

# 操作系统课程设计实验报告

实验名称： 进程控制

姓名/学号： 曾煜瑾/1120172765

## 一、 实验目的

设计并实现 Unix 的“time”命令。“mytime”命令通过命令行参数接受要运行的程序，创建一个独立的进程来运行该程序，并记录程序运行的时间。

## 二、 实验内容

在 Windows 下实现：

使用 `CreateProcess()` 来创建进程

使用 `WaitForSingleObject()` 在“mytime”命令和新创建的进程之间同步

调用 `GetSystemTime()` 来获取时间

在 Linux 下实现：

使用 `fork()/vfork /exec()` 来创建进程运行程序

使用 `wait()` 等待新创建的进程结束

调用 `gettimeofday()` 来获取时间

mytime 的用法： `$ mytime.exe program1`

要求输出程序 `program1` 运行的时间。`Program1` 可以为自己写的程序，也可以是系统里的应用程序。

`$ mytime.exe program2 t`

`t` 为时间参数，为 `program2` 的输入参数，控制 `program2` 的运行时间。最后输出 `program2` 的运行时间，应和 `t` 基本接近。

显示结果： \*\*小时\*\*分\*\*秒\*\*毫秒\*\*微秒

## 三、 实验环境

	名称	版本
Windows 操作系统	Windows 10	企业版
Windows IDE	Dev C++	5.11
Linux 操作系统	Ubuntu 16.04	内核 4.13.0
Linux IDE	gcc	5.4.0

## 四、 程序设计与实现

总的设计流程图：

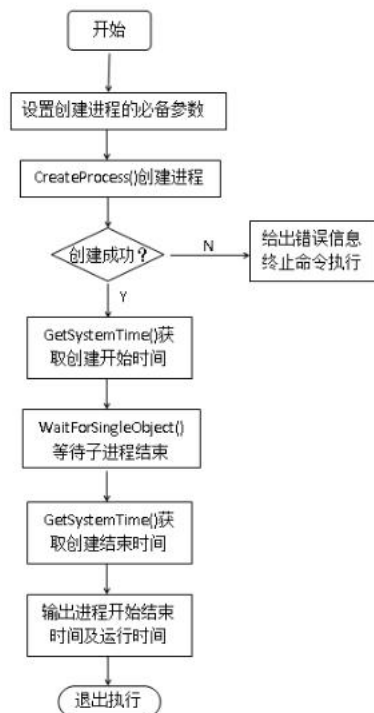


图1 windows 进程控制调用  
程序运行流程图

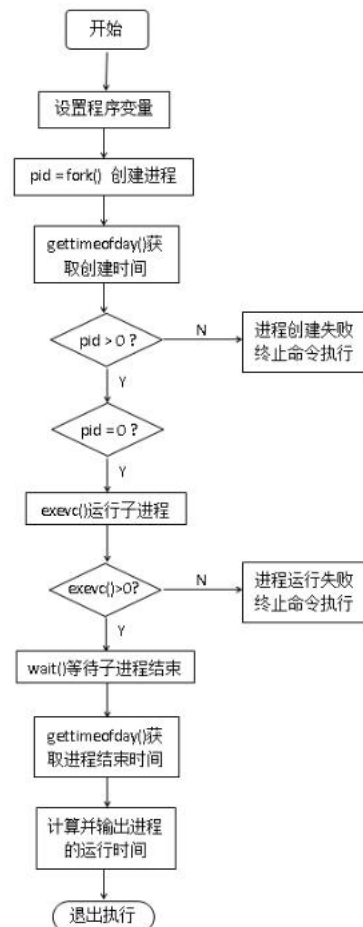


图2 linux 进程控制调用  
程序运行流程图

**Windows 环境下：**

程序设计：

(一) 启动新进程

```
//进程启动相关信息的结构体
STARTUPINFO si;
memset(&si,0,sizeof(STARTUPINFO));
si.cb = sizeof(STARTUPINFO);
si.dwFlags = STARTF_USESHOWWINDOW;
si.wShowWindow = SW_SHOW;

PROCESS_INFORMATION pi; //创建进程

if (!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
```

(二) 选择功能

通过判断传入参数的数量 argc 的数值选择不同功能

```

if (argc == 2)
    WaitForSingleObject(pi.hProcess, INFINITE);
else if(argc == 3)
{
    //主进程休眠
    double sleep_s;
    sleep_s = atof(argv[2]) * 1000;
    Sleep (sleep_s);

    //结束子进程
    DWORD exitCode;
    GetExitCodeProcess(pi.hProcess,&exitCode); //获取退出码
    TerminateProcess(pi.hProcess, exitCode); //终止进程
    //关闭句柄
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
    printf("Process has been killed\n");
}

