操作系统实验文档

XV6 GUI 实现

2012013311 张　凯

2012013292 邹豪风

2012013289 乔春雨

2012013299 傅展升

目录

1 概述 2
2 系统环境 2
操作系统 2
编辑器 2
3 系统设计 2
3.1 综述 2
3.2 架构设计 3
3.3 实现功能 3
应用层 3
接口层 3
内核层 4
4 具体实现 4
4.1 图形模式的切换 4
4.2 窗口队列 4
4.3 图形绘制 5
4.4 文件系统 5
4.5 位图支持 6
4.6 鼠标驱动 7
初始化工作 7
数据读取 7
事件处理 8
4.7 键盘驱动 8
4.8 文字支持 8
5 用户进程 9
5.1 桌面 9
5.2 文件浏览器 9
5.3 图片查看器 10
5.4 文本编辑器 10
6 GUI 数据结构 11
7 团队分工 13

1 概述

本次实验基于 xv6 系统实现 GUI，使用了 git://pdos.csail.mit.edu/xv6/xv6.git 发布的最新 xv6 内核。在实验中我们改进了文件系统，实现了图形界面，鼠标驱动以及键盘驱动。

项目管地地址为 https://github.com/nezharen/xv6。

2 系统环境

操作系统

Ubuntu 14.04 LTS 64-bit

Linux Mint 17.1 Cinnamon 64-bit

Kubuntu 14.04 LTS 64-bit

编辑器

Sublime Text 2

vim

gedit

3 系统设计

3.1 综述

我们仔细研究了上一届优秀 GUI 作业的代码和架构，虽然在效果上有较高的仿真度和观赏性，但仍有很多不足，如：不应将图片直接写入内核、窗口队列管理有优化空间、文件系统不完善、以及窗口全局绘制等。[1]

基于以上考虑，在图片读取上，我们使用了位图读取技术，使用系统调用将图片读入内存；而对于字符点阵的处理，由于在文本编辑器中要频繁调用，我们选择将其写入内核中，以提高效率；在文件系统方面，我们进行了适当扩展，实现了文件的外存存储；窗口绘制使用了局部刷新。[2]

在 GUI 函数库设计上，我们分离了应用程序接口和内核接口，使得逻辑清晰，调用方便，并在鼠标和键盘驱动的支持下，共同实现了图片浏览器、文本编辑器和文件系统资源管理器。

