华中科技大学

操作系统原理课程设计报告

姓　　名：　　叶 翔 宇

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术（卓越工程师）

班　　级：　　计卓 1601 班

学　　号：　　U201611143

指导教师：　　郑然

分数	
教师签名	

2019 年 3 月 15 日

目　录

1 实验一 熟悉和理解 Linux 编程环境

1.1 实验目的

掌握 Linux 的使用方法，熟悉和理解 Linux 编程环境。

1.2 实验内容

1 编写一个 C 程序，用 fread，fwrite 等库函数实现文件拷贝功能。命令形式：

copy <源文件名> <目标文件名>

2 编写一个 C 程序，用 QT 或 GTK 分窗口显示三个并发进程的运行(一个窗口实时显示当前时间，一个窗口按行显示/etc/fstab 文件的内容，一个窗口显示 1 到 1000 的累加求和过程，每秒刷新一次)。

1.3 实验设计

1.3.1 开发环境

1 **虚拟机运行环境**：Microsoft Windows 10 64 位 1803；

2 **虚拟机软件版本**：VMware Workstation 12 Pro 12.5.7 build-5813279；

3 **虚拟机资源分配**：

a) 处理器：Intel® Core™ i5-8250U CPU @ 1.60GHz × 4；

b) 内存：3.8 GB NVMe SSD；

c) 磁盘：115GB USB3.0 Device；

4 **Linux 内核版本**：Linux version 4.16.10；

5 **Ubuntu 版本**：Ubuntu 5.4.0-6ubuntu1~16.04.10；

6 **GTK 版本**：GTK 2.24.3

7 **文本编辑器**：Gedit 3.18.3

8 **编译器**：GCC 5.4.0 20160609

1.3.2 实验设计

- 实验内容一

1 总体设计

本实验中使用选择 fwrite 与 fread 等库函数来实现文件的拷贝功能，在本实验中使用了环形缓冲技术与多进程同步并发技术来实现文件拷贝。功能实现的过程中充分考虑了设计功能的完备性，程序的鲁棒性以及资源的利用效率。总体设计图如图 1.1 所示。

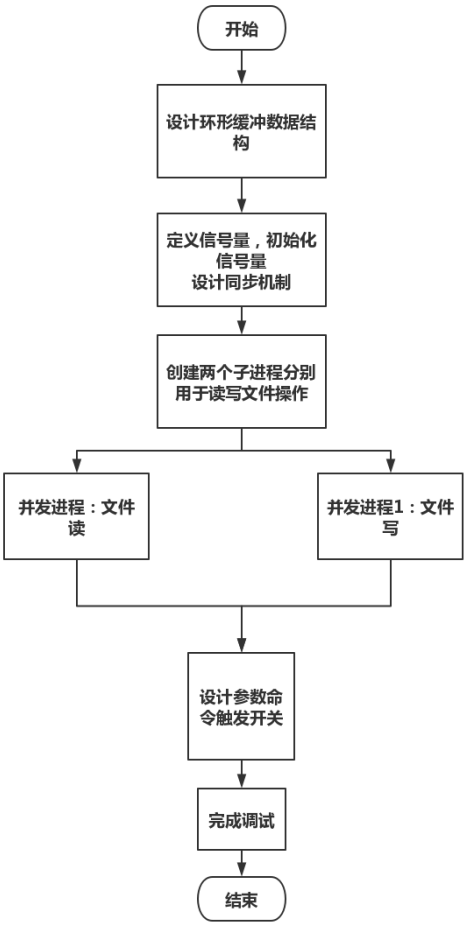

图 1.1 文件拷贝总体设计图

2 详细设计

a) 信号量设计：本实验设计两个信号灯，一个信号灯初值为 10，代表 10 个缓冲区初始均为可写状态；另一个缓冲区初始值 0，代表没有一个缓冲区初始可读；

b) 环形缓冲设计：环形缓冲采用共享内存进行进程间通讯，共享内存的结构应该包括一个第一维为 10 第二维为 100 的二维 char 类型数据组结构；还应当包括一个包含 10 个元素的数组用来记录每个缓冲区块的大小；最后还应当包含一个标志位来标记是否出现最后一个缓冲区。环形缓冲区用二维 char 型数组来表示，其中第一维为 10 第二维为 100 代表 10 个缓冲区，一个缓冲区为 100 个字节。环形缓冲结构如图 1.2 所示；

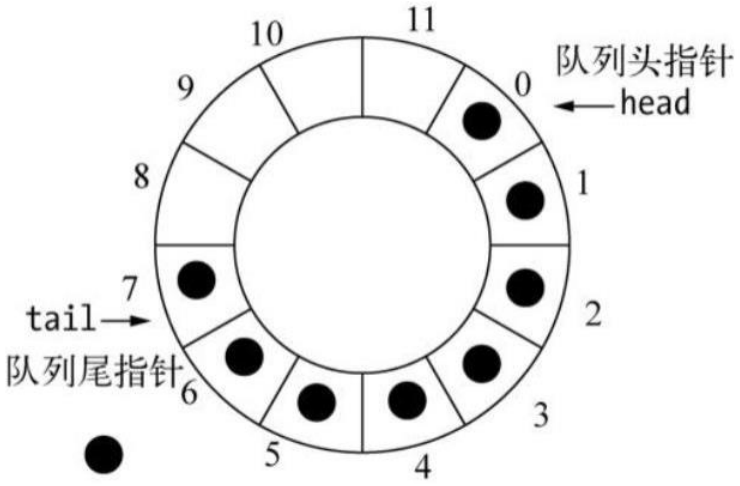

图 1.2 环形缓冲结构

c) 命令参数开关设计：判断调用函数时传入的参数类型，第二个参数为“copy”字符串，第三个参数为源文件的地址，第四个参数为目标文件的地址。

● 实验内容二

1 总体设计

本设计要求使用 C 语言完成并发进程的设计，同时使用 GTK 完成并发窗口的显示，按照要求动态显示当前内容。在本设计中，创建了三个进程分别用来分别显示“当前实时时间”、“/etc/fatsb 文件内容”，以及“1 到 1000 的累加过程”。再再每个进程中创建一个线程用来每秒动态刷新显示窗口显示内容。总体设计结构如图 1.3 所示。

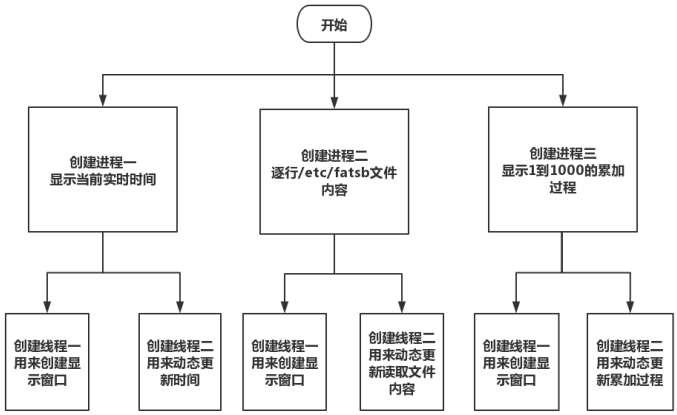

图 1.3 动态显示总体设计图

2 详细设计

a) 动态显示实时时间设计：首先在显示时间的子进程中创建两个线程，一个用

来动态获得时间，一个使用 GTK 创建一个窗口，两个线程共享进程的内存空间，通过设置每隔一秒重新读取一次系统时间改变窗口中标签内容来实现时间的动态显示；

b) 动态逐行读取文件设计：首先在逐行读取文件的子进程中创建两个线程，一个用来逐行读取“/etc/fatsb”文件内容，一个使用 GTK 创建一个窗口，两个线程共享进程的内存空间，通过设置每隔一秒读取下一行文件内容改变窗口中标签内容来实现逐行读取文件的动态显示；

c) 动态显示计算过程设计：首先在逐行读取文件的子进程中创建两个线程，一个用来逐行进行累加计算，一个使用 GTK 创建一个窗口，两个线程共享进程的内存空间，通过设置每隔一秒进行下一次累加运算改变窗口中标签内容来实现累加过程的动态显示；

1.4 实验调试

1.4.1 实验步骤

- 实验内容一

1 创建信号灯：创建一个包含两个信号量的信号量集，第一个信号量初始化为 10，第二个初始化为 0；

2 创建共享内存：共享内存的结构包括一个二维 char 类型数组结构；一个包含 10 个元素的数组用来记录每个缓冲区块的大小；一个标志位来标记是否出现最后一个缓冲区；

3 建立主进程模块：在主进程中创建信号灯与共享内存并创建两个子进程，程序执行进入子进程后，主进程等待子进程结束后删除信号灯，释放共享内存后返回；

4 建立写缓冲进程模块：写缓冲进程循环进行读取文件到缓冲区操作，读取文件时，每次读取 100 字节大小的数据块到环形缓冲的一个缓冲区，缓冲区计数器进行加 1 后模 10 操作，用来指示当前使用的是环形缓冲的哪一块缓冲区。当一次读取的文件大小不足 100 字节时，标记写到最后一个缓冲区块；

5 建立读缓冲进程模块：读缓冲进程循环进行读缓冲区数据到目标文件操作，读取文件时，每次从缓冲区读取 100 字节大小的数据块到目标文件，当一次读取的文件大小不足 100 字节时，标记写到最后一个缓冲区块；

6 建立开关参数模块：通过命令执行时传入的参数来模拟实现函数调用的命令，其中“copy”字符串和源文件与目标文件的地址为主进程传入的三个参数。

- **实验内容二**

1　建立动态显示实时时间模块：对于该模块下的子线程一，调用库函数 time()读取当前系统时间，使用库函数 sleep()实验每次刷新间隔一秒，使用 GTK 库函数 gtk_label_set_tex()将每次动态刷新后的数据装入窗口中的标签；对于该模块下的子线程二，使用 GTK 的窗口和组装盒结构设计显示文本内容的窗口，GTK 模块示意图如图 1.4 所示。

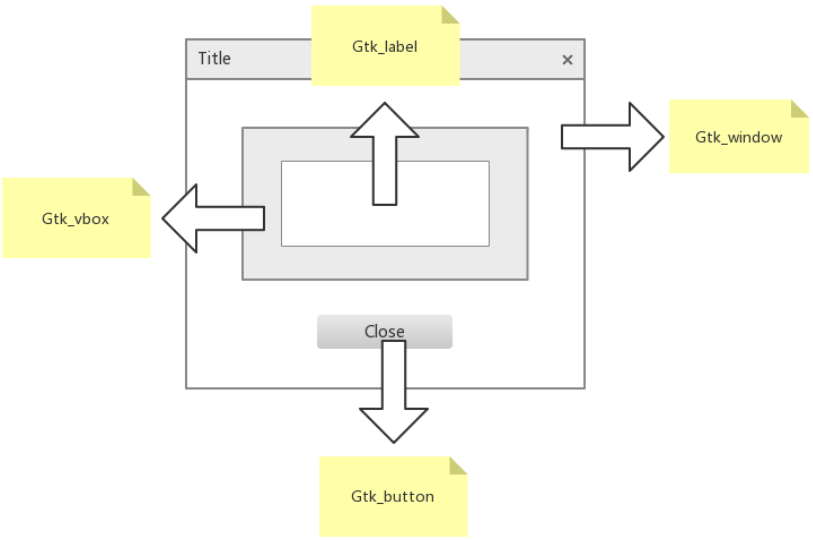

图 1.4 GTK 模块示意图

2　建立逐行读取文件内容模块：对于该模块下的子线程一，调用库函数 fgets()读取指定文件中一行的内容，使用库函数 sleep()实验每次刷新间隔一秒，使用 GTK 库函数 gtk_label_set_tex()将每次动态刷新后的某一行文件内容装入窗口中的标签；对于该模块下的子线程二，使用 GTK 的窗口和组装盒结构设计显示文本内容的窗口，GTK 模块示意图如图 1.4 所示。

3　建立动态显示累加过程模块：对于该模块下的子线程一，调用库函数 sprintf()编辑加法内容，使用库函数 sleep()将实验每次刷新间隔一秒，使用 GTK 库函数 gtk_label_set_tex()讲每次动态刷新后的加法等式内容装入窗口中的标签；对于该模块下的子线程二，使用 GTK 的窗口和组装盒结构设计显示文本内容的窗口，GTK 模块示意图如图 1.4 所示。

4　建立窗口销毁模块：使用 GTK 库函数 gtk_main_quit()实现窗口的销毁，并使用 GTK 库函数 g_signal_connect_swapped()将该函数和按钮点击事件进行绑定。

1.4.2 实验调试及心得

- 实验内容一

1 实验调试：

编写并编译代码后执行可执行文件，按照实验要求输入命令参数，选择的测试源文件为“test.mp4”，测试目标文件为：“test1.mp4”，在执行程序前，工程目录内的文档如图 1.3 所示。

