MINIGUI 移植开发文档

一．说明

本文档主要是讲述 MINIGUI 的开发步骤以及基于两种不同运行环境：PC 机和特定目标系统上的开发，最后讲述每一种运行环境的具体开发步骤。
MINIGUI 的开发一般分为几个步骤：
1：建立目标系统的编译环境
2：安装 MINIGUI 的资源（libminigui 和 minigui）
3: 用目标系统的编译环境编译程序
4：下载编译好的程序到目标系统运行

MINIGUI 开发通常有二种方式，基于 PC 机和特定的开发板。当开始开发时，一般是还在 PC 机上开发，也就是先以 PC 机为目标系统，用 PC 机 LINUX 为编译环境，用提供的测试程序编译在 PC 机上运行的程序，然后在 PC 机上运行所编译的程序。另一种是基于某种开发系统，比如我们的 AT91RM9200 开发板。首先也是要在 PC 机的 LINUX 系统上建立适合目标系统的交叉编译环境，其他的和在 PC 机上一样，主要是要用交叉编译环境来编译程序。

MINIGUI 有二种运行模式：MINIGUI-Theads 和 MINIGUI-Lite。二种方式的源代码有 99％的兼容，MINIGUI-Theads 采用线程机制（类似 Wince），MINIGUI-Lite 采用类似 C/S 的模式,选择哪一种模式可在配制时定义。（更多的可看《MiniGUI 用户手册》附录 A）

二．基于 PC 机为目标系统：

1. MINIGUI 的组成：

MINIGUI 图形系统由函数库、资源、演示程序三部分组成：

1） MINIGUI 的函数库部分由三个函数库组成的。Libminigui、libmgext 以及 libvcongui。Libminigui 是提供窗口管理和图形接口的核心函数库，也是提供了大量的标准控件；libmgext 是 Libminigui 的一个扩展库，提供了一些高级控件以及“文件打开”对话框等；libvcongui 则提供了一个应用程序可用的虚拟控制台窗口，从而可以方便地在 MINIGUI 环境中运行字符界页的应用程序。Libmgext 和 libvcongui 已经包含 在 libminigui 函数库的源代码中。

2） 运行 MINIGUI 所需的资源。这些资源包括运行 MINIGUI 应用程序需要的基本字体、图标、位图以及标光标等。这些资源包含在 minigui-res-1.3.x.tar.gz 软件包中。

2. 建立 MINIGUI 运行环境的前提：

1）. 支持 POSIX1.X 的 LINUX 的系统。这包括 LINUX2.0、2.2 和 2.4 等，也包括 uclinux 等非标准 LINUX 系统。

2）. LINUX FrameBuffer 的驱动程序功能正常。在 PC 机上，如果显示芯片是 VESA 没有 FrameBuffer 支持的 LINUX 系统，需要编写特定的图形引擎才能运行 MINIGUI。在这里要特别注意就是有些集成了显示芯片主板，像 I810、I815 的主板的显示芯片不是与 VESA 兼容的，要从网上下载支持这种主板的 FrameBuffer 的驱动。（见附录 A）

3）. 打开 FrameBuffer 的支持：

lilo:

编辑 /etc/lilo.conf

在 image=/boot/vmlinux 后

添加 vga=0x317

运行:lilo

重新启动计算机

grub:

编辑 /boot/grub/grub.conf

在 root=/dev/hdxx 后添加

root=/dev/hdxx vga=0x317

(vga=792 将会把 framebuffer 配置成 1024x768-76 24 bpp)

重新启动计算机

若有打开 FrameBuffer 的话就会在启动时左上角看到一个小企鹅的图标，进入系统后运行 fbset（若系统支持此命令）命令会显示其支持模式。

若没有打开 FrameBufferr 的支持，在运行测试程序时会有如下的信息：

```
No available video device.
GAL: Does not find matched engine: fbcon.
Error in step 3: Can not get graphics engine information!
InitGUI failure when using /usr/local/etc/MiniGUI.cfg as cfg file
```

3. 编译和安装 MINIGUI:

1) 下载 MiniGUI

安装好图形引擎之后, 到 www.minigui.com 下载如下的.tar.gz 的软件包：

- libminigui-1.3.x.tar.gz: MiniGUI 函数库源代码, 其中包括 libminigui, libmywins, libmgext, 和 libvcongui.
- minigui-res-1.3.x.tar.gz: MiniGUI 所使用的资源, 包括图标, 位图和鼠标光标.
- Mde-minigui-1.3.x.tar.gz MiniGUI 的示例程序

2） 安装 MiniGUI 的资源文件

我们首先要安装 MiniGUI 的资源文件．请按照如下步骤：

1) 使用 "tar" 命令解开 "minigui-res-1.3.x.tar.gz". 可使用如下命令：

```
$ tar zxf minigui-res-1.3.x.tar.gz
```

2) 改变到新建目录中，然后以超级用户身份运行 "make" 命令：

```
$cd minigui-res-1.3.x
$ make install
```

在安装时是把资源安装在/usr/local/lib 路径下面

3）配置和编译 MiniGUI

MiniGUI 使用了 "automake" 和 "autoconf" 接口，因而 MiniGUI 的配置和编译非常容易：

1) 使用 "tar" 解开 "libminigui-1.3.x.tar.gz" 到新的目录：

```
$ tar zxf libminigui-1.3.x.tar.gz
```

2) 改变到新目录，然后运行 "./configure"：

```
$ cd libminigui-1.3.x
$ ./configure
```

3) 运行下面的命令编译并安装 MiniGUI:

```
$ make
$make install
```

4) 默认情况下, MiniGUI 的函数库将安装在 `/usr/local/lib' 目录中．您应该确保该目录已经列在 "/etc/ld.so.conf" 文件中．并且在安装之后，要运行下面的命令更新共享函数库系统的缓存：

```
$ ldconfig
```

5) 如果要控制您的 MiniGUI 提供那些功能，则可以运行：

```
$ ./configure --help
```

查看完整的配置选项清单，然后通过命令行开关打开或者关闭某些功能．例如，如果您不希望 MiniGUI 使用 LoadBitmap 函数装载 JPEG 图片，则可以使用：

```
$ ./configure --disable-jpgsupport
```

更详细的配置清单可参考《MiniGUI 用户手册》。

6) 注意，某些 MiniGUI 特色依赖于其它函数库，请确保已安装了这些函数库．

4）运行 MiniGUI 的演示程序

在 mde-minigui-1.3.x.tar.gz 软件中包含了各种演示程序，

运行之前，应该解开并编译这些 tar.gz 包：

1) 使用 "tar" 命令将软件包解开到新的目录．

2) 依次运行 "./configure" 和 "make" 编译演示程序．

3) 尝试运行演示程序和应用程序．例如，如果将 MiniGUI 配置为 MiniGUI-Threads, 则可以进入 "mde-minigui-1.3.x/dlgdemo" 运行 "./dlgdemo",还可运行其他的程序。

4) 如果配置并安装了 MiniGUI-Lite, 则应该首先运行服务器，然后从服务器当中运行其它演示程序．编译 "mde-minigui-1.3.x" 将生成一个 "mginit" 程序，该程序将提供一个运行于 MiniGUI-Lite 的虚拟控制台．

5) 安装及配置示例

本示例假定用户使用的系统是 RedHat Linux 7.x 发行版，使用 Linux 内核 2.4.xx, 用户的目标是运行 MiniGUI-Threads(使用 MiniGUI Version 1.3.0).

