

嵌入式 linux GUI--DirectFB + GTK 至尊秘笈

www.directfb.com.cnkendych@sina.com.cn

版权申明: 本文档一切权利归本人(kendych@sina.com.cn)所有, 用于商业用途徐征得本人同意, 如无法联系到本人, 须征得 www.directfb.com.cn 所有者同意; 用于非商业用途的, 无需任何许可, 但请尊重本人的署名权, 并注明出处 www.directfb.com.cn 及附加本申明。

关键词: 嵌入式 GUI arm linux DirectFB GTK tslib

1 前言

数年前, 曾经开发过一个嵌入式的产品, 如今市场依然存在, 但由于电子产品的升级换代很快, 许多元器件都采购不到了, 为了延续产品的生命周期, 计划在 linux 平台上开发新的版本。而在 linux 上的 GUI 上成了大问题, 最开始有用 Minigui 的打算, 也同飞漫公司联系过, 但费用我这里无法承受。(Minigui 作为国产优秀的嵌入式 GUI, 如果不是费用的问题, 应该是最优的选择。) QT 我也看了下, 也是收费的, 没有仔细研究。起先我打算用 MicroWindow 的, 但后来发现这个东西好久没有更新了, bug 一大堆。最后的眼光停留在 GTK 上, 最开始无从下手, 不知道到底适合不适合做嵌入式 GUI, 最后不知道在哪里看到一个介绍说诺基亚有产品是用 GTK 的, 觉得既然别人能做得, 我也能做得。做这个的初始阶段, 能看到的相关资料太少, 一点底都没有。但经过两个月(长了点)的努力, 终于解决了所有问题, 才一颗石头落地。个人认为, 在本文的帮助下, 如果你拥有初中级的嵌入式 linux 的知识, 也许一天就能解决问题, 最长也不会超过一个星期。

2 准备工作

硬件环境: linux 主机一台, 如果你喜欢用 windows, 可以在 windows 主机上用 vmware 虚拟一个 linux 系统。嵌入式开发套件, 包括嵌入式开发板、带触屏的液晶屏及相关连接电缆。

软件环境: x86 linux 发行版, 配置好 framebuffer, 并安装好 ftp server、telnet server、tftp server、nfs server、gcc 及相关软件、交叉编译器、开发套件的 kernel 2.6 的源码包。

本人用的是 Ubuntu 7.10 的发行版, 嵌入式开发版采用三星的 2440 系列 cpu, 如何搭建开发环境不在本文讨论范围之类, 请参考其他文档。交叉编译器用的是自己编译的 3.4.1, 最开始采用的是开发套件带的 3.4.1, 为什么要用自己编译的而不用开发套件自带的呢? 这里说本文第一个秘笈: 由于 GTK 采用的矢量字体里一些算法使用了浮点运算, 而 24xx 系列的 cpu 硬浮点不支持(我没有相关知识, 这是我的一个同事说得, 是不是他本来意思, 我都不敢确定, 如果你的 cpu 不是这个系列的, 请查看相关资料), 只要在交叉编译器里加上软浮点运算支持, 就应该没有问题了, 所以在开始之前, 先检查交叉编译器里有没有 --with-float=soft 这个选项。在我使用开发版自带的编译完成编译后, 运行 gtk 的程序, 总是有这个提示" shape engine failure, expect ugly output. the offending font is", 而屏幕上所有字符

都显示不出来, `button` 由于字符无法显示, 被压缩成一条线, 这个问题困扰我好几天, 我开始以为是字库设置的问题, 后来求助同事, 他听了我的描述后, 猜测可能是浮点运算的问题, 让我用我们自己编译的交叉编译器重新编译一下, 看看能不能解决问题, 结果真的解决了。

3 源码包的选择

选用 GTK 做嵌入式 GUI 是个痛苦的选择, 如果选用商业 GUI 如 MiniGui, 别人都给你整好了, 你拿过来用就可以了, 或者用 Wince, 还是与 windows 兼容的。而选用 GTK 做 GUI, 不同组织(个人)编写的 15 个软件包, 各个软件包又有不同的版本, 而网上又没有权威的指南, 如何选择合适的版本以及如何整合确实是个非常复杂的问题。