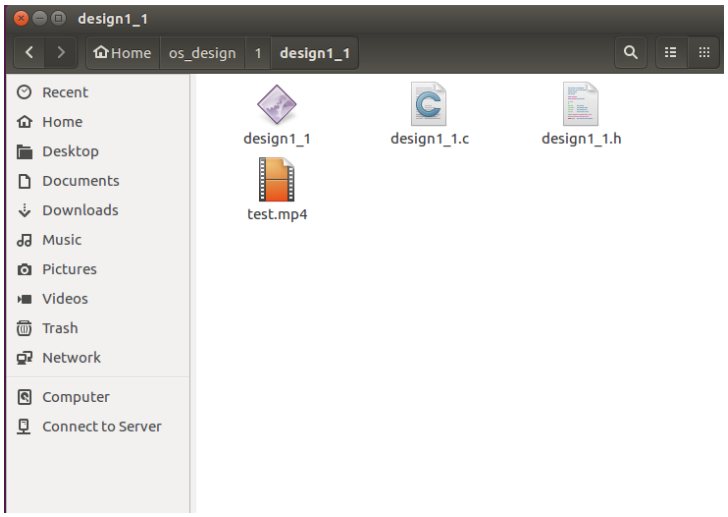

图 1.5 执行程序前工程目录内容

在程序执行过程中，控制台输出提示信息如图 1.6 所示。

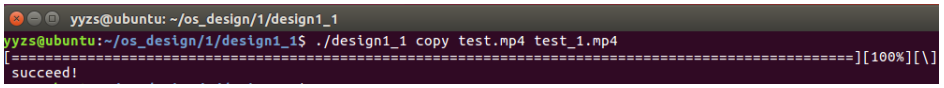

图 1.6 文件复制控制台输出信息

控制台提示执行成功后，工程目录内的文档如图 1.7 所示，此时目标文档“test_1.mp4”出现在工程目录中。

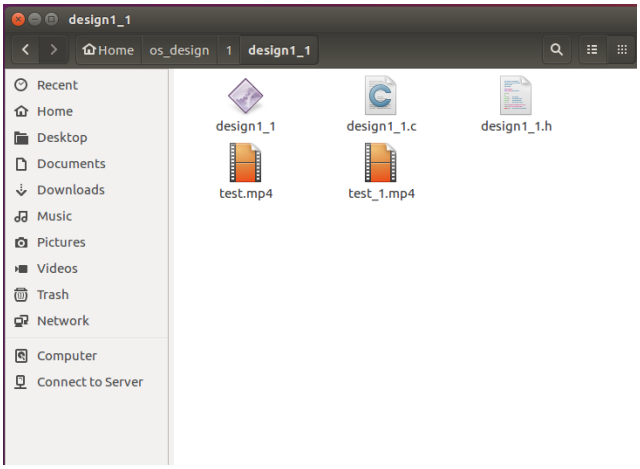

图 1.7 执行程序后工程目录内容

比较两文件大小如图 1.8 所示，显示文件大小一致，且复制后得到的文件可以正常打开，文件复制成功。

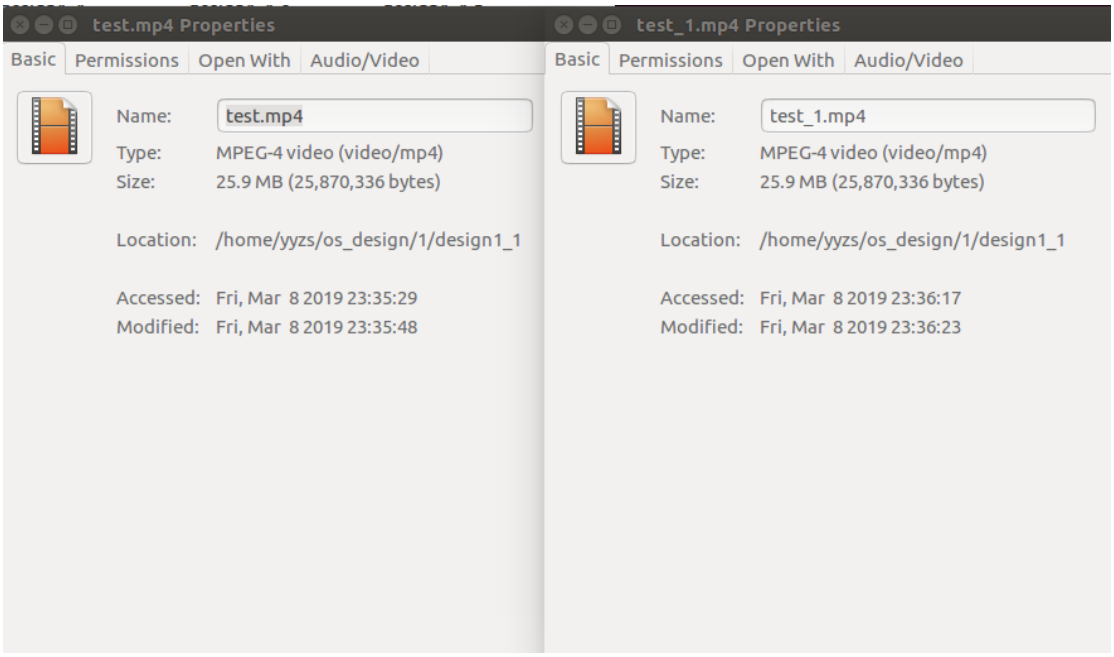

图 1.8 源文件与目标文件对比

2 实验心得

● 实验内容二

1 实验调试：

运行可执行文件，出现三个动态显示窗口如图 1.9 所示。

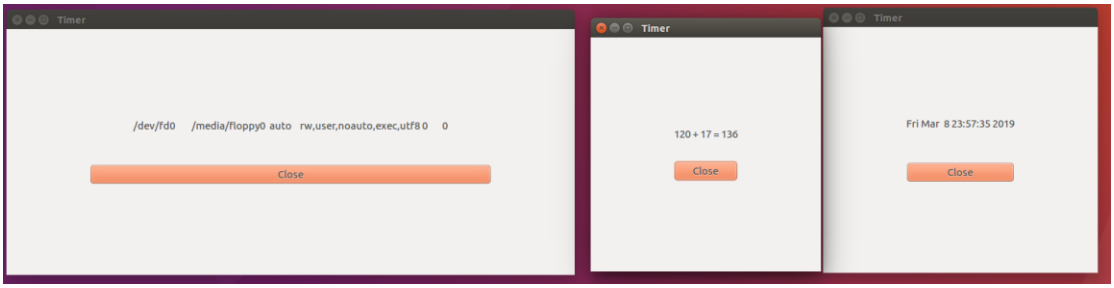

图 1.9 三个动态显示窗口

程序运行一段时间后，三个动态显示窗口的内容有实时更新，如图 1.10 所示。

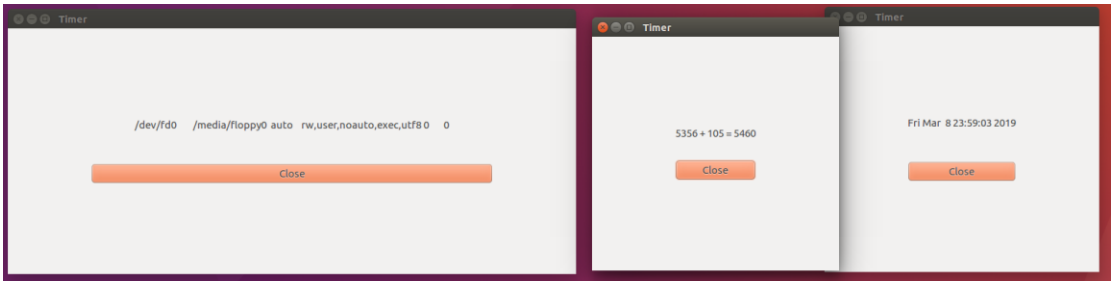

图 1.10 动态更新后的显示窗口

点击结束按钮。可以实现窗口逐个的销毁

2 实验心得：

通过本次实验，对于 Linux 下的编程有了进一步的了解，对于使用 Gtk 进行图形界面的编程有了最初的认识，这是第一次使用 C++完成图形界面的设计，在实验的过程中，遇到了诸如多线程窗口关闭会产生异常之类的关于线程安全的错误，经过调试之后，问题主要集中在设置窗口关闭的回调函数和多线程的销毁之间的并发关系没有充分考虑。这对之后的多线程调试有一定的启发意义。

附录 实验代码

- **design1_1.c**

```
#include "design1_1.h"
#include <unistd.h>
#include <sys/file.h>
#include <sys/wait.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
//读取文件大小函数
int file_size(char * filename)
{
    struct stat statbuf;
    stat(filename,&statbuf);//读取文件状态信息
    int size = statbuf.st_size;
    return size;
}

int main(int argc, char* argv[])
{
    if(argc != 4)
    {
        printf("the num of the parameters is wrong!\n");
        return 0;
    }
    else if(strcmp(argv[1],"copy") != 0)
    {
        printf("wrong instruction!\n");
        return 0;
    }
    strcpy(sourcefile,argv[2]);
    strcpy(targetfile,argv[3]);
    //创建信号灯
    int semID;
    semID = semget(semKey,2,IPC_CREAT|0666);
    union semun arg;
    arg.val = 10;//10 个缓冲区初始都为可写状态
    semctl(semID,0,SETVAL,arg);//初始化可写信号灯
    arg.val = 0;//没有一个缓冲区初始可读
    semctl(semID,1,SETVAL,arg);//初始化可读信号灯

    //创建共享内存
    int shID;
```

```
    shID = shmget(shKey,sizeof(shbuf),IPC_CREAT|0666);
    shbuf * Loop_BUF = (shbuf *)shmat(shID,NULL,0);
    Loop_BUF->last_block = 0;//初始标记最后一个缓冲区不存在
    int pid1 = -1,pid2 = -1;
    pid1 = fork();
    if(pid1 != 0 && pid2 != 0)
    {
        pid2 = fork();
    }
    if(pid1==0)//子进程 1
    {
        //get 信号灯
    int semID;
    semID = semget(semKey,2,IPC_CREAT|0666);//0666 为权限

    //get 共享内存
    int shID;
    shID = shmget(shKey,sizeof(shbuf),IPC_CREAT|0666);
    shbuf * Loop_BUF = (shbuf *)shmat(shID,NULL,0);


    //定义文件指针
    FILE * fp;
    if((fp = fopen(sourcefile,"rb"))==NULL)
    {
        printf("Can not open file\n");
    }


    int total_size = file_size(sourcefile);//文件总大小
    int Buf_Num = 0;//当前缓冲区编号
    double current_size = 0;//当前已读大小

    //准备进度条
    char buf[102];//进度条主体
    memset(buf,'\0',sizeof(buf));
    const char* lable="|/-\\";
    int i = 0;
    while(Loop_BUF->last_block!=1)
    {
        P(semID,0);//P 写操作
        Loop_BUF->len_of_block[Buf_Num] =
fread(Loop_BUF->buf_block[Buf_Num],sizeof(char),100,fp);//往一个缓冲区里写
        if(Loop_BUF->len_of_block[Buf_Num]<100)//当一个缓冲区没有被写满
        {

            Loop_BUF->last_block = 1;//标记遇为最后一个缓冲区出现
        }
        current_size += Loop_BUF->len_of_block[Buf_Num];
        printf("\033[?25l");

        //来做一个进度条
        printf("[%-100s][%.0f%%][%c]\r",buf,current_size/total_size*100,lable[i%4]);
        i++;
        int j = 0;
        for(j = 0;j<=(current_size*100/total_size);j++)
        {
```

```
            buf[j] = '=';
        }
        Buf_Num = (Buf_Num+1)%10;//环形读取下一个缓冲区
        V(semID,1);//V 读操作
    }
    fclose(fp);
    exit(0);
    }
    else if(pid2==0)//子进程 2
    {
        //获取信号灯
    int semID;
    semID = semget(semKey,2,IPC_CREAT|0666);

    //获取环形缓冲
    int shID;
    shID = shmget(shKey,sizeof(shbuf),IPC_CREAT|0666);
    shbuf * Loop_BUF = (shbuf *)shmat(shID,NULL,0);

    //打开要写入的文件
    FILE *fp2;
    if((fp2 = fopen(targetfile,"wb"))==NULL)
    {
        printf("Can not open the file\n");
    }

    int Buf_Num_2 = 0;
    while(Loop_BUF->last_block!=1 || Loop_BUF->len_of_block[Buf_Num_2]==100)
    {
        P(semID,1);//P 读
        fwrite(Loop_BUF->buf_block[Buf_Num_2],sizeof(char),Loop_BUF->len_of_block[Buf_Nu
m_2],fp2);//每次读取一个缓冲区的内容
        Buf_Num_2 = (Buf_Num_2+1)%10;//循环累加
        V(semID,0);//V 写
    }
    if(Loop_BUF->len_of_block[Buf_Num_2]!=0)//当标记为最后一块的缓冲区内还有可读内容
    {
        fwrite(Loop_BUF->buf_block[Buf_Num_2],sizeof(char),Loop_BUF->len_of_block[Buf_Nu
m_2],fp2);//读取最后一块缓冲区
    }
    fclose(fp2);
    exit(0);
    }
    else//主进程
    {
        wait(&pid1);
        wait(&pid2);
        printf("\n succeed!\n");
        //删除信号灯

        semctl(semID,1,IPC_RMID,0);
        //释放共享内存
        shmctl(shID,IPC_RMID,0);
    }
    return 0;
}
```

- **design1_2.c**

```c
#include<gtk/gtk.h>
#include<sys/types.h>
#include<unistd.h>
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

void destroy_progress(GtkWidget *widget);
void *set_time();
void *set_sum();
void *set_file();
void gtk_happen(GtkWidget * lable);


pid_t pid1,pid2,pid3;
char t[100];//记录实时时间
char file_content[1000];//记录文件每行内容
char csum[100];//记录累加结果
GtkWidget * label1;
GtkWidget * label2;
GtkWidget * label3;

int main(int argc, char* argv[])
{
  /*
  创建三个线程，分别用来:动态获得时间,动态获取文件信息,动态累加求和
  */
  pthread_t th1,th2,th3;
  int re;
  /*
  创建三个进程,父进程用来显示时间,pid1 用来逐行读取文件,pid2 用来动态求和
  */

  pid1 = fork();
  if(pid1 != 0)
  {
    pid2 = fork();
  }
  if(pid1 != 0 && pid2 != 0)
  {
    label1 = gtk_label_new(t);
    re = pthread_create(&th1, NULL, (void *)set_time, NULL);
    gtk_init(&argc, &argv);
    gtk_happen(label1);
    //re = pthread_join(th1, NULL);
  }
  if(pid1 == 0)
  {
    label2 = gtk_label_new(csum);
    re = pthread_create(&th2, NULL, (void *)set_sum, NULL);
    gtk_init(&argc, &argv);
    gtk_happen(label2);
    //re = pthread_join(th2, NULL);
  }
  else if(pid2 == 0)
  {
```

```
    label3 = gtk_label_new(file_content);
    re = pthread_create(&th3, NULL, (void *)set_file, NULL);
    gtk_init(&argc, &argv);
    gtk_happen(label3);
  }
}
//销毁 Gtk
void destroy_progress( GtkWidget *widget)
{
   gtk_main_quit ();
}
//线程 1
void *set_time()
{
  while(1)
  {
    time_t time1;
    time(&time1);
    sprintf(t,"%s",ctime(&time1));
    gtk_label_set_text(GTK_LABEL(label1),t);
    sleep(1);
  }
}

void *set_sum()
{
  int i = 1;
  int sum = 0;
  int pre_sum = 0;
  while(1)
  {
    pre_sum = sum;
    sum = pre_sum + i;
    i++;
    sprintf(csum,"%d + %d = %d",pre_sum,i,sum);
    gtk_label_set_text(GTK_LABEL(label2),csum);
    sleep(1);
  }
}

void *set_file()
{
  FILE *fp = NULL;
  fp = fopen("/etc/fstab","rb");
  while(!feof(fp))
  {
    fgets(file_content,10000,fp);
    gtk_label_set_text(GTK_LABEL(label3),file_content);
    sleep(1);
  }
}

//gtk 结构
void gtk_happen(GtkWidget * label)
{
    GtkWidget * window;//定义一个窗口
    GtkWidget * vbox;//定义一个组装盒
```

```
    GtkWidget * button;//定义一个按钮
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);//创建一个新的窗口
    gtk_window_set_resizable (GTK_WINDOW(window), TRUE); //修改窗体的伸缩属性
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);//设定窗口的位置
    g_signal_connect (G_OBJECT(window), "destroy", G_CALLBACK (destroy_progress), NULL);//
监听窗口的 destroy 事件
    gtk_window_set_title (GTK_WINDOW (window), "Timer");//设定窗口的标题
    gtk_container_set_border_width (GTK_CONTAINER(window), 20);//用来设定宽度

    //使用 gtk_vbox_new 函数建立纵向组装盒;
    //为了显示构件，必须将构件放入组装盒中,并将组装盒放在窗口内;
    vbox = gtk_vbox_new (FALSE, 10);
    gtk_container_set_border_width (GTK_CONTAINER (vbox), 100);//用来设定宽度;
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);

    //lable 构件放入组装盒
    gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 10);
    gtk_widget_show (label);

    button = gtk_button_new_with_label("Close");
    g_signal_connect_swapped (G_OBJECT (button), "clicked", G_CALLBACK (gtk_widget_destroy),
window);//信号登记函数,监听按钮的 clicked 事件
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 10);
    GTK_WIDGET_SET_FLAGS(button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default(button);
    gtk_widget_show (button);

    gtk_widget_show (window);
    //准备将窗口和所有的组件显示在屏幕上,这个函数必须在 GTK 程序的最后调用.
    gtk_main ();
}
```

2 实验二 掌握添加系统调用的方法

2.1 实验目的

掌握添加系统调用的方法。

2.2 实验内容

1 采用编译内核的方法，添加一个新的系统调用实现文件拷贝功能；
2 编写一个应用程序，测试新加的系统调用。

2.3 实验设计

2.3.1 开发环境

1 **虚拟机运行环境**：Microsoft Windows 10 64 位 1803；
2 **虚拟机软件版本**：VMware Workstation 12 Pro 12.5.7 build-5813279；
3 **虚拟机资源分配**：
   a) 处理器：Intel® Core™ i5-8250U CPU @ 1.60GHz × 4；
   b) 内存：3.8 GB NVMe SSD；
   c) 磁盘：115GB USB3.0 Device；
4 **Linux 内核版本**：Linux version 4.16.10；
5 **Ubuntu 版本**：Ubuntu 5.4.0-6ubuntu1~16.04.10；
6 **GTK 版本**：GTK 2.24.3
7 **文本编辑器**：Gedit 3.18.3
8 **编译器**：GCC 5.4.0 20160609

2.3.2 实验设计

1 总体设计

本实验中要求通过编译内核的方式把新加入的拷贝文件的功能永久性加入内核中，编译生成新的内核。设计的主要步骤包括添加源代码、连接新的系统调用，重建 Linux 内核和测试新的系统功能调用。其中完成系统功能调用的功能逻

