Perfect Wireless Experience
完美无线体验

FIBOCOM Linux gsmMuxd Client instructions

Version：V1.0.1

Date：2019-09-19

Applicability type

No.	Product model	Description
1	NL668-CN-00	NA
2	NL668-CN-01	NA
3	NL668-CN-02	NA
4	NL668-EAU-00	NA
5	NL668-EU-00	NA
6	NL668-EU-01	NA
7	NL668-JP-00	NA
8	NL668-JP-01	NA
9	NL668-CN-04	NA
10	NL668-CN-03	NA
11	NL668-LA-00	NA
12	NL668-757S-00-01	NA
13	NL668-757S-01-00	NA
14	M910-GL-00	NA
15	MA510-GL-00	NA
16	MA510-GL-01	NA

Copyright

Notice

The document is subject to update from time to time owing to the product version upgrade or other reasons. Unless otherwise specified, the document only serves as the user guide. All the statements, information and suggestions contained in the document do not constitute any explicit or implicit guarantee.

Trademark

The trademark is registered and owned by Fibocom Wireless Inc.

Versions

Version	Author	Assessor	Approver	Update Date	Description
V1.0.0	Lu Kai			2019-03-20	Initial version
V1.0.1	He Ruichen	Long Yiliang	Ma Zhixin	2019-09-19	Modify content

Contents

1 Document overview

This article briefly describes how to migrate and use gsmMuxd in a Linux system.gsmMuxd is the driver for the user to use the serial port to implement the basic functions of the GSM 07.10 protocol. gsmMuxd provides multiple virtual connections between the TE and the MS. The TE and the MS communicate with each other through a virtual channel. Each channel between the TE and the MS is called a Data Link Connection (DLC) and is sequentially established.

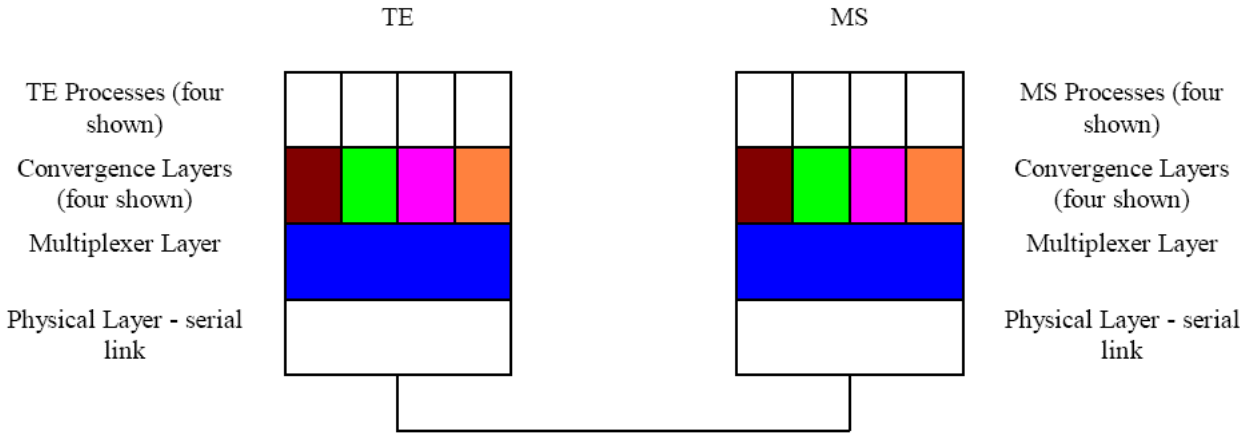

Figure 1

2 Transplant compilation

The gsmMuxd source directory structure is shown in Figure 2:

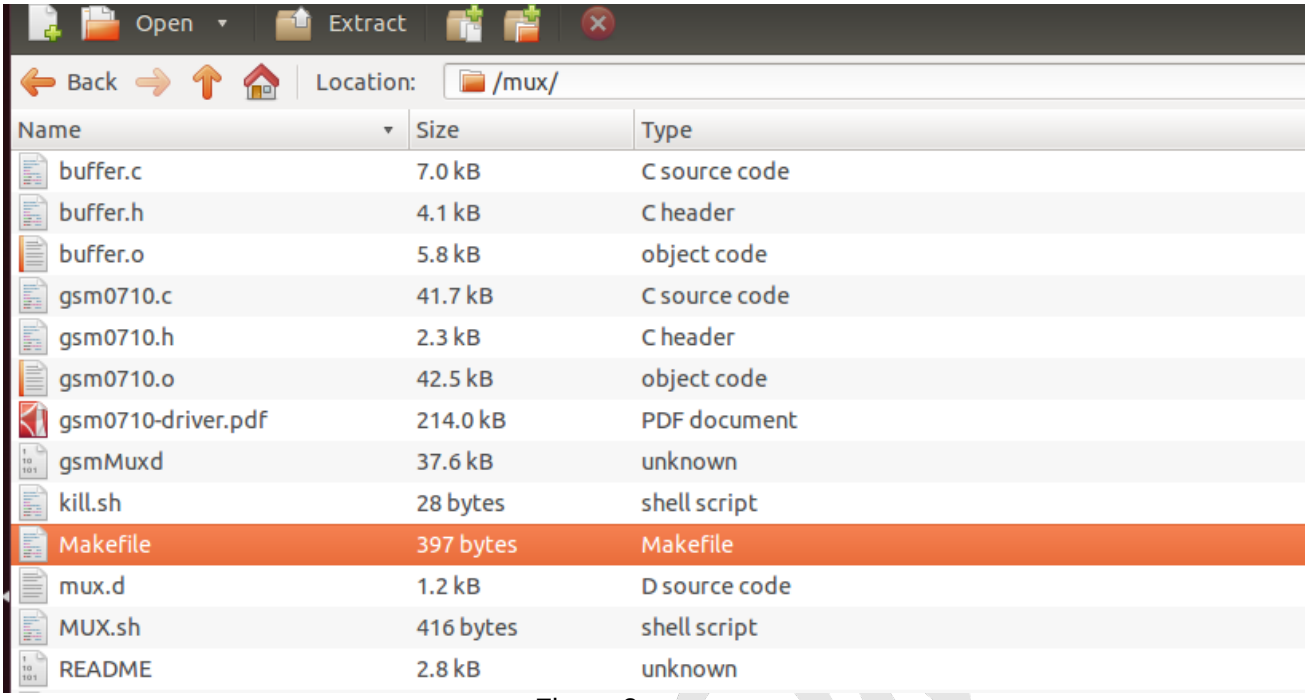

Figure 2

Compile, please use the Makefile under this directory to compile directly. If you want to cross-platform port, please modify the CC and LD options in the Makefile, as shown in Figure 3:

```
root@ubuntu: /home/ght/mux
# Comment/uncomment the following line to disable/enable debugging
DEBUG = y

TARGET = gsmMuxd
SRC = gsm0710.c buffer.c
OBJS = gsm0710.o buffer.o

CC = gcc
LD = gcc
CFLAGS = -Wall
LDLIBS = -lm

ifeq ($(DEBUG),y)
  CFLAGS += -DDEBUG
endif


all: $(TARGET)

clean:
        rm -f $(OBJS) $(TARGET)

%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<

$(TARGET): $(OBJS)
        $(LD) $(LDLIBS) -o $@ $(OBJS)

.PHONY: all clean
```

Figure 3

After compiling with the make command, the target file will be generated in the current directory: gsmMuxd

