



华中科技大学

操作系统课程设计报告

姓 名： 许荣仁

学 院： 计算机科学与技术

专 业： 计算机科学与技术

班 级： CS1409

学 号： U201414803

指导教师： 谢美意

分数	
教师签名	

2017 年 3 月 14 日

目录

实验一.....	- 4 -
1 实验目的.....	- 4 -
2 实验内容.....	- 4 -
3 实验设计.....	- 4 -
4 实验调试.....	- 5 -
实验二.....	- 7 -
1 实验目的.....	- 7 -
2 实验内容.....	- 7 -
3 实验设计.....	- 7 -
4 实验调试.....	- 8 -
实验三.....	- 10 -
1 实验目的.....	- 10 -
2 实验内容.....	- 10 -
3 实验设计.....	- 10 -
4 实验调试.....	- 11 -
实验四.....	- 13 -
1 实验目的.....	- 13 -
2 实验内容.....	- 13 -
3 实验设计.....	- 13 -
4 实验调试.....	- 15 -
附录 实验代码.....	- 20 -
实验一.....	- 20 -
实验二.....	- 27 -
实验三.....	- 28 -
实验四.....	- 31 -

实验一

1 实验目的

熟悉和理解 Linux 编程环境

2 实验内容

(1) 编写一个 C 程序，用 read、write 等系统调用实现文件拷贝功能。命令形式：

`copy <源文件名> <目标文件名>`

(2) 编写一个 C 程序，使用图形编程库 (QT/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系统时间，一个窗口循环显示 0 到 9，一个窗口做 1
到 1000 的累加求和，刷新周期均为 1 秒)。

3 实验设计

3.1 平台环境

实验平台为 deepin（基于 debian）15.3 64 位，内容（2）使用的开发工具为 Qt
creator。

3.2 方案设计

(1) 内容 1 用了两种方法，方法 1 在上学期做过一次实验，是使用两个线程完
成的，一个线程 1 负责读，线程 2 负责写；方法 2 则是仅仅用一个简单的循环，
读一次文件，写一次文件，直到把整个文件全部复制完成。

(2) 内容 2 需要显示三个窗口，需要使用三个进程分别控制，由于是使用 Qt，
所以要定义三个类，分别有定时刷新的方法，然后再 main 中使用。

4 实验调试

4.1 实验步骤

内容 1：（1）按照方案设计把实现代码，然后编译调试。

内容 2：（1）学习基本的 Qt 的用法，能够使用 label 等组件

（2）根据实验的要求，先设计出一个类，能够显示时间，再进一步学习使用槽函数，让 label 能够自动刷新；

（3）计算 CPU 利用率需要读取/proc/stat 文件，并把其中的内容读取出来，取一个时间间隔来计算当前的 CPU 利用率；

（4）完成另外两个类，并在一个 main 函数中使用这三个类，分别在三个进程中显示三个窗口；

（5）调整窗口的位置和大小。

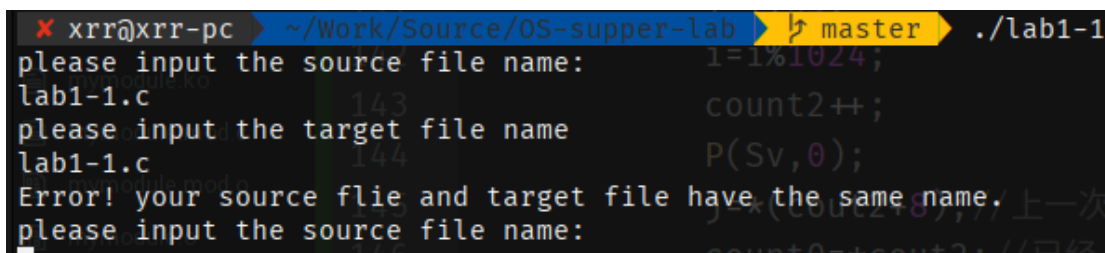
4.2 实验调试及结果

（1）内容 1：源文件为 lab1.c，目标文件为 new.c，运行程序以后，比较两个文件的内容，是一样的，结果如图 1-1 所示，内容 1 运行正确；

lab1-1.c	new.c
1 /*	1 /*
2 #include <stdio.h>	2 #include <stdio.h>
3 #include <stdlib.h>	3 #include <stdlib.h>
4 #include <string.h>	4 #include <string.h>
5	5
6 int main() {	6 int main() {
7 char src_name[255],	7 char src_name[255],
• tgt_name[255],	• tgt_name[255],
• swap[1024];	• swap[1024];
8 FILE *fp_src,	8 FILE *fp_src,
• *fp_tgt;// fp source	• *fp_tgt;// fp source
• and dp target;	• and dp target;
9 int j = 1024;//read	9 int j = 1024;//read
• 1024 bytes one time;	• 1024 bytes one time;
10 int flag_name_check =	10 int flag_name_check =
• 1;	• 1;
11 do {	11 do {
12 printf("please input	12 printf("please input

图 1-1 实验一内容 1 运行结果

源文件与目标文件同名时的错误提示，如图 1-2 所示：



```
xrr@xrr-pc ~/Work/Source/OS-supper-lab master ./lab1-1
please input the source file name:
lab1-1.c
please input the target file name:
lab1-1.c
Error! your source flie and target file have the same name.
please input the source file name:
```

图 1-2 实验一内容 1 错误提示

(2) 内容 2：查看三个窗口的内容，窗口 1 显示系统时间，刷新时间 1s，窗口 2 显示实时 CPU 利用率，刷新时间 2s，窗口 3 显示 1 到 100 累加求和的结果，刷新时间 3s，运行结果如图 1-3 所示：

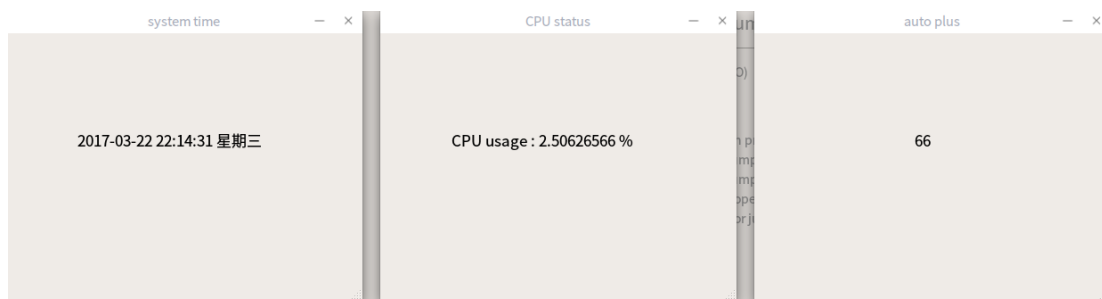


图 1-3 实验一内容 2 运行结果

4.3 实验心得

第一次实验还是比较简单的，内容 1 逻辑很简单，就是打开两个文件，读取源文件，写入目标文件，用循环来完成整个操作，最后关闭文件就可以了；难度在内容 2 上，主要原因是之前没有接触过 Qt/GTK 等图形编程库，所以第一次使用感觉还是比较困难的，很多时间都是用在了学习如何使用图形上；