步骤如下：

1) 确保您的 PC 机显示卡是 VESA 兼容的．大多数显示卡是 VESA 兼容的，然而某些内嵌在主板上的显示卡可能不是 VESA 兼容的，比如 Intel i810/i815 系列．如果显示卡是 VESA 兼容的，就可以使用 Linux 内核中的 VESA FrameBuffer 驱动程序了．

2) 确保您的 Linux 内核包含了 FrameBuffer 支持，并包含了 VESA FrameBuffer 驱动程序. RedHat Linux 7.x 发行版自带的内核中已经包含了该驱动程序．如果使用自己编译的内核，请检查您的内核配置．

3) 修改 /etc/lilo.conf 文件，在您所使用的内核选项段中，添加如下一行：

```
vga=0x0317
```

这样, Linux 内核在启动时将把显示模式设置为 1024x768x16bpp 模式．如果您的显示器无法达到这种显示分辨率，可考虑设置 vga=0x0314, 对应 800x600x16bpp 模式．

修改后的 /etc/lilo.conf 文件类似:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
linear
default=linux

image=/boot/vmlinuz-2.4.2
        vga=0x0317          ; 这一行设置显示模式.
        label=linux
        read-only
        root=/dev/hda6

other=/dev/hda1
        label=dos
```

4) 运行 lilo 命令，使所作的修改生效，并重新启动系统．

5) 如果一切正常，将在 Linux 内核的引导过程中看到屏幕左上角出现可爱的 Linux 吉祥物 -- 企鹅，并发现系统的显示模式发生变化．

6) 按照第 4 节所讲，下载 libminigui-1.3.0.tar.gz,minigui-res1.3.0.tar.gz,mde-minigui-1.3.0.tar.gz 等软件包．注意要安装正确的版本．

7) 以 root 用户身份安装 minigui-res-1.3.0.tar.gz

```
$ tar zxf minigui-res-1.3.0.tar.gz
$ cd minigui-res-1.3.0
$ make install
```

8) 在某个目录下解开 libminigui-1.3.0.tar.gz, 并进入新建的 libminigui-1.0.00 目录．

```
$ tar zxf libminigui-1.3.0.tar.gz
$ cd libminigui-1.3.0
```

9) 依次运行如下命令：

```
$ ./configure          （配制成 Threads 模式）
$ make
```

10) 以 root 身份运行 make install 命令：

```
$ make install
```

11) 修改 /etc/ld.so.conf 文件，将 /usr/local/lib 目录添加到该文件最后一行．修改后类似：

/usr/lib
/usr/X11R6/lib
/usr/i486-linux-libc5/lib
/usr/local/lib

12) 以 root 身份执行 ldconfig 命令：

```
# ldconfig
```

13) 在新目录中解开 mde-minigui-1.3.0.tar.gz，并进入新建目录：

```
$ tar zxf mde-minigui-1.3.0.tar.tz
$ cd mde-minigui-1.3.0
```

14) 依次运行如下命令：

```
$ ./configure
$ make
```

15) 如果编译正常后.

16) 切换到 ../dlgdemo 目录，执行其中的程序：

```
$ cd ../dlgdemo
$ ./dlgdemo
```

17) 如果能够在屏幕上看到 GUI 窗口，则表明一切 OK.

三．在特定的目标系统上运行 MINIGUI(at91RM9200 开发板)

在特定的目标系统上运行 MINIGUI 和在 PC 机上运行 MINIGUI 是一样的，不同的是在特定的目标系统上没有像 PC 机上的显卡,但是只要有类似 FrameBuffer 的驱动就能运行，也就是说在目标板上运行的 LINUX 要有类似 FrameBuffer 的驱动能够驱动目标板上的显示芯片。

下面以 at91RM9200 开发板为例讲述移植的步骤：

1. 建立 at91RM9200 开发板的交叉编译环境

我们的 at91RM9200 开发板的 LINUX 是以 cross-2.95.3.tar.gz 为编译环境的，首先要在 PC 机的 LINUX 上安装好 cross-2.95.3.tar.gz,可到网上下载 cross-2.95.3.tar.gz，在/usr/local 下建个 arm 的文件，把 cross-2.95.3.tar.gz 复制到 arm 文件下，解压就成了。

```
$ tar Ixvf cross-2.95.3.tar.gz
```

2. 安装 minigui-res-1.3.0.tar.gz

使用 "tar" 命令解开 "minigui-res-1.3.0.tar.gz". 可使用如下命令：

```
$ tar zxf minigui-res-1.3.0.tar.gz
```

改变到新建目录中，然后以超级用户身份运行 "make" 命令：

```
$cd minigui-res-1.3.0
$ make install
```

在安装时是把资源安装在/usr/local/lib 路径下面, 这时会在/usr/local/lib 产生一个 minigui 的文件，在 minigui 下有个 res 的文件，文件下面会有各种 minigui 的资源。

3. 安装 libminigui-1.3.0.tar.gz

使用 "tar" 解开 "libminigui-1.3.0.tar.gz" 到新的目录：

$ tar zxf libminigui-1.3.0.tar.gz

改变到新目录，然后运行 "./configure":

$ cd libminigui-1.3.0

$ CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc ./configure --prefix=/usr/local/arm/2.95.3 --host=i386-linux --target=arm-linux

同时你也可以用--disable-xxxx 或者–enable-xxx 来打开和禁止某些功能。在这里CC 是指编译时用到的编译器，prefix 是指要把文件库安装在那个地方，target 是指产生哪种目标代码。

运行下面的命令编译并安装 MiniGUI:

$ make

$ make install

这时就会在/usr/local/arm/2.95.3/下的 include lib 的文件下面产生 MINIGUI 的的头文件库和函数库，以便编译程序时能找到正确的函数库和头文件库，同时还会在 usr/local/arm/2.95.3 的 etc 下产生文件 MiniGUI.cfg 的配制文件。

4. 编译运行 helloworld 测试程序

$ cd helloword

$ make

5. 编译 mde-minigui-1.3.0.tar.gz 的演示程序（可不做）

在新目录中解开 mde-minigui-1.3.0.tar.gz，并进入新建目录:

$ tar zxf mde-minigui-1.3.0.tar.tz

$ cd mde-minigui-1.3.0

依次运行如下命令:

$ CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc ./configure --prefix=/usr/local/arm/2.95.3 --host=i386-linux --target=arm-linux

$ make

在这时可能会遇到一些错误，特别是找不到库文件，这时可修改 Makefile 文件或者 configure.in 的文件，使能正确找到文件库，修改完 configure.in 后要运行

$ ./autogen.sh

重新产生 configure 的配制文件，然后再重新配制、编译。在运行$ ./autogen.sh 时也可能产生错误。这时要确定在 linux 上有安装 autoconf 和 automake 或者不能识别到 configure.in 中的某些定义，要安装比较新的版本。

6. 运行程序

当目标板上运行 MINIGUI 时，他首先会在用户的目录下寻找 minigui.cfg，要是找不到会到/usr/local/etc 寻找 minigui.cfg，最后是/etc/下寻找 minigui.cfg，直到找到为止。因此要把 LINUX 上的 minigui.cfg 复制目标板到的/usr/local/etc 或者/etc/下，在编译时如果编译成动态库的话也要把 /usr/local/arm/2.95.3/lib/下的.s0 的动态库复制到目标板的 /usr/local/lib/下面，同时在安装 minigui-res-1.3.0.tar.gz 时在 /usr/local/lib/下有产生 minigui 的资源文件也要把他放到目标板 /usr/local/lib/的下面。后面的这两个文件可任意放，但是一定要在 minigui.cfg 文件上修改放置的路径，以便在运行 MINIGUI 初始化时能够找到文件。最后是把编译完的演示程序下载到目标板上运行。

在程序目录下输入：

$./hello

就可看到程序运行的结果了。

7. 下载文件到目标板上方法

在下载文件到目标板 LINUX 系统上的某个目录下时，如果在目标在启动 LINUX 下能和计算机通信就能直接下载。有时也可以把 MINIGUI 的资源和演示程序和文件系统做在一起，然后下载文件系统到目标板上运行，进入文件直接运行程序就行。

在文件系统上添加文件的方法（附录 B），有时在文件系统中太大文件添加不了，可重建一个文件系统（附录 C）。

8. minigui.cfg 说明

在 at91RM9200 中建这几项设为

[fbcon]
defaultmode=640x480-16bpp
# IAL engine
ial_engine=dummy
mdev=/dev/mouse
mtype=PS2

更多的 FAQ 见（附录 D: 使用 MiniGUI 的一些 FAQ）

四．附录

附录 A

关于 i810、i815 主板的 framebuffer 支持

关于 i810、i815 显示芯片，Linux 内核中没有对这个显示芯片的 FrameBuffer 支持（除标准的 VGA16 FrameBuffer 之外）。
网络上有一个专门为 i810 编写的 FrameBuffer 驱动程序，可以到这里下载：
http://www.visuelle-maschinen.de/ctfb/drivers/i810
下载了 FrameBuffer 驱动程序以后可以根据以下的步骤配置：
The steps described here are given by leon, and have been tested on RedHat Linux 7.1, Linux kernel 2.4.2.
1. the i810fb tar ball -- i810fb_1.0x.tbz
which is compressed by bz2.
2. ucompress the tarball (using 'bunzip2' and 'tar' commands),
and copy the i810fb.c i810fb.h to your kernel directory:
/usr/src/linux-2.4/drivers/video
3.modify the file: /usr/src/linux-2.4/drivers/video/fbmem.c
1) add these lines as declaration:
#ifdef CONFIG_FB_I810
extern int i810fb_init (void);

