event, NULL, NULL);
285:    mgmt_register(mgmt, MGMT_EV_ADVERTISING_REMOVED, index,
286:                    advertising_removed_event, NULL, NULL);
287:
288:    dev_name_len = snprintf((char *) dev_name, 26, "BlueZ Peripheral");
289:
290:    if (current_settings & MGMT_SETTING_POWERED) {
291:        val = 0x00;
292:        mgmt_send(mgmt, MGMT_OP_SET_POWERED, index, 1, &val,
293:                        NULL, NULL, NULL);
294:    }
295:
296:    if (!(current_settings & MGMT_SETTING_LE)) {
297:        val = 0x01;
298:        mgmt_send(mgmt, MGMT_OP_SET_LE, index, 1, &val,
299:                        NULL, NULL, NULL);
300:    }
301:
302:    if (current_settings & MGMT_SETTING_BREDR) {
303:        val = 0x00;
```

在gap.c的第288行可以更改这行代码来更改连接的BLE名称.

3.2.BLE服务创建

看到gatt.c这个.c文件内，在210行，这里的bt_uuid16_create创建一个蓝牙服务，不太理解，后面再看看

gap.c (D:\桌面\...\peripheral) | gatt.c (D:\桌面\...\peripheral) | main.c (D:\桌面\...\peripheral)

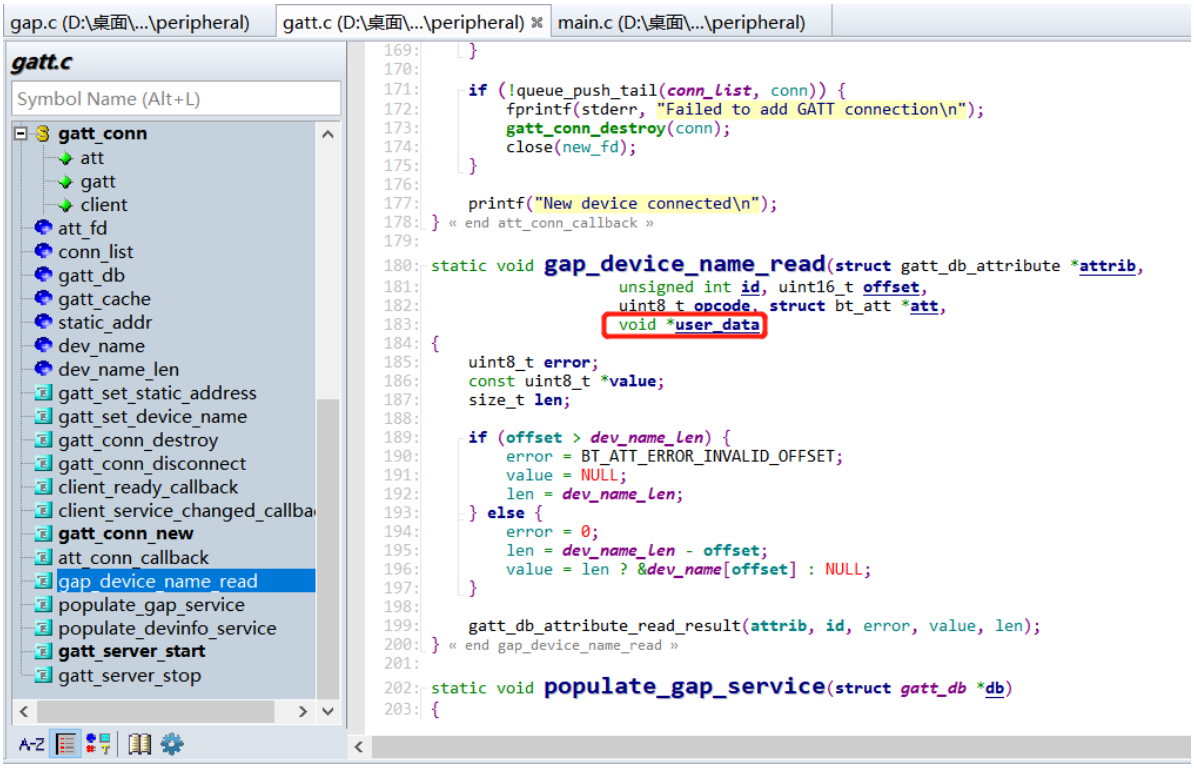
gatt.c

Symbol Name (Alt+L)

- **gatt_conn**
  - att
  - gatt
  - client
- att_fd
- conn_list
- gatt_db
- gatt_cache
- static_addr
- dev_name
- dev_name_len
- gatt_set_static_address
- gatt_set_device_name
- gatt_conn_destroy
- gatt_conn_disconnect
- client_ready_callback
- client_service_changed_callba
- **gatt_conn_new**
- att_conn_callback
- gap_device_name_read
- populate_gap_service
- populate_devinfo_service
- **gatt_server_start**
- gatt_server_stop

```
193:    } else {
194:        error = 0;
195:        len = dev_name_len - offset;
196:        value = len ? &dev_name[offset] : NULL;
197:    }
198:
199:    gatt_db_attribute_read_result(attrib, id, error, value, len);
200: } « end gap_device_name_read »
201:
202: static void populate_gap_service(struct gatt_db *db)
203: {
204:     struct gatt_db_attribute *service;
205:     bt_uuid_t uuid;
206:
207:     bt_uuid16_create(&uuid, UUID_GAP);