辑图如图 2.1 所示。

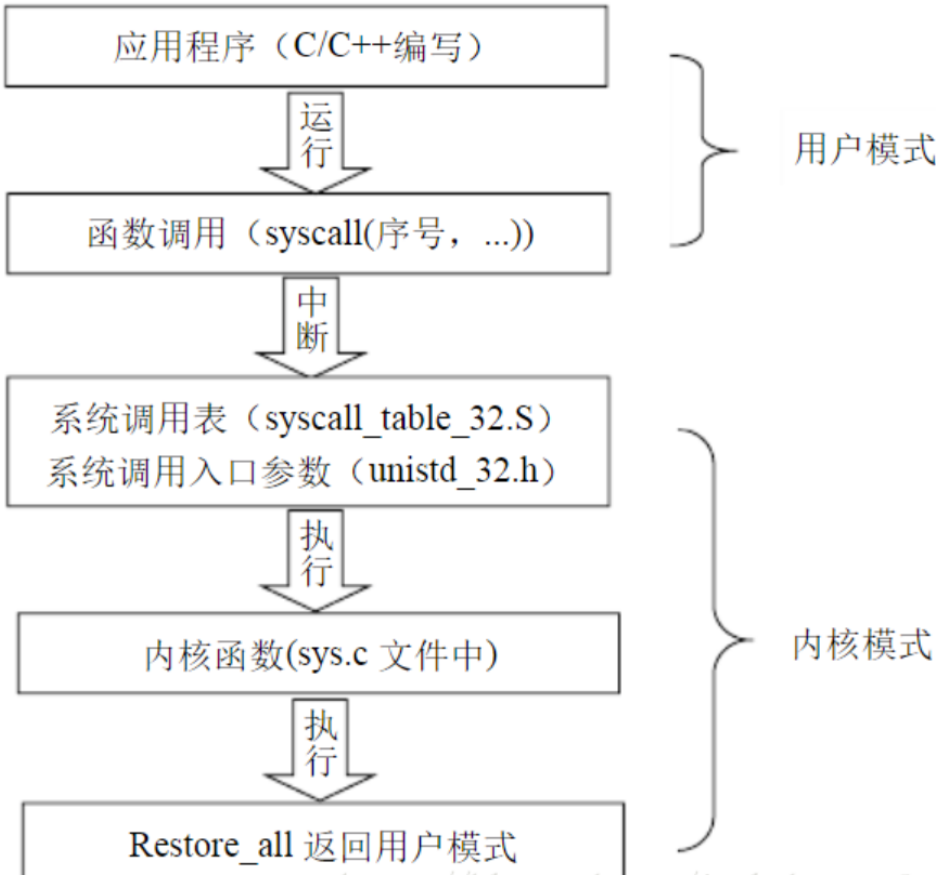

图 2.1 完成系统功能调用的功能逻辑图

2 详细设计

a) 文件拷贝功能源代码：首先在用户态使用系统功能调用实现文件的拷贝，再将代码内系统功能的调用方式改为内核态的系统功能调用方式，并在 sys.c 文件中添加修改后的代码，以实现文件内核态的文件拷贝功能；

b) 连接新的系统调用：为了使内核的其余部分知道该系统调用的存在，需要为系统功能调用分配一个系统功能调用号以及系统调用名，并将分配的系统调用号与系统调用名按格式写入 syscall_64.tbl 文件中。上述操作完成后，还需要在 syscalls.h 中对系统功能进行声明，已完成新的系统功能调用的连接；

c) 重建 Linux 内核：此步骤中首先需要生成新的内核配置文件，成功编译完成后，继续编译内核映像与内核模块。上述内容全部编译完成后，生成并安装模块，最终安装新的系统。为了使新系统能够顺利进入，需要修改引导程序 grub 的配置，选择新修改的内核。进入新的系统；

d) 测试系统功能调用：记录在连接新的系统调用步骤中为新的系统功能调用分配的系统功能调用命和调用号，在用户态编写程序，通过系统功能调用号或系统功能调用命完成对新的系统功能的调用。实现用户态文件复制的功能。

2.4 实验调试

2.4.1 实验步骤

1 下载 Linux 内核代码：在 www.kernel.org 中下载 Linux 内核原码，本次课程设计使用的内核版本为 4.16.10。下载完成后使用终端 xz 指令和 tar 指令进行解压得到 linux-4.16.10 文件夹，解压结果如图 2.2 所示；

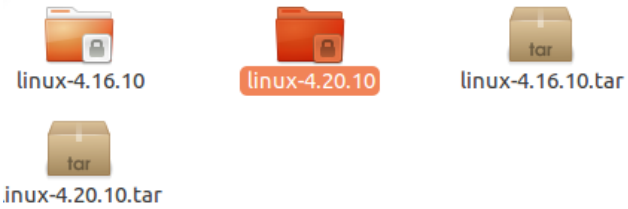

图 2.2 内核源码解压结果

2 编写文件拷贝功能代码：进入目录./kernel/sys.c 修改用户态的部分调用方式，完成函数在内核态的调用，内核态实现文件拷贝功能的流程图如图 2.3 所示。

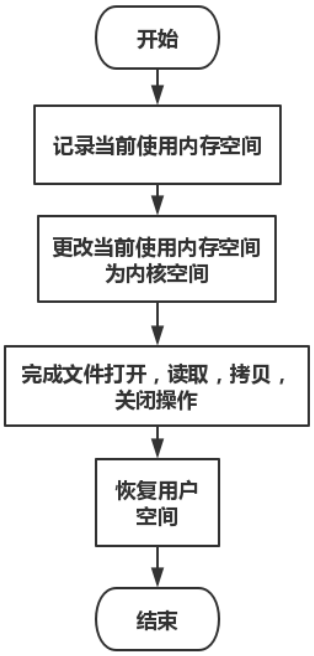

图 2.3 内核态实现文件拷贝功能的流程图

3 连接新的系统功能调用：通过./arch/x86/entry/syscalls/syscall_64.tbl 路径查看系统功能调用表。在系统功能调用表中，每个系统功能调用占一项，格式为：

<系统调用号> <64/x32/common> <系统调用名> <服务例程入口>

如图 2.4 所示选择一个未使用的系统调用号分配给新的系统功能调用。

```
332	common	statx	sys_statx
333	64	mysyscall	sys_mysyscall
```

图 2.4 选择一个未使用的系统调用号分配

通过./include/linux/syscalls.h 路径查看系统功能调用声明文件。在文件末尾按照系统调用的声明方式对新添加的系统功能调用进行声明。声明方式如图 2.5 所示。

```
asmlinkage long sys_mysyscall(char * sourcefile, char * targetfile);

#endif
```

图 2.5 声明新的系统功能调用

其中 asmlinkage 是一个必须的限定词，用于通知编译器仅从内核中提取该函数的参数，而不是从寄存器中，因为在执行服务例程之前系统已经将通过寄存器传递过来的参数压入内核堆栈。

4 重建 Linux 内核：首先安装编译内核所需要的库，本实验中编译时需要的库有 libncurses5-dev libssl-dev、build-essential openssl、zlibc minizip、libidn11-dev libidn11 等。在安装完成之后清除以前编译的残留文件，使用 make menuconfig 命令配置内核。对内核的配置保留默认值不做修改。完成编译内核，编译模块，安装内核散步操作，注意终端输出信息。

5 配置 grub 引导：计算机启动时，引导程序在对计算机系统进行初始化后，把操作系统的核心部分程序装入住存储器，修改修改/etc/default/grub 文件，注释掉 GRUB_HIDDEN_TIMEOUT=0，然后运行 update-grub 命令。

6 编写测试系统功能调用代码：利用事先记录的系统功能调用号在用户态完成对新的系统功能的调用。

2.4.2 实验调试及心得

1 实验调试：

完成对 sys.c 文件,syscall_64.tbl 文件，syscalls.h 文件的修改。如图 2.6 所示。

```
SYSCALL_DEFINE2(mysyscall,char* ,sourcefile, char* ,targetfile)
{
	char addr[10];
	int x,stream1,stream2;
	mm_segment_t old_fs = get_fs();
	set_fs(KERNEL_DS);
	stream1 = sys_open(sourcefile,O_RDONLY,0);
	stream2 = sys_open(targetfile,O_WRONLY|O_CREAT|O_TRUNC,0600);
	if(stream1 == -1 || stream2 == -1)
		return -1;
	do{
		x = sys_read(stream1,addr,10);
		sys_write(stream2,addr,x);
		printk("+");
	}
	while(x>0);
	sys_close(stream1);
	sys_close(stream2);
	set_fs(old_fs);
	return 1;
}
```

图 2.6 修改后的系统功能源码

安装编译内核需要的库，使用命令如下：

sudo apt-get install libncurses5-dev libssl-dev

sudo apt-get install build-essential openssl

sudo apt-get install zlibc minizip

sudo apt-get install libidn11-dev libidn11

执行配置内核指令，使用命令如下:

sudo make mrproper *//清除以前编译的残留文件*

sudo make clean

sudo make menuconfig *//配置内核*

在配置内核界面保留原设置退出，如图 2.7 所示。

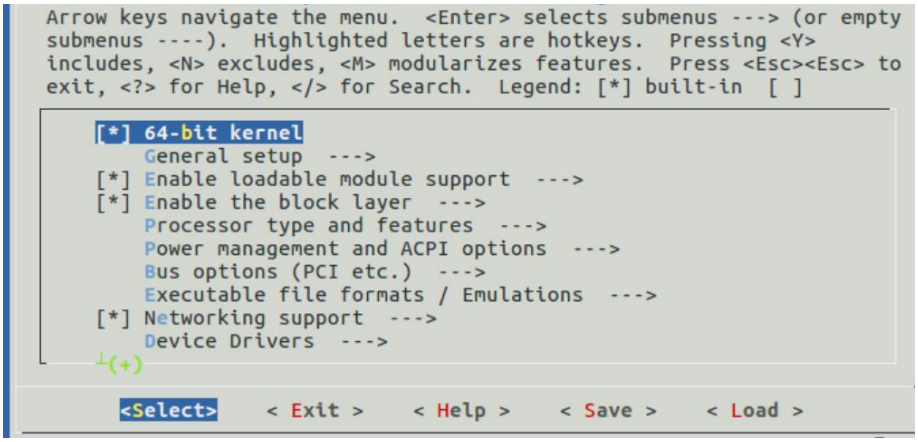

图 2.7 配置内核界面

编译内核，编译模块和安装内核，使用命令如下：

sudo make

sudo make modules

sudo make modules_install

sudo make install

完成 grub 配置文件的修改，如图 2.8 所示，重启计算机，在 grub 界面选择新编译的内核启动，如图 2.9 所示

```
GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
```

图 2.8 配置 grub 文件

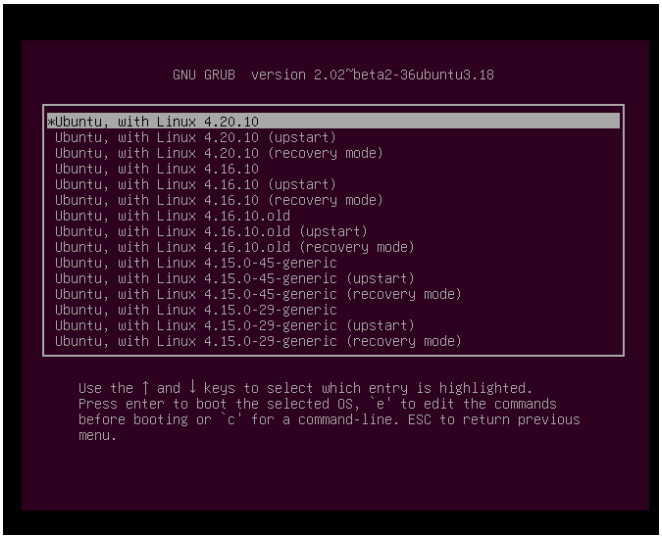

图 2.9 选择新编译的内核重新启动

运行测试程序，实现对文件的复制，文件复制结果如图 2.10 所示。测试系统功能调用结果正确。

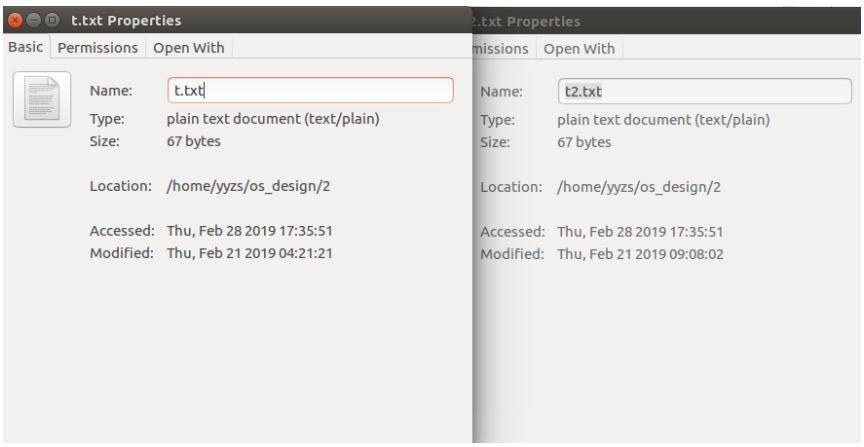

图 2.10 系统功能调用测试结果

2 实验心得:

本次实验第一次接触 Linux 内核，通过查询资料完成了对涉及系统功能调用的三个文件的修改，同时对操作系统引导程序也有了直观的认识，在编程中需要注意的是不同版本的 Linux 内核所使用的内核态库函数的调用方法有所不同，在设计时应当根据自己想要编译的内核的版本正确地使用内核态地函数调用。同时，本次实验地编译内核内核阶段会消耗大量时间，为了防止不必要地时间浪费，应当采用用户态单独调试功能，不添加新地系统功能调用编译内核，编译添加系统功能调用后地内核三步走地操作。

附录 实验代码

- design2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
    long x;
    x = syscall(333,"/home/yyzs/os_design/2/t.txt","/home/yyzs/os_design/2/t5.txt");
    if(x==1)
    printf("succeed!\n");
    else if(x==-1)
    printf("failed!\n");
    return 0;
}
```

- sys.c

```c
SYSCALL_DEFINE2(mysyscall,char* ,sourcefile, char* ,targetfile)
{
    char addr[10];
    int x,stream1,stream2;
    mm_segment_t old_fs = get_fs();
    set_fs(KERNEL_DS);
    stream1 = sys_open(sourcefile,O_RDONLY,0);
    stream2 = sys_open(targetfile,O_WRONLY|O_CREAT|O_TRUNC,0600);
    if(stream1 == -1 || stream2 == -1)
        return -1;
    do{
        x = sys_read(stream1,addr,10);
        sys_write(stream2,addr,x);
        printk("+");
    }
    while(x>0);
    sys_close(stream1);
    sys_close(stream2);
    set_fs(old_fs);
    return 1;
    }
```

3 实验三 掌握添加设备驱动程序的方法

3.1 实验目的

掌握添加设备驱动程序的方法。

3.2 实验内容

1 采用模块方法，添加一个新的设备驱动程序。
2 要求添加字符设备的驱动。
3 编写一个应用程序，测试添加的驱动程序

3.3 实验设计

3.3.1 开发环境

1 **虚拟机运行环境**：Microsoft Windows 10 64 位 1803；
2 **虚拟机软件版本**：VMware Workstation 12 Pro 12.5.7 build-5813279；
3 **虚拟机资源分配**：
a) 处理器：Intel® Core™ i5-8250U CPU @ 1.60GHz × 4；
b) 内存：3.8 GB NVMe SSD；
c) 磁盘：115GB USB3.0 Device；
4 **Linux 内核版本**：Linux version 4.16.10；
5 **Ubuntu 版本**：Ubuntu 5.4.0-6ubuntu1~16.04.10；
6 **GTK 版本**：GTK 2.24.3
7 **文本编辑器**：Gedit 3.18.3
8 **编译器**：GCC 5.4.0 20160609

3.3.2 实验设计

1 **总体设计**：

Linux 内核中的设备驱动程序是一组驻留内存的特权共享库，是一种低级硬件处理例程。对于用户程序，设备驱动程序隐藏设备的特定详细信息，并为各种

设备提供一致的界面。通常，设备映射到特殊设备文件，用户程序可以与其他文件相同。此设备文件运行。

Linux 支持三种类型的设备：字符设备，块设备和网络设备。

设备由主设备号和次设备号识别。主设备号唯一标识设备类型，即设备驱动程序类型，它是块设备表或字符设备表中设备表条目的索引。次设备号仅由设备驱动程序解释，通常用于在几个可能的硬件设备之间识别 I / O 请求中涉及的设备。

典型的驱动程序大致可分为几个部分：

a) 注册设备：

系统启动时或模块加载时，必须将设备注册到相应的设备阵列并返回设备的主设备号;

b) 定义功能函数：

对于每个驱动程序功能，有一些与此设备密切相关的功能。在最常用的块设备或字符设备中，有诸如 open()和 read()之类的操作。当系统调用这些调用时，将自动使用驱动程序函数中的特定模块。实现具体操作;

c) 卸载设备：

当不使用此设备时，可以将其卸载，主要是从/ proc 取消此设备的特殊文件。

本设计中完成添加设备驱动的总体设计流程图如图 3.1 所示。

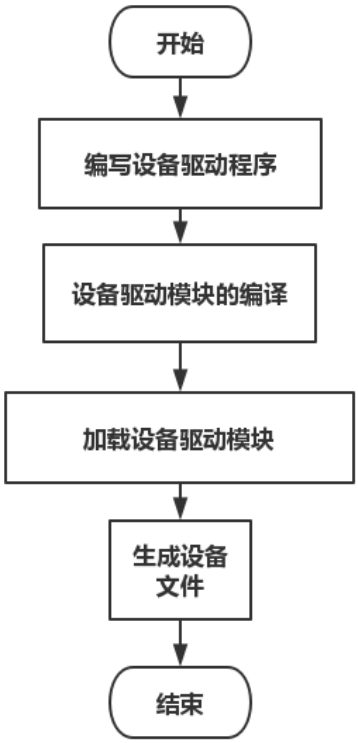

图 3.1 添加设备驱动总体设计流程图

2　详细设计：

a)　编写 Makefile 文件：本设计中可以使用通用的 Makefile 文件完成对驱动模块的编译。

b)　编写设备驱动程序：设备驱动程序主要模块图如图 3.2 所示。

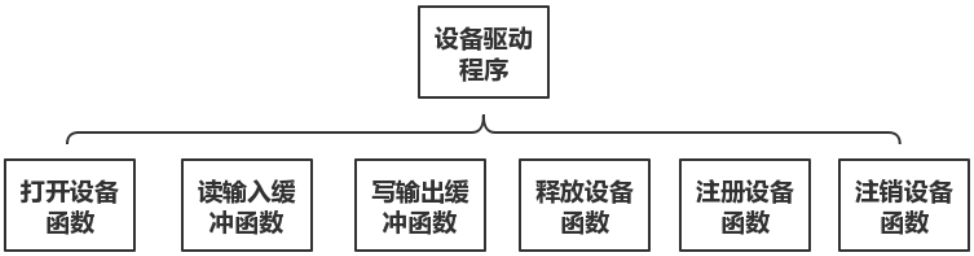

图 3.2 设备驱动程序主要模块图

c)　加载设备模块：insmod mydriver.ko 若加载成功，在文件/proc/devices 中能看到新增加的设备，包括设备名 mydriver 和主设备号。

d)　生成设备文件：mknod /dev/test c 254 0 其中，test 为设备文件名，254 为主设备号，0 为从设备号，c 表示字符设备。

3.4　实验调试

3.4.1　实验步骤

1　编写 Makefile 文件：调用 Makefile 文件之后，其具体过程如下：KERNELRELEASE 是在内核源码的顶层 Makefile 中定义的一个变量，在第一次读取执行此 Makefile 时，KERNELRELEASE 没有被定义，所以 make 将读取执行 else 之后的内容；如果 make 的目标是 clean，直接执行 clean 操作，然后结束。当 make 的目标为 all 时，-C $(KDIR)指明跳转到内核源码目录下读取那里的 Makefile；M=$(PWD) 表明然后返回到当前目录继续读入、执行当前的 Makefile。当从内核源码目录返回时，KERNELRELEASE 已被定义，内核的 build 程序 Kbuild 也被启动去解析 kbuild 语法的语句，make 将继续读取 else 之前的内容。else 之前的内容为 kbuild 语法的语句，指明模块源码中各文件的依赖关系，以及要生成的目标模块名。

2　编写设备功能函数：根据各个设备驱动模块的功能，编写设备驱动功能函数。

3　加载并安装设备：进入 Makefile 文件和 zcydriver.c 文件所在目录，清除 make 产生的残留文件。命令为：

make clean

删除先前可能加载过的模块，命令为：

rmmod /dev/mydriver

卸载设备：

rm /dev/mydriver

编译设备文件，产生模块文件

make

加载模块

insmod mydriver.ko

加载设备，分配设别号

mknod /dev/mydriver c 240 0

4 **运行测试程序**：测试程序首先实现向新的设备驱动文件写入文本信息，再把写入的文本信息从字符设备中读出，期间不断输出 dmesg 的信息，检测系统运行状态。

3.4.2 实验调试及心得

1 **实验调试**：

编写设备功能函数，编译完成后结果如图 3.3 所示。

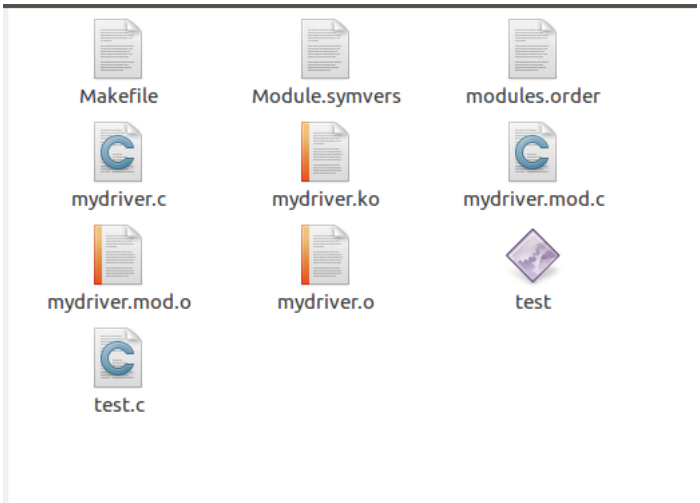

图 3.3 编译完成后结果

加载设备驱动模块如图 3.4 所示。