```
extern int i810fb_setup (char*);
#endif
before lines:
static struct {
const char *name;
int (*init)(void);
int (*setup)(char*);
} fb_drivers[] __initdata = {
...
```

2) add these lines as the entry for i810 FrameBuffer driver:

```
#ifdef CONFIG_FB_I810
{ "i810fb", i810fb_init, i810fb_setup },
#endif
before lines:
#ifdef CONFIG_FB_VESA
{ "vesa", vesafb_init, vesafb_setup }
#endif
...
```

4.if you compile this driver into the kernel 2.4.0 or above
you should modify the file: /usr/src/linux-2.4/drivers/char/agp/agpgart_be.c
Current version:

```
----------------
static int __init agp_init(void)
{
init ret_val;
...
```

Change to:

```
----------
int __init agp_init(void)
{
static int ret_val = 0;
static int woman = 0;
if (woman) return ret_val;
woman = 1;
...
```

5.edit the Makefile in the directory: /usr/src/linux-2.4/drivers/video add line:

```
obj-$(CONFIG_FB_I810) += i810fb.o
```

6.edit the Config.in in the directory: /usr/src/linux-2.4/drivers/video add these lines:

```
if [ "$CONFIG_PCI" = "y" ]; then
bool ' i810 support' CONFIG_FB_I810
fi
```

below the lines:

```
bool 'Support for frame buffer devices (EXPERIMENTAL)' CONFIG_FB_RIVA
```

if ["$CONFIG_FB" = "y"]; then
define_bool CONFIG_DUMMY_CONSOLE y
if ["$CONFIG_EXPERIMENTAL" = "y"]; then
if [" $CONFIG_PCI" = "y"]; then
tristate ' nVidia Riva support (EXPERIMENTAL)' CONFIG_FB_RIVA
fi
7. exexute the 'make menuconfig'
a, make sure:
Character devices --->
<*> /dev/agpgart (AGP Support)
¡¤ Intel I810/I815 (on-board) support
b, make sure:
Console drivers --->
Frame-buffer support --->
¡¤ i810 support
8, compile the kernel image:
make dep
make bzImage
9, then cp your-bzImage /boot
10, edit /etc/lilo.conf:
add lines:
image = /boot/your-bzImage
append = "video=i810fb:x:1280,xv:1280,y:1024,yv:1024, \
depth:16,pclk:7800,le:248,ri:16,up:38,lo:1, \
hs:144,vs:3,sync=3,vmode:0,accel:0,font:VGA8x8"
initrd = /boot/initrd
label = i810fb
note: if your monitor doesn't support the 1280x1024 mode
you can add this lines instead:
image = /boot/your-bzImage
append = "video=i810fb:x:1024,xv:1024,y:768,yv:768, \
depth:16,pclk:15384,le:168,ri:8,up:29,lo:3, \
hs:144,vs:6,sync=3,vmode:0,accel:0,font:VGA8x8"
initrd = /boot/initrd
label = i810fb
This is a 1024x768x16bpp mode with 60Hz vertical frequency.
11, Run 'lilo' command.
12, Congratulations!! you can reboot your computer now!

附录 B

在文件系统中添加文件和应用程序

加入应用程序的 ramdisk 文件系统映像制作
ramdisk-rmk7.gz 为 LINUX 的文件系统映像压缩文件。（在这时有点要注意的就是若文件系统是压缩文件但是没有.gz 的后缀可加入.gz 再进行解压操作）
用户可以在文件系统中加入自己的应用，例如可以将 ramdisk-rmk7.gz 拷贝到根目录下, 新建一个 ramdisk 目录并解开 ramdisk-rmk7.gz：
cp ramdisk-rmk7.gz /
cd /
mkdir ramdisk
gunzip ramdisk-rmk7.gz
在根目录下会生成 ramdisk-rmk7
ramdisk-rmk7 为解开后的 Linux 的文件系统映像文件；
再将 ramdisk-rmk7 文件系统映像文件 mount 到新建目录 ramdisk 中：
mount －o loop ramdisk-rmk7 ramdisk/
cd ramdisk/
ls
查看目录中的内容如下：
bin etc Linuxrc mnt sbin usr
dev lib lost+found proc tmp var
它完全就是 Linux 的文件系统（与目标板启动后的文件系统完全一样）；
此时用户可以加入自己的应用程序；
下面举例说明将 hello 加入到文件系统：
cd /ramdisk
mkdir application (新建目录名可以自己定义)
cd application
mkdir hello
cd hello
cp /hello . (拷贝 hello 到 hello 目录下，此时生成的可执行文件 hello 在/目录下，用户可自己指定自己的应用程序路径)
cd /
umount /ramdisk
压缩新生成的 ramdisk－rmk7 文件系统映像文件：
gzip ramdisk-rmk7 ramdisk-rmk7.gz
生成 Linux 内核映像文件 ramdisk-rmk7.gz，然后烧写 ramdisk-rmk7.gz 到 Flash 中
启动目标板
#ls
application bin etc Linuxrc mnt sbin usr
dev lib lost+found proc tmp var
文件系统中出现了 application 目录
cd application

cd hello
在 hello 目录中出现了可执行文件 hello；
运行文件:
./hello

附录 C

制作文件系统

一、什么是 RamDisk

顾名思义，Ram：内存，Disk：磁盘，RamDisk 就是指使用你的一部分内存空间来模拟出一个硬盘分区。不过这在硬盘越来越便宜的今天好象并不是很有用，但现在内存也越来越便宜呀，对于一些我们想让其访问速度很高的情况下，还是可以试一试的。

二、创建一个 RamDisk

其实创建一个 RamDisk 是一件很简单的事，由于 RedHat Linux 6.0 在默认安装下就支持了 RamDisk，你只需要格式化一个 RamDisk，并将其 mount 到某一个目录上就可以了。

1、 查一下所有可用的 ramdisk：