208:     service = gatt_db_add_service(db, &uuid, true, 6);
209:
210:     bt_uuid16_create(&uuid, GATT_CHARAC_DEVICE_NAME);
211:     gatt_db_service_add_characteristic(service, &uuid,
212:                     BT_ATT_PERM_READ,
213:                     BT_GATT_CHRC_PROP_READ,
214:                     gap_device_name_read, NULL, NULL);
215:
216:     gatt_db_service_set_active(service, true);
217: }
218:
219: static void populate_devinfo_service(struct gatt_db *db)
220: {
221:     struct gatt_db_attribute *service;
222:     bt_uuid_t uuid;
223:
224:     bt_uuid16_create(&uuid, 0x180a);
225:     service = gatt_db_add_service(db, &uuid, true, 17);
226:
227:     gatt_db_service_set_active(service, true);
```

其中gap_device_name_read是类似于多线程的那个函数的read_fun

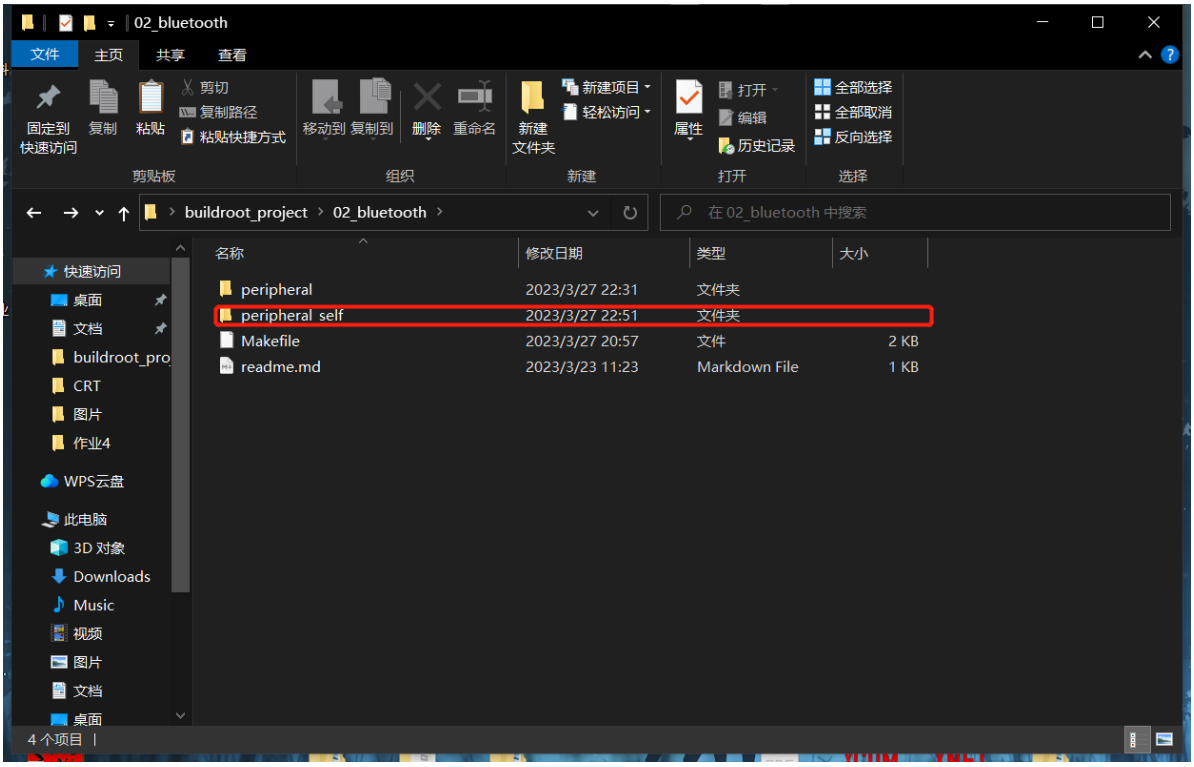


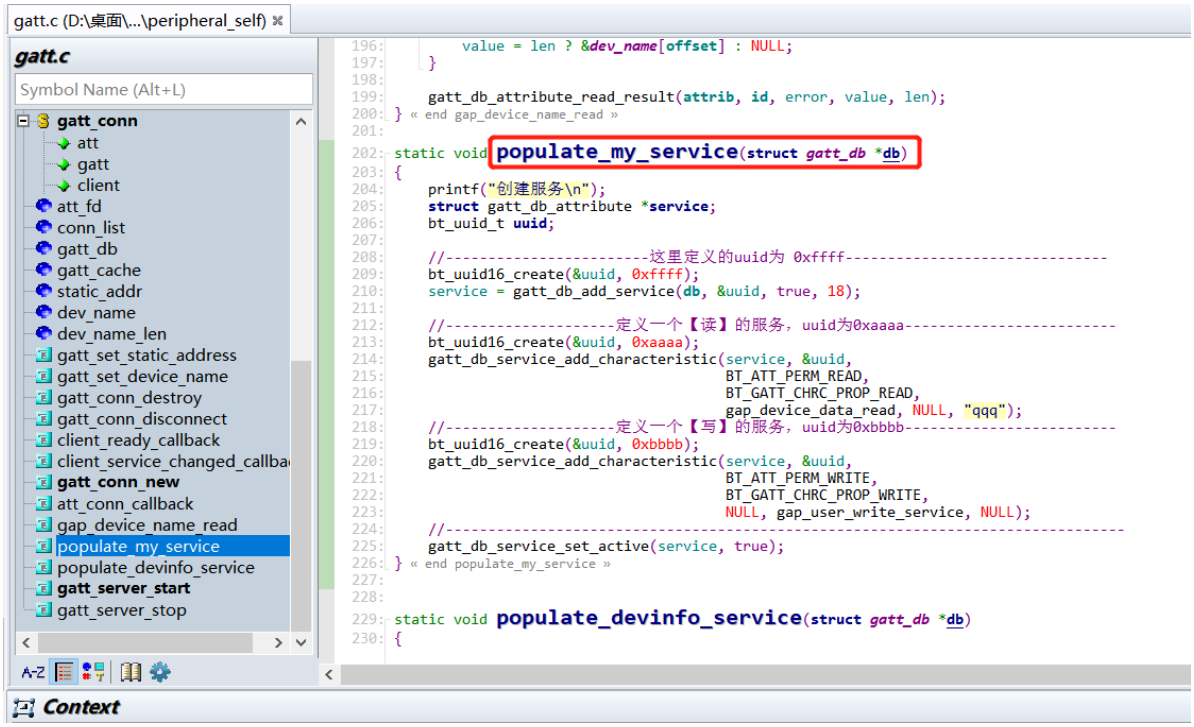
gatt_db_service_add_characteristic 的原型如下：

```
gatt_db_service_add_characteristic(struct gatt_db_attribute *attrib, --> service
                const bt_uuid_t *uuid,                                  -->
&uuid
                uint32_t permissions,                                   -->
BT_ATT_PERM_READ
                uint8_t properties,                                     -->
BT_GATT_CHRC_PROP_READ
                gatt_db_read_t read_func,                               --> 读取，
read_func
                gatt_db_write_t write_func,                             --> 写入，
write_func
                void *user_data)                                        --> 用户数
据？
```

4.源码编写

备份一下原来的代码，直接在原本的代码上编写自己的代码即可





用以上代码代替原本的populate_gap_service(struct gatt_db *db)程序

gatt.c (D:\桌面\...\peripheral_self)

gatt.c

Symbol Name (Alt+L)

- gatt_conn
  - att
  - gatt
  - client
- att_fd
- conn_list
- gatt_db
- gatt_cache
- static_addr
- dev_name
- dev_name_len
- gatt_set_static_address
- gatt_set_device_name
- gatt_conn_destroy
