QT 进程间通信详细介绍及 QProcess 机制分析

2013-07-27 | 分类: QT | 标签： | 浏览(1000)

1、QT 通信机制[喝小酒的网摘]http://blog.hehehehehe.cn/a/8732.htm

为了更好的实现 QT 的信息交互，在 QT 系统中创建了较为完善的**通信**机制。QT 的**通信**可分为 QT 内部**通信**和外部**通信**两大类。对于这两类**通信**机制及应用场合做如以下分析：

（1）QT 内部对象间**通信**

在图形用户界面编程中，经常需要将一个窗口部件的变化通知给窗口的其它部件使其产生相应的变化。对于这种内部对象间的**通信**，QT 主要采用了**信号**和**槽**的机制。这种机制是 QT 区别于其他 GUI 工具的核心机制。在大部分的 GUI 工具中，通常为可能触发的每种行为通过定义回调函数来实现。这种回调函数是一个指向函数的指针，在进行函数回调执行时不能保证所传递的函数参数类型的正确性，因此容易造成**进程**的崩溃。

在 QT 中，**信号**和**槽**的机制取代了这种繁杂的、易崩溃的对象**通信**机制。**信号**是当对象状态改变时所发出的。**槽**是用来接收发射的**信号**并响应相应事件的类的成员函数。信号和槽的连接是通过 connect（）函数来实现的。例如，实现单击按钮终止应用程序运行的代码 connect(button ， SIGNAL(clicked()) ， qApp ， SLOT(quit()));实现过程就是一个 button 被单击后会激发 clicked **信号**，通过 connect（）函数的连接 qApp 会接收到此信号并执行槽函数 quit()。在此过程中，信号的发出并不关心什么样的对象来接收此信号，也不关心是否有对象来接收此**信号**， 只要对象状态发生改变此信号就会发出。此时槽也并不知晓有什么的信号与自己相联系和是否有信号与自己联系，这样信号和槽就真正的实现了程序代码的封装，提 高了代码的可重用性。同时，信号和槽的连接还实现了类型的安全性，如果类型不匹配，它会以警告的方式报告类型错误，而不会使系统产生崩溃。

（2）QT 与外部设备间通信

QT 与外部通信主要是将外部发来的消息以事件的方式进行接收处理。外部设备将主要通过 socket 与 QT 应用程序进行连接。在此，以输入设备与 QT 应用程序的通信为例说明 QT 与外部通信的原理。

在 QT 的应用程序开始运行时，主程序将通过函数调用来创建并启动 qwsServer 服务器，然后通过 socket 建立该服务器与输入硬件设备的连 接。服务器启动后将会打开鼠标与键盘设备，然后将打开的设备文件描述符 fd 连接到 socket 上。等到 QT 应用程序进入主事件循环时，事件处理程序将通过 Linux 系统的 select 函数来检测文件描述符 fd 的状态变化情况以实现对 socket 的监听。如果文件描述符 fd 状态改变，说明设备有数据输入。此 时，事件处理程序将会发出信号使设备输入的数据能及时得到 QT 应用程序的响应。数据进入服务器内部就会以事件的形式将数据放入事件队列里，等待 QT 客户应 用程序接收处理。处理结束后再将事件放入请求队列里，通过服务器将事件发送到相应硬件上，完成外部输入设备与 QT 应用程序的整个通信过程。

2、 QProcess 机制分析

QProcess 类通常是被用来启动外部程序，并与它们进行通信的。QProcess 是把外部进程看成是一个有序的 I/O 设备，因此可通过 write（）函数实现对进程标准输入的写操作，通过 read（），readLine（）和 getChar（）函数实现对标准输出的读操作。

（1） QProcess 通信机制

QT 可以通过 QProcess 类实现前端程序对外部应用程序的调用。这个过程的实现首先是将前端运行的程序看成是 QT 的主进程，然后再通过创建主进 程的子进程来调用外部的应用程序。这样 QProcess 的通信机制就抽象为父子进程之间的通信机制。QProcess 在实现父子进程间的通信过程中是运用 Linux 系统的无名管道来实现的，因此为了能更加清楚的说明 QProcess 的通信机制，在此首先介绍关于无名管道实现父子进程间的通信机制。