3 Instructions

The method of using gsmMuxd is shown in Figure 4 below:

```
root@ubuntu:/home/ght/mux# ./gsmMuxd

Usage: ./gsmMuxd [options] <pty1> <pty2> ...
  <ptyN>               : pty devices (e.g. /dev/ptya0)

options:
  -p <serport>         : Serial port device to connect to [/dev/modem]
  -f <framsize>        : Maximum frame size [32]
  -d                   : Debug mode, don't fork
  -m <modem>           : Modem (mc35, mc75, generic, ...)
  -b <baudrate>        : MUX mode baudrate (0,9600,19200, ...)
  -P <PIN-code>        : PIN code to fed to the modem
  -s <symlink-prefix>  : Prefix for the symlinks of slave devices (e.g. /dev/mux)
  -w                   : Wait for deamon startup success/failure
  -r                   : Restart automatically if the modem stops responding
  -h                   : Show this help message
root@ubuntu:/home/ght/mux#
```

Figure 4

4 Important parameter description

./gsmMuxd -p /dev/ttyUSB0 -b 115200 -f 512 -s /dev/mux -w /dev/ptmx /dev/ptmx -d

Note: At present, it supports 4 virtual serial channels (except M910-GL project supports 3 virtual serial channels)。

-p /dev/ttyUSB0	-- Specify the physical serial port of the connected module
-b 115200	-- The specified baud rate is 115200
-f 512	-- The maximum length of the module to transmit the frame packet, the default is 31
-s /dev/mux -w /dev/ptmx /dev/ptmx	-- Generate two mux ports in the /dev/ directory
-d	-- Print debug information
-m	-- No pass, default generic

After the gsmMuxd command is executed successfully, two virtual mux ports will be generated in the /dev/ directory, as shown in Figure 5 below:

```
ght@ubuntu:~$ ls /dev/mux* -l
lrwxrwxrwx 1 root root 10 Nov 26 05:10 /dev/mux0 -> /dev/pts/3
lrwxrwxrwx 1 root root 10 Nov 26 05:10 /dev/mux1 -> /dev/pts/4
ght@ubuntu:~$
```

Figure 5

Keep the gsmMuxd process running in the background, Virtual successful mux0 or mux1 can perform AT-related operations, as well as data services such as external protocols or built-in protocol stacks. Figure6 below:

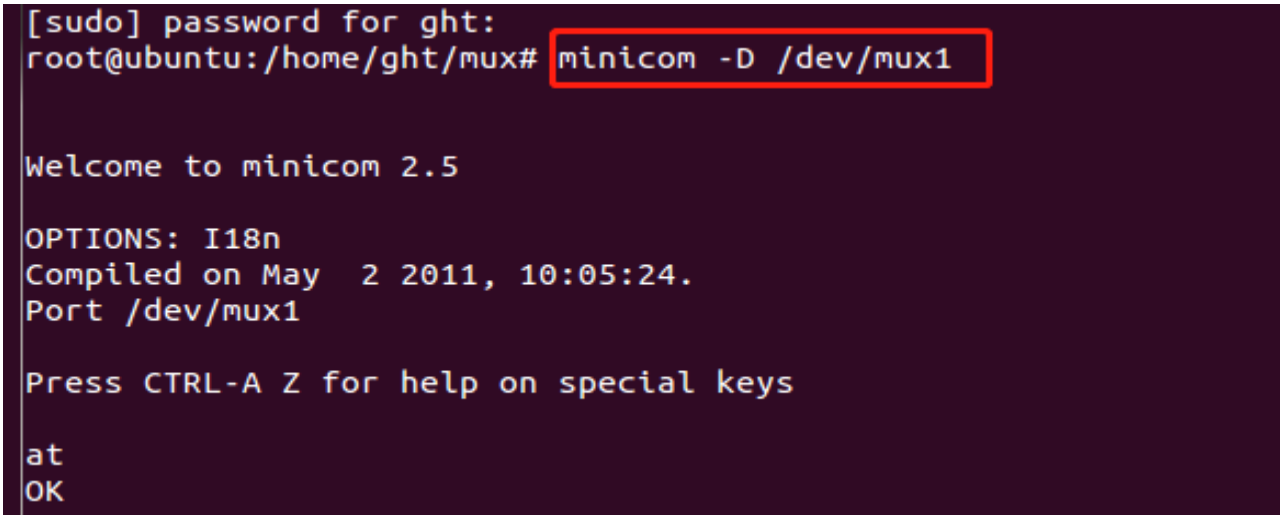

Figure 6

5 Key source overview

5.1 gsmMuxd program parameter analysis

When the client establishes the mux port, it will parse the passed parameters in the main function, as shown in Figure 7 below:


```
int main(int argc, char *argv[], char *env[])
{
#define PING_TEST_LEN 6
    static char ping_test[] = "\x23\x09PING";      //'#' ,'TAB'
    //struct sigaction sa;
    int sel, len;
    fd_set rfds;
    struct timeval timeout;
    unsigned char buf[4096], **tmp;
    char *programName;
    int i, size,t;

    unsigned char close_mux[2] = { C_CLD | CR, 1 };
    int opt;
    pid_t parent_pid;
    // for fault tolerance
    int pingNumber = 1;
    time_t frameReceiveTime;
    time_t currentTime;

    //command line like:gsmMuxd -p /dev/ttyUSB0 -b 115200 -s /dev/mux -w /dev/ptmx /dev/ptmx
    programName = argv[0];
    /*************************************/
    if(argc<2)
    {
        usage(programName);
        exit(-1);
    }
    _modem_type = GENERIC;

    serportdev="/dev/modem";

    while((opt=getopt(argc,argv,"p:f:h?dwrm:b:P:s:"))>0)
    {
        switch(opt)
        {
```

Figure 7

5.2 Enable the AT command of the MUX function

gsmMuxd parses the AT command that enables the MUX function, and passes the parameters to the modem to enable the MUX function, as shown in Figure 8 below:

```
/**
 * Function to start modems that only needs at+cmux=X to get-in mux state
 */
int initGeneric()
{
    char mux_command[20] = "AT+CMUX=0\r\n";
    unsigned char close_mux[2] = { C_CLD | CR, 1 };

    int baud = indexOfBaud(baudrate);
    if (baud != 0) {
        // Setup the speed explicitly, if given
        sprintf(mux_command, "AT+CMUX=0,0,%d,%d\r\n", baud,max_frame_size);
    }
                实际开启MUX功能的AT命令是：AT+CMUX=0,0,5,512；【512为控制模块一帧传输的
    /**         大字节数】
     * Modem Init for Siemens Generic like Sony
     * that don't need initialization sequence like Siemens MC35
     */
    if (!at_command(serial_fd,"AT\r\n", 10000))
    {
        if(_debug)
            syslog(LOG_DEBUG, "ERROR AT %d\r\n", __LINE__);

        syslog(LOG_INFO, "Modem does not respond to AT commands, trying close MUX mode");
        write_frame(0, close_mux, 2, UIH);
        at_command(serial_fd,"AT\r\n", 10000);
    }
        if (pin_code > 0 && pin_code < 10000)
        {
            // Some modems, such as webbox, will sometimes hang if SIM code
            // is given in virtual channel
            char pin_command[20];
            sprintf(pin_command, "AT+CPIN=%d\r\n", pin_code);
            if (!at_command(serial_fd,pin_command, 20000))
            {
        if(_debug)
            syslog(LOG_DEBUG, "ERROR AT+CPIN %d\r\n", __LINE__);
            }
        }

    if (!at_command(serial_fd, mux_command, 10000))
    {
        syslog(LOG_ERR, "MUX mode doesn't function.\n");
        return -1;
    }
        syslog(LOG_DEBUG, "fibocom debug AT+CMUX=%s %d \r\n", mux_command, __LINE__);
    return 0;
}
```