但是话说回来，图形界面在软件中是很重要的一个环节，因为用户看到的就是软件的界面，逻辑清晰，交互良好的界面才有可能让用户满意。

实验二

1 实验目的

掌握添加系统调用的方法；

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

3 实验设计

3.1 平台环境

试验环境是 windows8.1 下的 ubuntu14 32 位虚拟机，使用的 Linux 内核版本是 3.16.41。

3.2 方案设计

- (1) 实验的功能是拷贝文件，这个功能在第一次实验中已经写过，只要稍作修改就可以使用了，实验一中用的是 `fread` 和 `fwrite` 函数，但是这两个函数不在 Linux 内核中，所以需要替换为 `read` 和 `write` 函数，并且对应的参数也需要合适地修改。
- (2) 内核中不能进行 `printf` 操作，查看信息需要使用 `printk` 函数，用法与 `printf` 是相同的，不同之处是不会打印到屏幕上，需要使用 `dmesg` 命令来查看输出的内容。
- (3) 函数需要对源文件进行检查，如果源文件不存在或打开失败就不能继续进行，而是停止操作并输出错误信息（源文件不存在、源文件打开失败、目标文件打开失败等）。

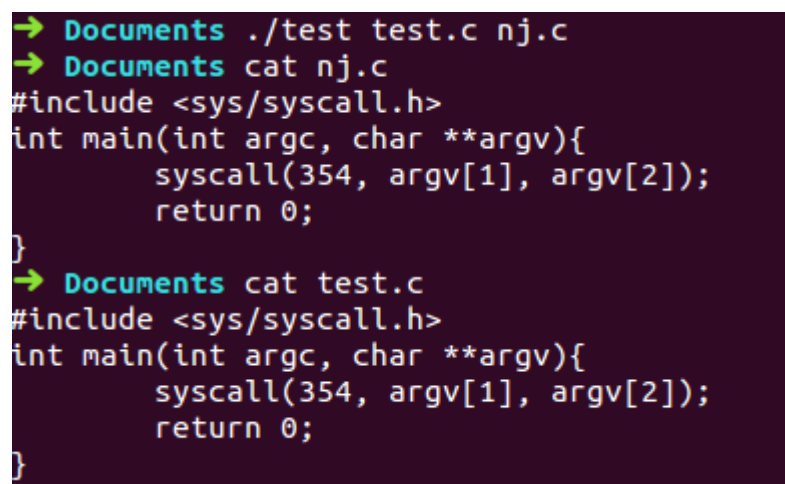
4 实验调试

4.1 实验步骤

- (1) 下载 Linux 内核源码
- (2) 文件拷贝的功能在实验 1 中已经完成,但是其中用的是 C 语言的库函数,不是标准的 Linux 函数,所以需要对其进行一些简单的修改,把对应的函数替换一下(fread→read, fwrite→write),然后修改为与其他函数相同的返回类型,添加到“sys.c”中
- (3) 连接新增的系统调用,需要编辑两个文件,系统调用清单“unistd.h”和内核函数指针表“syscall_table.s”
- (4) 重建 Linux 内核:生成内核配置文件,编译内核映像,编译内核模块,生成并安装模块,安装新的系统,修改 grub 文件,重启选择新的内核
- (5) 编写应用程序测试新增的系统调用。

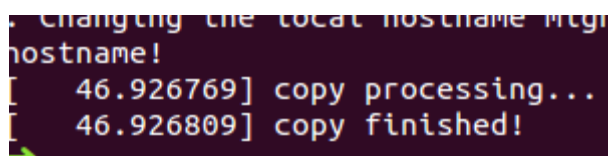
4.2 实验调试及结果

- (1) 实验结果,调用该系统功能调用后,成功拷贝了源文件到目标文件中,如图 2-1 所示,源文件是 test.c,目标文件是 nj.c;



```
→ Documents ./test test.c nj.c
→ Documents cat nj.c
#include <sys/syscall.h>
int main(int argc, char **argv){
    syscall(354, argv[1], argv[2]);
    return 0;
}
→ Documents cat test.c
#include <sys/syscall.h>
int main(int argc, char **argv){
    syscall(354, argv[1], argv[2]);
    return 0;
}
```

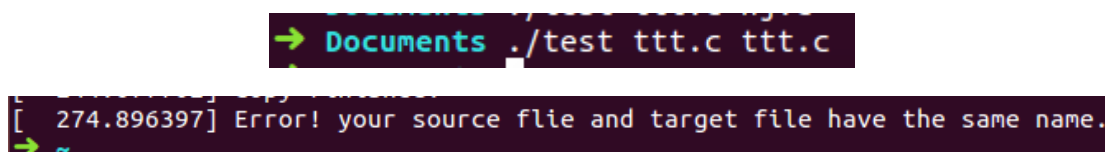
图 2-1 拷贝成功结果



```
Changing the local hostname to gn
hostname!
[ 46.926769] copy processing...
[ 46.926809] copy finished!
```

图 2-2 系统内核的输出信息

- (2) 当源文件不存在时，拷贝过程没有停下来，因为判断文件打开失败的功能没有根据 `open` 函数和 `fopen` 函数做对应的判断，导致判断文件打开失败的功能无效，考虑到了这种情况，但在实现上出现了问题；
- (3) 目标文件和源文件的文件名不能相同，这里是做了判断的，运行结果正常，会在 `dmesg` 中提示并且终止拷贝，如图 2-3 所示



```
Documents ./test ttt.c ttt.c
[ 274.896397] Error! your source flie and target file have the same name.
```

图 2-3 源文件与目标文件名相同时的提示信息

4.3 实验心得

给内核增加系统调用并不是很复杂，只要根据自己想实现的功能写一个或几个函数就可以了，函数可能会有一点限制，不会跟在外面写功能完全相同，但总有替代的方法；写完功能后就可以配置内核，进行编译了。

编译内核是一个比较漫长的过程，最好是在编译之前好好考虑自己写的程序是否还存在一些问题。尽量减少重新编译的次数，最好是能够一次通过。

用户对系统功能的调用不一定会按照完全正确的用法来使用，使用不当可能会出现问題，所以程序需要对用户的操作进行检验，当操作有误时不能继续进行，而是终止执行并提醒用户出现了错误操作。

实验三

1 实验目的

掌握添加设备驱动程序的方法

2 实验内容

- (1) 采用模块方法，添加一个新的字符设备的驱动程序，实现打开/关闭、读/写等基本操作
- (2) 编写一个应用程序，测试添加的驱动程序

3 实验设计

3.1 平台环境

实验平台为 deepin（基于 debian）15.3 64 位。

3.2 方案设计

- (1) 一个简单的字符设备驱动程序应当包含打开和关闭，读和写的基本操作，还应包括两个加载设备和卸载设备的函数，分别调用注册设备和卸载设备的函数。
- (2) 打开和关闭无需添加特别的功能，但是读和写需要记录每次读到或者已经写到的位置，以便能够在关闭设备之前继续上一次读（写）的位置连续读（写），就需要一个指针来记录当前读（写）的位置，每次操作之后都要进行修改。
- (3) 读和写除了考虑记录位置之外，还要考虑读写的数量，当要读的数目超过了缓冲区剩余的大小时，就需要只读剩余大小的内容，当要写的大小超过了剩余的大小时，就只能写剩余大小。
- (4) 读和写分别需要 `copy_to_user()`和 `copy_from_user()`函数。
- (5) 编写完设备驱动程序后需要编译模块，然后加载设备驱动模块，生成设备文件，再编写测试程序对添加的模块进行测试

4 实验调试

4.1 实验步骤

- (1) 根据方案设计编写出驱动程序的代码（打开、关闭、读、写、加载、卸载、file_operations 结构体）。
- (2) 编译内核模块。
- (3) 加载设备驱动模块。
- (4) 查看加载后的设备号，根据设备号生成设备文件。
- (5) 编写测试程序对驱动程序进行测试。