```
yyzs@ubuntu:~/os_design/3$ sudo insmod mydriver.ko
[sudo] password for yyzs:
yyzs@ubuntu:~/os_design/3$
```

图 3.4 加载设备驱动模块

在文件/proc/devices 中查看新增加的设备如图 3.5 所示，查看到设备名为

mydriver 主设备号为 240。

图 3.5 查看/proc/devices 中查看新增加的设备

生成设备文件，在/dev 中可以查看到设备文件 test，如图 3.6 所示。

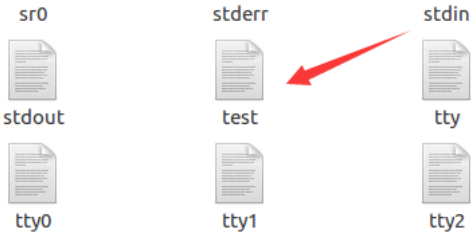

图 3.6 查看设备文件

运行测试程序结果如图 3.7 所示，程序运行结果符合预期，字符设备驱动添加成功。

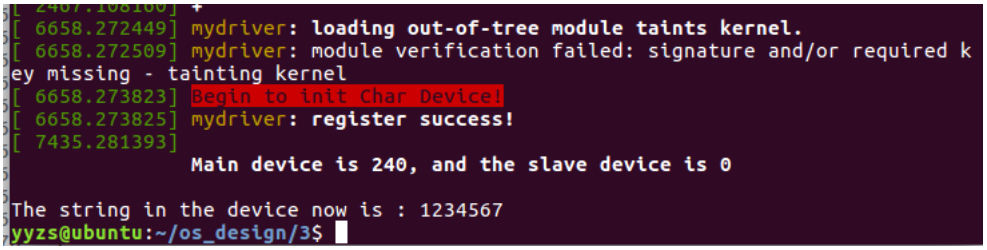

图 3.7 测试程序运行结果

2 实验心得：

本次实验接了模块化添加设备驱动的方法，对于 Linux 如何完成设备管理有了更进一步的认识。初次使用 MAKEFILE 文件进行模块的编译，同时学会了通过查看 demesg 输出的调试信息寻找程序中的 bug 的方法，这种方法在编译内核和添加设备驱动模块时都是极为有用的方法。

附录 实验代码

- **my_driver.c**

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/uaccess.h>

#if CONFIG_MODVERSIONS == 1
#define MODVERSIONS
#include <linux/version.h>
#endif

#define DEVICE_NUM 0 //随机产生一个设备号
```

```
static int device_num = 0; //用来保存创建成功后的设备号
static char buffer[1024] = "mydriver"; //数据缓冲区
static int open_nr = 0; //打开设备的进程数，用于内核的互斥

//函数声明
//inode：linux 下文件的管理号。
//file：linux 一切皆文件。文件结构体代表一个打开的文件，系统中的每个打开的文件在内核空间都有一个关联的 struct file。它由内核在打开文件时创建，并传递给在文件上进行操作的任何函数。在文件的所有实例都关闭后，内核释放这个数据结构
static int mydriver_open(struct inode *inode, struct file *filp);
static int mydriver_release(struct inode *inode, struct file* filp);
static ssize_t mydriver_read(struct file *file, char __user *buf, size_t count, loff_t *f_pos);
//loff-t:long long 型
static ssize_t mydriver_write(struct file *file, const char __user *buf, size_t count, loff_t
*f_pos); //__user 表明参数是一个用户空间的指针，不能在 kernel 代码中直接访问。
//size_t:一个基本的无符号整数的 C / C + +类型
//填充 file_operations 结构相关入口
static struct file_operations mydriver_fops = {
.read = mydriver_read,
.write = mydriver_write,
.open = mydriver_open,
.release = mydriver_release,
};

//打开函数
static int mydriver_open(struct inode *inode, struct file *filp)
{
    printk("\nMain device is %d, and the slave device is %d\n", MAJOR(inode->i_rdev),
MINOR(inode->i_rdev)); //把主从设备号传入
    if (open_nr == 0)
    {
        open_nr++;
        try_module_get(THIS_MODULE); //尝试打开模块
        return 0;
    }
    else
    {
        printk(KERN_ALERT "Another process open the char device.\n");//进程挂起
        return -1;
    }
}

//读函数
static ssize_t mydriver_read(struct file *file, char __user *buf, size_t count, loff_t *f_pos)
{
//if (buf == NULL) return 0;
    if (copy_to_user(buf, buffer, sizeof(buffer))) //读缓冲 ,第一个参数是 to：用户空间的地址，
第二个参数是 from，内核空间的地址，第三个参数是要从内核空间拷贝的字节数
    {
        return -1;
    }
    return sizeof(buffer);
}

//写函数,将用户的输入字符串写入
```

```
static ssize_t mydriver_write(struct file *file, const char __user *buf, size_t count, loff_t
*f_pos)
{
//if (buf == NULL) return 0;
    if (copy_from_user(buffer, buf, sizeof(buffer))) //写缓冲
    {
        return -1;
    }
    return sizeof(buffer);
}

//释放设备函数
static int mydriver_release(struct inode *inode, struct file* filp)
{
    module_put(THIS_MODULE); //释放模块
    open_nr--; //进程数减 1
    printk("The device is released!\n");
    return 0;
}

//注册设备函数
static int __init mydriver_init(void)
{
    int result;
    printk(KERN_ALERT "Begin to init Char Device!"); //注册设备
    //向系统的字符登记表登记一个字符设备
    result = register_chrdev(DEVICE_NUM, "mydriver", &mydriver_fops); //第一个参数等于 0，则
表示采用系统动态分配的主设备号；不为 0，则表示静态注册。 第二个参数命名， 第三个参数为其地址
    if (result < 0)
    {
        printk(KERN_WARNING "mydriver: register failure\n");
        return -1;
    }
    else
    {
        printk("mydriver: register success!\n");
        device_num = result;
        return 0;
    }
}

//注销设备函数
static void __exit mydriver_exit(void)
{
    printk(KERN_ALERT "Unloading...\n");
    unregister_chrdev(device_num, "mydriver"); //注销设备
    printk("unregister success!\n");
}

//模块宏定义
module_init(mydriver_init); //模块加载函数
module_exit(mydriver_exit); //设备卸载函数
MODULE_LICENSE("GPL"); // "GPL" 是指明了 这是 GNU General Public License 的任意版本
```

- **test.c**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define MAX_SIZE 1024
int main(void)
{
int fd;
char buf[MAX_SIZE];

char get[MAX_SIZE];

char devName[20], dir[49] = "/dev/";

system("ls /dev/");

printf("Please input the device's name you wanna to use :");

gets(devName);

strcat(dir, devName);

fd = open(dir, O_RDWR | O_NONBLOCK);

if (fd != -1)

{

read(fd, buf, sizeof(buf));

printf("The device was inited with a string : %s\n", buf);

 // 测试写

printf("Please input a string  :\n");

gets(get);

write(fd, get, sizeof(get));

//测试读

read(fd, buf, sizeof(buf));

system("dmesg");

printf("\nThe string in the device now is : %s\n", buf);

close(fd);

return 0;

}

else

{
```

```
printf("Device open failed\n");