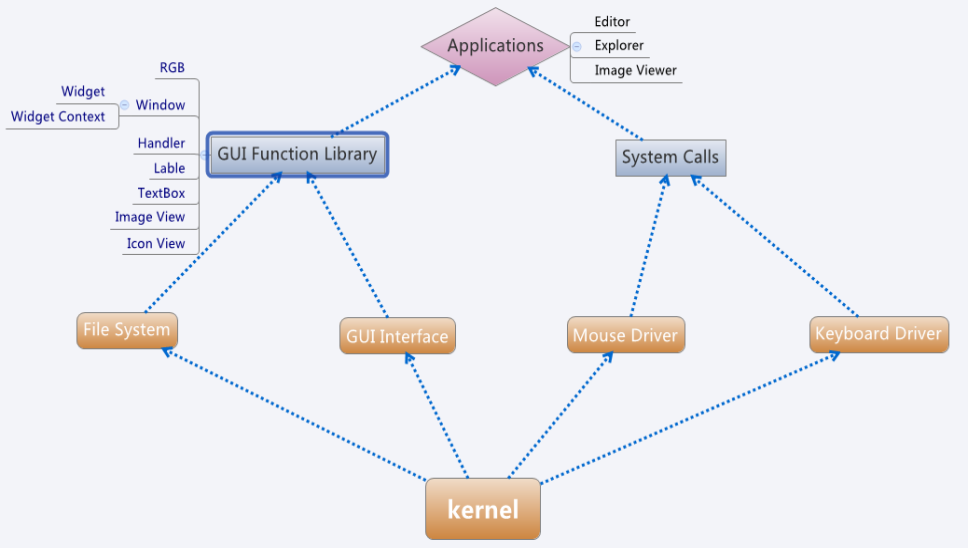
---

[1] 实际上，在新版的 xv6 中，由于内存限制上述方法已经不可行。

[2] 另从效率考虑，我们只使用单核 CPU

3.2 架构设计

逻辑架构设计使用了自顶向下的结构，分为应用层、接口层和内核层。逻辑设计架构图如下所示：



架构设计图

3.3 实现功能概览

3.3.1 应用层

【文本编辑器】

用户进程 editor 实现了简单的文本编辑器，支持文件的读取、修改、存储、新建。

【资源管理器】

用户进程 explorer 实现了文件系统的管理，支持新建文件夹和文本文件。

【图片浏览器】

用户进程 imageViewer 实现了图片浏览功能。

3.3.2 接口层

【GUI 函数库】

为了使用户进程更方便地对界面进行绘制，我们需要准备一个功能全面的 GUI 函数库供进程调用。我们的 GUI 函数库中提供了多种 GUI 组件，包括窗口、颜色值、处理句柄、标签、编辑框、图片浏览、图标浏览。值得注意的是，我们将窗口的数据结构分为用户态和内核态两部分以区别使用[3]。

[3] 实际上在内核态和用户态上使用的数据结构要求一致，区分是为了逻辑清晰，详见 window.h 和 uwindow.h。

【系统调用】

系统调用包括了鼠标驱动、键盘驱动、文件存取以及图像绘制。

3.3.3 内核层

【文件系统】

文件组织层次由支持一级索引到支持二级索引，实现了文件存储到外存等功能。

【位图支持】

从外存读取图片。

4 具体实现

4.1 图形模式的切换

在内核启动时将显卡的工作模式设置为图形模式，代码如下：

```
movw    $0x4f02,%ax
movw    $0x411B,%bx
int     $0x10

movw    $GUI_BUF,%bx
movw    %bx, %es
movw    $0x0000,%di
movw    $0x4f01,%ax
movw    $0x411B,%cx
int     $0x10
```

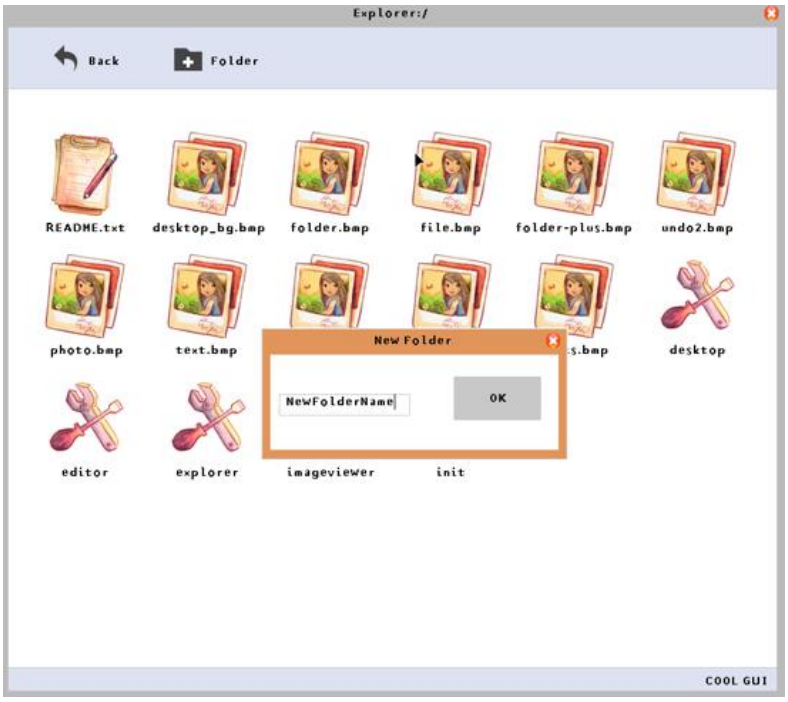
通过在寄存器 ax 中设置 0x4f02，在寄存器 bx 中设置图形模式类型，执行 0x10 BIOS 中断设置图形模式为 1280x1024x24 色分辨率，模式号为 0x11B。然后在寄存器 ax 中设置 0x4f01，执行 0x10 BIOS 中断获取显卡信息到 GUI_BUF 中。GUI_BUF 定义在 gui.h，表示内存地址 0x9000。在 main.c 中会调用 initGUI 函数，函数的主要作用是根据 GUI_BUF 中的内容获取显卡信息，其中最重要的便是显存地址。

4.2 窗口队列

window.c 主要负责窗口队列的维护，并为应用程序提供创建窗体及销毁窗体的系统调用。使用链表来维护窗口队列，并利用链表天然的顺序关系来表示窗口的层次关系。比较重要的一点是在内核中无法使用 malloc，因此创建新的链表节点时无法动态获取内存空间，只能通过预先定义好一块静态数组 windowLine，然后从这个静态数组中获取空间。