4.2 实验调试及结果

- (1) 测试程序分为写和读两部分，先打开一个文件，写一部分字符（测试用例是 abc），再写一部分字符（测试连续写的功能，用例是 defg），然后关闭文件；再打开这个文件，从里面读取一部分刚刚写入的字符（测试用例是 abcd），再继续读取一部分字符（测试连续读的功能，用例是 efg）。测试结果如图 3-1 所示：

```
char buf[] = {"abc"}, buf2[16];
testdev = open("/dev/mydev", O_RDWR);
if ( testdev == -1 ) {
    printf("Cann't open file\n");
    return 0;
}
write(testdev, buf, 3);
write(testdev, "defg", 4);
```

图 3-1(a) 设备驱动程序测试（两次写设备）

```
read(testdev, buf2, 4);
read(testdev, buf3, 3);
close(testdev);
printf("%s\n%s\n", buf2, buf3);
```

图 3-1(b) 设备驱动程序测试（两次读设备）

```
xrr@xrr-pc ~/Work/Source/OS-supper-lab master sudo mknod /dev/myd
ev c 247 0
xrr@xrr-pc ~/Work/Source/OS-supper-lab master sudo ./lab3_test
abcd
efg
```

图 3-1(c) 设备驱动程序测试（输出）

4.3 实验心得

在 Linux 系统中，设备都是当作文件来处理的，所以对字符设备的操作就是对一个文件的操作，整个实验比较麻烦的地方是对读写操作边界的判断，如果要读或者要写的数据量比剩余的要多，就只能完成一部分的读写甚至不操作（当前指针已经在文件的尾部了）。通过这次实验，自己对 Linux 系统设备的处理方式有了简单的了解，对边界检验的重要性也有了更高的认识，对自己以后编程的逻辑和代码风格会有比较大的影响。

实验四

1 实验目的

理解和分析/proc 文件

2 实验内容

- (1) 了解/proc 文件的特点和使用方法
- (2) 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 和内存利用率、所有进程信息等(可自己补充、添加其他功能)

3 实验设计

3.1 平台环境

实验环境为 deepin（基于 debian）15.3 64 位，开发工具是 electron，开发语言是 javascript。

3.2 方案设计

electron 生成的应用程序本质上是一个封装的浏览器，所以需要用到 html、css 和 javascript 来控制显示的内容，也就是界面的显示，而程序的逻辑是由 javascript 控制的。

- (1) 主进程 main.js 创建一个窗口，并且引导加载界面文件 index.html。
- (2) 界面由两大部分组成，一部分是选项卡，另一部分是每一个选项的具体内容。共有三个选项卡，分别是 CPU 信息，内存信息和进程信息。
- (3) CPU 信息显示 CPU 的型号和 CPU 利用率，每隔 1 秒刷新一次，并以折线图的形式表示出来。
 - a) CPU 的型号是从 /proc/cpuinfo 文件获取的，CPU 利用率是从 /proc/stat 文件读取并计算的。处理方法：从文件中读取到的内容是一个字符串，需要调用字符串对象的 split() 方法，指定分割依据 (/proc/stat 信息是使用一

- 个或多个空格的正则表达式 `/+/`），把一个长的字符串分割为多个小的字符串，分割的结果保存在一个数组中，根据需要按照下标取出计算需要的元素，然后用 `Number()` 方法把字符串转换为数字进行计算。
- b) 定时刷新是用 `setInterval()` 函数，每隔 1 秒读取一次文件，把本次与上次的读取结果联合计算出 CPU 利用率，并显示在对应的 html 元素中。
 - c) 折线图的绘制。绘制图像的方法是用 js 脚本控制 `canvas` 元素，在上面画线，每隔 1s 读取出一个 CPU 利用率，作为点的纵坐标，时间作为点横坐标，依次把点连起来，并把画布左移一定距离，实现折线图的自动滚动。
- (4) 内存信息显示总的内存容量，剩余内存和内存使用率，并用一个饼图把内存使用率表示出来，刷新时间为 1 秒。
- a) 内存信息是从 `/proc/meminfo` 文件中读取的，读的方法是先存为 1 个长的字符串，然后用回车分为字符串数组，再用冒号把需要用到的行分开，取出 `total`, `cached`, `free` 等多个数值，每隔 1 秒读取一次，计算出使用率后与需要显示的原数据一起送到 html 中对应的元素中。
 - b) 饼图的绘制，同样使用 js 脚本和 html 的 `canvas` 元素，先画一个圆，然后填充颜色，再根据内存使用率会出一段对应的弧线，连接到圆心，再连接到起点，然后填充另一种颜色，就画出了一个饼图。
 - c) 饼图也要每隔 1 秒刷新一次，操作方法仍是使用 `setInterval()` 定时执行的函数，每隔 1 秒取一次利用率，然后擦除已经绘制的图形，再重新绘制图形，达到动态刷新的效果。
- (5) 进程信息显示当前系统中的全部进程（先遍历 `/proc` 文件夹，然后判断其中的项目是不是数字，即一个进程的 pid 号，然后从每个进程的 `/proc/"pid"/stat` 文件读取信息），显示进程名、pid 号、状态和内存使用量四个信息，并且可以按照不同的依据进行排序，刷新时间为 1 秒，另外还有根据 pid 号杀死进程的功能。
- a) 获得进程信息的方法是遍历 `/proc` 文件夹，读取每一项 `/proc/pid/stat` 的信息，得到一个字符串，再把它分割成多个字符串，放到一个数组里，把需要的项（进程名、PID、状态和内存使用情况）取出来，作为一个对象的成员保存。
 - b) 进程信息是用列表元素显示的，每读取出一个进程的信息，就生成一个新的列表项，并把四个信息作为列表项的子元素添加进去，再把列表项添加到列表元素中，直到所有的项目都添加进去。
 - c) 信息刷新，设置每隔 1 秒刷新一次，操作与前面基本相同，还是 `setInterval()`

函数，每次在遍历之前先把之前的所有数据也就是列表的所有子元素清空，然后才开始添加列表项。

- d) 排序功能。排序是在获取到所有的进程的信息后进行的，每个进程用一个对象表示，其中有四个成员（name, pid, status, memory），js 的数组原生就有排序方法，修改一下比较函数就可以实现按照自定义的规则依据排序。把 NAME、PID、STATUS 和 MEM USAGE 四个元素添加鼠标点击的监听事件，当检测到鼠标点击时就修改排序依据，实现可以按照不同的依据排序（是在下一次刷新的时候展现出来的）。
- e) 按照进程的 PID 杀死进程。通过 nodejs 的 exec 模块可以发送信息让 Linux 系统执行 shell 脚本，这里就是通过执行“kill pid”完成的，在执行之前先检测 pid 是否为空，为空则输出警告弹窗提示输入 pid，同时也可以获取命令执行的输出信息，当产生错误时可以在控制台查看到错误信息。

4 实验调试

4.1 实验步骤

- (1) 写出 main.js、index.html 和 style.css，画出程序的主体框架。
- (2) 编写 cpu_info.js、cpu_calculate.js、mem_calculate.js 等模块，计算出 cpu 和内存的利用率。
- (3) 编写 cpu_line、mem_line、progress_calculate.js 模块，画出 CPU 利用率折线图，内存使用率饼图，显示出进程信息。

4.2 实验调试及结果

调试方法是在终端中执行 electron .，就启动了程序，可以在控制台中查看用 console.log() 语句输出的内容，还可以查看每一个元素的布局，计算大小与样式。