无名管道是一种只能够在同族父子之间通信，并且在通信过程中，只能从固定的一端写，从另一端读的单向的通信方式。该无名管道是通过调用 pipe（）函数而创建的。创建代码如下：

```
#include <unistd.h>
int pipe(int fd[2]) ;    返回：若成功则为 0，若出错则为-1
```

创建后经参数 fd 返回两个文件描述符：fd[0]为读而打开，fd[1]为写而打开。经过 fork（）函数创建其子进程后，子进程将拥有与父进程相 同的两个文件描述符。如果想要实现父进程向子进程的通信则关闭父进程的读端 fd[0]，同时关闭子进程的写端 fd[1]。这样就建立了从父进程到子进程的 通信连接。

由于无名管道的单向通信性，所以如果要应用无名管道实现父子进程之间的双向通信则至少需要应用双管道进行通信。QProcess 类的通信原理就是利 用多管道实现了父子进程之间的通信。然而对于外部运行的应用程序大都是通过标准输入而读得信息，通过标准输出而发送出信息，因此只通过建立管道并不能完成 内外进程?之间的通信。要解决此问题，就如该模块开始时所说，QProcess 是把外部进程看成是一个 I/O 设备，然后通过对 I/O 设备的读写来完成内外进程的通信。

在 QProcess 中父子进程之间是通过管道连接的，要实现子进程能从标准输入中读得父进程对管道的写操作，同时父进程能从管道中读得子进程对标准 输出或标准容错的写操作，就要在子进程中将管道的读端描述符复制给标准输入端，将另外管道的写端描述符复制给标准输出端和标准容错端，即实现管道端口地址 的重定向。这样子进程对标准输入、标准输出及标准容错的操作就反应到了管道中。

QProcess 在正常渠道模式下具体实现共用了五个无名管道进行通信。五个管道的描述符分别用 childpipe[2]，stdinChannelpipe[2]，stdoutChannelpipe[2]，stderrChannelpipe[2] 和 deathpipe[2]五个数组来保存。deathpipe 指代的管道会用在消亡的子进程与父进程之间。当子进程准备撤销时会发送一个表示该子进程消 亡的字符给父进程来等待父进程进行处理。stdinChannelpipe，stdoutChannelpipe 和 stderrChannelpipe 所 指代的管道分别与标准输入，标准输出和标准容错进行绑定，实现了与外部程序的通信。childpipe 指代的管道主要是为父子进程之间的通信而建立的。

如果在管道中有新数据写入，就会通知相应进程去读。另外图 2 是 QProcess 在正常渠道模式下的通信原理图，如果是在融合渠道模式下，将没有容错 管道，此时原理图中将没有第一个管道，也就不会有管道描述符。同时，标准容错端和标准输出端将共同挂接到子进程的 stdoutChannelpipe 的写 端，来实现内外进程的通信。

（2） QProcess 应用方式

由于 QProcess 类实现了对底层通信方式较为完善的封装，因此利用 QProcess 类将更为方便的实现对外部应用程序的调用。在此，通过在 QT 界面中调用外部 mplayer 的例子来简单说明 QProcess 的应用方式。

```
const QString mplayerPath("/mnt/yaffs/mplayer");
const QString musicFile("/mnt/yaffs/music/sound.mp3");
QProcess* mplayerProcess=new QProcess();   QStringList args;
args<<"-slave";
args<<"-quiet";
args << "-wid";
args<<musicFile;
mplayerProcess->setProcessChannelMode(QProcess::MergedChannels);
yProcess->start(mplayerPath,args);
```

第一行指明了所要调用的外部应用程序 mplayer 的位置。第二行指明了所要播放的歌曲文件及地址。第五行设置 mplayer 为后台模式。在此模式 下，mplayer 将从标准输入中读得信息，并通过标准输出向主进程发送信息。六七行为 mplayer 运行的参数。第九行为设置进程渠道的模式为融合模 式，即将标准输出和标准容错绑定到同一个管道的写端。第十行为启动外部应用程序 mplayer。内核中管道及通信环境的建立都是在此步中完成的。

mplayer 在 slave 模式下运行会自动从标准输入中读取信息并执行。由 QProcess 的通信原理可知，管道的读端描述符 stdinChannelpipe[0]复制给了标准输入，即标准输入的描述符也为 stdinChannelpipe[0]，因此按照标准输入的描述符去 读信息就是到 stdinChannelpipe 所对应的管道中读取信息。所以如果想在 QT 的主进程中发送命令使 mplayer 退出，只需在主程序中向 stdinChannelpipe[1]端写入命令 quit 就可以，执行语句为 myProcess->write（”quit ”）;（此处的 write（）函数为 QProcess 类的成员函数，具体实现就是向 stdinChannelpipe[1]端写入信息）