我是先从在 x86 上搭建 GTK 环境开始的, 首先我选用最新版本, 然后, 看到介绍说, GTK 在 framebuffer 上运行有两种模式: DirectFB 和 linux-fb, 而 linux-fb 的项目好像已经停止, 主要方向是 DirectFB, 后来查的有个 DirectFB + GTK 的英文文档, 基本都选用最新的版本, 而且很多包都可以使用系统自带的, 编译必须的源码就可以了, 最开始怎么也编译不成功, 我快绝望的时候, 发现释放出一个最新的 GTK 源码版本 (大概是 08 年 7 月 4 日), 我第一时间下载下来, 很快就编译出来了, 并且在 framebuffer 模式下 gtk-demo 以及一些 test 运行一点问题都没有。

熟悉了编译过程后, 我先尝试用交叉编译这个最新版本, 编译第一个包 glib-2.16.4 就失败了。我在网上阅读了很多关于交叉编译 DirectFB+GTK 的资料, 有个用 shell 脚本写的好像不错, 但是看不懂呀, 虽然我接触 unix 都十多年了, 还曾经在 unix 下做个一比较大型的项目, 但后来, linux 都是个人兴趣, 看了一些, 玩了一些, 没做过东西, shell 编程看过, 好多东西一知半解, 他那篇文档看的我都傻了, 这人真牛。但源码包的选择是可以借鉴的。

➤ 我的选择:

1) tslib-1.0.tar.bz2

触摸屏本来我是最后弄的, 但如果你需要触屏支持, 最好在一开始就搞定他。这个其实是 tslib.14, 为啥成了 1.0 的了, 我也不知道, 如何编译不是很难, 难在怎么配置, 相关文档请阅读本人的另外一篇文档《arm-linux 之 tslib》, 当然哪片文档可能并不能帮你搞定, 你需要动用你的聪明才智, 甚至你还要阅读 kernel 里触屏驱动的源码。我的开发套件触屏控制器是用的 cpu 本身的 AD 转换器, kernel 里的驱动是兼容 H3600 的, 所以我在 ts.conf 文件里 module_raw 模块设置为 h3600。如果你用的三星的 24XX 系列的 cpu, 也许使用的方式跟我相同, 使用同样的配置就可以了。如果不行, 需要阅读触屏驱动的源码及 tslib 的 module_raw 的源码, 找到匹配的模块就行了。

《arm-linux 之 tslib》的位置:

<http://www.directfb.com.cn/viewthread.php?tid=388&extra=page%3D1>

2) freetype-2.3.5.tar.bz2

3) glib-2.12.13.tar.bz2

4) libpng-1.2.19.tar.bz2

这个模块本来我是不需要的，我在编译 GTK 的时候想 disable 这个模块，configure 无法完成。

5) zlib-1.2.3.tar.bz2

6) jpegsrvc.v6b.tar.gz

这个模块是我必须的，但编译 GTK 的时候，configure 说找不到这个模块的 jpeglib.h，我也研究了好久，

7) tiff-3.7.4.tar.gz

这个模块我不需要，但编译了，最终，我不会采用这个模块。

8) DirectFB-1.1.1.tar.gz

最开始我采用的是 1.1.0，基本正常，最后弄触屏的时候，运行程序 DirectFB 怎么也加载不成功 inputdrivers 里的模块 libdirectfb_tslib.so，阅读了相关代码后，发现这个版本的 DFB 的这个模块加载后，不读取 tslib 的相关环境变量，直接加载/dev/input/tslibn 些设备，而我的开发板起来后，根本没有这些设备，1.1.0 之后的版本，相关代码做过修正，我也尝试了最新版本的 DFB 1.2.1，但这个版本的 DFB 跟好核后面选用的 GTK 的版本配合有点问题，无法编译后面的 GTK。我阅读 DFB 1.1.1 的相关代码，发现 1.1.0 拥有的这个问题已经修正。升级为 DFB1.1.1 后，运行 GTK 程序，DFB 开始加载 tslib 的相关模块，但只加载成功 tslib 的 module_raw 的模块，后面的模块加载失败。什么原因这里不做解释，后面再说。

9) **atk-1.19.3.tar.bz2**

10) **expat-2.0.1.tar.gz**

11) **libxml2-2.6.29.tar.gz**

12) **fontconfig-2.4.2.tar.gz**

13) **pango-1.16.4.tar.bz2**

14) **cairo-1.4.10.tar.gz**

15) **gtk+-2.10.14.tar.bz2**

这个模块最后编译，编译完成意味着我们成功完成了编译工作，我 `configure` 这个模块的时候碰到了两个问题：第一找不到 `pango`，最后看了好多资料，才知道在 `configure` 前需要设置这个环境变量 `export LDFLAGS="-L$PREFIX/lib -Wl,-rpath,$PREFIX/lib"`，这个啥意思，我不知道，为什么这样设我也不知道；第二找不到 `jpeglib.h`，我分析了 `log`，发现是在测试 `g++` 编译的时候，找不到 `jpeglib.h`，而测试 `gcc` 编译是没有问题的，我在 `configure` 前设置了这个 `export CPPFLAGS="-I$PREFIX/include"` 环境变量，告诉 `g++` 编译器到哪里找 `jpeglib.h`。