程序执行的结果如下：

- (1) CPU 的信息和使用率如图 4-1 所示：

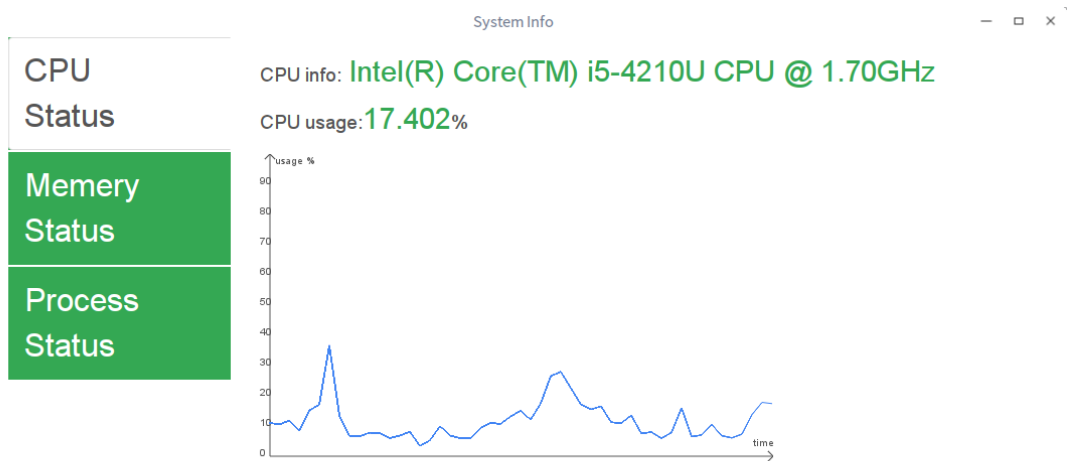


图 4-1 CPU 信息图

(2) 内存使用情况如图 4-2 所示:

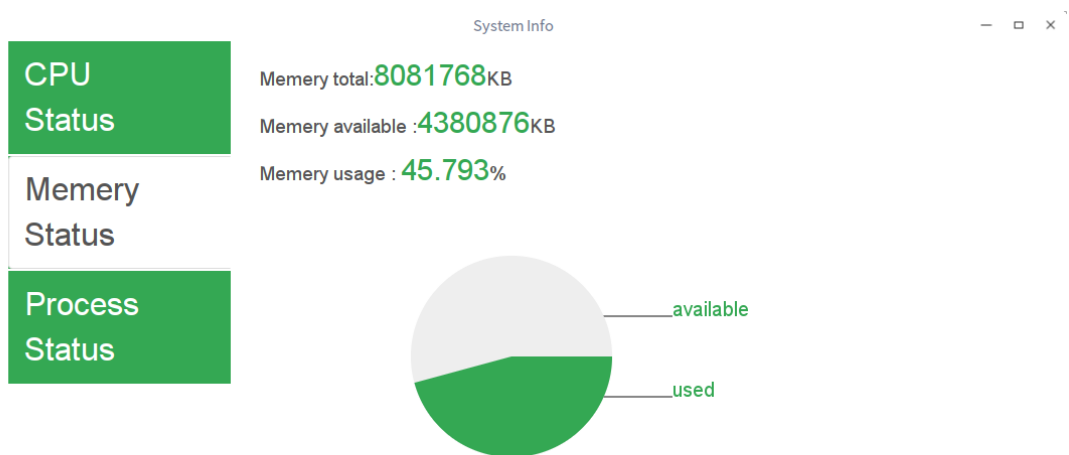


图 4-2 内存信息图

(3) 进程信息如图 4-3 所示:

CPU Status Memery Status Process Status	System Info			
	processeeName	PID	Status	MemUsage/KB
	mark-desktopfil	12873	S	93800
	QQApp.exe	12847	S	2685792
	QQ.exe	12838	S	2737012
	QQ.exe	12808	S	2780352
	explorer.exe	12803	S	2680344
	winedevice.exe	12795	S	2649620
	winedevice.exe	12789	S	2649556
	kworker/0:0	12779	S	0
	QQProtect.exe	12778	S	2702188
	plugplay.exe	12771	S	2647236
	winedevice.exe	12763	S	2652420
	services.exe	12759	S	2648584
	wineserver	12754	S	8808
	winewrapper.exe	12744	S	2676704
	kworker/2:2	12555	S	0
	kworker/1:2	11863	S	0
	kworker/3:1	11861	S	0
	electron	11549	S	1107204
	electron	11548	R	1306080
				kill

图 4-3(a) 进程信息图(PID 降序)

CPU Status Memery Status Process Status	System Info			
	processeeName	PID	Status	MemUsage/KB
	sogou-qimpanel	1579	S	3122820
	QQ.exe	12808	S	2780352
	QQ.exe	12838	S	2737012
	QQProtect.exe	12778	S	2702188
	QQApp.exe	12847	S	2685792
	explorer.exe	12803	S	2680344
	winewrapper.exe	12744	S	2676704
	winedevice.exe	12763	S	2652420
	winedevice.exe	12795	S	2649620
	winedevice.exe	12789	S	2649556
	services.exe	12759	S	2648584
	plugplay.exe	12771	S	2647236
	wps	4903	S	2380940
	startdde	997	S	2237884
	dde-session-dae	1228	S	2236664
	chrome	3726	S	1763276
	chrome	2223	S	1714572
	chrome	9200	R	1613344
	chrome	3189	S	1570340
				kill

图 4-3(b) 进程信息图(内存使用量降序)

(4) 杀死进程功能测试:

a) 未输入 pid 时的提示如图 4-4 所示:

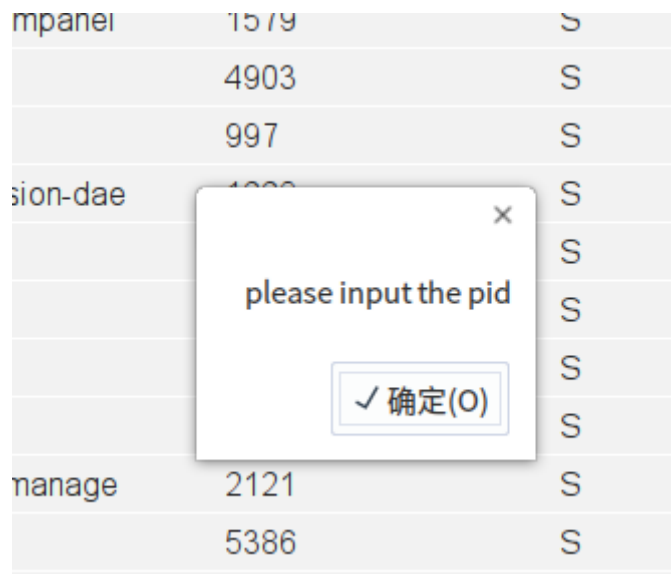


图 4-4 未输入 PID

- b) 正确输入 PID（1288，QQ 的一个进程的 PID）后的进程信息如图 4-5 所示，可以看到对应的进程被杀死了，因为 QQ 被强行杀死了 1 个进程，所以整个 QQ 程序崩溃，所有相关进程也结束了：

System Info			
CPU Status			
Memory Status			
Process Status			
processName	PID	Status	MemUsage/KB
sogou-qimpanel	1579	S	3122820
wps	4903	S	2385132
startdde	997	S	2237884
dde-session-dae	1228	S	2236664
chrome	3189	S	1849428
chrome	3726	S	1763276
chrome	2223	S	1714572
chrome	9200	R	1612660
dde-file-manage	2121	S	1534316
atom	5386	S	1455192
dde-session-ini	1144	S	1407924
electron	11517	S	1377084
atom	5453	S	1314612
electron	11548	R	1309932
dde-desktop	1214	S	1234460
ss-qt5	1367	S	1193648
electron	11549	S	1124876
dde-launcher	1324	S	1124256
chrome	2956	S	1059260

kill 12808

图 4-5 杀死 PID 为 12808 后的进程图

4.3 实验心得

这次试验要读取并解析系统的 `proc` 文件，并用图形界面把信息展示出来，使用的框架是 `electron`，可以通过 `fs` 模块读取文件和文件夹，取得信息后以字符串的形式进行分割选择，对字符串的操作在 `js` 语言中是比较灵活的。花费时间比较多的是图形界面的绘制，基础的样式很简单，`html` 可以较容易地画出基本模型，之后的数值、折线图、饼图和进程列表的显示和刷新等逻辑功能就需要用 `js` 脚本实现。

