综合实训案例

实验环境

- 硬件：UP-Magic 魔法师实训平台，配套模块(温湿度传感器、热释红外传感器、光谱气体传感器、光敏声响开关、直流电机模块、LED 蜂鸣器模块、点阵 LCD 模块、双数码管模块)，PC 机 Pentium 500 以上， 硬盘 40G 以上,内存大于 256M
- 软件：Vmware Workstation +Fedora8 + MiniCom/Xshell + ARM-LINUX 交叉编译开发环境
  qt-embedded-linux-opensource-src-4.4.0.tar.bz2 tslib-1.4.tar.bz2
  arm-linux-gcc-3.4.1.tar.bz2 UP-Magic_Demo.tar.bz2

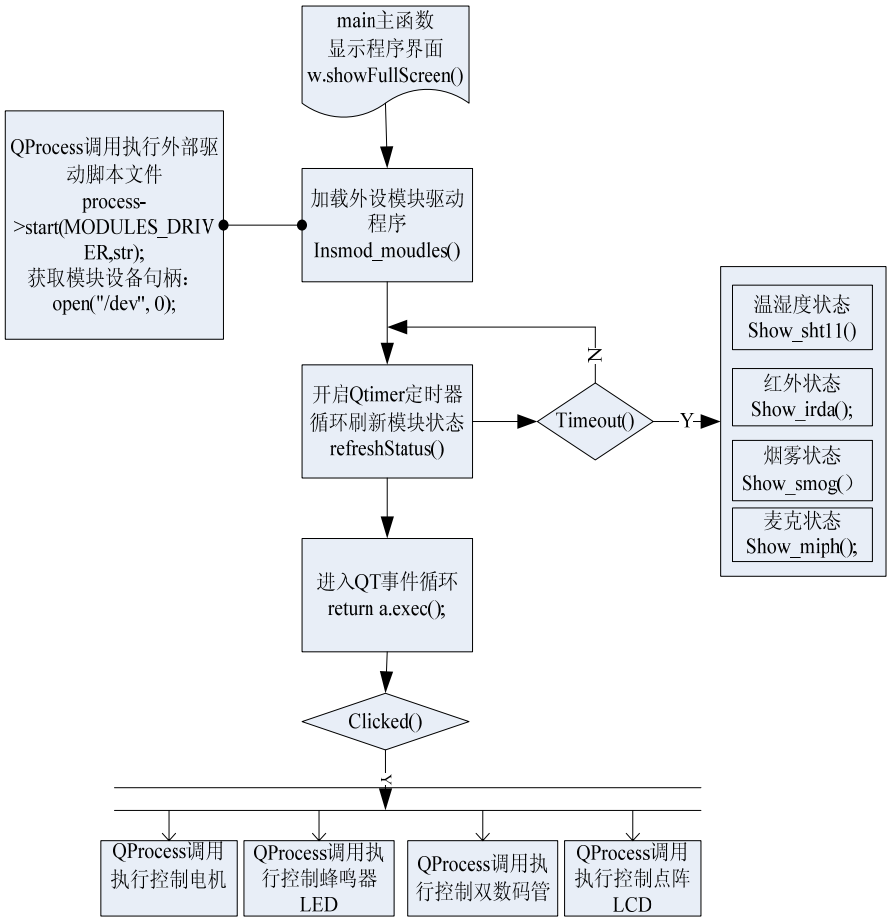
实验内容

- **内容描述**

  基于 UP-Magic 魔法师实训平台，在 ARM-Linux 系统下，利用 QT 图形用户开发工具，实现对外围硬件模块的控制和传感器数据采集的人机交互界面程序设计。

实验原理

- **程序流程图**



◆ 关键代码分析

软件构造函数：

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);//构建 UI 工程界面
    //设置界面背景
    QPalette palette1=this->palette();
    palette1.setBrush(QPalette::Window,QBrush(QPixmap("./images/bg.png")));
    this->setPalette(palette1);
    this->setAutoFillBackground(true);