利用窗口队列，我们可以在界面上展示出窗口层叠的效果。最上面的窗口为当前窗口。其下所有窗口可能被遮挡，需要逐个处理。

效果展示：



窗口队列效果

4.3 图形绘制与局部刷新

GUI 函数库负责鼠标、窗口、控件、图片、文字等的绘制。由于鼠标的移动以及窗口的拖动速度很快，如果每次都重新绘制整个界面，效率将会十分低下。为了提高绘制效率，我们采用了局部刷新的机制。

除了显存之外，我们使用了两级缓存。第一级缓存保存除当前活动窗口和鼠标外的其余窗口按照层次顺序绘制好的界面，第二级缓存则为除鼠标外的所有窗口按照层次顺序绘制好的界面，而显存内容则为第二级缓存再加上鼠标。

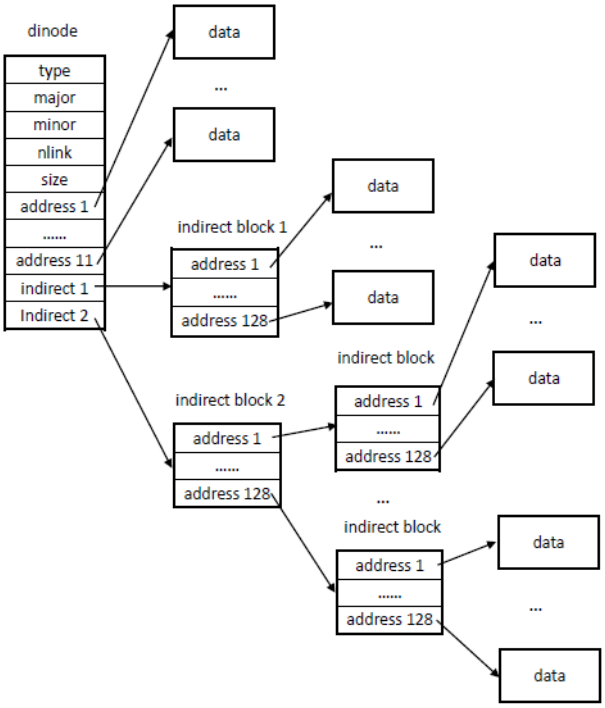
仅有鼠标在移动时，我们只需将显存中鼠标原位置的内容用第二级缓存相应位置的内容覆盖，然后在显存中鼠标新位置绘制鼠标即可。窗口拖动也是类似，由于只有当前活动窗口需要重绘，只需要将第二级缓存中活动窗口原位置的内容用第一级缓存相应位置的内容覆盖，然后在第二级缓存中活动窗口新位置绘制活动窗口即可。第二级缓存更新后再用第二级缓存去更新显存。

实现多级缓存与局部刷新之后，绘制效率获得了明显的提升。

4.4 文件系统

为了配合 GUI 完成相关的文件操作，我们增加了原文件系统所能容纳的文件的大小，并对文件系统进行了更加深入的扩展。

xv6 默认的 inode 文件系统包括 12 个直接索引块和 1 个一级索引块，共支持 70 KB 的文件大小。我们将 xv6 的文件系统改为了 11 个直接索引块，1 个一级索引块和 1 个二级索引块，从而将文件系统支持的容量扩大至 8.1 MB。



文件系统

由于文件的读写操作在磁盘的角度来看只是对某个扇区的读写操作，所以在修改文件系统时只需修改扇区的定位函数(fs.c 中 bmap)即可。而 xv6 需要在 Qemu 中虚拟化运行，运行前需要先创建一个硬盘镜像，这个镜像的生成程序的源代码是 mkfs.c，因此也有需要对其作出修改，修改的内容除了修改扇区定位函数外，还包括硬盘大小等常量的定义，以及对位图块(bitmap block)初始化的修改。

相关代码请见：fs.c, fs.h, mkfs.c[4]