这次试验的收获还是很大的，一是了解了 `proc` 文件的格式，系统的信息都可以从中获取；二是了解了 `electron` 这样一个用 `js` 开发程序（可以方便地实现跨平台开发）的工具，同时对 `js` 的使用有了更多的学习，之前虽然学习过一些 `js` 的知识，但是没有在实践中使用过，仅仅是停留在简单的语法层面上，可以说是这次实验才让自己真正对一门课外学习的内容有了真正意义上的入门；三是在完成开发的过程中遇到了很多问题和疑惑，搜索了很多别人写过的博客，看官方的文档，向另一个同样使用 `electron` 开发的同学请教，才最终完成了程序，提高了自己搜索知识解决问题的能力。

附录 实验代码

实验一

内容 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main( ) {
    char src_name[255], tgt_name[255], swap[1024];
    FILE *fp_src, *fp_tgt; // fp source and dp target;
    int j = 1024; // read 1024 bytes one time;
    int flag_name_check = 1;
    do {
        printf("please input the source file name:\n");
        scanf("%s", src_name);
        getchar();
        printf("please input the target file name\n", tgt_name);
        scanf("%s", tgt_name);
        getchar();
        flag_name_check = !strcmp(src_name, tgt_name);
        if (flag_name_check) {
            printf("Error! your source flie and target file have the same name.\n");
        }
    } while(flag_name_check);
    printf("source file: %s\ntarget file: %s\n", src_name, tgt_name);
    printf("copy processing...\n");
    fp_src = fopen(src_name, "r");
    fp_tgt = fopen(tgt_name, "w");
    if (!fp_src) {
        printf("error open source file\n");
        return 0;
    }
    if (!fp_tgt) {
        printf("error open target file\n");
        return 0;
    }
    while (j == 1024) { // read uncomplete
        j = fread(swap, 1, 1024, fp_src);
```

```

        fwrite(swap, 1, j, fp_tgt);
    }
    fclose(fp_src);
    fclose(fp_tgt);

    printf("copy finished!\n");

    return 0;
}

```

内容 2:

mainwindow.h:

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```

#include <QMainWindow>
#include <QLabel>

```

```

namespace Ui {
class MainWindow;
}

```

```

class MainWindow : public QMainWindow
{
    Q_OBJECT

```

```

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    QLabel *timestr;
private slots:
    void updateTime();

```

```

private:
    Ui::MainWindow *ui;
};

```

```

#endif // MAINWINDOW_H

```

cpustatus.h:

```

#ifndef CPUSTATUS_H
#define CPUSTATUS_H

```

```

#include <QMainWindow>
#include <QLabel>

namespace Ui {
class cpuStatus;
}

class cpuStatus : public QMainWindow
{
    Q_OBJECT

public:
    explicit cpuStatus(QWidget *parent = 0);
    ~cpuStatus();
    FILE *fp;
    double user[2], nice[2], system[2], idle[2], total[2];
    char info[200], cpu[20][2];
    QLabel *cpuStr;
    char estat[100];

private slots:
    void updateStatus();

private:
    Ui::cpuStatus *ui;
};

#endif // CPUSTATUS_H

```

```

autoplus.h:
#ifndef AUTO_PLUS_H
#define AUTO_PLUS_H

```

```

#include <QMainWindow>
#include <QLabel>

```

```

namespace Ui {
class auto_plus;
}

```

```

class auto_plus : public QMainWindow

```

```

{
    Q_OBJECT

public:
    explicit auto_plus(QWidget *parent = 0);
    ~auto_plus();
    qint32 srcNum, sum;
    QLabel *sumStr;

private slots:
    void updateSum();

private:
    Ui::auto_plus *ui;
};

```

```

#endif // AUTO_PLUS_H

```

mainwindow.cpp:

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QLabel>
#include <QWidget>
#include <QTime>
#include <QDate>
#include <QTimer>

```

```

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timestr = new QLabel(this);
    QFont ft;
    ft.setPointSize(13);
    timestr->setFont(ft);
    QDateTime dateTime;
    QString currentTime;
    dateTime = QDateTime::currentDateTime();
    currentTime = dateTime.toString("yyyy-MM-dd hh:mm:ss dddd");
    timestr->setText(currentTime);
}

```

```

        timestr->setGeometry(80,100,240,40);
        QTimer *timer;
        timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), this, SLOT(updateTime()));
        timer->start(1000);
    }

```

```

MainWindow::~MainWindow()
{
    delete ui;
}

```

```

void MainWindow::updateTime()
{
    //获取系统现在的时间
    QDateTime time = QDateTime::currentDateTime();
    //设置系统时间显示格式
    QString str = time.toString("yyyy-MM-dd hh:mm:ss dddd");
    //在标签上显示时间
    timestr->setText(str);
}

```

cpustatus.cpp:

```

#include "cpustatus.h"
#include "ui_cpustatus.h"
#include <QLabel>
#include <QTimer>
#include <unistd.h>

```

```

cpuStatus::cpuStatus(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::cpuStatus)
{
    ui->setupUi(this);
    cpuStr = new QLabel(this);
    QFont ft;
    ft.setPointSize(13);
    cpuStr->setFont(ft);
    cpuStr->setGeometry(80,100,220,40);
    sprintf(cstat,"CPU usage :..... %%");
    cpuStr->setText(cstat);
    QTimer *timer;

```

```

        timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), this, SLOT(updateStatus()));
        timer->start(2000);

    }

cpuStatus::~cpuStatus()
{
    delete ui;
}

void cpuStatus::updateStatus()
{
    fp = fopen("/proc/stat", "r");
    fgets(info, 200, fp);
    sscanff(info, "%s%lf%lf%lf%lf", cpu[0], &user[0], &nice[0], &system[0], &idle[0]);
    fclose(fp);
    total[0] = (user[0] + nice[0] + system[0] + idle[0]);
    sleep(1);
    fp = fopen("/proc/stat", "r");
    fgets(info, 200, fp);
    sscanff(info, "%s%lf%lf%lf%lf", cpu[1], &user[1], &nice[1], &system[1], &idle[1]);
    fclose(fp);
    total[1] = (user[1] + nice[1] + system[1] + idle[1]);
    sprintf(cstat, "CPU usage : %.8f%%", (1-(idle[1]-idle[0])/(total[1]-total[0]))*100);
    cpuStr->setText(cstat);
}

```

autoplus.cpp:

```

#include "auto_plus.h"
#include "ui_auto_plus.h"
#include <QTimer>

auto_plus::auto_plus(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::auto_plus),
    srcNum(0),
    sum(0)
{
    ui->setupUi(this);
    sumStr = new QLabel(this);
    QFont ft;

```

```

        ft.setPointSize(13);
        sumStr->setFont(ft);
        sumStr->setText(QString::number(sum,10));
        sumStr->setGeometry(180,100,100,40);
        QTimer *timer;
        timer = new QTimer(this);
        connect(timer, SIGNAL(timeout()), this, SLOT(updateSum()));
        timer->start(3000);
    }

```