```

### (三) 计算运行时间

设置两个结构体：SYSTEMTIME time\_start, time\_end，分别记录程序开始和结束的时间，相减即为子程序运行时间。

```

void ShowTime(SYSTEMTIME start, SYSTEMTIME end)
{
    int hour, min, sec, ms;
    ms = end.wMilliseconds - start.wMilliseconds;
    sec = end.wSecond - start.wSecond;
    min = end.wMinute - start.wMinute;
    hour = end.wHour - start.wHour;
    if (ms < 0)
    {
        sec--;
        ms += 1000;
    }
    if (sec < 0)
    {
        min--;
        sec += 60;
    }
    if (min < 0)
    {
        hour--;
        min += 60;
    }
    printf("Cost Time: %d hour %d min %d sec %d ms\n", hour, min, sec, ms);
}

```

程序实现：

题目中要求实现两个功能，即输入两个参数和三个参数，第三个参数为控制子程序运行的时间。另外还要求运行的程序既可以是自己写的程序，也可以是系统里的应用程序，所以在 Windows 系统中需验证 4 次，Linux 系统下同样如此。

我把源代码 mytime.cpp 放在桌面下，自己写的程序是 test.cpp，编译后生成 test.exe，功能是打印出一个字母图形，和 mytime 一起放在桌面下。系统里的应用程序我选择的是 eclipse.exe，绝对路径为 E:\编程软件\eclipse\eclipse。

#### (一) Dev C++中编译 mytime.cpp，生成 mytime.exe

```
C:\Users\lenovo\Desktop\mytime.exe
Create Fail!
-----
Process exited after 0.03917 seconds with return value 1
请按任意键继续. . .
```

注：Create Fail 的原因是因为没有输入子进程，实现代码如下：

```
if (!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
    cout <<"Create Fail!"<< endl;
    exit(1);
}
```

（二）运行程序 test，不输入第三个参数

```
C:\Users\lenovo\Desktop>mytime test
Create Success!
Begin Time: 2019:11:19-8:26:27:323

A                                     A
A B                                 B A
Z   C                             C Z
Y   D                             D Y
X   E                             E X
W   F                             F W
V   G                             G V
U   H                             H U
T   I I                           I T
S R Q P O N M L K J K L M N O P Q R S

End Time: 2019:11:19-8:26:27:339
Cost Time: 0 hour 0 min 0 sec 16 ms
```

注：只精确到 ms，而不是 us，因为时间结构体 SYSTEMTIME 只能精确到 ms，而在 linux 代码中选择另一种记录时间结构体 struct timeval，可精确到 us。

（三）运行程序 test，输入第三个参数，为 2，单位为秒

```

C:\Users\lenovo\Desktop>mytime test 2
Create Success!
Begin Time: 2019:11:19-8:32:31:991

A
A B
Z C
Y D
X E
W F
V G
U H
T I I
S R Q P O N M L K J K L M N O P Q R S

A
B A
Z
Y
X
W
V
U
T
S

Process has been killed
End Time: 2019:11:19-8:32:33:993
Cost Time: 0 hour 0 min 2 sec 2 ms

```

(四) 运行程序 `eclipse`，不输入第三个参数，一段时间后关闭

```

C:\WINDOWS\system32\cmd.exe - mytime E:\编程软件\eclipse\eclipse

```

```

Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\lenovo>cd desktop

C:\Users\lenovo\Desktop>mytime E:\编程软件\eclipse\eclipse
Create Success!
Begin Time: 2019:11:19-8:38:25:563

```



```

C:\WINDOWS\system32\cmd.exe

```

```

Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\lenovo>cd desktop

C:\Users\lenovo\Desktop>mytime E:\编程软件\eclipse\eclipse
Create Success!
Begin Time: 2019:11:19-8:38:25:563

End Time: 2019:11:19-8:38:58:546
Cost Time: 0 hour 0 min 32 sec 983 ms

```

(五) 运行程序 eclipse，输入第三个参数，为 30，单位为秒

```
C:\Users\lenovo\Desktop>mytime E:\编程软件\eclipse\eclipse 30
Create Success!
Begin Time: 2019:11:19-8:40:18:390

Process has been killed
End Time: 2019:11:19-8:40:48:394
Cost Time: 0 hour 0 min 30 sec 4 ms
```