return -1;}
}
```

4 实验四 /PROC 文件分析

4.1 实验目的

理解和分析/PROC 文件分析。

4.2 实验内容

1 了解/proc 文件的特点和使用方法
2 周期性监控系统状态，显示系统部件的使用情况
3 用图形界面监控系统状态，包括 CPU、内存、磁盘和网络使用情况（要求可排序）、进程信息等(可自己补充、添加其他功能)

4.3 实验设计

4.3.1 开发环境

1 **虚拟机运行环境**：Microsoft Windows 10 64 位 1803；
2 **虚拟机软件版本**：VMware Workstation 12 Pro 12.5.7 build-5813279；
3 **虚拟机资源分配**：
   a) 处理器：Intel® Core™ i5-8250U CPU @ 1.60GHz × 4；
   b) 内存：3.8 GB NVMe SSD；
   c) 磁盘：115GB USB3.0 Device；
4 **Linux 内核版本**：Linux version 4.16.10；
5 **Ubuntu 版本**：Ubuntu 5.4.0-6ubuntu1~16.04.10；
6 **GTK 版本**：GTK 2.24.3
7 **文本编辑器**：Gedit 3.18.3
8 **编译器**：GCC 5.4.0 20160609

4.3.2 实验设计

1 **总体设计**：本次设计可以分为 GUI 模块，文件读取模块与字符串处理模块，其中 GUI 模块由 GTK 实现，包括 CPU，内存使用情况的折线图动态实时更新；磁盘网络使用情况的动态输出；模拟实现进程管理器（包括进程按关键字排序，进程的关闭，进程使用资源情况的查看等）。文件读取模块根据需要获取的信息不同读取/proc 下不同文件的内容，且设置时间间隔对文件读取内容进行实时更

新，并将读取的结果作为输入参数传递到字符串处理模块中。字符串处理模块负责把文件读取模块中读取的动态信息保存到预先定义的结构体中，在结构体上实现对数据的各种处理，并将处理完毕的格式化数据传入 GUI 模块；同时在结构体基础上定义实现排序，刷新等功能，作为 GTK 时间响应的回调函数。完成各种基本功能的实现。本次实验中各个模块的关系图如图 4.1 所示。

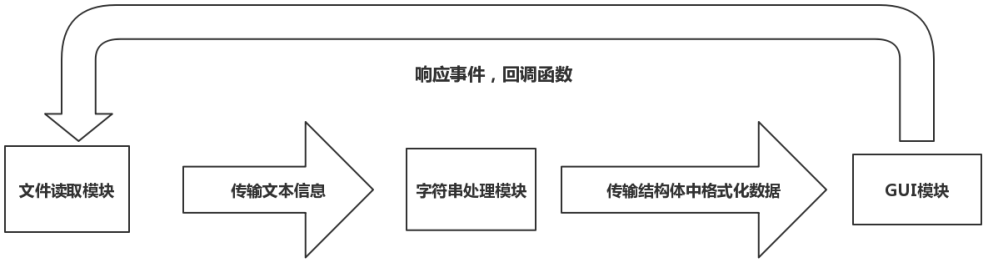

图 4.1 各个模块关系图

2 模块设计：

a) GUI 模块设计：GUI 设计基于 GTK 的 notebook 控件实现用户对于多个界面的切换，各个选项卡的标签为“SYSTEM”，“PROCESS”，“RESOURCE”，“NET&DISK”。分别表示系统信息，进程信息，资源信息，网络和磁盘信息。各个界面的 UI 设计概念图如图 4.2 至图 4.5 所示。

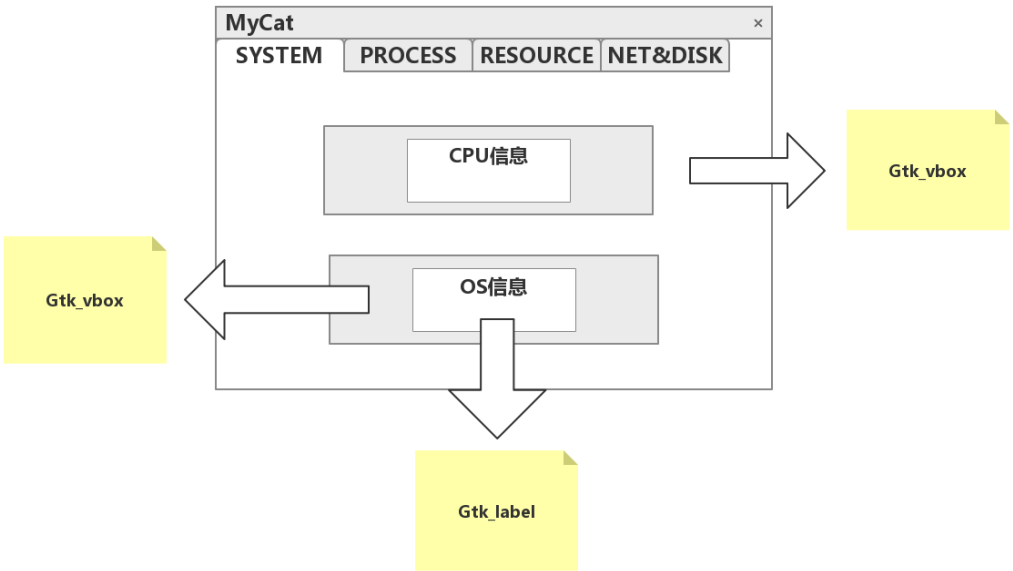

图 4.2 “SYSTEM”界面设计图

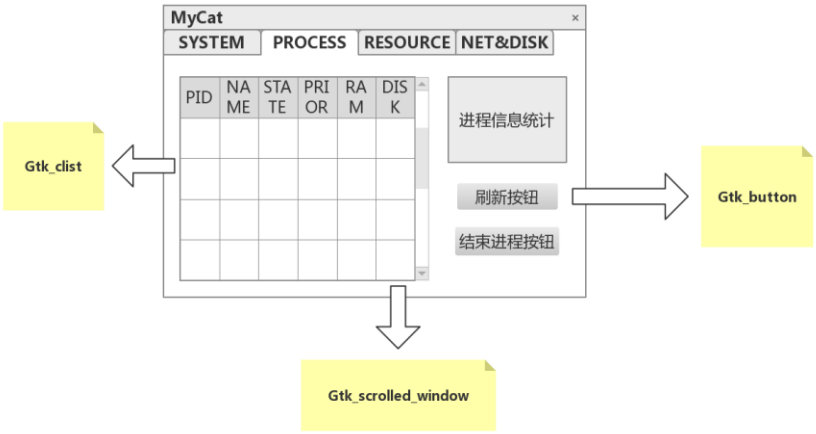

图 4.3 “PROCESS”界面设计图

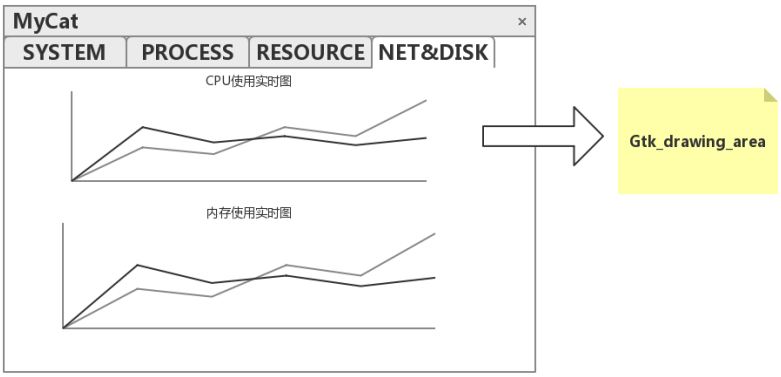

图 4.4 “RESOUCE”界面设计

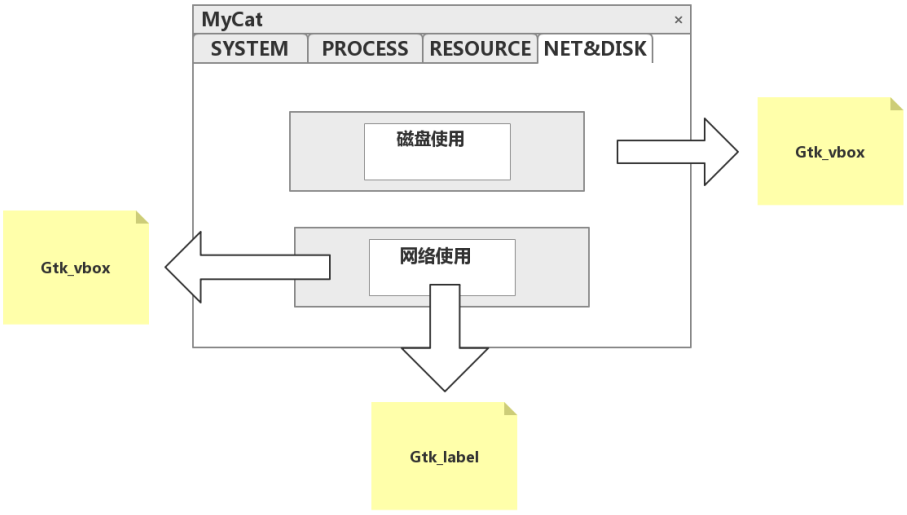

图 4.5 “NET&DISK”界面设计

b) 文件读取模块设计：本次实验读取的主要从/proc 文件中获取信息。Linux 的 proc 文件系统是进程文件系统和内核文件系统的组成的复合体，是将内核数据对象化为文件形式进行存取的一种内存文件系统，是监控内核的一种用户接口。它拥有一些特殊的文件（纯文本），从中可以获取系统状态信息。对 Linux 下的/proc 文件。

对于进程信息主要分析起关键作用的文件包括以下几个部分：/proc/cpuinfo：保存 CPU 信息，包括名称、型号、数目、主频、Cache 等；/proc/version：保存操作系统信息，包括系统版本、内核型号、GCC 版本等；/proc/stat：保存 CPU 活动信息数据，用来测算 CPU 使用率。

对于处理器和操作系统的分析起关键作用的文件包括以下几个部分：主机名：/proc/sys/kernel/hostname；系统启动时间：/proc/uptime；系统运行时间：/proc/uptime；系统版本号：/proc/sys/kernel/ostype , /proc/sys/kernel/osrelease；CPU 信息：/proc/cpuinfo；CPU 使用情况：/proc/stat；内存使用情况：/proc/meminfo。内存空间和交换空间实时大小可用来计算内存和交换分区使用率，MemTotal 表示内存空间总大小，MemFree 表示内存剩余空间大小，SwapTotal 表示交换分区总大小，SwapFree 表示交换分区剩余空间大小。据此可以算出内存和交换分区的使用率。

c) 字符串处理模块设计：字符串处理模块在每次读取文件信息时都能够使用到，主要目的是为了将文件中记录的信息格式化读取出，并存放在设计好的数据结构中，再在数据结构的基础上实现数据刷新以及数据排序等基本操作。本次实验设计的数据结构以进程信息为基础，具体设计见图表 4.1。

表 4.1 进程信息数据结构

名称	类型	占用空间	说明
pid	int	4B	进程的 id 号
name	char []	100B	进程名称
state	char	1B	进程状态
priority	int	4B	进程优先级
ram	int	4B	进程占用内存
disk	int	4B	进程占用磁盘
net	int	4B	进程占用网络

完成进程数据结构的设计后，对 int 类型的数据定义排序操作，对 char 类型的数据定义按字典序排序的操作，再在每次完成上述排序操作后调用数据刷新函数即完成了排序并显示操作。

4.4 实验调试

4.4.1 实验步骤

1 根据 GUI 界面设计图使用 Gtk 构建完成基本的操作界面框架搭建，每个视图界面对应 notebook 的一个页面；

2 根据需要读取的处理器和操作系统信息从/proc/sys/kernel/ostype 和/proc/cpuinfo 等文件中读取字符串信息，格式化处理后放入对应 Gtk_label 中进行格式化显示；

3 根据实验设计中设计的数据结构对进程信息文件读取并解析，并对进程信息的每一个要素设计排序函数，作为点击 Gtk_clist 的回调函数。使用数据结构中的 pid 作为接口使用系统功能调用 kill -9 实现对单个进程的中止操作。

4.4.2 实验调试及心得

1 实验调试：

首先测试“SYSTEM”选项卡对应界面，如图 4.6 所示。所显示的内容格式正确且为当前操作系统和处理器的正确的信息。

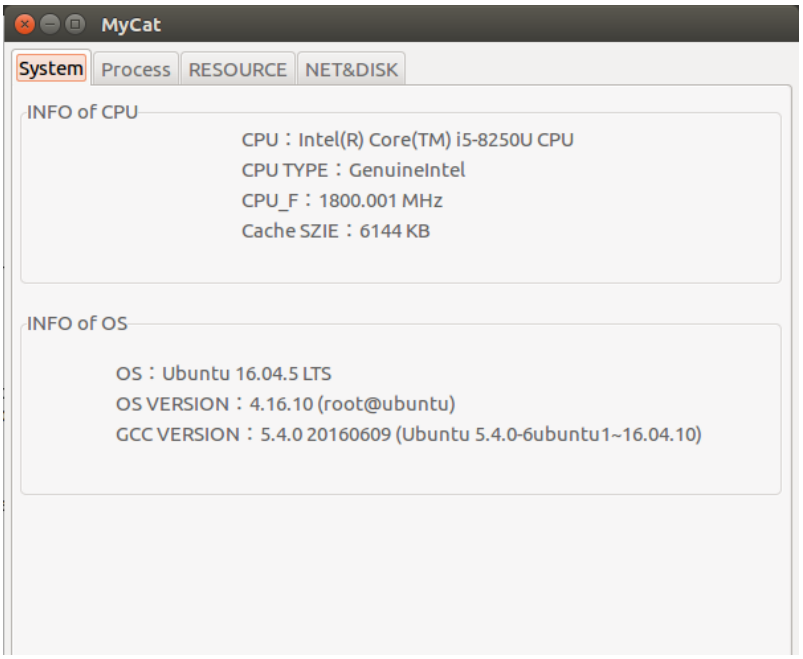

图 4.6 “SYSTEM”选项卡对应实际界面

测试“PROCESS”选项卡对应界面，首先初始化数据显示正确，如图 4.7 所示。

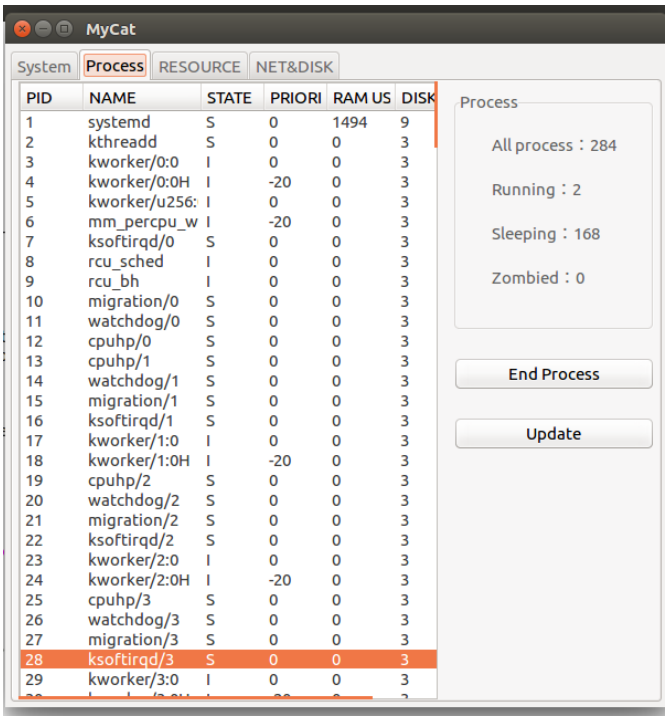

图 4.7 “PROCESS”选项卡对应实际界面

测试进程中止功能，选中要中止进程的 pid，点击“End Process”按钮，观察到所选中 pid 对应的进程（gedit）中止如图 4.8 和图 4.9 所示，该功能测试正确。

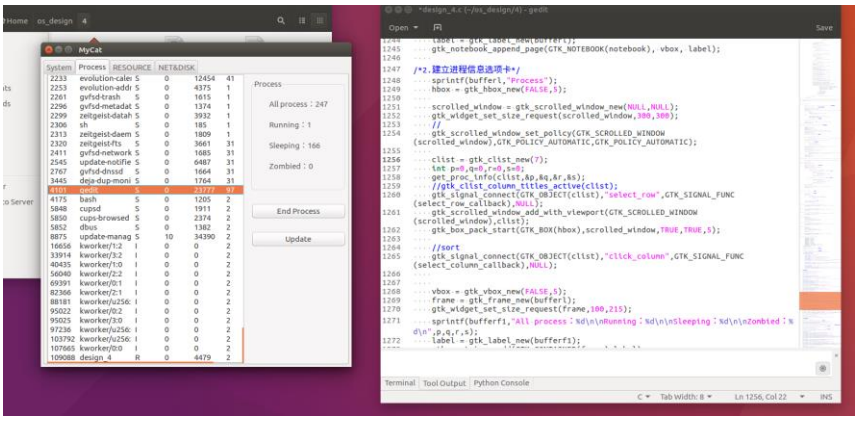

图 4.8 结束进程操作前

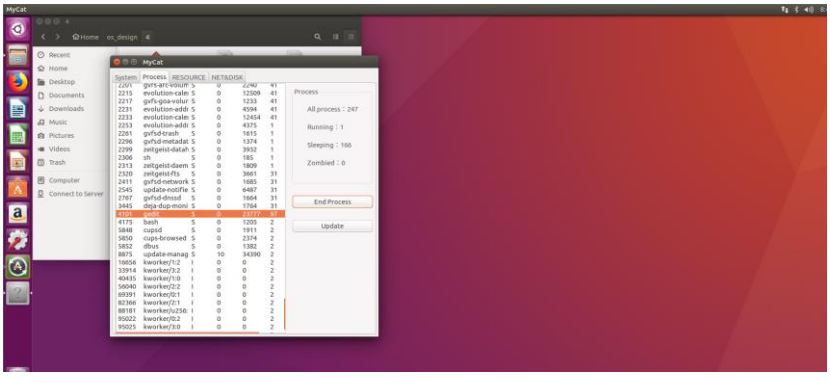

图 4.9 结束进程操作后

分别点击按“NAME”排序，按“PRIOR”排序，测试排序功能，如图 4.10 和图 4.11 所示。根据图示信息，排序功能测试正确。

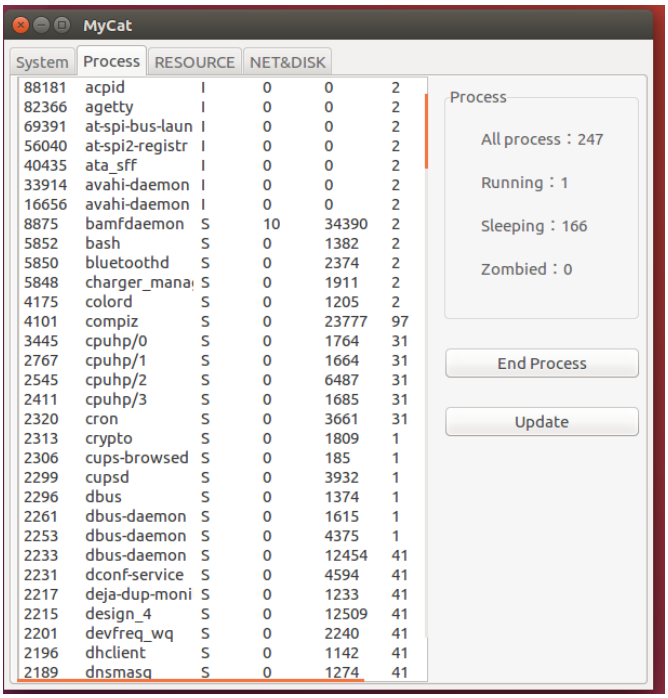

图 4.10 按“NAME”排序后的结果

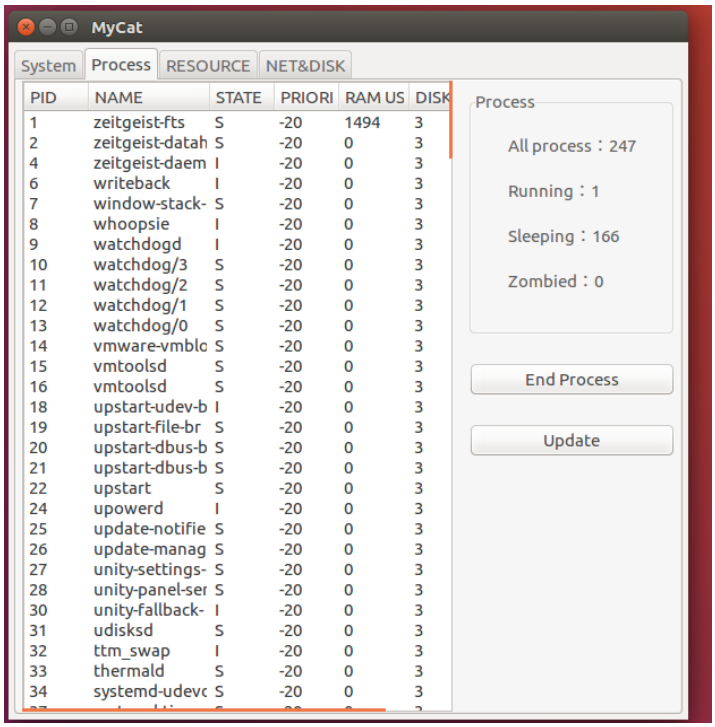

图 4.11 按“PRIORI”排序后的结果

测试“SOURCE”选项卡对应界面，如图 4.12 所示，动态显示 CPU 使用情况和内存使用情况的折线图。测试结果正确。

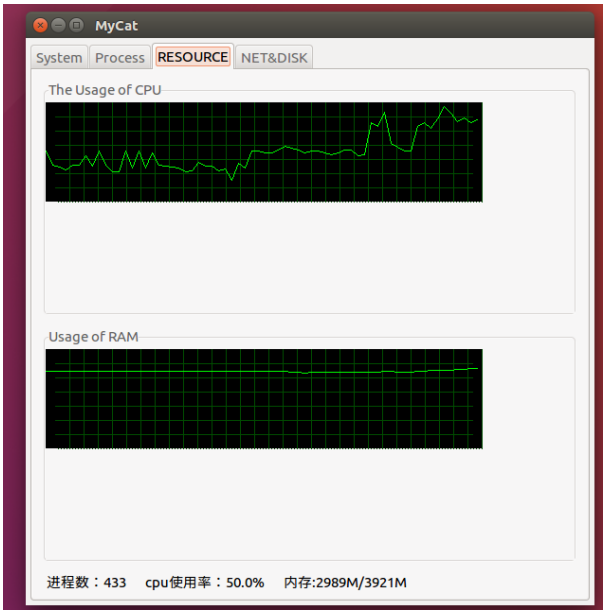

图 4.12 “RESOURCE”选项卡对应实际界面

测试“NET&DISK”选项卡对应界面，如图 4.13 所示，动态显示网络使用情况和磁盘使用情况。测试结果正确。

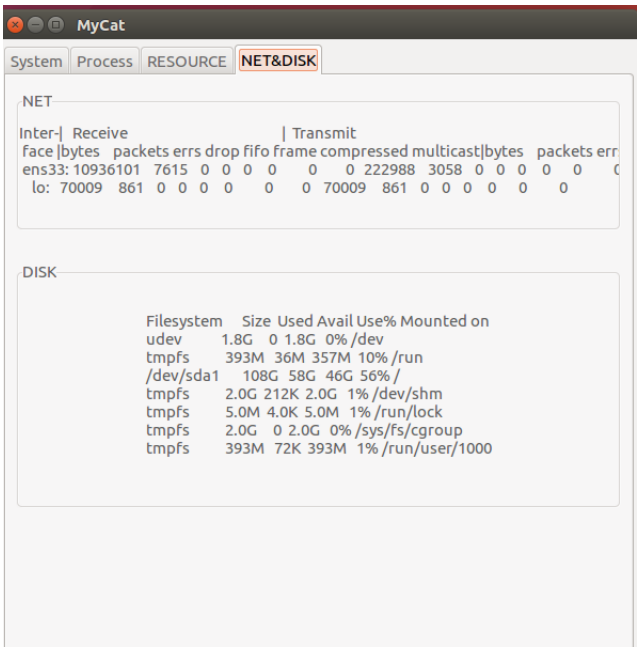

图 4.13 “NET&DISK”选项卡对应实际界面

2 实验心得：

本次实验主要的困难在于如何使用 Gtk 构架以及回调函数逻辑完成各种功能的可视化实现，实验中对于排序功能的设计最为复杂，原本方案为 C++运算符重载，但是考虑到各种不同进程信息排序的依据并不都相同，最后采用的方案是根据不同的数据类型单独设计排序函数，排序的方法是基于冒泡排序，效率有待进一步提高。同时，设计复杂的 Gtk 窗口也对 GUI 的设计有了初步的认识，未来

无论采用什么设计工具来完成 GUI 设计，都必不可少这种模块化设计的思想。

附录 实验代码

- design4.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gtk/gtk.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/vfs.h>
#include <sys/time.h>
#include <dirent.h>

typedef struct Info
{
    int pid;
    char name[100];
    char state[1];
    int priority;
    int ram;
    int disk;
    int net;
    //int flag = 0;
}info;

info pro_info[1000];
int n_info  =0;
int p=0,q=0,r=0,s=0;
static float cpu_used_percent=0;     /*cpu 使用率*/
static int cpu_start_position=15;    /*绘制 cpu 移动的线条*/
static float cpu_data[66];  /*cpu 历史数据*/
static int flag2=0;      /*初始化 cpu_data 数组中数据的标志*/
static int cpu_first_data=0;     /*第一个数据，既最早的数据，下一个要淘汰的数据*/
static long idle,total;      /*计算 cpu 时的数据*/
static int flag=0;       /*计算 cpu 使用率时启动程序的标志*/
static int flag1=0;      /*计算单个进程 cpu 使用率时使用的标志*/
static char temp_cpu[50];    /*cpu 使用率*/
static char temp_mem[50];    /*内存使用*/
static char temp_process[50];    /*进程数*/
static long mem_total;       /*内存总大小*/
static long mem_free;        /*空闲内存*/
static long long ustime[32768]; /*前一次记录的用户态和核心态的总时间*/
static long mtime[32768];    /*前一次记录的时刻*/
static float mem_data[66];  /*cpu 历史数据*/
static int flag3=0;      /*初始化 cpu_data 数组中数据的标志*/
static int mem_first_data=0;     /*第一个数据，既最早的数据，下一个要淘汰的数据*/
static int mem_start_position=15;    /*绘制内存移动的线条*/
char *txt_pid = NULL;
char temp_disc[1024];
char temp_nett[1024];
```

```
void select_row_callback(GtkWidget *clist,gint row,gint column, GdkEventButton *event,
gpointer data)
{
    gtk_clist_get_text(GTK_CLIST(clist),row,column,&txt_pid);
    printf("%s\n",txt_pid);
}



void start(GtkWidget* clist,gint co)
{
    int i = 0;
    int j = 0;
    int n;
    char text[7][100];
    gchar *txt[7];
    n = n_info;
    switch(co)
    {
        case 0:
            for(i = 0;i<n-1;i++)
            {
                for(j = 0;j<n-i-1;j++)
                {
                    if(pro_info[j].pid<pro_info[j+1].pid)
                    {
                        int temp_pid;
                        temp_pid=pro_info[j].pid;
                        pro_info[j].pid=pro_info[j+1].pid;
                        pro_info[j+1].pid=temp_pid;
                    }
                }
            }
            for(i = n-1;i>=0;i--)
            {
                sprintf(text[0],"%d",pro_info[i].pid);
                sprintf(text[1],"%s",pro_info[i].name);
                sprintf(text[2],"%s",pro_info[i].state);
                sprintf(text[3],"%d",pro_info[i].priority);
                sprintf(text[4],"%d",pro_info[i].ram);
                sprintf(text[5],"%d",pro_info[i].disk);
                sprintf(text[6],"%d",pro_info[i].net);
                txt[5]=text[5];
                txt[6]=text[6];
                 txt[0]=text[0];
                 txt[1]=text[1];
                 txt[2]=text[2];
                 txt[3]=text[3];
                 txt[4]=text[4];
                gtk_clist_append(GTK_CLIST(clist),txt);
            }
            break;
        case 1:
            for(i = 0;i<n-1;i++)
            {
                for(j = 0;j<n-i-1;j++)
```

```
        {
            if(strcmp(pro_info[j].name,pro_info[j+1].name)<0)
            {
                char temp_name[100];
                strcpy(temp_name,pro_info[j].name);
                strcpy(pro_info[j].name,pro_info[j+1].name);
                strcpy(pro_info[j+1].name,temp_name);
            }
        }
    }
    for(i = n-1;i>=0;i--)
    {
        sprintf(text[0],"%d",pro_info[i].pid);
        sprintf(text[1],"%s",pro_info[i].name);
        sprintf(text[2],"%s",pro_info[i].state);
        sprintf(text[3],"%d",pro_info[i].priority);
        sprintf(text[4],"%d",pro_info[i].ram);
        sprintf(text[5],"%d",pro_info[i].disk);
        sprintf(text[6],"%d",pro_info[i].net);
        txt[5]=text[5];
        txt[6]=text[6];
         txt[0]=text[0];
         txt[1]=text[1];
         txt[2]=text[2];
         txt[3]=text[3];
         txt[4]=text[4];
        gtk_clist_append(GTK_CLIST(clist),txt);
    }
    break;
case 2:
    for(i = 0;i<n-1;i++)
    {
        for(j = 0;j<n-i-1;j++)
        {
            if(strcmp(pro_info[j].state,pro_info[j+1].state)>0)
            {
                char temp_state[100];
                strcpy(temp_state,pro_info[j].state);
                strcpy(pro_info[j].state,pro_info[j+1].state);
                strcpy(pro_info[j+1].state,temp_state);
            }
        }
    }
    for(i = n-1;i>=0;i--)
    {
        sprintf(text[0],"%d",pro_info[i].pid);
        sprintf(text[1],"%s",pro_info[i].name);
        sprintf(text[2],"%s",pro_info[i].state);
        sprintf(text[3],"%d",pro_info[i].priority);
        sprintf(text[4],"%d",pro_info[i].ram);
        sprintf(text[5],"%d",pro_info[i].disk);
        sprintf(text[6],"%d",pro_info[i].net);
        txt[5]=text[5];
        txt[6]=text[6];
         txt[0]=text[0];
         txt[1]=text[1];
         txt[2]=text[2];
```

```
            txt[3]=text[3];
            txt[4]=text[4];
            gtk_clist_append(GTK_CLIST(clist),txt);
        }
        break;
    case 3:
        for(i = 0;i<n-1;i++)
        {
            for(j = 0;j<n-i-1;j++)
            {
                if(pro_info[j].priority>pro_info[j+1].priority)
                {
                    int temp_pri;
                    temp_pri=pro_info[j].priority;
                    pro_info[j].priority=pro_info[j+1].priority;
                    pro_info[j+1].priority=temp_pri;
                }
            }
        }
        for(i = 0;i<n;i++)
        {
            sprintf(text[0],"%d",pro_info[i].pid);
            sprintf(text[1],"%s",pro_info[i].name);
            sprintf(text[2],"%s",pro_info[i].state);
            sprintf(text[3],"%d",pro_info[i].priority);
            sprintf(text[4],"%d",pro_info[i].ram);
            sprintf(text[5],"%d",pro_info[i].disk);
            sprintf(text[6],"%d",pro_info[i].net);
            txt[5]=text[5];
            txt[6]=text[6];
            txt[0]=text[0];
            txt[1]=text[1];
            txt[2]=text[2];
            txt[3]=text[3];
            txt[4]=text[4];
            gtk_clist_append(GTK_CLIST(clist),txt);
        }
        break;
    case 4:
        for(i = 0;i<n-1;i++)
        {
            for(j = 0;j<n-i-1;j++)
            {
                if(pro_info[j].ram<pro_info[j+1].ram)
                {
                    int temp_ram;
                    temp_ram=pro_info[j].ram;
                    pro_info[j].ram=pro_info[j+1].ram;
                    pro_info[j+1].ram=temp_ram;
                }
            }
        }
        for(i = 0;i<n;i++)
        {
            sprintf(text[0],"%d",pro_info[i].pid);