```

auto_plus::~auto_plus()
{
    delete ui;
}

```

```

void auto_plus::updateSum()
{
    if(srcNum <= 100){
        sum = sum + srcNum;
        srcNum = srcNum + 1;
        sumStr->setText(QString::number(sum,10));
    }
}

```

```

main.cpp:
#include "mainwindow.h"
#include "auto_plus.h"
#include "cpustatus.h"
#include <QApplication>
#include <QLabel>
#include <unistd.h>

```

```

pid_t childpid1, childpid2;
int main(int argc, char *argv[])
{

    while((childpid1 = fork()) == -1);
    if(childpid1 == 0) {
        QApplication b(argc, argv);
        auto_plus sum;
        sum.setWindowTitle("auto plus");
        sum.show();
    }
}

```

```

        sum.move(860,300);
        sum.setMinimumSize(400,300);
        sum.setMaximumSize(400,300);
        return b.exec();
    }
    else {
        while((childdpid2 = fork()) == -1);
        if(childdpid2 == 0) {
            QApplication c(argc, argv);
            cpuStatus status;
            status.setWindowTitle("CPU status");
            status.show();
            status.move(440,300);
            status.setMinimumSize(400,300);
            status.setMaximumSize(400,300);
            return c.exec();
        }
        else{
            QApplication a(argc, argv);
            MainWindow w;
            w.setWindowTitle("system time");
            w.show();
            w.move(20,300);
            w.setMinimumSize(400,300);
            w.setMaximumSize(400,300);
            return a.exec();
        }
    }
}

```

实验二

系统调用的函数：

```

asmlinkage long sys_copyfile(const char *src_name, const char *tgt_name ) {
    char swap[1024];
    int fp_src, fp_tgt;// fp source and fp target;
    int j = 1024;//read 1024 bytes one time;
    int flag_name_check;
    mm_segment_t old_fs=get_fs();
    set_fs (KERNEL_DS);
    flag_name_check = strcmp(src_name, tgt_name);

```

```

if (flag_name_check == 0) {
    printk("Error! your source flie and target file have the same name.\n");
    return 0;
}

printk("copy processing...\n");
fp_src = sys_open(src_name, O_RDONLY, 0);
fp_tgt = sys_open(tgt_name, O_WRONLY|O_CREAT|O_TRUNC, 0600);
if (!fp_src) {
    printk("error open source file\n");
    return 0;
}
if (!fp_tgt) {
    printk("error open target file\n");
    return 0;
}
while (j == 1024) { //read uncomplete
    j = sys_read(fp_src, swap, 1024);
    sys_write(fp_tgt, swap, j);
}
sys_close(fp_src);
sys_close(fp_tgt);

printk("copy finished!\n");
set_fs(old_fs);
return 0;
}

```

实验三

驱动程序文件：

```

#include <linux/module.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <linux/cdev.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <asm/system.h>

```

```

#ifndef MAXSIZE
#define MAXSIZE 4096
#endif

char data[MAXSIZE]
unsigned int cur_prt, devnum = -1;

int ch_open(struct inode *inode, struct file *filep) { //open
    cur_prt = 0;
    return 0;
}

int ch_release(struct inode *inode, struct file *filep) { //close
    return 0;
}

static ssize_t ch_read(struct file *filep, char __user *buf, size_t size, loff_t *ppos) {
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;
    if (p > MAXSIZE) {
        return 0;
    }
    if (count > MAXSIZE - p) {
        count = MAXSIZE - p;
    }
    if (copy_to_user(buf, (void *) (data + p), count)) {
        ret = -EFAULT;
    } else {
        *ppos += count;
        ret = count;
        printk(KERN_INFO "read %d bytes from %d\n", count, p);
    }
    return ret;
}

static ssize_t ch_write(struct file *filep, const char __user *buf, size_t size, loff_t *ppos) {
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;
    if (p > MAXSIZE) {
        return 0;
    }
    if (count > MAXSIZE - p) {

```

```

        count = MAXSIZE - p;
    }
    if (copy_from_user(data + p, buf, count)) {
        ret = -EFAULT;
    } else {
        *ppos += count;
        ret = count;
        printk(KERN_INFO "write %d bytes from %d\n", count, p);
    }
    return ret;
}

```

```

static const struct file_operations ch_fops = {
    .owner = THIS_MODULE,
    .read = ch_read,
    .write = ch_write,
    .open = ch_open,
    .release = ch_release,
};

```

```

static int init_module(void) {
    devnum = register_chrdev(0, "mydev", &ch_fops);
    if (devnum) {
        printk("Succeed!\n");
    } else {
        printk("Failed!\n");
    }
    return devnum;
}

```

```

static int cleanup_module(void) {
    if (devnum >= 0) {
        unregister_chrdev(devnum, "mydev");
    }
}

```

测试文件:

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

```

```

int main()
{
    int testdev;
    int i;
    char buf[] = {"abc"}, buf2[10], buf3[10];
    testdev = open("/dev/mydev",O_RDWR);
    if ( testdev == -1 ) {
        printf("Can't open file \n");
        return 0;
    }
    write(testdev, buf, 3);
    write(testdev, "defg", 4);
    close(testdev);
    testdev = open("/dev/mydev",O_RDWR);
    read(testdev,buf2,4);
    read(testdev,buf3,3);
    close(testdev);
    printf("%s\n%s\n",buf2, buf3);
    return 0;
}

```

实验四

main.js:

```

const electron = require('electron');
const {app} = electron;
const {BrowserWindow} = electron;

```

let mainWindow;

```

function creatWindow() {
    mainWindow = new BrowserWindow({width: 1100,
        height: 663,
        minHeight: 663,
        minWidth: 1100,
        resizable: true});
    mainWindow.loadURL('file:///${__dirname}/index.html');
    mainWindow.webContents.openDevTools();
    mainWindow.setMenu(null);
    mainWindow.on('closed',() => {
        mainWindow = null;
    });
}

```

```

}
app.on('ready', creatWindow);
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
app.on('activate', () => {
  if (mainWindow === null) {
    creatWindow();
  }
});

```

index.html:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>System Info</title>
    <link href="http://cdn.static.runoob.com/libs/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.static.runoob.com/libs/foundation/5.5.3/css/foundation.min.css">
    <link rel="stylesheet" href="style.css">
    <script>
      window.$ = window.jQuery = require(`${__dirname}/node_modules/jquery/dist/jquery.min.js`);
      require(`${__dirname}/node_modules/bootstrap/dist/js/bootstrap.min.js`);
    </script>
    <script src="cpu_calculate.js" charset="utf-8"></script>
    <script src="memory_calculate.js" charset="utf-8"></script>
    <script src="cpu_info.js" charset="utf-8"></script>
    <script src="progress_calculate.js" charset="utf-8"></script>
  </head>

  <body class="col-xs-12">
    <div class="col-xs-3">
      <ul id="myTab" class="nav nav-tabs tabs-left">
        <li class="active">
          <a href="#cpu" data-toggle="tab">
            CPU<br> Status
          </a>
        </li>
        <li><a href="#memory" data-toggle="tab">Memory<br> Status</a></li>
        <li><a href="#progress" data-toggle="tab">Process<br> Status</a></li>
      </ul>
    </div>
  </body>
</html>

```