4 交叉编译

在宿主系统上交叉编译后的所有包的安装目录为 `/usr/gtkdfb`，当然，你可以使用其他的目录，但绝不能跟主机环境相冲突，将交叉编译后东西安装到宿主系统的默认位置，可能会导致你的宿主系统某些不可预知的后果。下面开始编译

1) Tslib

```
export PREFIX=/usr/gtkdfb
echo "ac_cv_func_malloc_0_nonnull=yes" > arm-linux.cache
./configure --host=arm-linux --prefix=$PREFIX --cache-file=arm-linux.cache --enable-inputapi=no
make
make install
```

编译前只需要指定 `PREFIX` 一个环境变量，这个模块编译下面的编译基本都需要以下三个

```
export LDFLAGS=-L$PREFIX/lib
export CFLAGS="-g -I$PREFIX/include"
export PKG_CONFIG_PATH=$PREFIX/lib/pkgconfig
```

2) glib

```
echo ac_cv_type_long_long=yes>arm-linux.cache
echo glib_cv_stack_grows=no>>arm-linux.cache
echo glib_cv_uscore=no>>arm-linux.cache
echo ac_cv_func_posix_getpwuid_r=yes>>arm-linux.cache
CC=arm-linux-gcc ./configure --host=arm-linux --build=i686-pc-linux --prefix=$PREFIX
--cache-file=arm-linux.cache
make
make install
```

3) atk

```
./configure --host=arm-linux --prefix=$PREFIX
make
make install
```

4) jpeg-6b

```
./configure --prefix=$PREFIX --enable-shared --enable-static
```

修改生成的 Makefile 文件:

The name of your C compiler:

CC= gcc 改成 CC=arm-linux-gcc (根据你自己交叉编译器的位置修改)

library (.a) file creation command

AR= ar rc 改成 AR= arm-linux-ar rc (同上)

second step in .a creation (use "touch" if not needed)

AR2= ranlib 改成 AR2=arm-linux-ranlib (同上)

```
mkdir $PREFIX/man
```

```
mkdir $PREFIX/man/man1
```

```
make
```

```
make install-lib
```

5) zlib

```
CC=arm-linux-gcc ./configure --prefix=$PREFIX --shared  
make  
make install
```

6) libpng

```
./configure --host=arm-linux --prefix=$PREFIX  
make  
make install
```

7) expat

```
./configure --host=arm-linux --prefix=$PREFIX  
make  
make install
```

8) freetype

```
./configure --host=arm-linux --prefix=$PREFIX  
make  
make install
```

9) libxml

```
./configure --host=arm-linux --prefix=$PREFIX  
make  
make install
```

10) fontconfig

```
export LIBXML2_CFLAGS=-I$PREFIX/include/libxml2  
export LIBXML2_LIBS="-L$PREFIX/lib -lxml2"  
./configure --host=arm-linux --prefix=$PREFIX --with-freetype-config=$PREFIX/bin/freetype-config  
--with-arch=arm  
make  
make install
```

11) tiff

```
./configure --host=arm-linux --prefix=$PREFIX --enable-shared  
make  
make install
```

12) DirectFB

```
./configure --host=arm-linux --prefix=$PREFIX --with-gfxdrivers=none --with-inputdrivers=all  
--enable-png --enable-jpeg --disable-tiff --enable-zlib --enable-sdl=no --enable-gif=no  
--disable-x11  
make  
make install
```

13) cairo

```
./configure --host=arm-linux --prefix=$PREFIX --without-x --disable-xlib --disable-xlib-xrender  
--enable-directfb --enable-freetype --disable-win32 --enable-pdf --enable-ps --disable-svg  
--enable-png  
make  
make install
```

不知道为什么，gtk 非要有 pdf 和 ps 的支持，没有就无法完成 configure，没办法，我只好在打开，其实也不能真正支持，因为 pdf 等东西根本没有加进来。不知道后来的 gtk 版本有无改进。