            sprintf(text[1],"%s",pro_info[i].name);
            sprintf(text[2],"%s",pro_info[i].state);
```

```
        sprintf(text[3],"%d",pro_info[i].priority);
        sprintf(text[4],"%d",pro_info[i].ram);
        sprintf(text[5],"%d",pro_info[i].disk);
        sprintf(text[6],"%d",pro_info[i].net);
        txt[5]=text[5];
        txt[6]=text[6];
         txt[0]=text[0];
         txt[1]=text[1];
         txt[2]=text[2];
         txt[3]=text[3];
         txt[4]=text[4];
        gtk_clist_append(GTK_CLIST(clist),txt);
    }
    break;
case 5:
    for(i = 0;i<n-1;i++)
    {
        for(j = 0;j<n-i-1;j++)
        {
            if(pro_info[j].disk<pro_info[j+1].disk)
            {
                int temp_disk;
                temp_disk=pro_info[j].disk;
                pro_info[j].disk=pro_info[j+1].disk;
                pro_info[j+1].disk=temp_disk;
            }
        }
    }
    for(i = 0;i<n;i++)
    {
        sprintf(text[0],"%d",pro_info[i].pid);
        sprintf(text[1],"%s",pro_info[i].name);
        sprintf(text[2],"%s",pro_info[i].state);
        sprintf(text[3],"%d",pro_info[i].priority);
        sprintf(text[4],"%d",pro_info[i].ram);
        sprintf(text[5],"%d",pro_info[i].disk);
        sprintf(text[6],"%d",pro_info[i].net);
        txt[5]=text[5];
        txt[6]=text[6];
         txt[0]=text[0];
         txt[1]=text[1];
         txt[2]=text[2];
         txt[3]=text[3];
         txt[4]=text[4];
        gtk_clist_append(GTK_CLIST(clist),txt);
    }
    break;
case 6:
    for(i = 0;i<n-1;i++)
    {
        for(j = 0;j<n-i-1;j++)
        {
            if(pro_info[j].net<pro_info[j+1].net)
            {
                int temp_net;
                temp_net=pro_info[j].net;
                pro_info[j].net=pro_info[j+1].net;
```

```
                    pro_info[j+1].net=temp_net;
                }
            }
        }
        for(i = 0;i<n;i++)
        {
            sprintf(text[0],"%d",pro_info[i].pid);
            sprintf(text[1],"%s",pro_info[i].name);
            sprintf(text[2],"%s",pro_info[i].state);
            sprintf(text[3],"%d",pro_info[i].priority);
            sprintf(text[4],"%d",pro_info[i].ram);
            sprintf(text[5],"%d",pro_info[i].disk);
            sprintf(text[6],"%d",pro_info[i].net);
            txt[5]=text[5];
            txt[6]=text[6];
             txt[0]=text[0];
             txt[1]=text[1];
             txt[2]=text[2];
             txt[3]=text[3];
             txt[4]=text[4];
            gtk_clist_append(GTK_CLIST(clist),txt);
        }
        break;
    default: break;
  }
}

void select_column_callback(GtkWidget *clist, gint column)
{
    gtk_clist_clear(GTK_CLIST(clist));
    start(clist, column);
}

gint delete_event( GtkWidget *widget, GdkEvent *event, gpointer data )
{
    gtk_main_quit ();
    return FALSE;
}

void kill_proc(void)
{
    char buf[20];
     sprintf(buf,"kill -9 %s",txt_pid);
    system(buf);
}

void refresh(GtkWidget *clist)
{
    p = 0;
    q = 0;
    r = 0;
    s = 0;
        DIR *dir;
        struct dirent *ptr;
        int i,j;
        FILE *fp;
        char buf[1024];
```

```
    char bufy[1024];

    char _buffer[1024];
    char _buffery[1024];

    char *buffery = _buffery;
    char *buffer=_buffer;
    char *buffer2;
    char proc_pid[1024];
    char proc_name[1024];
    char proc_stat[1024];
    char proc_pri[1024];
    char proc_takeup[1024];
    char disk_1[1024];
    char disk_2[1024];
    char net[1024];
    char text[7][1024];
    gchar *txt[7];
    int y = 0;
gtk_clist_clear(GTK_CLIST(clist));

dir=opendir("/proc");

    while(ptr=readdir(dir)){
           if((ptr->d_name)[0]>=48&&(ptr->d_name)[0]<=57){
                  p++;
                  sprintf(buf,"/proc/%s/stat",ptr->d_name);
                   sprintf(bufy,"/proc/%s/net/dev",ptr->d_name);



                   FILE * fpy = fopen(bufy,"r");
                   int i3;
                   for(i3 = 0; i3<3;i3++)
                   {
                       fgets(buffery,1024,fpy);
                   }
                   fclose(fpy);
                   int j3 = 0,j4 = 0;
                   for(j3 = 0,j4 = 0;j4 != 2;j3++)
                   {
                       while(buffery[j3]==' ')
                       {
                           j3++;
                       }
                       j4+=1;
                       if(j4==2)
                       {
                           break;
                       }
                       while(buffery[j3]!=' ')
                       {
                           j3++;
                       }
                   }
                   for(j4=0;buffery[j3]!=' ';j3++)
```

```
 {
     net[j4]=buffery[j3];
     j4+=1;
 }
 net[j4] = '\0';



fp=fopen(buf,"r");
fgets(buffer,1024,fp);
fclose(fp);
int i1,j1;
for(i1=0,j1=0;i1<1024&&j1<9;i1++)
{
    if(buffer[i1]==' ') j1++;
}
i1++;
for(j1=0;;i1++)
{
    if(buffer[i1]==' ')
    {
        disk_1[j1]='\0';
        break;
    }
    else disk_1[j1] = buffer[i1];
    j1++;
}


for(i1=0,j1=0;i1<1024&&j1<11;i1++)
{
    if(buffer[i1]==' ') j1++;
}
i1++;
for(j1=0;;i1++)
{
    if(buffer[i1]==' ')
    {
        disk_1[j1]='\0';
        break;
    }
    else disk_2[j1] = buffer[i1];
    j1++;
}

for(i=0;i<1024;i++){
    if(buffer[i]==' ') break;
}
buffer[i]='\0';
strcpy(proc_pid,buffer);
i+=2;
buffer+=i;
for(i=0;i<1024;i++){
    if(buffer[i]==')') break;
}
buffer[i]='\0';
strcpy(proc_name,buffer);
```

```
        i+=2;
        buffer2=buffer+i;
        buffer2[1]='\0';
        strcpy(proc_stat,buffer2);
        for(i=0,j=0;i<1024&&j<15;i++){
            if(buffer2[i]==' ') j++;
        }
        buffer2+=i;
        for(i=0;i<1024;i++){
            if(buffer2[i]==' ') break;
        }
        buffer2[i]='\0';
        strcpy(proc_pri,buffer2);
        for(j=0;i<1024&&j<4;i++){
            if(buffer2[i]==' ') j++;
        }
        buffer2+=i;
        for(i=0;i<1024;i++){
            if(buffer2[i]==' ') break;
        }
        buffer2[i]='\0';
        strcpy(proc_takeup,buffer2);

        if(!strcmp(proc_stat,"R")) q++;
        if(!strcmp(proc_stat,"S")) r++;
         if(!strcmp(proc_stat,"Z")) s++;

        sprintf(text[0],"%s",proc_pid);
        pro_info[y].pid = atoi(proc_pid);
        sprintf(text[1],"%s",proc_name);
        sprintf(pro_info[y].name,"%s",proc_name);
        sprintf(text[2],"%s",proc_stat);
        sprintf(pro_info[y].state,"%s",proc_stat);
        sprintf(text[3],"%s",proc_pri);
        pro_info[y].priority = atoi(proc_pri);

         sprintf(text[4],"%s",proc_takeup);
         pro_info[y].ram = atoi(proc_takeup);
         int disk = atoi(disk_1)+atoi(disk_2);
         pro_info[y].disk = disk;
         sprintf(text[5],"%d",disk);
         sprintf(text[6],"%s",net);
         pro_info[y].net = atoi(net);
         y+=1;

         txt[0]=text[0];
         txt[1]=text[1];
         txt[2]=text[2];
         txt[3]=text[3];
         txt[4]=text[4];
         txt[5]=text[5];
         txt[6]=text[6];


        gtk_clist_append(GTK_CLIST(clist),txt);
}
n_info = y;
```

```
        }
        closedir(dir);
}

void refresh_lable(GtkWidget *label)
{
    char bufff[1000];
        sprintf(bufff,"All
process: %d\n\nRunning: %d\n\nSleeping: %d\n\nZombied: %d\n",p,q,r,s);
        gtk_label_set_text(GTK_LABEL(label),bufff);
        gtk_widget_show(label);
}

char *get_cpu_name(char *_buf1)
{

    FILE * fp;
    int i=0;
    char *buf1=_buf1;

        fp=fopen("/proc/cpuinfo","r");
    for(i=0;i<5;i++){
    fgets(buf1,256,fp);
    }
    for(i=0;i<256;i++){
    if(buf1[i]==':') break;
    }
    i+=2;
    buf1+=i;
    buf1[31]='\0';
    fclose(fp);
    return buf1;
}

char *get_cpu_type(char *_buf2)
{

    FILE * fp;
    int i=0;
    char *buf2=_buf2;

        fp=fopen("/proc/cpuinfo","r");
    for(i=0;i<2;i++){
    fgets(buf2,256,fp);
    }
    for(i=0;i<256;i++){
    if(buf2[i]==':') break;
    }
    i+=2;
    buf2+=i;
    buf2[12]='\0';
    fclose(fp);
    return buf2;
}

char *get_cpu_f(char *_buf3)
{
```

```
    FILE * fp;
    int i=0;
    char *buf3=_buf3;

        fp=fopen("/proc/cpuinfo","r");
    for(i=0;i<8;i++){
    fgets(buf3,256,fp);
    }
    for(i=0;i<256;i++){
    if(buf3[i]==':') break;
    }
    i+=2;
    buf3+=i;
    buf3[8]='\0';
    fclose(fp);
    return buf3;
}

char *get_cache_size(char *_buf4)
{

    FILE * fp;
    int i=0;
    char *buf4=_buf4;

    fp=fopen("/proc/cpuinfo","r");
    for(i=0;i<9;i++){
    fgets(buf4,256,fp);
    }
    for(i=0;i<256;i++){
    if(buf4[i]==':') break;
    }
    i+=2;
    buf4+=i;
    buf4[10]='\0';
    fclose(fp);
    return buf4;
}

char *get_system_type(char *_buf1)
{
    FILE * fp;
       int i=0;
    char *buf1=_buf1;

       //fp=fopen("/proc/version","r");
       fp=fopen("/etc/issue","r");
       fgets(buf1,256,fp);
       for(i=0;i<256;i++){
    if(buf1[i]=='\\') break;
    }
       buf1[i]='\0';
       fclose(fp);
       return buf1;
}
```

```
char *get_system_version(char *_buf2)
{
    FILE * fp;
       int i=0;
       int j=0;
    char *buf2=_buf2;

       fp=fopen("/proc/version","r");
       fgets(buf2,256,fp);
       for(i=0,j=0;i<256&&j<2;i++){
    if(buf2[i]==' ') j++;
    }
       buf2+=i;
       for(i=0;i<256;i++){
    if(buf2[i]==')') break;
    }
       buf2[i+1]='\0';
       fclose(fp);
       return buf2;
}

char *get_gcc_version(char *_buf3)
{
    FILE * fp;
       int i=0;
       int j=0;
    char *buf3=_buf3;

       fp=fopen("/proc/version","r");
       fgets(buf3,256,fp);
       for(i=0,j=0;i<256&&j<6;i++){
    if(buf3[i]==' ') j++;
    }
       buf3+=i;
       for(i=0;i<256;i++){
    if(buf3[i]==')') break;
    }
       buf3[i+1]='\0';
       fclose(fp);
       return buf3;
}

gint cpu_record_draw(GtkWidget *widget)/*cpu 使用记录绘图函数*/
{
    int i;
    int my_first_data;
    GdkColor color;
    GdkDrawable *canvas;
    GdkGC *gc;
    GdkFont *font;
    canvas = widget->window;
    gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];

    gdk_draw_rectangle(canvas, gc, TRUE, 0, 5, 460, 105);
    color.red = 0;
    color.green = 20000;
    color.blue = 0;
```

```
    gdk_gc_set_rgb_fg_color(gc, &color);
    for(i=20;i<110;i+=15)   /*绘制横线*/
    {
        gdk_draw_line(canvas, gc, 10, i, 450, i);
    }

    for(i=10;i<450;i+=15)
    {
        gdk_draw_line(canvas, gc, i+cpu_start_position,5,
i+cpu_start_position,110);
    }
    cpu_start_position-=3;
    if(cpu_start_position==0)  cpu_start_position=15;

    if(flag2==0)    /*第一次清空数据*/
    {
        for(i=0;i<66;i++)
            cpu_data[i]=0;
        flag2=1;
        cpu_first_data=0;
    }

    cpu_data[cpu_first_data]=cpu_used_percent/100;
    cpu_first_data++;