## Linux 环境下:

### 程序设计:

#### (一) 启动新进程

Windows 中是用 CreateProcess 函数启动新进程的过程中传入多个参数，打开运行程序。而 Linux 中是先用 fork 函数启动新进程，此时不传参，然后通过 fork 函数的返回值 pid 初次进入子进程，在子进程中用 exec 函数打开运行程序。代码如下：

```
pid_t pid = fork(); // return 0 to child process, return pid of child to parent process, return -1 while failure
if (pid < 0) //如果出错
{
    printf("Create Fail");
    exit(1);
}
else if (pid == 0) //如果是子进程
{
    printf("Child process is running!\n");
    gettimeofday(&time_start, NULL);
    printf("time_start.tv_sec: %ld second\n", time_start.tv_sec); //子进程开始运行的系统秒数
    printf("time_start.tv_usec: %ld microsecond\n", time_start.tv_usec); //子进程开始运行的系统微秒数
    printf("\n");
    execv(argv[1], &argv[1]); //在子进程中调用execv函数在命令行中来运行一个程序
}
else //如果是父进程
{
    printf("Parent process is running!\n");
    gettimeofday(&time_start, NULL);
```

#### (二) 选择功能

通过判断传入参数的数量 argc 的数值选择不同功能



```

if (argc == 2)
    wait(NULL); //等待子进程结束
if (argc == 3)
{
    //主进程休眠
    double sleep_us;
    sleep_us = atof(argv[2]) * 1000000;
    usleep(sleep_us);

    //结束子进程
    long kill_ret_val = 0;
    kill_ret_val = kill(pid, SIGKILL);
    if (kill_ret_val == -1) // return -1, fail
    {
        printf("Kill Failed.\n");
        perror("kill");
    }
    else if (kill_ret_val == 0) // return 0, success
        printf("process %d has been killed\n", pid);
    printf("\n");
}

```

### (三) 计算运行时间

设置两个结构体：struct timeval time\_start, time\_end; 分别记录程序开始和结束的时间，相减即为子程序运行时间。

```

time_use = (time_end.tv_sec - time_start.tv_sec)*1000000 + (time_end.tv_usec - time_start.tv_usec); //总运行时间
time_hour = time_use / (60 * 60 * 1000 * 1000); //运行小时
time_left = time_use % (60 * 60 * 1000 * 1000);
time_min = time_left / (60 * 1000 * 1000); //运行分钟
time_left %= (60 * 1000 * 1000);
time_sec = time_left / (1000 * 1000); //运行秒
time_left %= (1000 * 1000);
time_ms = time_left / 1000; //运行毫秒
time_left %= 1000;
time_us = time_left; //运行微妙
printf("Time Cost: %ld hour, %ld min, %ld sec, %ld ms, %ld us\n", time_hour, time_min, time_sec, time_ms, time_us);

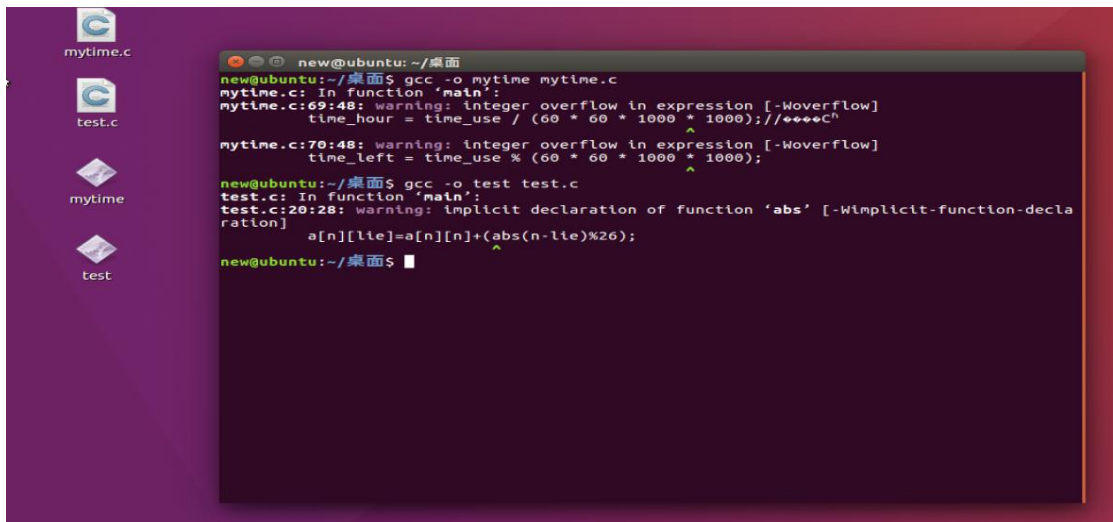
```