```

</ul>
</div>
<div class="col-xs-8">
  <div id="myTabContent" class="tab-content">
    <div class="tab-pane fade in active" id="cpu">
      <p>CPU info:<span id="cpuinfo">.....</span></p>
      <p>CPU usage:<span id="cpuusage">.....</span>%</p>
      <div id="canvasContener">
        <canvas id="axis" width="510" height="305"></canvas>
        <div id="canvasContener2">
          <canvas id="cpuCanvas" width="10000" height="305"></canvas>
        </div>
      </div>
    </div>
    <div class="tab-pane fade" id="memory">
      <p>Memery total:<span id="memtotal">.....</span>KB</p>
      <p>Memery available :<span id="memava">.....</span>KB</p>
      <p>Memery usage : <span id="memusage">.....</span>%</p>
      <canvas id="memCanvas" width="500" height="305"></canvas>
    </div>
    <div class="tab-pane fade" id="progress">
      <div class="div-table-title">
        <div class="col-xs-3" id="pName">
          processeeName
        </div>
        <div class="col-xs-3" id="pid">
          PID
        </div>
        <div class="col-xs-3" id="stu">
          Status
        </div>
        <div class="col-xs-3" id="mem">
          MemUsage/KB
        </div>
        <div class="clear-float">
          </div>
      </div>
      <ol class="div-table" id="prgInfoTable">
        </ol>
      </div>
    </div>
  </div>

```

```

        <div class="">
            <input type="Number" name="" value="" id="inpt">
            <button type="button" name="button" id="btn">kill</button>
        </div>
    </div>
</div>
</div>
<script src="cpu_line.js" charset="utf-8"></script>
<script src="mem_line.js" charset="utf-8"></script>
</body>
</html>

```

package.json:

```

{
  "name": "system info",
  "version": "0.1.0",
  "main": "main.js",
  "dependencies": {
    "jquery": "^3.1.1"
  }
}

```

cpu_info.js:

```

var cpufs = require('fs');
function cpuStringToValueInfo(datain) {
    var arrStringCPU = datain.split("\n");
    var arrinfo = arrStringCPU[4].split(":");
    var cpuValueInfo = arrinfo[1];
    return cpuValueInfo;
}
var cpuValueInfo;
function calCPUinfo() {
    setInterval(function() {
        cpufs.readFile('/proc/cpuinfo', 'utf-8', (err, data) => {
            cpuValueInfo = cpuStringToValueInfo(data);
        });
        document.getElementById("cpuinfo").innerHTML = cpuValueInfo;
    }, 1000);
}
calCPUinfo();

```

cpu_calculate.js:

```

//calculate CPU usage
var cpuFs = require('fs');
function createCpuValue() {
    var oTemp    = new Object;
    oTemp.user    = 0;
    oTemp.nice    = 0;
    oTemp.system = 0;
    oTemp.idle    = 0;
    oTemp.total   = 0;
    return oTemp;
}
var cpuValue=[], cpuusage = 0;
cpuValue[0] = createCpuValue();
cpuValue[1] = createCpuValue();
function cpuStringToValue(cpuValue, datain) {
    var arrStringCPU = datain.split(/ +/);
    cpuValue.user    = Number(arrStringCPU[1]);
    cpuValue.nice    = Number(arrStringCPU[2]);
    cpuValue.system  = Number(arrStringCPU[3]);
    cpuValue.idle    = Number(arrStringCPU[4]);
    cpuValue.total   = cpuValue.user + cpuValue.nice + cpuValue.system + cpuValue.idle;
}
function calCPU() {
    var i = 0;
    setInterval(function() {
        cpuFs.readFile('/proc/stat', 'utf-8', (err, data) => {
            cpuStringToValue(cpuValue[i], data);
            i = i + 1;
            i = i%2;
        });
        cpuusage = (cpuValue[1].total != cpuValue[0].total) ? (1 - (cpuValue[1].idle -
cpuValue[0].idle)/(cpuValue[1].total - cpuValue[0].total))*100.0 : 0;
        cpuusage = cpuusage.toFixed(3);
        document.getElementById("cpuusage").innerHTML = cpuusage;
    },1000);
}
//calculate CPU usage end
calCPU();

cpu_line.js:

```

```

function strokCpu() {
    var canv = document.getElementById("cpuCanvas");
    var context = canv.getContext("2d");
    context.beginPath();
    var usg = document.getElementById("cpuusage").innerHTML, count = -10, width = 510;
    context.moveTo(0,300);
    var objj = document.getElementById("canvasContener2");
    setInterval(function() {
        usg = document.getElementById("cpuusage").innerHTML;
        context.lineTo(count + 10 ,300-usg*3);
        if (count > 490) {
            objj.style.overflow = "auto";
            objj.scrollLeft += 10;
        }
        count = count + 10;
        context.strokeStyle = "#4285f4";
        context.stroke();
    },1000);
}

function strokAxis() {
    var canv = document.getElementById("axis");
    var context = canv.getContext("2d");
    context.beginPath();
    context.moveTo(10,300);
    context.lineTo(10,0);
    context.lineTo(5,5);
    context.moveTo(10,0);
    context.lineTo(15,5);
    context.moveTo(10,300);
    context.lineTo(510,300);
    context.lineTo(505,305);
    context.moveTo(510,300);
    context.lineTo(505,295);
    context.font = "bold 16px";
    context.fillStyle = "#555";
    context.fillText("usage %", 15, 10);
    context.fillText("time", 490, 290);
    var i = 0;
    while (i<100) {
        context.fillText(100-10*i, 0, 30*i);
        i = i + 1;
    }
}

```

```

        context.strokeStyle = "#555";
        context.stroke();
        var usg = document.getElementById("cpuusage").innerHTML, count = 0;
    }
    strokAxis();
    strokCpu();

mem_calculate.js:
var memFs = require('fs');
function createMemValue() {
    var oTemp      = new Object;
    oTemp.total    = 0;
    oTemp.avable   = 0;
    oTemp.usage    = 0;
    return oTemp;
}
var memValue  = createMemValue();
function memStringToValue(memValue, datain) {
    var arrStringMem = datain.split("\n");
    var memStringTotal = arrStringMem[0].split(/ +/);
    var memStringAvaib = arrStringMem[2].split(/ +/);
    memValue.total = Number(memStringTotal[1]);
    memValue.avable = Number(memStringAvaib[1]);
    memValue.usage = ((memValue.total - memValue.avable)/memValue.total)*100.0;
    memValue.usage = memValue.usage.toFixed(3);
}
function calMem() {
    setInterval(function(){
        memFs.readFile('/proc/meminfo', 'utf-8', (err, data) => {
            memStringToValue(memValue, data);
        });
        document.getElementById("memusage").innerHTML = memValue.usage;
        document.getElementById("memtotal").innerHTML = memValue.total;
        document.getElementById("memava").innerHTML = memValue.avable;
    },1000);
}

calMem();

mem_line.js:
function strokArc() {
    var canv = document.getElementById("memCanvas");

```

```

var context = canv.getContext("2d");
context.translate(250,150);
context.beginPath();
setInterval(function() {
    context.clearRect(-200,-150,500,300);

    context.beginPath();
    context.moveTo(30, -40);
    context.lineTo(160,-40);
    context.fillText("available",160,-40);
    context.moveTo(30,40);
    context.lineTo(160,40);
    context.font = "20px sans-serif";
    context.fillText("used",160,40);
    context.strokeStyle = "#101010";
    context.stroke();
    context.beginPath();
    context.arc(0,0,100,0,2*Math.PI);
    context.fillStyle="#eeeeee";
    context.fill();
    var angu = document.getElementById("memusage").innerHTML;
    angu = Number(angu)/100*2*Math.PI;
    context.beginPath();
    context.arc(0,0,100,0,angu);
    context.lineTo(0,0);
    context.lineTo(100,0);
    context.fillStyle="#34a853";
    context.fill();
}, 1000);
}
strokeArc();
function draw() {
    var angu = document.getElementById("memusage");
    angu = Number(angu)/100*2*Math.PI;
    context.arc(0,0,100,0,angu);
    context.lineTo(0,0);
    context.lineTo(100,0);
    context.fillStyle="#34a853";
    // context.stroke();
    context.fill();
}

```