4.5 位图支持

由于需要将图片以位图文件格式存放在磁盘镜像中，我们完成了对于位图文件格式的支持，包括位图文件的读取、存放、渲染等。

一个位图文件至少包括 3 个部分：

1） 位图文件头

2） DIB 头

3） 像素数组

由于位图文件本身也含多种格式，本次实验中只实现了 24 位位图这一种格式，这种文件将包括 3 个部分：

1） 位图文件头

2） 位图信息头

[4] mkfs.c 严格来说不算是 xv6 操作系统中的一部分。

3） 像素数组

相关信息说明如下：

【位图文件头】

用于文件识别，大小为 14 个字节。

【位图信息头】

包含位图文件的详细信息，如宽度、高度、颜色所占位数等。大小为 40 字节

【像素数组】

每一行所占的字节数必须为 4 的位数，读取下一行时须跳过这些填充字节。关于位图文件的更详尽描述，可参阅位图的相关文档。

4.6 鼠标驱动

通过查阅资料及向学长们请教，我们在 GUI 中实现了鼠标驱动。该实现基于鼠标 ps/2 协议。

标准的 PS/2 鼠标支持下面的输入：X（左右）位移，Y（上下）位移，左键，中键和右键。鼠标以一个固定的频率读取这些输入并更新不同的计数器，然后标记出反映的移动和按键状态。鼠标读取它的输入，并通过硬件中断发送数据包给计算机，由 OS 进行相应的处理。

具体步骤如下：

1）初始化工作

```
outb(0x64, 0xa8);
outb(0x64, 0xd4);
outb(0x60, 0xf4);
outb(0x64, 0x60);
outb(0x60, 0x47);
```

主机需要通过 0x60、0x64 端口读取鼠标的相关数据包。因此，首先需要向这两个端口发送相应信号，打开端口监听。接着，还需要设置鼠标中断及相关的锁：

```
initlock(&mouse_lock, "mouse");
picenable(IRQ_MOUSE);
ioapicenable(IRQ_MOUSE, 0);
```

2）数据读取

当中断事件发生时，系统将会调用中断处理函数。该函数需要 OS 来实现。在该函数(mouseint)中，可以通过 0x60、0x64 端口读取相应数据。

标准的 PS/2 鼠标发送位移和按键信息给主机采用如下的 3 字节数据包格式：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							

字节 1 包含了鼠标位移方向(sign bit)、是否越界(overflow)、左右中键按下(Left/Right/Middle Btn)等位信息。字节 2 和字节 3 则表示鼠标在 X 方向、Y 方向上的位移值，需注意的是该值为位移量的补码，因此方向位为 1 时需要对位移量进行转码。

3）事件处理

在获取了数据之后，需要完成两个工作：（1）根据鼠标位移量，更新鼠标位移信息，在界面上重绘鼠标；（2）判断鼠标事件。

我们在 mouse.h 中定义了关于鼠标的结构体：

```
typedef struct mouseinfo
{
  int x_position;
  int y_position;
  int last_draw_x;
  int last_draw_y;
  int event;
}mouseinfo;
```

其中 x_position、y_position 记录鼠标当前位置，last_draw_x、last_draw_y 记录上一次鼠标绘制的位置，event 记录事件（包括单击、双击、拖动等）；

对于（1），只要根据从端口读取的数据更新 x_position、y_position，然后调用 GUI 相关的函数绘制鼠标。绘制鼠标同样运用了局部刷新机制。

对于（2），判断鼠标事件并不困难，但比较繁琐。我们在 mouse.c 的 updateMouseEvent 完成了鼠标事件更新。具体实现可参见代码，这里不再赘述。

根据鼠标的位置、事件和状态，OS 将分派相应的相应函数给用户进程。我们利用鼠标驱动实现的相关功能有：按住左键拖动窗口，点击图标显示选中焦点、点击窗口更改窗口队列顺序及窗口焦点等。

4.7 键盘驱动

实际上，xv6 已经完成了键盘驱动(kbd.c、kbd.h)，我们按照自身需求，稍微修改了原本的代码，将从键盘读取的字符值作为输入，传递给用户进程函数。

我们利用键盘驱动，实现了简单的文本编辑器。其实现机制为：每次键盘中断发来数据信号时，由 OS 负责处理该输入数据。OS 首先定位到当前正在编辑的文本控件(textBox)，然后系统切换到用户进程，更改该控件中字符串的数据，完成字符的添加(insertCharacter)、删除(deleteCharacter)及光标的移动(moveCursor)等数据更新，之后由 GUI 库相应的函数完成绘制。具体的实现请见 kbd.c 代码。

4.8 文字支持

对于 GUI 而言，文字支持是必不可少的。考虑到系统需要，只需要实现英文字符绘制即可。

由于文字支持需要频繁使用（在文本编辑器中），若使用读取 bitmap 文件的方式绘制字符将导致效率低下，故需要将文字字符点阵写入内核，直接调用。使用程序直接生成的字符点阵不够美观，我们人工对字符点阵进行了修饰和美化。