14) Pango

修改 configure 文件，将下面一些参数改成 true

```
have_cairo=true  
have_cairo_png=true  
have_cairo_ps=true  
have_cairo_pdf=true  
have_cairo_freetype=true
```

```
./configure --host=arm-linux --prefix=$PREFIX --enable-cairo --without-x  
make  
make install
```

15) gtk

```
export LDFLAGS="-L$PREFIX/lib -Wl,-rpath,$PREFIX/lib"
export CPPFLAGS="-I$PREFIX/include"
./configure --host=arm-linux --prefix=$PREFIX --with-gdktarget=directfb --without-x
--without-libtiff
make
make install
```

5 开发板运行环境

开发板的内核是 2.6 的，一切驱动都已完毕，文件系统采用 nfs 加载，在宿主系统上位于 /arm/root/root_nfs 上，我是编译好后，才弄板子的，但最好，在编译之前，最好先熟悉熟悉开发板，我弄好好几天才使开发板运行起来，因为 2.6 的 kernel 使用的 uboot 里某些东西跟 2.4 kernel 使用的 uboot 不同。我以前根本没有弄过，困扰了好几天。

在宿主系统上，将 /usr/gtkdfb 目录复制到 /arm/root/root_nfs/usr 目录下，开发板运行起来后，编译好的东西都会在 /usr/gtkdfb 目录下。

1) Pangorc

```
mkdir /arm/root/root_nfs/usr/gtkdfb/etc/pango
创建文件 /arm/root/root_nfs/usr/gtkdfb/etc/pango/pangorc 文件内容如下：
# pangorc file for uninstalled operation.
# We set the path as ../modules, such that it works from any of
# top level build subdirs.
#
```

```
[Pango]
ModuleFiles = /usr/gtkdfb/etc/pango/pango.modules
ModulesPath = /usr/gtkdfb/lib/pango/1.6.0/modules
上面的命令在宿主系统上运行；
下面的命令在开发板上运行，
/usr/gtkdfb/bin/pango-querymodules > /usr/gtkdfb/etc/pango/pango.modules
```

2) gfxdrivers

下面的命令本行在开发板上运行，只是消除一个警告，
`mkdir /usr/gtkdfb/lib/directfb-1.1-0/gfxdrivers`

3) gdk-pixbuf.loaders

下面的命令本行在开发板上运行，

```
mkdir /usr/gtkdfb/etc/gtk-2.0  
/usr/gtkdfb/bin/gdk-pixbuf-query-loaders > /usr/gtkdfb/etc/gtk-2.0/gdk-pixbuf.loaders
```

4) gtk.immodules

下面的命令本行在开发板上运行，

```
/usr/gtkdfb/bin/gtk-query-immodules-2.0 > /usr/gtkdfb/etc/gtk-2.0/gtk.immodules
```

5) fonts.conf

在/usr/gtkdfb/etc/fonts 目录下有 fonts.conf 这么个文件，这个文件配置了一些字库的信息，在宿主系统中打开文件/arm/root/root_nfs/usr/gtkdfb/etc/fonts/fonts.conf，修改<!-- Font directory list -->这行以下的东西，设置正确的字库目录。当然前提需要将字库复制到/arm/root/root_nfs 的合适目录下。比如 ubuntu 的设置如下：

```
<!-- Font directory list -->  
    <dir>/usr/share/fonts</dir>  
    <dir>/usr/X11R6/lib/X11/fonts</dir>  
    <dir>~/fonts</dir>
```

最简单的方法就是将 ubuntu 的这些目录里的内容复制到/arm/root/root_nfs 相对目录下，这样，这个文件都可以不修改。但最后你肯定不能这样做，因为嵌入式是不需要这么多字库的，我们只要需要的，不需要的统统砍掉，了解一下这个文件也应该是必须的。

6) directfbrc

这个文件我还没用过，我曾经相弄它，还读过相关的源码，因为当初碰到一个问题以为是他的问题，但不是它的问题，所以到现在为止，我还没弄。论坛上 VCVC0 说，没有这个文件，程序加载很慢，有了，就快很多，这里给个链接，我就不详述

<http://www.directfb.com.cn/viewthread.php?tid=373>

下面是帖子的内容。也现在也不臆想我的用法了。

如何配置基于 arm 的 directfbrc 配置文件？

我在我的开发板里跑了一下应用。如果/etc 目录下没有 directfbrc 文件的话，程序需要很久才能显示出来。

我简单设置了一下 directfbrc 内容如下：

```
system=fbdev  
fbdev=/dev/fb/0  
wm=default
```

```
mode=640x480
```

```
depth=32
```

```
pixelformat=RGB32
```