    if(cpu_first_data==66) cpu_first_data=0;

    color.red = 0;
    color.green = 65535;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);

    my_first_data=cpu_first_data;
    for(i=0;i<65;i++)
    {
        gdk_draw_line(canvas,gc,i*7,110-104*cpu_data[my_first_data%66],(i+1)*7,110-104*c
pu_data[(my_first_data+1)%66]);
        my_first_data++;
        if(my_first_data==66)   my_first_data=0;
    }
    color.red = 0;
    color.green = 0;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    return TRUE;
}

void get_proc_info(GtkWidget *clist,int *p,int *q,int *r,int *s)
{
        DIR *dir;
        struct dirent *ptr;
        int i,j;
        FILE *fp;
        char buf[1024];

        char bufy[1024];

        char _buffer[1024];
```

```
char _buffery[1024];

char *buffery = _buffery;
char *buffer=_buffer;
char *buffer2;
char proc_pid[1024];
char proc_name[1024];
char proc_stat[1024];
char proc_pri[1024];
char proc_takeup[1024];
char disk_1[1024];
char disk_2[1024];
char net[1024];
char text[7][1024];
gchar *txt[7];
int y = 0;
gtk_clist_set_column_title(GTK_CLIST(clist),0,"PID");
gtk_clist_set_column_title(GTK_CLIST(clist),1,"NAME");
 gtk_clist_set_column_title(GTK_CLIST(clist),2,"STATE");
 gtk_clist_set_column_title(GTK_CLIST(clist),3,"PRIORITY");
 gtk_clist_set_column_title(GTK_CLIST(clist),4,"RAM USAGE");
 gtk_clist_set_column_title(GTK_CLIST(clist),5,"DISK");
 gtk_clist_set_column_title(GTK_CLIST(clist),6,"NET");
gtk_clist_set_column_width(GTK_CLIST(clist),0,50);
gtk_clist_set_column_width(GTK_CLIST(clist),1,100);
gtk_clist_set_column_width(GTK_CLIST(clist),2,50);
gtk_clist_set_column_width(GTK_CLIST(clist),3,50);
gtk_clist_set_column_width(GTK_CLIST(clist),4,55);
gtk_clist_set_column_width(GTK_CLIST(clist),6,55);

gtk_clist_column_titles_show(GTK_CLIST(clist));
dir=opendir("/proc");

while(ptr=readdir(dir)){
       if((ptr->d_name)[0]>=48&&(ptr->d_name)[0]<=57){
              (*p)++;
              sprintf(buf,"/proc/%s/stat",ptr->d_name);
               sprintf(bufy,"/proc/%s/net/dev",ptr->d_name);



               FILE * fpy = fopen(bufy,"r");
               int i3;
               for(i3 = 0; i3<3;i3++)
               {
                   fgets(buffery,1024,fpy);
               }
               fclose(fpy);
               int j3 = 0,j4 = 0;
               for(j3 = 0,j4 = 0;j4 != 2;j3++)
               {
                   while(buffery[j3]==' ')
                   {
                       j3++;
                   }
                   j4+=1;
                   if(j4 == 2)
```

```
        {
            break;
        }
        while(buffery[j3]!=' ')
        {
            j3++;
        }
    }
    for(j4=0;buffery[j3]!=' ';j3++)
    {
        net[j4]=buffery[j3];
        j4+=1;
    }
    net[j4] = '\0';



fp=fopen(buf,"r");
fgets(buffer,1024,fp);
fclose(fp);
int i1,j1;
for(i1=0,j1=0;i1<1024&&j1<9;i1++)
{
    if(buffer[i1]==' ') j1++;
}
i1++;
for(j1=0;;i1++)
{
    if(buffer[i1]==' ')
    {
        disk_1[j1]='\0';
        break;
    }
    else disk_1[j1] = buffer[i1];
    j1++;
}


for(i1=0,j1=0;i1<1024&&j1<11;i1++)
{
    if(buffer[i1]==' ') j1++;
}
i1++;
for(j1=0;;i1++)
{
    if(buffer[i1]==' ')
    {
        disk_1[j1]='\0';
        break;
    }
    else disk_2[j1] = buffer[i1];
    j1++;
}

for(i=0;i<1024;i++){
    if(buffer[i]==' ') break;
}
```

```
buffer[i]='\0';
strcpy(proc_pid,buffer);
i+=2;
buffer+=i;
for(i=0;i<1024;i++){
    if(buffer[i]==')') break;
}
buffer[i]='\0';
strcpy(proc_name,buffer);
i+=2;
buffer2=buffer+i;
buffer2[1]='\0';
strcpy(proc_stat,buffer2);
for(i=0,j=0;i<1024&&j<15;i++){
    if(buffer2[i]==' ') j++;
}
buffer2+=i;
for(i=0;i<1024;i++){
    if(buffer2[i]==' ') break;
}
buffer2[i]='\0';
strcpy(proc_pri,buffer2);
for(j=0;i<1024&&j<4;i++){
    if(buffer2[i]==' ') j++;
}
buffer2+=i;
for(i=0;i<1024;i++){
    if(buffer2[i]==' ') break;
}
buffer2[i]='\0';
strcpy(proc_takeup,buffer2);

if(!strcmp(proc_stat,"R")) (*q)++;
if(!strcmp(proc_stat,"S")) (*r)++;
 if(!strcmp(proc_stat,"Z")) (*s)++;

sprintf(text[0],"%s",proc_pid);
pro_info[y].pid = atoi(proc_pid);
sprintf(text[1],"%s",proc_name);
sprintf(pro_info[y].name,"%s",proc_name);
sprintf(text[2],"%s",proc_stat);
sprintf(pro_info[y].state,"%s",proc_stat);
sprintf(text[3],"%s",proc_pri);
pro_info[y].priority = atoi(proc_pri);

 sprintf(text[4],"%s",proc_takeup);
 pro_info[y].ram = atoi(proc_takeup);
 int disk = atoi(disk_1)+atoi(disk_2);
 pro_info[y].disk = disk;
 sprintf(text[5],"%d",disk);
 sprintf(text[6],"%s",net);
 pro_info[y].net = atoi(net);
 y+=1;

 txt[0]=text[0];
 txt[1]=text[1];
 txt[2]=text[2];
```

```
                        txt[3]=text[3];
                        txt[4]=text[4];
                        txt[5]=text[5];
                        txt[6]=text[6];


                        gtk_clist_append(GTK_CLIST(clist),txt);
                }
                n_info = y;
        }
        closedir(dir);
}

gboolean cpu_record_callback(GtkWidget *widget,GdkEventExpose *event, gpointer  data)
{
    gtk_timeout_add(100,(GtkFunction)cpu_record_draw,(gpointer)widget);
    return TRUE;
}




char* stat_read()
{
    long  user_t, nice_t, system_t, idle_t,total_t; /*此次读取的数据*/
    long total_c,idle_c;    /*此次数据与上次数据的差*/
    char cpu_t[10],buffer[70+1];
    int fd;
    fd=open("/proc/stat",O_RDONLY);
    read(fd,buffer,70);
    sscanf(buffer, "%s %ld %ld %ld %ld", cpu_t, &user_t, &nice_t, &system_t, &idle_t);

    if(flag==0)
    {
        flag=1;
        idle=idle_t;
        total=user_t+nice_t+system_t+idle_t;
        cpu_used_percent=0;
    }
    else
    {
        total_t=user_t+nice_t+system_t+idle_t;
        total_c=total_t-total;
        idle_c=idle_t-idle;
        cpu_used_percent=100*(total_c-idle_c)/total_c;
        total=total_t;  /*此次数据保存*/
        idle=idle_t;
    }
    close(fd);
    sprintf(temp_cpu,"cpu 使用率：%0.1f%%",cpu_used_percent);
    //puts(temp_cpu);
    return temp_cpu;
}

gint mem_record_draw(GtkWidget *widget) /*内存记录绘图函数*/
{
    int i;
```

```
    int my_first_data;
    GdkColor color;
    GdkDrawable *canvas;
    GdkGC *gc;
    GdkFont *font;
    canvas = widget->window;
    gc = widget->style->fg_gc[GTK_WIDGET_STATE(widget)];

    gdk_draw_rectangle(canvas, gc, TRUE, 0, 5, 460, 105);
    color.red = 0;
    color.green = 20000;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    for(i=20;i<110;i+=15)   /*绘制横线*/
    {
        gdk_draw_line(canvas, gc, 10, i, 450, i);
    }

    for(i=10;i<450;i+=15)
    {
        gdk_draw_line(canvas, gc, i+mem_start_position,5,
i+mem_start_position,110);
    }
    mem_start_position-=3;
    if(mem_start_position==0)  mem_start_position=15;

    if(flag3==0)    /*第一次清空数据*/
    {
        for(i=0;i<66;i++)
            mem_data[i]=0;
        flag3=1;
        mem_first_data=0;
    }

    mem_data[mem_first_data]=(float)(mem_total-mem_free)/mem_total;
    mem_first_data++;
    if(mem_first_data==66) mem_first_data=0;

    color.red = 0;
    color.green = 65535;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);

    my_first_data=mem_first_data;
    for(i=0;i<65;i++)
    {
        gdk_draw_line(canvas,gc,i*7,109-104*mem_data[my_first_data%66],(i+1)*7,109-104*m
em_data[(my_first_data+1)%66]);
        my_first_data++;
        if(my_first_data==66)   my_first_data=0;
    }
    color.red = 0;
    color.green = 0;
    color.blue = 0;
    gdk_gc_set_rgb_fg_color(gc, &color);
    return TRUE;
}
```

```
gboolean mem_record_callback(GtkWidget *widget,GdkEventExpose *event,gpointer data) /*内存记录回调函数*/
{
    gtk_timeout_add(100,(GtkFunction)mem_record_draw,(gpointer)widget);
    return TRUE;
}

char* meminfo_read()
{
    char buffer[100+1];
    char data[20];
    long total=0,free=0;    /*总内存和用户内存*/
    int i=0,j=0,counter=0;
    int fd;
    fd=open("/proc/meminfo",O_RDONLY);
    read(fd,buffer,100);

    for(i=0,j=0;i<100;i++,j++)
    {
        if (buffer[i]==':') counter++;
        if (buffer[i]==':'&&counter==1) /*MemTotal 总内存*/
        {
            while(buffer[++i]==' ');
            for(j=0;j<20;j++,i++)
            {
                if(buffer[i]=='k')  break;
                data[j]=buffer[i];
            }
            data[--j]='\0';
            total=atol(data)/1024;
        }

        if (buffer[i]==':'&&counter==2) /*MemFree 空闲内存*/
        {
            while(buffer[++i]==' ');
            for(j=0;j<20;j++,i++)
            {
                if(buffer[i]=='k')  break;
                data[j]=buffer[i];
            }
            data[--j]='\0';
            free+=atol(data)/1024;
        }
    }
    mem_total=total;
    mem_free=free;
    sprintf(temp_mem,"内存:%ldM/%ldM",total-free,total);
    close(fd);
    return temp_mem;
}



char* procsum_read()    /*进程数*/
{
    int i,sum=0;    /*进程总数*/
```

```
    int fd;
    char path[30];

    for(i=1;i<32768;i++)
    {
        sprintf(path,"/proc/%d/statm",i);
        if( !((fd=open(path,O_RDONLY))<0))
        {
            sum++;
            close(fd);
        }
    }
    sprintf(temp_process,"进程数：%d",sum);
    //puts(temp_process);
    return temp_process;
}


gint process_refresh(gpointer process_label)
{
    gtk_label_set_text(GTK_LABEL(process_label),procsum_read());
    gtk_widget_show(process_label);
    return TRUE;
}


gint cpu_refresh(gpointer cpu_label)
{
    gtk_label_set_text(GTK_LABEL(cpu_label),stat_read());
    gtk_widget_show(cpu_label);
    return TRUE;
}


gint mem_refresh(gpointer mem_label)
{
    gtk_label_set_text(GTK_LABEL(mem_label),meminfo_read());
    gtk_widget_show(mem_label);
    return TRUE;
}


char* disc_read(){
    FILE *fp;
    char txt[100];
    temp_disc[0] = '\0';
    system("df -hl > disc.txt");
    fp = fopen("disc.txt","r");
    while((fgets(txt,100,fp))!=NULL){
        strcat(temp_disc,txt);
    }
    fclose(fp);
    return temp_disc;
}

gint disc_refresh(gpointer disc_label){
    gtk_label_set_text(GTK_LABEL(disc_label), disc_read());
```

```
    gtk_widget_show(disc_label);
    return TRUE;
}


char* net_read(){
    FILE *fp;
    char txt[100];
    temp_nett[0] = '\0';
    fp = fopen("/proc/net/dev","r");
    while((fgets(txt,100,fp))!=NULL){
        strcat(temp_nett,txt);
    }
    fclose(fp);
    return temp_nett;
}

gint net_refresh(gpointer net_label){
    gtk_label_set_text(GTK_LABEL(net_label), net_read());
    gtk_widget_show(net_label);
    return TRUE;
}


int main(int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button1;
    GtkWidget *button2;
    GtkWidget *button3;
    GtkWidget *table;
    GtkWidget *notebook;
    GtkWidget *label;
    GtkWidget *frame;
    GtkWidget *frame2;
    GtkWidget *clist;
    GtkWidget *cpu_hbox;
    GtkWidget *mem_hbox;