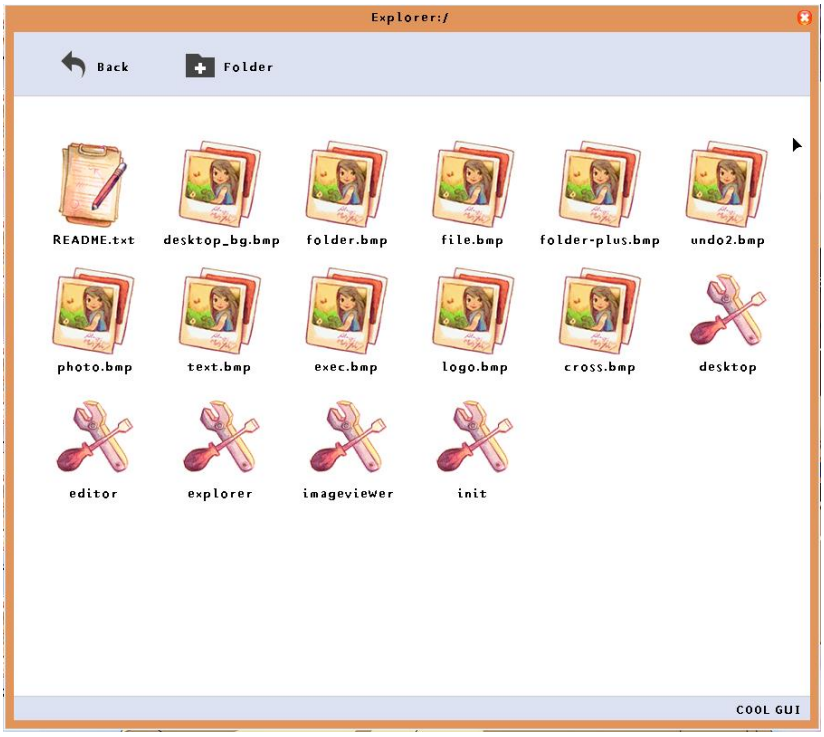
5 用户进程

5.1 桌面

桌面是继 init 进程后的第一个进程，也是常驻系统的进程。桌面会显示出文件系统根目录的所有文件和目录，对于不同的文件类型，会使用不同的图标。同时双击图标也会根据文件类型调用相应的应用程序去打开它。如 bmp 文件会调用图片查看器 imageviewer，txt 文件会调用文本编辑器 editor，而可执行程序则会直接执行。