    QTimer *timer = new QTimer( this );
   //进行定时器信号连接
    connect( timer, SIGNAL(timeout()), this, SLOT(refreshStatus()) );
    //加载模块硬件驱动脚本程序
    Insmod_moudles();
```

```
    sleep(1);
    //开启刷新状态定时器
    timer->start(1000); // 1 seconds single-shot timer
    GetStatus();//显示模块初始状态

}
```

Insmod_moudles()函数使用 Qprocess 调用外部脚本程序，并获取模块设备文件句柄：

```
void Widget:: Insmod_moudles()
{
    QProcess *process = new QProcess;
    QStringList str;
    str << "";
    process->start(MODULES_DRIVER,str);
    process->waitForStarted();
    sleep(1);

    fd_irda = open("/dev/irda", 0);
    if (fd_irda < 0) {
        printf("Can't open /dev/irda\n");
    }
    fd_smog = open("/dev/smog", 0);
    if (fd_smog < 0) {
        printf("Can't open /dev/smog\n");
    }
    fd_miph = open("/dev/miph", 0);
    if (fd_miph < 0) {
        printf("Can't open /dev/miph\n");
    }
    fd_sht11 = open("/dev/sht11", 0);
    if (fd_sht11 < 0) {
        printf("Can't open /dev/sht11\n");
    }
}
```

refreshStatus()槽函数调用模块状态子函数

```
void Widget::refreshStatus()
{
    GetStatus();
}
void Widget::GetStatus()
```

```
{
    Show_sht11();//温湿度获取子函数
    Show_irda(); //红外状态子函数
    Show_smog(); //烟雾状态子函数
    Show_miph(); //麦克声音状态子函数
}
```

按钮控制方式使用 QT 自带的信号与槽的命名规则方式进行信号和槽的连接：

```
// 启动电机槽函数
void Widget::on_pb_motor_on_clicked()
{
    QProcess *process = new QProcess;
    process->start("/root/motor/dcm_test");
    process->waitForStarted();

}
// 关闭电机槽函数
void Widget::on_pb_motor_off_clicked()
{
    QProcess *process = new QProcess;
    process->start("/root/motor/motor_stop.sh");
    process->waitForStarted();

}
// 开启双数码管槽函数
void Widget::on_pb_led_on_clicked()
{
    QProcess *process = new QProcess;
    process->start("/root/leds/magicleds_test");
    qDebug()<<"Exec leds app...";
    process->waitForStarted();
}
// 关闭双数码管槽函数
void Widget::on_pb_led_off_clicked()
{
    QProcess *process = new QProcess;
    process->start("/root/leds/leds_stop.sh");
    process->waitForStarted();
}
// 开启蜂鸣器 LED 槽函数

void Widget::on_pb_buzzer_on_clicked()
{
```

```
    QProcess *process = new QProcess;
    QStringList str;

    str.clear();
    str << "1" << "1";
    process->start("/root/buzzer/gpio_test",str);
    process->waitForStarted();
}
// 关闭蜂鸣器 LED 槽函数

void Widget::on_pb_buzzer_off_clicked()
{
    QProcess *process = new QProcess;
    QStringList str;
    str.clear();
    str << "0" << "0";
    process->start("/root/buzzer/gpio_test",str);
    process->waitForStarted();
}
// 开启点阵 LCD 槽函数

void Widget::on_pb_matrix_on_clicked()
{
    QProcess *process = new QProcess;

    process->start("/root/lcd/s3c24xx_lcd_test");
    process->waitForStarted();
}
// 关闭点阵 LCD 槽函数

void Widget::on_pb_matrix_off_clicked()
{
    QProcess *process = new QProcess;
    process->start("/root/lcd/lcd_stop.sh");
    process->waitForStarted();
}
```

实验步骤

- 实验目录: magic_demo

编译 QT/E 环境

以下实验假定实验目录为/home/sprife/qt4/for_arm

1、拷贝并解压 QT/E 库及触摸屏库到实验目录 for_arm