```
ls -al /dev/ram*
```

这就会列出你现在可用的 ramdisk，这些 ramdisk 现在还不占用内存，除非你对它进行格式化。

2、 创建一个 RamDisk：

```
mke2fs /dev/ram0      格式化一个 ramdisk
```

执行该命令将出现以下提示

```
mke2fs 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
1024 inodes, 4096 blocks
204 blocks (4.98%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
1024 inodes per group
```

如果该操作执行失败，那么有可能是因为你的内核不支持 ramdisk，这时你只有重新配置、编译内核。在配置时将 CONFIG_BLK_DEV_RAM 设置为 Enable。

3、 将其 mount 上来使用：

mkdir /tmp/ramdisk0 新建一个目录

mount /dev/ram0 /tmp/ramdisk0 将其 mount 到刚才新建的目录上

这样，我们就可以使用这个新建的 RamDisk--/tmp/ramdisk0

另外，我们可以执行 df 命令来查看一下该 RamDisk 的大小：

```
>df -k /dev/ram0
Filesystem 1k-blocks Used Available Use% Mounted on
/dev/ram0 3963 13 3746 0% /tmp/ramdisk0
```

要注意的一点是：由于这个部分是内存，因此在系统重新启动的时候，将刷新这个区域。所以不要将任何没有拷贝的数据放在这个区域。如果你对这个目录进行了修改，并且需要保留这些修改，采取一些办法进行备份。也就是说，这个目录只能用于为了提高访问速度而暂时存储数据的地方。

三、创建大小合适的 RamDisk

上面我们创建了一个 RamDisk，但是无法指定大小，有时系统最大的只能创建最大为 4M RamDisk, 这时就要使用下面的方方法来创建合适的 RamDisk，现在我们来学习如何创建大小合适的 RamDisk。

1、 使用 LILO 配置：

我们在 lilo.conf 文件中加入一行：

ramdisk_size=10000 (如果是老版内核，则写 ramdisk=10000)

然后运行 /sbin/lilo 生成新的 LILO。

这样当你重新启动计算机之后，ramdisk 的默认大小将会是 10M。

2、使用/etc/conf.modules 配置：

如果你的内核是以模块的形式编译 ramdisk 的话，那么就可以在加载的时候决定 ramdisk 的大小。也就是可以通过修改/etc/conf.modules 的选项设置来做到，加入：

```
options rd rd_size=10000
```

当然也可以在命令行中指定参数给 ismod 来实现：

```
insmod rd rd_size=10000
```

当使用以上两种方法创建了合适大小的 ramdisk 时，我们还需要执行以下命令来使用它：

```
mke2fs /dev/ram0
mkdir /tmp/ramdisk0
mount /dev/ram0 /tmp/ramdisk0
```

而当不使用时，应执行：

```
umount /tmp/ramdisk0
```

若是使用 insmod 来加载模块的，应该再执行：

```
```

```
rmmod rd
```

下面是创建一个 20M 大小的 Ramdisk 的方法：

```
dd if=/dev/zero of=/dev/ram bs=1k count=20000
mke2fs -vm0 /dev/ram 20000
mkdir /mnt/ramisk
mount –t ext2 /dev/ram /mnt/ramdisk  这时就可操作 ramdisk 文件，在里面添加文件
cp –av filesystems /mnt/ramdisk
umount /mnt/ramdisk
dd if=/dev/ram bs=1k count=20000 | gzip -v9 > /tmp/ ramdisk.gz
```

四、使用 RamDisk 的一个实例

下面，我们通过一个使用 Ramdisk 做 WEB 服务器的实例来说明其应用。

1、 首先将 WEB 服务器移到另外一个地方

```
mv /home/httpd/ /home/httpd_real
mkdir /home/httpd
mkdir /home/httpd/cgi-bin
mkdir /home/httpd/html
mkdir /home/httpd/icons
```

2、 将以下命令加入到/etc/rc.d/init.d/httpd.init 中去：

```
/sbin/mkfs -t ext2 /dev/ram0
/sbin/mkfs -t ext2 /dev/ram1
/sbin/mkfs -t ext2 /dev/ram2
mount /dev/ram0 /home/httpd/cgi-bin
mount /dev/ram1 /home/httpd/icons
mount /dev/ram2 /home/httpd/html
tar -C /home/httpd_real -c . | tar -C /home/httpd -x
```

这样就可以了，但是请记住，你更新数据时，应更新 httpd_real 目录，而不要更新 httpd 目录，否则在系统重新启动时，所有的更新都将丢失。你最好设置一个 cron 进程，让其监视 httpd_real 是否有改变，一有改变就将其复制到 ramdisk 中去。

五、使用 RamDisk 做/tmp 目录

我还想推荐一个更酷的方法，如果你的内存太多，那么可以将其中一部分做为/tmp 目录，这样将大大提高你的系统执行速度。而且，/tmp 将会在系统重新启动时被删除，多么惬意呀。

附录 D

使用 MiniGUI 的一些 FAQ

一．MiniGUI 配置使用 FAQ

下面是找到的关于 miniGUI 的一些 hints，应该覆盖了 miniGUI 使用中的很多问题，不过，个人觉得，在移植 miniGUI 至某个特殊平台时，关于系统的经验还是很重要的，甚至超过下面这些小技巧的重要性；另外，仔细阅读文档也是相当重要的一环，仔细读文档能帮助你避免很多麻烦。

1．Framebuffer 支持：使用 miniGUI 时，需要系统启动 framebuffer 支持，在 lilo.conf 中添加一行：

vga=0x0317

在启动时进入 framebuffer 模式，也可用 fbset 设置；

2．如果启动时提示 init 错误，可以考虑给 lilo 传递 init 参数：

init=/bin/sh rw

3.如果 miniGUI 启动错误，考虑可能是以下原因：

A)没有启动 framebuffer，即 lilo.conf 中的 vga=XXXX，显示模式参照表如下：

640x480 800x600 1024x768 1280x1024

256 0x3010x3030x3050x307

32k 0x3100x3130x3160x319

64k 0x3110x3140x3170x31A

16M 0x3120x3150x3180x31B

B)启动了 framebuffer，但无 fb 设备文件，可以自己创建/dev/fb0 文件：

mknod fb0 c 29 0

C)miniGUI 需要/var/tmp 目录，如果不存在也会导致错误；

4. MiniGUI 提供抓屏，用 PrintScr 键，抓到的图片为 tmp 模式，存在当前目录下。

5.有些时候，在试了很多方法之后设备仍然不能使用，可以考虑换个知名品牌的硬件试试，会有惊喜。

6．MiniGUI 需要 Unix Domain socket 和 System V IPC 支持，内核里需要这两个。

7． 启动时的输入条可以用 ctrl 键关闭/ 启动，不过可能响应比较慢，要多摁几下；另外，也可以通过配置 MiniGUI.cfg 文件里的 imenumber 选项：

[imeinfo]

imenumber=2

将 imenumber＝2 改为 imenumber＝0，启动时就不会出现输入条了；

8．iniGUI 对串口鼠标的支持：

MiniGUI 的 Native IAL engine 目前不支持串口鼠标。

如果要使用鼠标，可以通过 gpm 程序：

1．运行 gpm -k 命令杀掉正在运行的 gpm。

2．运行 mouse-test 命令确认自己的鼠标设备和协议。

3．运行 gpm，指定鼠标设备和协议：

gpm -R -t <yourmousetype> -m <yourmousedevice>

4．编辑 /etc/MiniGUI.cfg 文件，将 mtype 设置为 gpm。

将 mdev 设置为 /dev/gpmdata。

重新启动就 OK 了。

注意：用 gpm 设置鼠标格式的时候，可以使用-R参数，gpm 的 -R 参数是指定将原有鼠标协议转换成哪种鼠标协议，并 出现在 /dev/gpmdata 文件上的。

gpm -R -t ms -m /dev/psaux
ms3 这种鼠标协议目前的 ial 引擎中 gpm 类型的设备所不支持的。
9. Framebuffer 选项需要在内核中打开试验选项，即第一项：
Code maturity level options—》Prompt for development and/or incomplete code/drivers
10.popt.h 文件的用处：
这东西专门用于命令行解析，如果你不准备使用 miniGUI 的虚拟控制终端，大可以将其扔掉。还有不少交叉编译环境都不支持这玩意，小心点哦。 如果你要在目标版上支持控制终端，内核得有 pty 与 pts 支持
14.
现象：我用的是 EP7212 的板子，BlueCat 的开发平台，下载了三个资源文件，libminiGUI1.1.0pre7,mde-0.2.7，资源文件安装了，然后解开 libminiGUI1.1.0，运行./autogen.sh，再运行./configure,再运行 make，出现这么一个错误：
Making all in server
make[3]: Entering directory
'/BlueCat/demo.clep7212/developer/src/minigui/libminigui-1.1.0/src/server'
/bin/sh ../../libtool --moe=complile gcc -DHAVE_CONFIG_H -I. -I. -I../.. -I../..
/include -I../include -g -02 -D__MINIGUI_LIB__ -02 -Wall -Wstrict-prototypes
-pipe -DNDEBUG -c server.c
gcc -DHAVE_CONFIG_H -I. -I. -I../.. -I../../include -I../include -g -02
-D__MINIGUI_LIB__ -02 -Wall -Wstrict-prototypes -pipe -DNDEBUG
-Wp,-MD,.deps/server.pp -c server.c -o server.o
../../gcc/pexecute.c:245: Internal compiler error in function pexecute
make[3]: ***[server.lo] Error 1
make[3]: Leaving directory
'/BlueCat/demo.clep7212/developer/src/minigui/libminigui-1.1.0/src'
make[1]: *** [all-recursive] Error1
解决方法：
I just wanted to let you know that the latest CVS does not build
without changes on Cygwin.
The problem is that the gcc distributed with Cygwin 20.1 barfs
on the -pipe option with an internal compiler error:
make all-recursive
Making all in data
Making all in default
make[3]: Nothing to be done for `all'.
Making all in civ1
make[3]: Nothing to be done for `all'.
Making all in classic
make[3]: Nothing to be done for `all'.
Making all in common
gcc -DHAVE_CONFIG_H -I. -I. -I.. -g -O2 -Wall -pipe -c capability.c
/home/noer/src/b20/comp-tools/devo/gcc/pexecute.c:245: Internal compiler
error in function pexecute
make[2]: *** [capability.o] Error 33

make[1]: *** [all-recursive] Error 1
make: *** [all-recursive-am] Error 2
/usr/local/src/freeciv/freeciv>
I fixed it by patching configure.in to no longer use the -pipe option.
I'm going to be reporting this problem to the Cygwin list too.
My patch to configure.in is:

```
--- configure.in        Sun Feb 28 10:52:22 1999
+++ configure.in.orig   Sun Feb 28 10:51:40 1999
@@ -104,8 +104,7 @@
 dnl  AC_PROG_MAKE_SET

   if test -n "$GCC"; then
-    dnl CFLAGS="$CFLAGS -Wall -pipe"
-    CFLAGS="$CFLAGS -Wall"
+   CFLAGS="$CFLAGS -Wall -pipe"
     fi

  if test "$CVS_DEPS" = "maybe"; then
```

Thanks,

Todd

11.MiniGUI 移植到 EP7211 的问题：

我最近几天一直忙着移植 MiniGUI 到 EP7212 板上，碰到不少问题，在此一次列出，请大家指点：

1).Cygwin 下编译 pexecute.c 错误问题，去掉 autogen.sh 产生的 configure.in 中的-pipe；

2).Libminigui-1.1.0 编译根据舵主手谕：./configure --disable-newgal --enable-fblin4 --enable-ep7211ial 。编译 ok,但我编译的 libminigui 都是静态库。我在 redhat 上同样编译却是动态连接库，奇怪？

3).库文件找不到，Makefile 中手工加-lm；，编译后文件都 2M 多，如何编译成动态库？

4).编译 mde-0.2.7，bomb,same 等程序出错，好像调用 newgdi/下的 attr.c 中的函数出错，为什么调 newgdi 呢？我编译库时指定了--disable-newgal，难道 mde-0.2.7 不能运行在 ep7212?

不管三七二十一，删掉 Makefile 里的 subdirs 相关目录，先编译过了再说。

5).仔细看发现 InitEP7211Input（）函数里打开设备/dev/ts，还有一个文件/etc/ts.conf 难道设备不用 MiniGUI.CFG 里指定的？ts.conf 那来的？mtype=应该是什么？

6).strip mginit 后，变成 600 多 k，勉强倒到板上，运行出错，凭我的想象，修改 /etc/MiniGUI.cfg:

```
gal_engine=native
ial_engine=native
mdev=/dev/tpanel     #我用 mknod -m 644 /dev/tpanel   c 10 12 产生的
mtype=PS2            #我不清楚 touchpanel 改用什么，索性不改
defaultmode=320x240x4
```

运行，出现错误：

Error in step 8: Can not initialize low level event!
InitGUI failure when using /etc/MiniGUI.cfg as cfg file.
看来时 ial 出错，查来查去好像 init-lite.c 文件调用引起，难道编译了 lite 版本？EP7212 上能用 lite 版本么？
看来你还是做了许多研究，但没有找到主要问题。
MiniGUI 中给出了针对 EP7211 的 IAL 引擎，但并没有针对 EP7212 的 IAL 引擎。
如果两者不同，你需要自己编写一个引擎，就像 EP7211 的引擎一样，并且要在 /etc/MiniGUI.cfg 文件中指定 ial_engine=ep7212，而不是 native。
native 的 ial 引擎是用于 PC 的，操作的是键盘和鼠标，你用在你的 EP7212 板子上能行吗？
mdev 和 mtype 是给 IAL 引擎使用的一些可配置参数。某些 IAL 引擎（比如 EP7211）根本就没有可配置的选项，所以程序不用这些配置参数，直接将设备名称硬编码到代码当中了，所以，是否设置 mdev 和 mtype，这无关紧要。
defaultmode 是用于新的 GAL 引擎的，你使用旧的 GAL 引擎，没有任何作用。
对于你这种情况，建议如下：
1). 在配置 MiniGUI 的时候，打开 Dummy ial 引擎，即使用如下的参数：
--enable-dummyial
其他的参数和你以前使用的一样。既然 EP7211 的引擎和你 EP7212 的引擎不一样，就不需要使用这个引擎了。Dummy IAL 引擎是个什么工作都不做的引擎。使用这个引擎你可以首先将 MiniGUI 在目标板上运行起来，然后再进一步参照其他的引擎编写适合自己的引擎。
2). 在 /etc/MiniGUI.cfg 中，指定如下设置：
gal_engine=native
ial_engine=dummy
其他参数可以忽略。
至于无法生成动态库的问题，可能和你的编译环境设置有关，我就帮不上什么忙了。
12.newgal 和 gal 的区别：
newgal 只支持 8 位色以上的线性显示模式，功能强大，
而低于 8 位色的那些模式（比如 16 色及低于 16 色的灰度等等）不被支持。
13.设置 ial 引擎的问题：
大家新年号！请问：
在 MiniGUI.cfg 文件中，如设置
ial_engine=Dummy
则 mdev，mtype 如何设置？
设置 ial_engine=Dummy 后，演示程序 mginit 能否运行？
那两个参数随便取值即可。
在使用 Dummy 引擎之后，MiniGUI 是可以正常运行的，
只是你无法控制程序而已，因为没有任何的输入。
14.系统时间对 miniGUI 也会有影响。譬如，如果系统时间不对，生成的文件时间比源文件旧，会导致 make 进行 clock skew
15.about 鼠标的问题：
现在我的 pc104 平台上已经显示出 bomb 的画面，真令人激动。现在还有两个小问题想请教
1。bomb 只在屏幕上显示了一部分，功能正常，** 但我无法修改图形模式**，
我的显卡（C&T 65535 1M）可以用 SVGALib 的测试程序，最高可设为 1024x768x256

我在 MiniGUI.cfg 中设置了
[SVGALib]
defaultmode=1024x768x256(1024x768x256 或 1024x768-256 ,1024x768x-8bpp，
640x480-256)
从 bomb 退出后均提示无效的图形模式。
若设成：defaultmode=G1024x768x256（或 G640x480x256） 则黑屏
我看了网站上的文档，好象没有具体的说明, 不知如何设 置
2。我看网站上有贴是关于串行口鼠标支持的一段修改的程序（用 ms3）, 我下载后编译成功，但鼠标无法驱动。是否此修改不能在 SVGALib 上用？
第一个问题：
正确的设置应该是
[SVGALib]
defaultmode=G1024x768x256
如果是黑屏，可能是驱动不支持的原因，也许要修改 /etc/vga/libvga.conf 中的设置。
第二个问题：
那段程序是用来支持 native IAL 引擎的，SVGALib 的鼠标支持，需要修改 /etc/vga/libvga.conf 文件。
将 mousetype 设置为 Microsoft，并正确设定鼠标设备文件。
我现在用 PS2 鼠标，显卡正常的设置应该是 G320x200x256 ,我看了一下文件 src/gal/svgalib.c 似乎在模式非法时缺省返回 G320x200x256,然后我将 MiniGUI.cfg 的模式也改成这个后，可以正常。改成 G640x480x256 ,或 G1024x768x256 都会黑屏，但这两种模式我安装好 SVGALib-1.4.0 后运行其自带的 vgatest 测试程序都可以正常显示。不知 SVGALib 图形引擎支持的模式，是否 minigui-1.1.0 都支持?黑屏跟其他设置有关吗，比如鼠标，键盘？
另外，在执行./configure ----disable-newgal --disable-nativegal --enable-svgalib 后 似乎仍然没有将 SVGALib 编译进去，我看了以下生成的文件 config.h,有一行
/*undef _SVGALIB*/
必须还要加上--disable-lite 生成的 config.h 里就有了#define _SVGALIB 1,不知是否是这样？
应该和鼠标键盘没有关系。
至于配置选项，的确应该加上 --disable-lite，因为 MiniGUI-Lite 版本不能在 SVGALIB 上运行。所以，我估计应该是这个问题造成的。
16.
Q:如何安装, 配置及运行 minigui?
A:请参考 kongming 老大写的文档:MiniGUI 编译、安装和配置,
http://www.minigui.org/ibmdocs/minigui-2/index.html
Q:运行 mginit 死机后, 重起怎么不能再运行 mginit?
A:请删除/var/tmp/下的 mginit 和 minigui,再重新运行.
Q:运行 mginit 后, 鼠标乱动怎么办?
A:修改/usr/local/etc/Minigui.cfg 中的 mtype 为你机子符合的类型.
一般来说, 首先要保证进入 minigui 前, 你的 linux 下能有正常的鼠标, 这是可以将 mtype 改为
PS2,这样问题基本上能解决. 记得如果失效后, 最好重起
机子, 这样后面的配置才不会受前面的影响的.

Q:我编写一个应用程序, 编译时出现了很多 undefined reference 的信息, 怎么回事?
A:那是因为 Makefile 中没有添加-lminigui 或者-lmgext 来引用 minigui 的库.
Q:我编写一个应用程序, 编译通过, 怎么不能运行?
A:必须先运行 mginit,然后可以在那里的虚拟控制台来运行应用程序.
17.miniGUI 的裁减
裁剪 MiniGUI 的三个步骤：
1). 在运行 ./configure 时进行定制，取消某些不需要的功能。
2). 修改 /etc/MiniGUI.cfg，删除某些不需要的字体文件。比如，
如果你只使用 GB2312、12 点阵的字体，就可以只保留系统字体，
而删除其他的字体。当然了，删除的方法是修改 fontnumber 键
的值，而不是删除这个段。对应的那些字体文件，就可以从系统
中删除了。
3). 那三个资源包中实际有用的东西，都列在了 /etc/MiniGUI.cfg 文件中。
如果哪个文件没有出现在 /etc/MiniGUI.cfg 中，就删除这个文件。
这样，一般可以将 MiniGUI 及其所使用的文件缩小到 2M 以内，有的情况
下可以缩小到 1M 以内。
移植时，光修改配置文件是不够的。 默认情况下，许多针对具体嵌入式系统的 IAL 引擎并没有编译到 MiniGUI 函数库 当中。你要在运行 ./configure 脚本的时候指定所需要的引擎。 具体方法，可参阅 http://www.minigui.org/cdoc.shtml 中有关配置的文章。 也可以参阅源代码当中的 ./buildlib-* 文件。
如果你使用的是 MiniGUI 1.1.0PreX 版本，请使用老 GAL 引擎来提供
对 4bpp 灰度 LCD 的支持。也就是在配置时使用
--disable-newgal
选项。新的 GAL 引擎，不支持低于 8 bpp 的演示模式。
另外，你还需要在配置时指定
--enable-fblin4
--enable-ep7211ial
等选项。
18.编译时的-m286 和-m486 选项的作用和意义：
from gcc-HowTo 中文版:
这之中最重要的有是-m386 和-m486 这两种，用来告诉 gcc 该把正在编译的程序代码视作专为 386 或是 486 机器所写的。不论是用哪一种-m来编译程序代码，都可以在彼此的机器上执行，-m486 编译出来的码会比较大，不过拿来在 386 的机器上跑也不会比较慢就是了。
目前尚无-mpentium 或是-m586 的旗号。Linus 建议我们可以用-m486 -malign-loops=2 -malign-jumps=2-malign-functions=2 来得到最佳编码的 486 程序代码，这样做正好就可以避免 alignment（Pentium 并不需要）有过大的 gaps 发生。
19.如果 mginit 不能正常启动的话，可以考虑减少 mginit.rc 的 nr 值，譬如修改为 6，这个文件是 mginit 的配置文件。
二．编程 FAQ
1.函数：
1.触发窗口重画的消息，类似 VC 的 MSG_PAINT.
InvalidateRect(hWnd,NULL,TRUE);
2.CreateWindow("控件名称","Caption",风格，ID,x,y,长，宽，hwnd,0);
3.获取和设置对话框中控件的值：

GetDlgItemText(hDlg,nIDDlgItem,String,nMaxCount) ----将控件 Edit 的值传给 String；

SetDlgItemText(hDlg,nIDDlgItem,String)-----设置控件 Edit 的值为 String。

4.菜单条显灰：

UINT EnableMenuItem(hmnu,item,flag);

flag 为 TRUE 时则显灰，返回菜单原来的状态。

5.限制编辑框输入的字符长度：

在生成编辑框之后

SendMessage(hEdit,EM_LIMITTEXT,10,0)-----10 即为编辑框输入的字符长度。

6. 装载图标：LoadIconFromFile(HDC_SCREEN,"图标路径",0)

7.若给控件 button 加上 BS_DEFPUSHBUTTON 属性，则按下回车键就默认为单击了该 button。

2.编译技巧

1.若你的应用程序编译能通过，但是运行时无法显示窗口，可尝试在主程序中加如下语句：

#Ifdef_LITE_VERSION

SetdesktopRect(0,0,640,480)

#endif

若想在屏幕上显示更大的窗口，可重新设置(640，480)参数，否则窗口最大就只有那么大.

2.创建对话框模板时，有一项参数为对话框内控件的数目，必须与所创建的控件数目相等，否则运行时无法 显示所有控件。

3.若用到了"打开文件"对话框，编译时要加入：-lmywins -lmgext。

以及要包含两个头文件：minigui/mywindows.h minigui/filedlg.h

4.结束对话框 EndDialog 会自动销毁 Dialog 中的所有控件，而无须手工一项项删除。

3.Minigui 源码解读：

1.mginit.rc 中的 nr 参数为在控制栏中显示几个应用程序的图标。

2.mginit.rc 中的参数 autostart = n 表示启动 minigui 时, 第 n 个应用程序便会自动启动。

3.改变 Minigui.cfg 中的 imenumber ＝ 1 则启动时会不出现输入法状态条。

4.mginit.c 中第 283 行：

if(SetDesktopRect(0,g_rcscr.bottom- HEIGHT_TASKBAR- HEIGHT_IMEWIN,

g_rcscr.right,g_rcscr.bottom) == 0)

去掉 HEIGHT_IMEWIN(即不减去该项)，即可消掉输入法所占的一片空白。

5.用一个"土办法"实现菜单快捷键控制:

1)在 menu.c 中，PopupmenuTrackProc()是处理菜单被击活后的按键消息，如上、下、左、右移等。 TrackMenuBar 是击活菜单，被 PopupMenuTrackProc()函数的按键或鼠标消息调用（主要是向左或右移 动弹出另一个菜单）故肯定有一个地方专门用来处理菜单未被击活时的键盘消息，以击活菜单之后再调 用 PopupMenuTrackProc()进行处理。

desktop.comm.c 的 deskTrackPopupMenu 函数即为此。

2)在 desktop_comm.c 中第 1772 行：更改 MSG_SYSKEYDOWN 为系统默认的菜单弹出键（可设多个）。

3)修改/src/gui/window.c 第 1096 行，其调用 TrackMenuBar 的代码，使定义键与菜单位置一一对应起来。

（TrackMenuBar 中的位置参数可用 0,1,2 等表示。）

4.小例子：

1.装载一幅位图，可如下：