- gatt_conn_disconnect
- client_ready_callback
- client_service_changed_callba
- gatt_conn_new
- att_conn_callback
- gap_device_data_read
- populate_my_service
- populate_devinfo_service
- gatt_server_start
- gatt_server_stop

```
172:         fprintf(stderr, "Failed to add GATT connection\n");
173:         gatt_conn_destroy(conn);
174:         close(new_fd);
175:     }
176:
177:     printf("New device connected\n");
178: } « end att_conn_callback »
179:
180: static void gap_device_data_read(struct gatt_db_attribute *attrib,
181:                                  unsigned int id, uint16_t offset,
182:                                  uint8_t opcode, struct bt_att *att,
183:                                  void *user_data)
184: {
185:     printf("执行读取函数 gap_device_data_read\n");
186:     uint8_t error = 0;
187:     const uint8_t *value;
188:     if (0 == check_internet())
189:     {
190:         printf("CONNECTED\n");
191:         value = "connect";
192:     }
193:     else
194:     {
195:         printf("DISCONNECTED\n");
196:         value = "disconnect";
197:     }
198:
199:     size_t len = strlen(value);
200:     printf("len = %d\n", len);
201:
202:     gatt_db_attribute_read_result(attrib, id, error, value, len);
203: } « end gap_device_data_read »
204:
205:
206: static void populate_my_service(struct gatt_db *db)
```

Context

代替原本的gap_device_name_read程序

gatt.c (D:\桌面\...\peripheral_self)

gatt.c

Symbol Name (Alt+L)

- gatt_conn
  - att
  - gatt
  - client
- att_fd
- conn_list
- gatt_db
- gatt_cache
- static_addr
- dev_name
- dev_name_len
- gatt_set_static_address
- gatt_set_device_name
- gatt_conn_destroy
- gatt_conn_disconnect
- client_ready_callback
- client_service_changed_callba
- gatt_conn_new
- att_conn_callback
- gap_device_data_read
- gap_user_write_service
- populate_my_service
- populate_devinfo_service