```
#cd /home/sprife/qt4/
#mkdir for_arm
#cd for_arm
#cp /UP-Magic/gui/Qt/src/qt-embedded-linux-
 opensource-src-4.4.0.tar.bz2 ./
#cp /UP-Magic/gui/Qt/src/tslib-1.4.tar.bz2 ./
#tar xjvf qt-embedded-linux-opensource-src-4.4.0.tar.bz2
#tar xjvf tslib-1.4.tar.bz2
```

2、编译 tslib1.4 触摸屏库

```
#cd tslib-1.4
#vi build.sh
```

修改该脚本文件为如下：

```
#/bin/sh
export CC=arm-linux-gcc
./autogen.sh
echo "ac_cv_func_malloc_0_nonnull=yes" >arm-linux.cache
./configure --host=arm-linux --cache-file=arm-linux.cache
-prefix=$PWD/../tslib1.4-install
make
make install
```

推出保存后编译：

```
#./build.sh
```

3、编译 QT/E 库

```
#cd /home/sprife/qt4/for_arm/
#cp -a tslib1.4-install/lib/*  qt-embedded-linux-opensource-src-4.4.0/lib/
#cp -a tslib1.4-install/include/ts*
qt-embedded-linux-opensource-src-4.4.0/include/
#cd qt-embedded-linux-opensource-src-4.4.0
#./configure -embedded arm -xplatform qws/linux-arm-g++ -nomake demos -nomake
examples -no-stl -no-qt3support -no-phonon -no-svg -no-webkit -no-openssl -no-nis
-no-cups -no-iconv -no-pch -no-dbus -no-separate-debug-info -depths 8,16 -fast
-little-endian -qt-mouse-linuxtp -qt-mouse-tslib
-I$PWD/../tslib1.4-install/include -L$PWD/../tslib1.4-install/lib -prefix
/usr/local/Trolltech/qt-embedded-4.4.0
```

```
#make
#make install
```

./configure 配置选项过长，使用该命令时候请仔细检查不要写错。

-prefix 选项要特殊注意，该指定目录必须要与实际 NFS 目录对应.

如果出现如下错误提示：

../../include/QtGui/private/../../../src/gui/kernel/qapplication_p.h:347: error: multiple parameters named 'screen'

则更改 qapplication_p.h 并找到相应的位置,将源代码的 screen 改为 screen_t

```
#vi src/gui/kernel/qapplication_p.h
#void setScreenTransformation(QScreen *screen, int screen_t, int transformation);
```

若提示：

painting/qdrawhelper.cpp:3453: error: explicit template specialization cannot have a storage class

则更改 qdrawhelper.cpp 并找到相应位置,将错误所在函数的 static 注释掉

```
#vi src/gui/painting/qdrawhelper.cpp
#/*static*/ inline void madd_4(qargb8565 *dest, const quint32 a, const qargb8565 *src)
```

4、测试触摸屏及 QT/E 程序

1）建立 NFS 共享目录/home/sprife（此目录如果存在就不用在建立了）

```
#cd /home/sprife
#mkdir Trolltech
#cd Trolltech
#mkdir qt-embedded-4.4.0
#cd qt-embedded-4.4.0
#cp    /home/sprife/qt4/for_arm/qt-embedded-linux-opensource-src-4.4.0/lib/ ./ -arf
#cp /home/sprife/qt4/for_arm/tslib1.4-install/etc/ ./ -arf
#cp /home/sprife/qt4/for_arm/tslib1.4-install/bin/ ./ -arf
```

2）ARM 端挂载 NFS 共享目录

```
#mount -o nolock,rsize=4096,wsize=4096 192.168.1.43:/home/sprife   /mnt/nfs
```

具体 LINUX 主机 IP 需要具体设置

3）设置环境变量（ARM 端）