```
hdc=BeginPaint(hwnd);
LoadBitmap(hdc,&bitmap,"位图路径");
Fillboxwithbitmap(hdc,x,y,bitmap.bmwidth,bitmap.bmHeight,&bitmap);
UnloadBitmap(&bitmap);
EndPaint(hwnd,hdc);
```

2.编辑框的子类化：

子类化编辑框步骤如例子所示，但如果是一个应用窗口中弹出的对话框来进行控件子类化的话，应注意以下两点:

1)创建对话框中的控件可不用在 CTRLDATA CtrlInitProgress 中完成，可在对话框消息处理函数中响应 MSG_CREATE 消息，用 CreateWindow 的方法生成。

2)不能如此生成所有控件，还必须在 CtrlInitProgress 中生成至少一个以上控件。（即该结构不能是空的）

3.制作闪屏：

1)要另外写一个 Main 函数，做为程序的入口；

2)该 MiniGUIMain 中，要加上如下代码来装载图片层:

```
#ifdef _LITE_VERSION
 if(JoinLayer("sp","sp",0,0,1024,768) == INV_LAYER_HANDLE)
   {
fprintf(stderr,"JoinLayer:invalid layerhandle \n");
exit(1);
}
 #endif
```

3)原来的主函数要更名；

4)在原主函数中的：

```
#ifdef _LITE_VERSION
 SetDesktopRect(0,0,800,600);
 #endif
```

要屏蔽掉，否则闪屏后不能退出图片层，而是会另开辟一个层给应用程序，导致程序退出后无法关闭图片层，下次无法运行。

5.桌面程序的层：

今天总算把 MiniGUI 给装上了，真爽呀！但是为什么不管运行什么程序都是在屏幕右边三分之二处有一条线把屏幕分成二部分了？

MiniGUI-Lite 原本是每个客户进程在自己的层中运行的。

后来可以让许多进程在同一个层中运行，但不能互相重叠。

那个线就是表示一个客户在一个层中所占有的矩形区域。

kongming 大哥，你的意思是不是说，除了不能互相重叠之外，是可以在桌面上同时看到两个矩形区域，分别有不同的客户程序在运行呢？可是在 mde 中除了能看到一个客户程序和桌纸、任务栏外，即便还运行着别的客户程序也是看不到的呀。

如果要做一个比较完善的窗口管理器的话，是不是可以做到“让许多进程在同一个层中运行，并能互相重叠”的效果呢，是不是还可以象 windows 下那样随意移动窗口的位置和改变窗口的大小呢？

我是新手，刚刚把 MiniGUI 安装好，还不懂其编程呢，所以问题很幼稚，请大侠们不要见笑，不吝赐教哟：）

那需要在运行程序时添加一个特殊的选项，

比如：

./fminigui -layer vcongui

指定它加入某个层，否则会新建一个层。

其实，在许多嵌入式设备当中，有这种层，以及进程内的多窗口完全支持，就足够了。

6.在 Embedded Planet 上安装 MiniGUI.1.1.0 的注意事项：

1. 修改 helio.c 文件

#define _HELIO_BUTTON 1

改为 #undef _HELIO_BUTTON

2. 修改 configure 文件

CC = “°$ac_cv_prog_c”1

改为 CC = ppc_8xx-gcc

3. 修改 config.h 文件

#undef _EXT_CTRL_TREEVIEW

改为 #define _EXT_CTRL_TREEVIEW

4. 修改 tools/目录下的 Makefile 文件：

LIBS = -lminigui

改为：LIBS = -lminigui –lm

5. taskbar.c 中去掉

hlMEwnd = lMEWndowEx……¡-

6. 配置使用 build_helio

7. 头文件、四个库文件、资源文件以及配置文件 MiniGUI.cfg 都要复制到嵌入式系统的对应目录中。

8. 根据 mginit 目录中 minigui.rc 文件的要求将 mginit 及其它 demo 复制到相应目录下。

7.如何在启动的任务栏中添加自己的程序：

我们在 redhat7.2 上编译好 minigui1.1.x 后，准备学习在该环境下编译调试 minigui 应用程序。我们以源程序中附带的 hello world 程序为例。该程序要求画一个窗口并在其中显示 Hello world。

于是大家摸索如何在 mginit 中添加一个图标，并使该图标的点击启动 hello world 程序。基本做法如下：

1、在 mde0.3.0 中添加一个 hello 文件包，并将 hello world 源程序复制到其中。

2、将 notebook 中的 Makefile 等三个文件以及 res 包复制到 hello 中。

3、分别修改三个 make 文件，将其中的 notebook 字样全部替换为 hello。

4、用 linux 下的任何图像编辑工具设计一个图标存为 hello.png 和 hello.gif，存到 mginit 包下的 res 包中。

5、修改 mginit 包下的 mginit.rc 文件：将其中的 nr＝10，改为 nr＝11；

复制其中的一组程序段到最后，段头改为 apply10；其中对应的字样也改为 hello

6、修改 mginit 包下的三个 make 文件，其中有一个地方列出了很多应用程序可执行文件名，在其中添加 hello。

至此，基本修改结束。执行 make 命令。成功后，运行 mginit，则应能看到 mginit 窗口下多了一个图标，那就是你设计的。点击之，应该能看到 hello world 程序运行结果。

不一定描述准确，仅供参考。

8. miniGUI 应用程序的问题：

各位兄弟：

大家好! 我做了一个应用，使板子一上电 Linux 引导之后就自动运行 MiniGUI 服务器进程和 MiniGUI 应用程序，但发现服务器进程 mginit 每次均能正常运行，而应用程序时好时坏(有时能起来，有时却起不来)。我估计这种现象可能是服务器与应用程序之间的通讯造成的(即服务器还没有准备好，而应用程序却开始运行了)。但发现在 mginit&之后加延时再跑应用程序也存在着这个问题。下面是我写的简单服务器进程 mginit.c 和应用程序 test.c,可能存在着桌面、层、通讯方面的问题与不足，请大家不吝赐教，谢谢!

显示：单色 LCD(320*240)

一、服务器进程 mginit.c