- gatt_server_start

```
196:         value = "disconnect";
197:     }
198:
199:     size_t len = strlen(value);
200:     printf("len = %d\n", len);
201:
202:     gatt_db_attribute_read_result(attrib, id, error, value, len);
203: } « end gap_device_data_read »
204:
205: static void gap_user_write_service(struct gatt_db_attribute *attrib,
206:                                    unsigned int id, uint16_t offset,
207:                                    const uint8_t *value, size_t len,
208:                                    uint8_t opcode, struct bt_att *att,
209:                                    void *user_data)
210: {
211:     if (len != 0)
212:     {
213:         printf("收到长度: %d\n", len);
214:         printf("收到内容:%s\n", value);
215:
216:         // 这里对收到的 value 进行处理
217:
218:         gatt_db_attribute_write_result(attrib, id, 0);
219:         return;
220:     }
221:     else
222:         printf("收到null\n");
223:
224:     gatt_db_attribute_write_result(attrib, id, 0);
225: } « end gap_user_write_service »
226:
227: static void populate_my_service(struct gatt_db *db)
228: {
229:     printf("创建服务\n");
230:     struct gatt_db_attribute *service;
```

Context

增加一个gap_user_write_service程序

gatt.c (D:\桌面\...\peripheral_self)

```
287:         close(att_fd);
288:         att_fd = -1;
289:         return;
290:     }
291:
292:     if (listen(att_fd, 1) < 0) {
293:         fprintf(stderr, "Failed to listen on ATT server socket: %m\n");
294:         close(att_fd);
295:         att_fd = -1;
296:         return;
297:     }
298:
299:     gatt_db = gatt_db_new();
300:     if (!gatt_db) {
301:         close(att_fd);
302:         att_fd = -1;
303:         return;
304:     }
305:
306:     memset(rec_data, 0, 1024);
307:
308:     //populate_gap_service(gatt_db);
309:
310:     populate_devinfo_service(gatt_db);
311:
312:     populate_my_service(gatt_db);
313:
314:     gatt_cache = gatt_db_new();
315:
316:     conn_list = queue_new();
317:     if (!conn_list) {
318:         gatt_db_unref(gatt_db);
319:         gatt_db = NULL;
320:         close(att_fd);
321:         att_fd = -1;
322:         return;
```

populate_devinfo_service Function in gatt.c (D:\桌面\...\peripheral_self) at line 254 (10 lines)

在void gatt_server_start(void)内添加如上内容

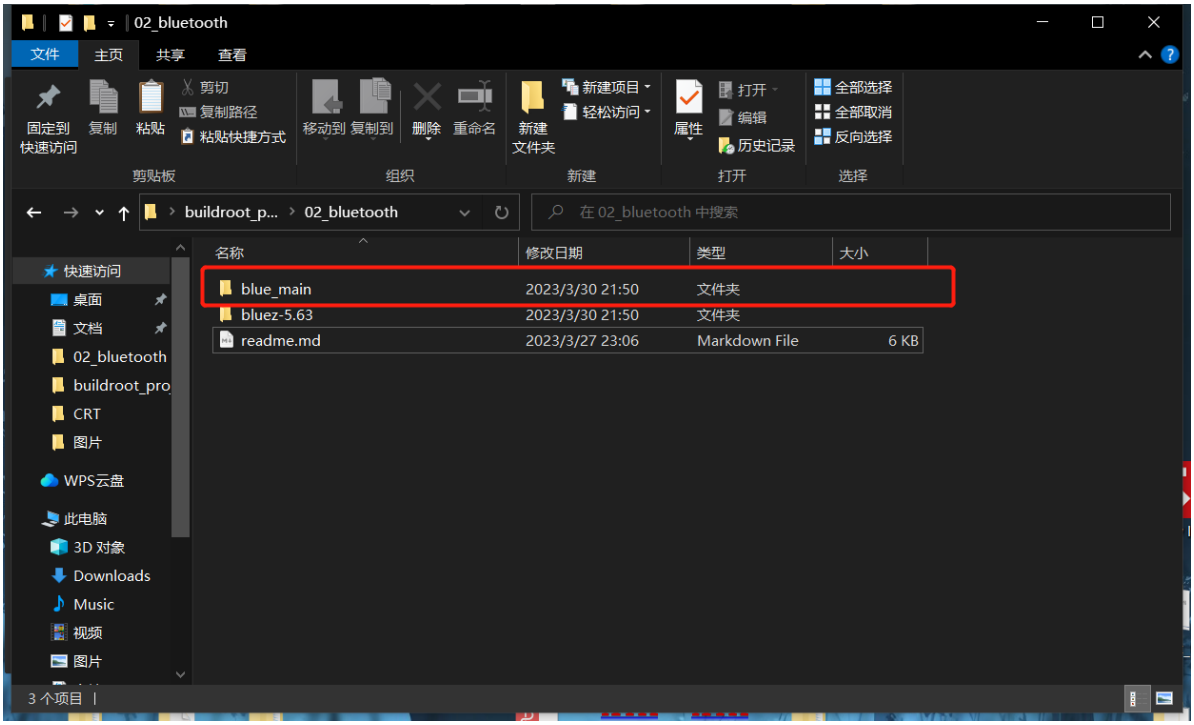
gatt.c (D:\桌面\...\peripheral_self)

```
95: }
96:
97: static struct gatt_conn *gatt_conn_new(int fd)
98: {
99:     struct gatt_conn *conn;
100:    uint16_t mtu = 0;
101:
102:    conn = new0(struct gatt_conn, 1);
103:    if (!conn)
104:        return NULL;
105:
106:    conn->att = bt_att_new(fd, false);
107:    if (!conn->att) {
108:        fprintf(stderr, "Failed to initialze ATT transport layer\n");
109:        free(conn);
110:        return NULL;
111:    }
112:
113:    bt_att_set_close_on_unref(conn->att, true);
114:    bt_att_register_disconnect(conn->att, gatt_conn_disconnect, conn, NULL);
115:
116:    //bt_att_set_security(conn->att, BT_SECURITY_MEDIUM);
117:
118:    conn->gatt = bt_gatt_server_new(gatt_db, conn->att, mtu, 0);
119:    if (!conn->gatt) {
120:        fprintf(stderr, "Failed to create GATT server\n");
121:        bt_att_unref(conn->att);
122:        free(conn);
123:        return NULL;
124:    }
125:
126:    conn->client = bt_gatt_client_new(gatt_cache, conn->att, mtu, 0);
127:    if (!conn->gatt) {
128:        fprintf(stderr, "Failed to create GATT client\n");
129:        bt_gatt_server_unref(conn->gatt);
130:        bt_att_unref(conn->att);
```

gatt_conn Structure in gatt.c (D:\桌面\...\peripheral_self) at line 37 (5 lines)

在static struct gatt_conn *gatt_conn_new(int fd)函数中注释掉bt_att_set_security(conn->att, BT_SECURITY_MEDIUM);这行代码，这行的代码是蓝牙设备与手机通信时的验证配对操作

代码编写完可以去试试了



编译好的代码如上,后续再添加别的东西(蓝牙点灯等..)