```
#cd /mnt/nfs/Trolltech/qt-embedded-4.4.0
#export QTDIR=$PWD
#export LD_LIBRARY_PATH=$PWD/lib
#export TSLIB_TSDEVICE=/dev/event0
#export TSLIB_PLUGINDIR=$PWD/lib/ts
#export TSLIB_CONSOLEDEVICE=none
```

```
#export TSLIB_CONFFILE=$PWD/etc/ts.conf
#export POINTERCAL_FILE=$PWD/etc/ts-calib.conf
#export QWS_MOUSE_PROTO=tslib:/dev/event0
#export TSLIB_CALIBFILE=$PWD/etc/ts-calib.conf
#export LANG=zh_CN
#export QWS_DISPLAY="LinuxFb:mmWidth160:mmHeight120:0"
#export QT_QWS_FONTDIR=$PWD/lib/fonts
```

4）执行触摸屏校准程序(在 qt-embedded-4.4.0 下)

```
#./bin/ts_calibrate
```

如果出现如下错误提示：

```
Couldnt load module pthres
No raw modules loaded.
ts_config: Success
```

则更改 ts.conf 配置文件选择一个输入设备

```
#vi /mnt/nfs/Trolltech/qt-embedded-4.4.0/etc/ts.conf
# Uncomment if you wish to use the linux input layer event interface
# module_raw input
```

更改为

```
# Uncomment if you wish to use the linux input layer event interface
module_raw input
```

若提示：

```
/dev/touchscreen/ucb1x00: No such file or directory
```

则一般是环境变量没设置好的问题，可以重新仔细检查环境变量的设置

再次执行触摸屏校准程序即可，程序会自动存储坐标校准信息以便应用程序 使用。

此时如无错误则进入 5 点触摸屏校准程序并存储配置文件

还可以进行其他触摸屏测试程序 ts_test、ts_print 等

5）执行 QT/E 带触摸屏的例子程序

在宿主机端拷贝编译好的 ARM 端可执行程序

```
#cp    /home/sprife/qt4/for_arm/qt-embedded-linux-opensource-src-4.4.0/
examples/widgets/digitalclock/digitalclock
/UP-Magic/Trolltech/qt-embedded-4.4.0/
```

ARM 端执行该程序

```
#./digitalclock  -qws
```

如出现如下错误：

```
error while loading shared libraries: libz.so.1: cannot open shared
object file: No such file or directory
```

则在宿主机端交叉编译器目录下搜索该库文件，拷贝到 lib 目录下(具体库　　文件位置可能不尽相同)

```
#cp /opt/host/armv4l/armv4l-unknown-linux/lib/libz.so.1
 /home/sprife/Trolltech/qt-embedded-4.4.0/lib/
```

若出现错误

error while loading shared libraries: libstdc++.so.6: cannot open shared object file: No such file or directory

拷贝

```
#cp    /home/bc/gcc-3.4.2-glibc-2.2.5/arm-linux/arm-linux/lib/libstdc++.so.6 /home/sprife/Trolltech/qt-embedded-4.4.0/lib/
```

若

error while loading shared libraries: libgcc_s.so.1: cannot open shared object file: No such file or directory

则

```
#cp    /home/bc/gcc-3.4.2-glibc-2.2.5/arm-linux/arm-linux/lib/libgcc_s.so.1 /home/sprife/Trolltech/qt-embedded-4.4.0/lib/
```

如果想让 QT/E 在 ARM 端支持 USB 鼠标，可以在 ARM 端配置环境变量

```
#export QWS_MOUSE_PROTO=MouseMan:/dev/input/mouse0
```

即可。

实验中具体用到的触摸屏及鼠标设备要根据具体设备而定义。

♦ 编译源程序

1、进入实验目录/home/sprife 将实验 DEMO 源码压缩包解压至该目录下：