```
 int DisplayImeProc(HWND hWnd, int nMsg, WPARAM wParam, LPARAM lParam);
int MiniGUIMain (int args, const char* arg[])
{
 MSG Msg;
 MAINWINCREATE CreateInfo;
 HWND hMainWnd;
if (!ServerStartup ())
 {
    fprintf(stderr, "Can not start mginit.\n");
    return 1;
 }
 if(SetDesktopRect (0, 207, 320, 240)==0)
 {
   fprintf(stderr, "Can not get the required desktop rectangle.\n");
   return 2;
 }
 if(!InitMiniGUIExt())
 {
     fprintf(stderr, "Can not init mgext library.\n");
    return 3;
 }
CreateInfo.dwstyle =  WS_BORDER;
 CreateInfo.dwExstyle = WS_EX_NONE;
 CreateInfo.hMenu =(HMENU)NULL;
 CreateInfo.hCursor =(HCURSOR)(IDC_ARROW);
 CreateInfo.hIcon = GetSmallSystemIcon (IDI_APPLICATION);
 CreateInfo.lx=0;
 CreateInfo.ty=207;
 CreateInfo.rx=320;
 CreateInfo.by=240;
 CreateInfo.hMenu=(HMENU)NULL;
```

```
 CreateInfo.MainWindowProc=DisplayImeProc;
 CreateInfo.iBkColor=COLOR_lightwhite;
 CreateInfo.dwAddData=0;
 CreateInfo.hHosting=HWND_DESKTOP;
hMainWnd=CreateMainWindow(&CreateInfo);
   if(hMainWnd==HWND_INVALID)
 {
     fprintf(stderr,"Main Window invalid!\n");
      return -1;
 }
 ShowWindow(hMainWnd,SW_HIDE);
 while(GetMessage(&Msg,hMainWnd))
 {
  DispatchMessage(&Msg);
 }
 MainWindowThreadCleanup(hMainWnd);
   return 0;
}
int DisplayImeProc(HWND hWnd, int nMsg, WPARAM wParam, LPARAM lParam)
{
switch (nMsg)
{
   case MSG_CREATE:
    {
IMEWindow(hWnd);
return 0;
    }
}
return DefaultMainWinProc(hWnd, nMsg, wParam, lParam);
}
```

二、应用程序 test.c

```
int MiniGUIMain (int args, const char* arg[])
{
 MSG Msg;
 MAINWINCREATE       CreateInfo;
 HWND hMainWnd;
 HHOOK hhook;
SetDesktopRect (0, 0, 320, 207);
 CreateInfo.dwstyle =  WS_BORDER;
 CreateInfo.dwExstyle = WS_EX_NONE;
 CreateInfo.hMenu =(HMENU)NULL;
 CreateInfo.hCursor = GetSystemCursor (IDC_ARROW);
 CreateInfo.hIcon = GetSmallSystemIcon (IDI_APPLICATION);
 CreateInfo.MainWindowProc = MainWinProc;
```

```
 CreateInfo.lx = 0;
 CreateInfo.ty = 0;
 CreateInfo.rx = 320;
 CreateInfo.by = 207;
 CreateInfo.iBkColor=COLOR_lightwhite;
 CreateInfo.dwAddData = 0;
 CreateInfo.hHosting = HWND_DESKTOP;
if(!InitMiniGUIExt())
 {
   return 1;
 }
hMainWnd = CreateMainWindow (&CreateInfo);
if (hMainWnd == HWND_INVALID)
     return 2;
ShowWindow (hMainWnd, SW_SHOWNORMAL);
 hhook = RegisterKeyMsgHook(hMainWnd, AllKeyboardProc);
 while( GetMessage (&Msg, hMainWnd))
 {
 TranslateMessage(&Msg);
    DispatchMessage (&Msg);
 }
UnregisterHook (hhook);
 MainWindowThreadCleanup (hMainWnd);
return 0;
}
```

三、存在的问题：

1.mginit 进程与应用程序进程 test 存在着通讯上的问题(即服务器进程能正常工作，而 应用程序有时能起来，有时却起不来) 。如何解决？

具体实现，在/etc/rc.sysinit 文件中加入：

```
insmod ...    //加入一些驱动，如 DOC、键盘。
mginit &
./test &
```

2.在应用程序中的编辑框中，按左右方向键不能在编辑框中的字符间移动，但左右方向键在 MiniGUI 上能通过按键消息得到。是否与 Caret 的设置有关？

3.服务器与应用程序中相应的配置是否还存在着问题？ (主要是桌面、层方面)

1. 如果是你怀疑的那个愿意，你可以试着在 mginit 当中 fork 然后执行 test 程序。就像 MDE 中的 mginit 那样。

2. 是在你的目标板上不正常吗？

3. 似乎没有什么问题。

1.为了使 mginit 快速启动，我把 mginit 和运行库放到了 FLASH 中了，而应用程序、字库、输 入法放在 DOC 中。但大多数情况下，应用程序都能起来。

2.板子好像没有问题，但可能驱动大多造成了服务器与应用间的通讯。我有 DOC、AD、显示、键盘等驱动。

至于光标，跑例子程序中的 notebook，左右方向键仍然不能在字符间移动。

Kong Ming:
您好!
问题已经解决，原来是我在 mginit 中处理 MSG_PAINT 消息后使用了 return 0;语句，而应使用 break;结束, 从而引起了 mginit 死机! 还有光标问题基本上已得到解决，原来我在键盘驱动中处理左右方向键时，扫描码没有按照 MiniGUI 的要求做。修改之后，光标已能在 Edit 框中左右移动了，但在 ComboBox 框中设置了焦点，仍不出现闪烁的竖线光标，请问是什么原因？
请问你是怎么设置 ComboBox 控件的焦点的？
使用 SetFocus(hComboOperator);其中，hComboOperator 是所创建组合框的窗口柄。谢谢!
似乎不应该出问题啊。你用的是什么版本的 MiniGUI？检查一下 src/control/combobox.c 中 SETFOCUS 消息的处理。
我用的是 MiniGUI1.10Pre9,在这个版本中怎么没有 SETFOCUS 的消息处理呢？
那看来是版本太老的原因，建议你还是升级到最新的 1.2.1 吧。
9.其它：
1.MSG_COMMAND 消息的 wparam 参数指定的是该消息来自哪个控件或菜单项。
2.主函数（main）有主函数的消息处理函数，
对话框有对话框自己的消息处理函数（DialogBoxProc）。
3.响应键盘的消息参数：
wParam:保存虚拟键码；
lParam:是 Ctrl、Alt 和 Shift 键按下或锁定的状态。