Figure 8

5.3 MUX data loop processing

Loop through the data of the MUX port in the following code, as shown in Figure 9 below:

```
if (FD_ISSET(serial_fd, &rfds))
{
    // input from serial port
    if(_debug)
        syslog(LOG_DEBUG, "Serial Data\n");
    if ((size = gsm0710_buffer_free(in_buf)) > 0 && (len = read(serial_fd, buf, min(size, sizeof(buf)))) > 0)
    {
        gsm0710_buffer_write(in_buf, buf, len);
        // extract and handle ready frames
        if (extract_frames(in_buf) > 0 && faultTolerant) {
            frameReceiveTime = currentTime;
            pingNumber = 1;
        }
    }
}

// check virtual ports
for (i = 0; i < numOfPorts; i++)
    if (FD_ISSET(ussp_fd[i], &rfds))
    {

        // information from virtual port
        if (remaining[i] > 0)
        {
            memcpy(buf, tmp[i], remaining[i]);
            free(tmp[i]);
        }
        if ((len = read(ussp_fd[i], buf + remaining[i], sizeof(buf) - remaining[i])) > 0)
            remaining[i] = ussp_recv_data(buf, len + remaining[i], i);
        if(_debug)
            syslog(LOG_DEBUG,"Data from ptya%d: %d bytes\n",i,len);
        if(len<0)
        {
                                    // Re-open pty, so that in
            remaining[i] = 0;
```

循环解析MUX端口接收到的数据

Figure 9

5.4 MUX frame format and data legality check

The MUX frame format verifies part of the code, as shown in Figure 10 below:

```
GSM0710_Frame *frame = NULL;

// Find start flag
while (!buf->flag_found && gsm0710_buffer_length(buf) > 0) {
  if (*buf->readp == F_FLAG)
    buf->flag_found = 1;
  INC_BUF_POINTER(buf, buf->readp);
}
if (!buf->flag_found) // no frame started
  return NULL;

// skip empty frames (this causes troubles if we're using DLC 62)
while (gsm0710_buffer_length(buf) > 0 &&
    (*buf->readp == F_FLAG)) {
  INC_BUF_POINTER(buf, buf->readp);
}

if (gsm0710_buffer_length(buf) >= length_needed) {
  data = buf->readp;
  frame = malloc(sizeof(GSM0710_Frame));

  frame->channel = ((*data & 252) >> 2);
  fcs = r_crctable[fcs^*data];
  INC_BUF_POINTER(buf,data);

  frame->control = *data;
  fcs = r_crctable[fcs^*data];
  INC_BUF_POINTER(buf,data);

  frame->data_length = (*data & 254) >> 1;
  fcs = r_crctable[fcs^*data];
  if ((*data & 1) == 0) {
    /* Current spec (version 7.1.0) states these kind of frames to be invalid
     * Long lost of sync might be caused if we would expect a long
     * frame because of an error in length field.*/
    INC_BUF_POINTER(buf,data);
    frame->data_length += (*data*128);
    fcs = r_crctable[fcs^*data];
    length_needed++;
    /*
    free(frame);
    buf->readp = data;
    buf->flag_found = 0;
    return gsm0710_buffer_get_frame(buf);*/
  }
  length_needed += frame->data_length;
```

Figure 10

The MUX frame packet length check code is as shown in Figure 10. If the transmitted data is larger than the set max_frame_size, the data is discarded.

```
if (frame->data_length > max_frame_size) /* Field Sanity check if payload is bigger than the max size negotiated in +CMUX */
{
  syslog(LOG_ALERT, "Dropping frame: Corrupt! Length field indicated %d. Max %d allowed",frame->data_length, max_frame_size);
  free(frame);
  return NULL;
}
```

Figure 11

6 Common problem

./gsmMuxd -p /dev/ttyUSB0 -b 115200 -f 512 -s /dev/mux -w /dev/ptmx /dev/ptmx -d

In the above command usage process, if the MUX channel fails to be established, two points should be investigated:

------> Verify whether the module is powered on normally

------> Verify that UART AT port is properly communicated and baud rate is properly configured

------> Make sure that you have permission to operate / dev / ttyUSB0 port, and there may be a jump in multiple operations under PC. Make sure that ttyUSB0 port is actually opened.