```

process_calculate.js:
var prgFs = require('fs');
var count = 0, prgValueInfo = [], sortSelect = "mem", UorD = 0, UorD1 = 0, UorD2 = 0, UorD3 = 0, UorD4 = 0;
function creatPrgObj() {
    var oTemp = new Object;
    oTemp.pName = "name";
    oTemp.mem = 0;
    oTemp.pid = 0;
    oTemp.statu = "S";
    return oTemp;
}
function prgToValue(datain) { //info from file to an object
    var arrStringPrg = datain.split(/ +/);
    var prgValue = creatPrgObj();
    prgValue.pName = arrStringPrg[1];
    prgValue.pName = prgValue.pName.slice(1,-1); //delete brackets
    prgValue.pid = Number(arrStringPrg[0]);
    prgValue.mem = Number(arrStringPrg[22])/1024;
    prgValue.statu = arrStringPrg[2];
    return prgValue;
}
function opendir() {
    const nameArray = prgFs.readdirSync('/proc/');
    return nameArray;
}
function openfile(nameArray) {
    var reg = new RegExp("^[0-9]*$");
    nameArray.forEach(function(itemName, index) {
        if (reg.test(itemName)) {
            const fileContent = prgFs.readFileSync('/proc/'+itemName+'/stat', 'utf-8');
            prgValueInfo[count] = prgToValue(fileContent);
            count += 1;
        }
    });
}

var by = function(name, upordown){
    return function(o, p){
        var a, b;
        if (typeof o === "object" && typeof p === "object" && o && p) {
            a = o[name];
            b = p[name];

```

```

        if (a === b) {
            return 0;
        }
        if (typeof a === typeof b) {
            if (upordown === 0) {
                return a < b ? -1 : 1;
            } else {
                return a > b ? -1 : 1;
            }
        }
        if (upordown === 0) {
            return typeof a < typeof b ? -1 : 1;
        }
        else {
            return typeof a > typeof b ? -1 : 1;
        }
    }
    else {
        throw ("error");
    }
}
}

```

```

function calPrg() {
    var nameArray = []
    setInterval(function() {
        nameArray.splice(0,nameArray.length);
        nameArray = opendir();
        count = 0;
        prgValueInfo.splice(0,prgValueInfo.length);
        openfile(nameArray);
        prgValueInfo.sort(by(sortSelect, UorD));
        updateDOM();
    }, 1000);
    setTimeout("setBtnCik()",1000);
}

function updateDOM() {
    $("li").remove(".new");
    prgValueInfo.forEach(function(item,index) {
        var textMem = $('<div class = "col-xs-3 new"></div>').text(item.mem);
        var textStatus = $('<div class = "col-xs-3 new"></div>').text(item.statu);
        var textPID = $('<div class = "col-xs-3 new"></div>').text(item.pid);
    });
}

```

```

var textName = $('<div class = "col-xs-3 new"></div>').text(item.pName);
var clrflt = $('<div class = "clear-float"></div>');

var newli = $('<li class = "new"></li>').append(textName, textPID, textStatus, textMem, clrflt);
$("#prgInfoTable").append(newli);
});
var ppiidd = document.getElementById("pid");
ppiidd.onclick = function() {
    sortSelect = "pid";
    UorD1 = !UorD1;
    UorD = UorD1;
}
var mmeemm = document.getElementById("mem");
mmeemm.onclick = function() {
    sortSelect = "mem";
    UorD2 = !UorD2;
    UorD = UorD2;
}
var nnaame = document.getElementById("pName");
nnaame.onclick = function() {
    sortSelect = "pName";
    UorD3 = !UorD3;
    UorD = UorD3;
}
var ssttatu = document.getElementById("stu");
ssttatu.onclick = function() {
    sortSelect = "statu";
    UorD4 = !UorD4;
    UorD = UorD4;
}
}
calPrg();

var exec = require('child_process').exec;
var cmd0 = "kill ", cmd, pidd;
function setBtnCik() {
    var buttn = document.getElementById("btn")
    buttn.onclick= killProcess;
}
function killProcess() {
    pidd = document.getElementById("inpt");
    if (pidd.value=="") {

```

```

        alert("please input the pid");
        return;
    }
    cmd = cmd0 + pidd.value;
    exec(cmd,{encoding:'utf8'},function (err,stdout,stderr){
        if (err){
            console.log(err);
            return;
        }
    });
}

```

style.css:

```

html {
    background-color: #eeeeee;
}
body {
    min-width: 1072px;
}
.tabs-left {
    border-bottom: none;
    padding-top: 2px;
}
.tabs-left {
    border-right: none;
}
.tabs-left>li {
    float: none;
    margin-bottom: 2px;
}
.tabs-left>li {
    margin-right: -1px;
    background-color: #34a853;
    font-size: 2em;
}
.tabs-left>li.active>a,
.tabs-left>li.active>a:hover,
.tabs-left>li.active>a:focus {
    border-bottom-color: #ddd;
    border-right-color: transparent;
    transition: background 0.5s, color 0.5s;
}

```

```
.tabs-left>li>a {
    border-radius: 4px 0 0 4px;
    margin-right: 0;
    display: block;
    color: #ffffff;
    transition: background 0.5s, color 0.5s;
}
```

```
.tabs-left>li>a: hover {
    color: #000000;
    transition: background 0.5s, color 0.5s;
}
```

```
.div-table {
    list-style-type: none;
    max-height: 550px;
    overflow-y: auto;
    margin: 0;
    padding: 0;
}
```

```
.tab-pane>p {
    font-size: 1.2em;
    line-height: 1.3em;
    color: #555;
}
```

```
.tab-pane>p>span{
    font-size: 1.5em;
    color: #34a853;
}
```

```
.tab-content {
    margin-top: 1.5em;
}
```

```
#canvasContener2 {
    max-width: 500px;
    overflow: hidden;
    position: relative;
    left: 10;
    top: -311px;
    left: 9px;
```

```

margin-top: 0;
margin-bottom: 0;
padding-top: 0;
padding-bottom: 0;

}
#axis {
    position: relative;;
    left: 0;
    top: 0;
    margin-top: 0;
    margin-bottom: 0;
    padding-top: 0;
    padding-bottom: 0;
}
#canvasContener {
    max-height: 400px;
    overflow-y: hidden;
}
.div-table-title {
    background-color: #36bd51;
    font-size: 1.1em;
    color: #eeeeee;
    line-height: 2em;
}
.clear-float {
    clear: both;
}
.div-table>li {
    line-height: 1.8em;

    margin-bottom: 1px;
    border-color: #36bd51;
    background-color: #f2f2f2;
    transition: color 0.2s, background 0.2s;
}
.div-table>li:hover {
    background-color: #36bd51;
    color: #ffffff;
    transition: color 0.2s, background 0.2s;
}
#pid:hover, #mem:hover, #pName:hover, #stu:hover {

```

```
        background-color: #34a853;
        cursor:pointer;
    }
    #pid, #mem, #pName, #stu {
        transition: background 0.3s;
    }
    div::-webkit-scrollbar {
        width: 0;
    }
    #inpt {
        width: 200px;
        float: right;
        height: 30px;
    }
    #btn {
        width: 100px;
        float: right;
        height: 30px;
        padding: 0;
        background-color: #34a853;
    }
}
```