```
[root@vm-dev ~]# cd /home/sprife/
[root@vm-dev sprife]# tar xjvf UP-Magic_Demo.tar.bz2
```

2、编译程序

```
[root@vm-dev sprife]# cd UP-Magic_Demo
[root@vm-dev UP-Magic_Demo]#
[root@vm-devUP-Magic_Demo]#
/home/sprife/qt4/for_arm/qt-embedded-linux-opensource-src-4.4.0/bin/qmake
[root@vm-dev UP-Magic_Demo]# make clean
[root@vm-dev UP-Magic_Demo]# make
```

使用前面编译 QT/E 环境生成的配套 qmake 工具生成 Makefile 文件，否则编译改工程会报错。

当前目录下生成 DEMO 程序 UP-Magic

将改应用程序及资源文件拷贝到 NFS 共享目录

/home/sprife/Trolltech/qt-embedded-4.4.0/

```
[root@vm-dev UP-Magic_Demo]# cp UP-Magic
/home/sprife/Trolltech/qt-embedded-4.4.0/
[root@vm-dev UP-Magic_Demo]# cp -a images
/home/sprife/Trolltech/qt-embedded-4.4.0/
```

♦ NFS 挂载实验目录测试

1、启动 UP-Magic 魔法师实训开发板，连好网线、串口线。将配套模块插入底板扩展端口。通过串口终端挂载宿主机实验目录。
设置开发板 IP：192.168.1.199（默认宿主机 LINUX IP 192.168.1.43,NFS 共享目录 /home/sprife）

```
up-tech:~ #ifconfig eth0 192.168.1.199
up-tech:~ #mount -t nfs -o nolock,rsize=4096,wsize=4096 192.168.1.43:/home/sprife /mnt/nfs
```

2、进入串口终端的 NFS 共享实验目录，设置 QT 环境变量。

```
#export QTDIR=$PWD
#export LD_LIBRARY_PATH=$PWD/lib
#export TSLIB_TSDEVICE=/dev/event0
#export TSLIB_PLUGINDIR=$PWD/lib/ts
#export TSLIB_CONSOLEDEVICE=none
#export TSLIB_CONFFILE=$PWD/etc/ts.conf
#export POINTERCAL_FILE=$PWD/etc/ts-calib.conf
#export QWS_MOUSE_PROTO=tslib:/dev/event0
#export TSLIB_CALIBFILE=$PWD/etc/ts-calib.conf
#export LANG=zh_CN
#export QWS_DISPLAY="LinuxFb:mmWidth160:mmHeight120:0"
#export QT_QWS_FONTDIR=$PWD/lib/fonts
```

3、执行程序

```
#./UP-Magic -qws
```

4、执行效果

程序的顺利执行要依赖ARM端的资源，如模块驱动和脚本程序，运行本实验前，确保UP-Magic

魔法师系统中已经在/mnt/yaffs 目录下烧写好了实验配套的程序(即需要将 UP-Magic_Demo_install.tar.bz2 压缩包解压至/mnt/yaffs/目录，烧写解压方法参考产品光盘的配套烧写手册)。

- **备注:**

连接外设模块时候，注意具体模块的连线方式。温湿度模块连接底板的 P1 端口、热释红外传感器模块连接 P5 端口、光谱气体传感器连接 P2 端口、光敏声响开关模块连接 P3 端口、直流电机模块连接 P6 端口、双数码管模块连接 P7 和 P8 端口、LED 蜂鸣器模块连接 P8 端口、点阵 LCD 模块连接 P7 和 P8 端口。注意以上硬件连接是由软件设置的，不能随意连接，且连线端口冲突的，只能同时连接一个模块(如双数码管模块和点阵 LCD 模块同时只能连接一个)。

本次实验使用的环境可能与用户实验环境不尽相同,用户需要根据具体实验环境进行设置。例如内核源码目录，NFS 共享目录,IP 地址等等。