    GtkWidget *cpu_record_drawing_area;
    GtkWidget *cpu_record;
    GtkWidget *mem_record;
    GtkWidget *mem_record_drawing_area;
    GtkWidget *vbox;
    GtkWidget *hbox;
    GtkWidget *process_label;   /*进程数*/
    GtkWidget *cpu_label;   /*cpu 使用率*/
    GtkWidget *mem_label;   /*内存使用情况*/
    char buf1[256],buf2[256],buf3[256],buf4[256];
    char bufferl[10];
    char bufferf1[1000];
    char bufferf2[1000];
    char bufferf3[1000];
    GtkWidget *scrolled_window;

    //new window
    gtk_init(&argc, &argv);
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
    //set title
    gtk_window_set_title(GTK_WINDOW(window), "MyCat");
    //set size
    gtk_widget_set_size_request(window,600,600);
    //set signal
    g_signal_connect (G_OBJECT(window), "delete_event",G_CALLBACK(delete_event), NULL);
    //set border
    gtk_container_set_border_width(GTK_CONTAINER(window),5);


    //new table
    table = gtk_table_new(3,6,FALSE);//YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
    //add table to window
    gtk_container_add(GTK_CONTAINER(window),table);


    //new notebook
    notebook = gtk_notebook_new();
    //set position
    gtk_notebook_set_tab_pos(GTK_NOTEBOOK(notebook), GTK_POS_TOP);
    //attach  notebook to table
    gtk_table_attach_defaults(GTK_TABLE(table),notebook,0,6,0,1);//YYYYYYYYYYYYYYYYY
    gtk_widget_show(notebook);


/*1.建立系统信息选项卡*/
    sprintf(bufferf1, "INFO of CPU");
    sprintf(bufferf2,"INFO of OS");
    sprintf(bufferl, "System");

    //new vbox
    vbox = gtk_vbox_new(FALSE,0);


    //new frame bufferf1 as title
    frame = gtk_frame_new(bufferf1);
    gtk_container_set_border_width(GTK_CONTAINER(frame),5);
    //set frame size
    gtk_widget_set_size_request(frame,500,150);
    sprintf(bufferf1, "    CPU: %s\n    CPU TYPE: %s\n    CPU_F: %s MHz\n    Cache
SZIE: %s\n",get_cpu_name(buf1),get_cpu_type(buf2),get_cpu_f(buf3),get_cache_size(buf4));
    //set label
    label = gtk_label_new(bufferf1);
    //add label to frame
    gtk_container_add(GTK_CONTAINER(frame),label);
    gtk_widget_show(label);
    gtk_box_pack_start(GTK_BOX(vbox),frame,FALSE,FALSE,5);
    gtk_widget_show(frame);

    //new frame2 bufferf2 as title
    frame2 = gtk_frame_new(bufferf2);
    gtk_container_set_border_width(GTK_CONTAINER(frame2),5);
    gtk_widget_set_size_request(frame2,500,150);
    sprintf(bufferf2, "    OS: %s\n    OS VERSION: %s\n    GCC
VERSION: %s\n",get_system_type(buf1),get_system_version(buf2),get_gcc_version(buf3));
    label = gtk_label_new(bufferf2);
    gtk_container_add(GTK_CONTAINER(frame2),label);
```

```
    gtk_widget_show(label);
    gtk_box_pack_start(GTK_BOX(vbox),frame2,FALSE,FALSE,5);
    gtk_widget_show(frame2);

    gtk_widget_show(vbox);
    //set page of notebook
    label = gtk_label_new(bufferl);
    gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

/*2.建立进程信息选项卡*/
    sprintf(bufferl,"Process");
    hbox = gtk_hbox_new(FALSE,5);

    scrolled_window = gtk_scrolled_window_new(NULL,NULL);
    gtk_widget_set_size_request(scrolled_window,300,300);
    //

gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled_window),GTK_POLICY_AUTOMATIC
,GTK_POLICY_AUTOMATIC);

    clist = gtk_clist_new(7);
    int p=0,q=0,r=0,s=0;
    get_proc_info(clist,&p,&q,&r,&s);
    //gtk_clist_column_titles_active(clist);

gtk_signal_connect(GTK_OBJECT(clist),"select_row",GTK_SIGNAL_FUNC(select_row_callback),N
ULL);
    gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrolled_window),clist);
    gtk_box_pack_start(GTK_BOX(hbox),scrolled_window,TRUE,TRUE,5);

    //sort

gtk_signal_connect(GTK_OBJECT(clist),"click_column",GTK_SIGNAL_FUNC(select_column_callba
ck),NULL);


    vbox = gtk_vbox_new(FALSE,5);
    frame = gtk_frame_new(bufferl);
    gtk_widget_set_size_request(frame,100,215);
    sprintf(bufferf1,"All
process：%d\n\nRunning：%d\n\nSleeping：%d\n\nZombied：%d\n",p,q,r,s);
    label = gtk_label_new(bufferf1);
    gtk_container_add(GTK_CONTAINER(frame),label);
    gtk_box_pack_start(GTK_BOX(vbox),frame,FALSE,FALSE,10);
    button1 = gtk_button_new_with_label("End Process");
    g_signal_connect(G_OBJECT(button1),"clicked",G_CALLBACK(kill_proc),"End Process");
    gtk_box_pack_start(GTK_BOX(vbox),button1,FALSE,FALSE,10);

    button2 = gtk_button_new_with_label("Update");
    g_signal_connect_swapped(G_OBJECT (button2), "clicked",G_CALLBACK (refresh), clist);
    g_signal_connect_swapped (G_OBJECT (button2), "clicked",G_CALLBACK (refresh_lable),
label);
    gtk_box_pack_start(GTK_BOX(vbox), button2, FALSE, FALSE, 10);

    gtk_box_pack_start(GTK_BOX(hbox), vbox, TRUE, TRUE, 5);

    gtk_widget_show_all(hbox);
```

```
    label = gtk_label_new(bufferl);
    gtk_notebook_append_page(GTK_NOTEBOOK(notebook),hbox,label);

/* 3.建立内存资源选项卡 */
    GtkWidget *capability;
    capability = gtk_vbox_new(FALSE,0); /*内存资源*/
    gtk_container_set_border_width(GTK_CONTAINER(capability),5);
    gtk_widget_set_size_request(capability,200,320);
    gtk_widget_show(capability);
    label = gtk_label_new("RESOURCE");
    gtk_notebook_append_page(GTK_NOTEBOOK(notebook),capability,label);
    gtk_notebook_set_current_page(GTK_NOTEBOOK(notebook),3);    /*把此页设为当前显示页*/
    cpu_hbox = gtk_hbox_new(FALSE,0);       /*cpu 横向 box*/
    gtk_box_pack_start(GTK_BOX(capability),cpu_hbox,TRUE,TRUE,2);
    gtk_widget_show(cpu_hbox);
    mem_hbox = gtk_hbox_new(FALSE,0);       /*mem 横向 box*/
    gtk_box_pack_start(GTK_BOX(capability),mem_hbox,TRUE,TRUE,2);
    gtk_widget_show(mem_hbox);

    //begin to draw
    //draw cpu
    cpu_record = gtk_frame_new("The Usage of CPU");
    gtk_container_set_border_width(GTK_CONTAINER(cpu_record),5);
    gtk_widget_set_size_request(cpu_record,1000,130);
    gtk_widget_show(cpu_record);
    gtk_box_pack_start(GTK_BOX(cpu_hbox),cpu_record,TRUE,TRUE,2);
    cpu_record_drawing_area = gtk_drawing_area_new();
    gtk_widget_set_size_request(cpu_record_drawing_area,50,50);
    //g_signal_connect(G_OBJECT(cpu_record_drawing_area),
"expose_event",G_CALLBACK(cpu_record_callback),NULL);
    gtk_timeout_add(1000,(GtkFunction)cpu_record_draw,(gpointer)cpu_record_drawing_area);
    gtk_container_add(GTK_CONTAINER(cpu_record),cpu_record_drawing_area);
    gtk_widget_show(cpu_record_drawing_area);

    //draw ram
    mem_record = gtk_frame_new("Usage of RAM");
    gtk_container_set_border_width(GTK_CONTAINER(mem_record),5);
    gtk_widget_set_size_request(mem_record,1000,130);
    gtk_widget_show(mem_record);
    gtk_box_pack_start(GTK_BOX(mem_hbox),mem_record,TRUE,TRUE,2);

    mem_record_drawing_area = gtk_drawing_area_new ();
    gtk_widget_set_size_request (mem_record_drawing_area, 50,50);
    //g_signal_connect (G_OBJECT(mem_record_drawing_area),
"expose_event",G_CALLBACK(mem_record_callback),NULL);
    gtk_timeout_add(1000,(GtkFunction)mem_record_draw,(gpointer)mem_record_drawing_area);
    gtk_container_add (GTK_CONTAINER(mem_record), mem_record_drawing_area);
    gtk_widget_show (mem_record_drawing_area);

    //text
    hbox = gtk_hbox_new(FALSE,0);
    gtk_box_pack_start(GTK_BOX(capability),hbox,FALSE,FALSE,2);
    gtk_widget_show(hbox);

    process_label = gtk_label_new("");
    cpu_label = gtk_label_new("");
```

```
    mem_label = gtk_label_new("");

    gtk_timeout_add(100,(GtkFunction)process_refresh,(gpointer)process_label);  /*进程数
刷新*/
    gtk_timeout_add(100,(GtkFunction)cpu_refresh,(gpointer)cpu_label);  /*cpu 使用率刷新*/
    gtk_timeout_add(100,(GtkFunction)mem_refresh,(gpointer)mem_label);  /*内存使用刷新*/


    gtk_label_set_justify(GTK_LABEL(process_label),GTK_JUSTIFY_RIGHT);
    gtk_label_set_justify(GTK_LABEL(cpu_label),GTK_JUSTIFY_RIGHT);
    gtk_label_set_justify(GTK_LABEL(mem_label),GTK_JUSTIFY_RIGHT);
    gtk_box_pack_start(GTK_BOX(hbox),process_label,FALSE,FALSE,10);
    gtk_box_pack_start(GTK_BOX(hbox),cpu_label,FALSE,FALSE,10);
    gtk_box_pack_start(GTK_BOX(hbox),mem_label,FALSE,FALSE,10);
    gtk_widget_show(process_label);
    gtk_widget_show(cpu_label);
    gtk_widget_show(mem_label);


    //4
   /* GtkWidget * disk_label;

    GtkWidget *diskbox = gtk_hbox_new(FALSE,0);
    GtkWidget *netbox = gtk_hbox_new(FALSE,0);
    GtkWidget *whole = gtk_vbox_new(FALSE,0);
    GtkWidget *net_label = gtk_label_new("NET STATE");

    gtk_container_set_border_width(GTK_CONTAINER(whole),5);
    gtk_widget_set_size_request(whole,200,320);
    gtk_widget_show(whole);
    label = gtk_label_new("DISK&NET");
    gtk_notebook_append_page(GTK_NOTEBOOK(notebook),whole,label);
    gtk_notebook_set_current_page(GTK_NOTEBOOK(notebook),4);



    disk_label = gtk_label_new("DISK STATE");
    gtk_box_pack_start(GTK_BOX(whole),diskbox,FALSE,FALSE,10);
    gtk_timeout_add(1000,(GtkFunction)disc_refresh,(gpointer)disk_label);
    gtk_box_pack_start(GTK_BOX(diskbox),disk_label,FALSE,FALSE,10);
    gtk_widget_show(disk_label);

    net_label = gtk_label_new("DISK STATE");
    gtk_box_pack_start(GTK_BOX(whole),netbox,FALSE,FALSE,10);
    gtk_timeout_add(1000,(GtkFunction)net_refresh,(gpointer)net_label);
    gtk_box_pack_start(GTK_BOX(netbox),net_label,FALSE,FALSE,10);
    gtk_widget_show(net_label);


    gtk_widget_show(table);
    gtk_widget_show(window);
    gtk_main();*/

    char buff[1024];


    GtkWidget *frame1;
```

```
    GtkWidget *net_label;
    //网络信息选项卡
    sprintf(buff,"NET");
    vbox = gtk_vbox_new(FALSE, 0);

    frame1 = gtk_frame_new(buff);
    gtk_container_set_border_width(GTK_CONTAINER(frame1), 10);
    gtk_widget_set_size_request(frame1, 500, 150);
    net_label = gtk_label_new("");
    gtk_timeout_add(100,(GtkFunction)net_refresh,(gpointer)net_label);
    gtk_container_add(GTK_CONTAINER(frame1), net_label);
    gtk_widget_show(net_label);
    gtk_box_pack_start(GTK_BOX(vbox), frame1, FALSE, FALSE, 5);
    gtk_widget_show(frame1);

    gtk_widget_show(vbox);



   GtkWidget *frame3;
    GtkWidget *disk_label;
    //GtkWidget *vbox2;
    //网络信息选项卡
    sprintf(buff,"DISK");
    //vbox2 = gtk_vbox_new(FALSE, 0);

    frame3 = gtk_frame_new(buff);
    gtk_container_set_border_width(GTK_CONTAINER(frame3), 10);
    gtk_widget_set_size_request(frame3, 500, 250);
    disk_label = gtk_label_new("");
    gtk_timeout_add(100,(GtkFunction)disc_refresh,(gpointer)disk_label);
    gtk_container_add(GTK_CONTAINER(frame3), disk_label);
    gtk_widget_show(disk_label);
    gtk_box_pack_start(GTK_BOX(vbox), frame3, FALSE, FALSE, 5);
    gtk_widget_show(frame3);

    gtk_widget_show(vbox);


    label = gtk_label_new("NET&DISK");
    gtk_notebook_append_page(GTK_NOTEBOOK(notebook), vbox, label);

    gtk_widget_show(table);
    gtk_widget_show(window);
    gtk_main();


    return 0;

}
```