（3）QProcess 的发展及分析

QProcess 类伴随着 QT/Embedded 的发展逐渐趋于完善。在 QTE2 及其更前版本中还没有 QProcess 类，如果想实现与外部应用程 序的通信，必须要自己实现对管道或 socket 的建立与重定向。到了 QTE3 版本，就实现了对 QProcess 类的封装。在 QTE3 的版本 中，QProcess 类的实现是通过应用 socket 来建立主进程与外部应用程序之间通信的。通信原理与图 3 所示基本相同，只是将图中的管道描述符改为是 socket 的描述符即可。QT 主程序在建立成对 socket 描述符时需要调用 Linux 系统函数 socketpair（）。在生成的成对 socket 描 述符之间可以实现父子进程之间的双向通信，即无论是 socket 的 0 套接口还是 1 套接口都可进行读写。

但为了避免出现通信过程中父子进程对同一个 socket 的争夺，例如，在子进程还未将父进程发送的信息全部读出时，子进程又要求将自己产生的数据返 回给父进程。如果父子进程双向通信只用一个 socket 来完成，就会出现父子进程发送的信息混乱情况。因此，对于 QProcess 的实现仍然必须通过多个 socket 来共同完成。

由上面的描述可知，尽管 socket 有双向通信功能，但在实现 QProcess 过程中只是利用 socket 实现了单向通信功能。因此既浪费了对资源 的利用又增加了系统的开销。为了解决此

问题，QTE4 版本将 QProcess 的通信连接方式由 socket 改为了只能实现单向通信的无名管道来实现。通信 原理就是以上 3.1 QProcess 通信机制中所描述的。

3、其它通信方式

除了上面介绍的无名管道和 socket 通信方式外，一般操作系统中常用的进程间通信机制也都可以用于 QT 系统内部不同进程之间的通信，如消息队列、共享内存、信号量、有名管道等机制。其中信号量机制在 QT 中已经重新进行了封装；有些机制则可以直接通过操作系统的系统调用来实现。另外，如果我们只是想通过管道或 socket 来实现较简单的外部通信，也可以重新创建管道或 socket 来实现自己要求的功能。例如，还是在 QT 主程序中调用外部 mplayer。如果我们只是想在 QT 主程序中控制 mplayer，而不要求得到 mplayer 输出的信息。则可以按照以下方式来实现：

```
const char* mplayerPath = "/mnt/yaffs/mplayer";
const char* musicFile = "/mnt/yaffs/music/sound.mp3";
const char* arg[5];
arg[0] = mplayerPath;
arg[1] = "-slave";
arg[2] = "-quiet";
arg[3] = musicFile;
arg[4] = NULL;
int fd[2],pid;
if(pipe(fd)<0)
    printf("creating pipe is error");
else
    while((pid=fork())<0);
if(pid==0){
::close(fd[1]);
::dup2(fd[0],STDIN_FILENO);
execvp(arg[0],(const* char*)arg);
}
else{
::close(fd[0]);
}
```

第 1 到 8 行与前面 QProcess 类实现调用 mplayer 一样，是用来指明 mplayer 运行时参数的。第 10 行是创建一个管道。第 13 行是创建一个子进程。15，20 行是关闭父子进程中没用的管道描述符。此时可结合图 2.1 和图 2.2 来理解从父进程到子进程通信环境的建立。第 16 行是把子进程的读端与标准输入绑定，以便 mplayer 能够接收到父进程发出的命令。17 行就是从子进程中调用外部 mplayer 的实现。此时，程序执行后，mplayer 就可以运行起来。如果想在 QT 主程序中通过发送命令使 mplayer 退出，就在管道的写端写入命令"quit"就可以。实现语句为 write(fd[1], "quit",strlen("quit"));

该例子说明了 **QT 通信**方式运用的灵活性，可以根据实际情况进行应用。同时该例子的实现方式正是利用了 QProcess 类实现的机制，因此可以结合这个例子更加深刻的理解 QProcess 类的实现机制。

[喝小酒的网摘]http://blog.hehehehehe.cn/a/8732.htm