程序的启动速度就快了很多，不知道是什么原因。

还有问题就是 `direct` 的 `wm` 能否管理 `gtk` 的窗口。或者有没有基于嵌入式开发的窗口管理器可以使用的。

6 最后的秘笈

现在你在板子上运行 `GTK` 源码包里的 `test`，应该是没有问题，如果有问题，就要回头苦修了。最后的一个问题来了，`usb` 鼠标可以控制屏幕上的鼠标指针，触屏不反应，明明我的 `tslib` 测试程序运行一定问题也没有，`DFB` 编译也正确识别了 `tslib` 这个库，怎么回事呢？这就是最后的秘笈，我只所以敢将本文档称为至尊秘笈，也就因为这个秘笈，我认为，解决这个问题需要相当的技术水平。

首先，要说明这个问题都是复杂的。先说 `DFB` 的初始化，`DFB` 加载先读 `directfbrc`，读取一些设置，然后加载辅助模块，模块的加载都是动态的，采用 `dlopen` 方式加载的。`tslib` 加载一些模块也是采用这种方式加载的。应用程序动态链接到 `DFB` 的库，加载能加载的模块，同时加载了 `libdirectfb_tslib.so`，这个模块动态链接到 `libts.so` 这个库，动态加载 `libdirectfb_tslib.so` 同时做一些初始化 `tslib` 的工作，初始 `tslib` 又会动态加载 `tslib` 的一些模块，这四个模块 `pthres`、`variance`、`dejitter`、`linear` 会用到 `libts.so` 这个库里的一个函数 `tslib_parse_vars`，由于 `libdirectfb_tslib.so` 是用 `dlopen` 加载的，加载 `tslib` 的上四个模块时，根本就不知道到何处去找函数 `tslib_parse_vars`，而 `tslib` 的测试程序已经动态链接了 `libts.so`，加载 `tslib` 的上速模块时能够正确识别函数 `tslib_parse_vars`。

原因知道了，解决问题就简单了。链接应用程序时动态链接到库 `libts.so` 就可以了，我修改 `gtk` 源码 `tests` 目录下的 `Makefile`

```
LDLFLAGS = -L/usr/gtkdfb/lib -lts -Wl,-rpath,/usr/gtkdfb/lib
```

重新编译了这些测试程序，复制到开发板上，运行，一切都 `OK` 了。如果你将来写应用程序，不管是基于 `GTK` 的应用程序还是基于 `DFB` 的应用程序，编译的时候链接到动态库 `libts.so`，触屏的支持都是没有问题的。

7 结语

回想起来，搭建一个嵌入式的 `DFB+GTK` 的 `GUI` 开发平台真的很困难，且不说如此多不同组织和个人编写的包，就是每个人的环境也是千差万别的，能够给你提供帮助的人几乎没有，碰到问题，你只能在浩瀚的网络里找寻解决的方法。解决任何问题也许都要耗费巨大的精力。如果你的项目资金允许，还是请支持一下国产的优秀 `GUI` 软件——`MiniGUI`，你碰到的问题，应该都很得到迅速，圆满的解决，无疑会加快你的项目开发进度。当然，`DFB+GTK` 也是很好的选择，在你自己解决问题的时候，你会学到很多东西，还会带给你带来巨大的成就感。

最后，要感谢这些开源软件包的写作者们，是他们无私的奉献，才让我们能够有机会学习这些优秀的代码。还要感谢哪些在网上共享了自己经验的先驱们，是他们的点点星火，给了我灵感，为我指引了解决问题的方向。谨以本文献给他们，再次感谢他们。同时，将本文献给 www.directfb.com.cn 及坛主 [echo](#) 先生以及论坛的每一个现在的和将来的会员。我只不过在论坛上发表了两篇很短小的陋文，[echo](#) 先生竟邀请我当版主，使我倍感荣幸。

环境的搭建工作我基本完成了，项目的需要，我要做其他的事情了。如果你也在做类似的工作，碰到了一些问题，希望本文能给你帮助。如果你在其他地方看到本文（本人希望有网站转载本文），请访问 www.directfb.com.cn，也许你能得到更多的信息。

2008 年 9 月

✧ 免责声明：本人不保证本文档完全正确，如果你看到有什么不合适的地方，欢迎与本人交流。