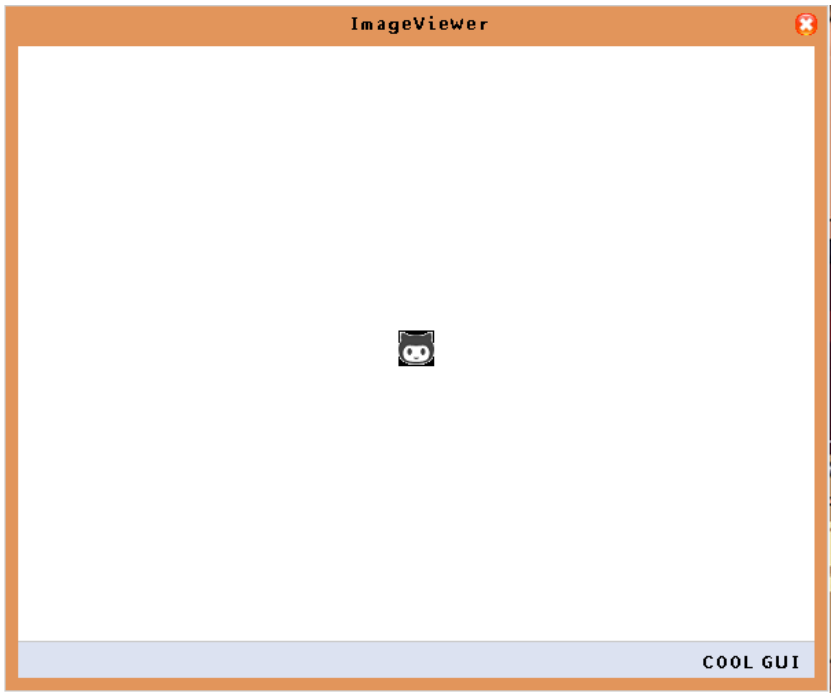
5.2 文件浏览器

文件浏览器的功能与桌面基本类似，但它可以查看非根目录下的文件和目录，同时可以新建目录。



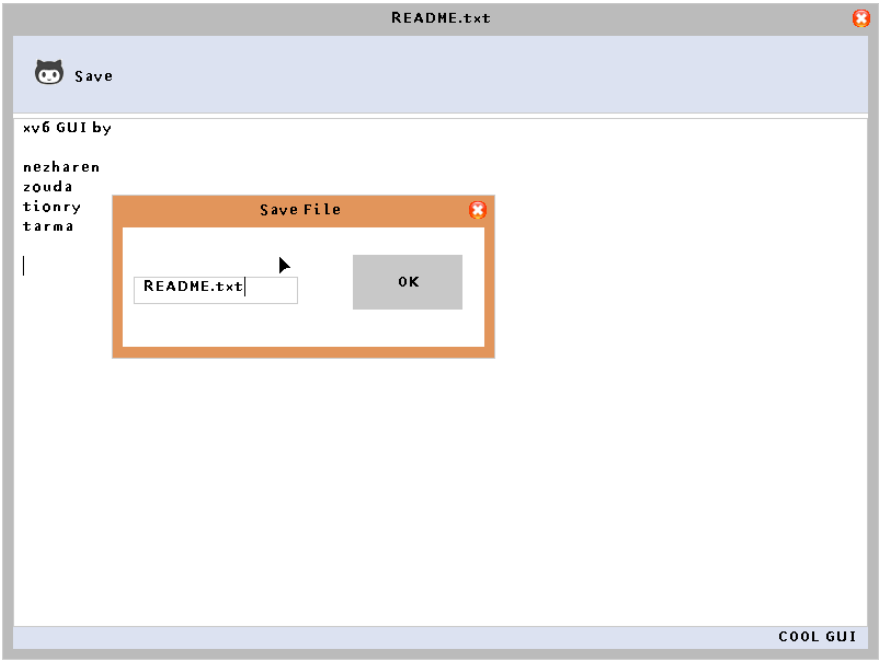
5.3 图片查看器

图片查看器可以加载文件系统中的 24 位色 Bitmap 文件并显示出它的内容。



5.4 文本编辑器

实现了简单的文本编辑器（editor），支持光标的上下左右移动，字符插入和删除，支持文件的新建和存储并存到 fs.img 中，关闭系统并重新启动仍可以看到。



6 GUI 相关数据结构

我们在下面列出了所有 GUI 函数用到的数据结构。其中 RGB 是颜色结构体，Window 是窗口的结构体，其余结构体都是基于窗口的控件。

RGB：颜色

```
typedef struct RGB
{
  unsigned char B;
  unsigned char G;
  unsigned char R;
} RGB;
```

Window： 窗口

```
typedef struct Window
{
  int show;
  int hasCaption;
  int last_leftTopX, last_leftTopY;
  int leftTopX, leftTopY, width, height;
  char caption[MAX_STRING_NUM];
  int widgetsNum;
  Widget widgets[MAX_WIDGET_NUM];
  Handler onFileSystemChangedHandler;
} Window;
```

Handler：事件处理句柄

```
typedef struct Handler
{
  int triggered;
  void (*handlerFunction)(Widget *widget, Window *window);
} Handler;
```

Label：标签控件

```
typedef struct Label
{
  int leftTopX, leftTopY, width, height;
  char text[MAX_STRING_NUM];
} Label;
```

TextBox：文本编辑框控件

```
typedef struct TextBox
{
  int fixed;
  int leftTopX, leftTopY, width, height, cursor;
  int textLength;
  char text[MAX_STRING_NUM];
} TextBox;
```

ImageView：图片控件

```
typedef struct ImageView
{
  int leftTopX, leftTopY, width, height;
  struct RGB *image;
  Handler onLeftClickHandler;
} ImageView;
```

IconView

```
typedef struct IconView
{
  int leftTopX, leftTopY, width, height;
  struct RGB* image;
  char text[MAX_STRING_NUM];
  Handler onLeftDoubleClickHandler;
} IconView;
```

在明确地定义了这些窗体控件之后，我们在绘制用户进程的窗口界面时就能更加得心应手。

7 团队分工

姓名	分工
张　凯	GUI 函数库、GUI 接口、窗口队列、资源管理器
邹豪风	鼠标驱动、键盘驱动、界面绘制与 UI 设计
乔春雨	GUI 接口、文字支持、文本编辑器
傅展升	文件系统改进、位图（BMP 格式）支持

THANKS FOR YOUR PATIENCE