程序实现：

同 Windows 一样，在 Linux 操作系统上同样需要验证四个功能，不再赘述。

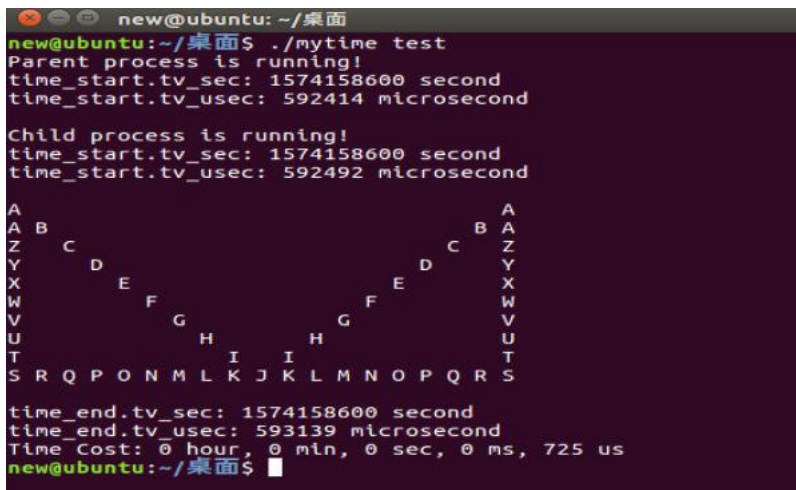
我把源代码 mytime.c 和测试代码 test.c 放在桌面上，系统的应用程序选择为 firefox 浏览器，绝对路径为/usr/lib/firefox/firefox。

(一) 编译 mytime.c 和 test.c，以生成可执行文件 mytime 和 test



```
new@ubuntu: ~/桌面
new@ubuntu:~/桌面$ gcc -o mytime mytime.c
mytime.c: In function 'main':
mytime.c:69:48: warning: integer overflow in expression [-Woverflow]
    time_hour = time_use / (60 * 60 * 1000 * 1000); //****c^
                                                ^
mytime.c:70:48: warning: integer overflow in expression [-Woverflow]
    time_left = time_use % (60 * 60 * 1000 * 1000);
                                                ^
new@ubuntu:~/桌面$ gcc -o test test.c
test.c: In function 'main':
test.c:20:28: warning: implicit declaration of function 'abs' [-Wimplicit-function-decl
ration]
    a[n][lie]=a[n][n]+(abs(n-lie)%26);
                           ^
new@ubuntu:~/桌面$
```

(二) 运行程序 test，不输入第三个参数



```
new@ubuntu:~/桌面$ ./mytime test
Parent process is running!
time_start.tv_sec: 1574158600 second
time_start.tv_usec: 592414 microsecond

Child process is running!
time_start.tv_sec: 1574158600 second
time_start.tv_usec: 592492 microsecond

A B C D E F G H I J K L M N O P Q R S
A B C D E F G H I J K L M N O P Q R S
Z C D E F G H I J K L M N O P Q R S
Y D E F G H I J K L M N O P Q R S
X E F G H I J K L M N O P Q R S
W F G H I J K L M N O P Q R S
V G H I J K L M N O P Q R S
U H I J K L M N O P Q R S
T I J K L M N O P Q R S
S R Q P O N M L K J K L M N O P Q R S

time_end.tv_sec: 1574158600 second
time_end.tv_usec: 593139 microsecond
Time Cost: 0 hour, 0 min, 0 sec, 0 ms, 725 us
new@ubuntu:~/桌面$
```

(三) 运行程序 test，输入第三个参数 3.12345，单位为秒



```
new@ubuntu:~/桌面$ ./mytime test 3.12345
Parent process is running!
time_start.tv_sec: 1574158680 second
time_start.tv_usec: 445946 microsecond

Child process is running!
time_start.tv_sec: 1574158680 second
time_start.tv_usec: 446195 microsecond

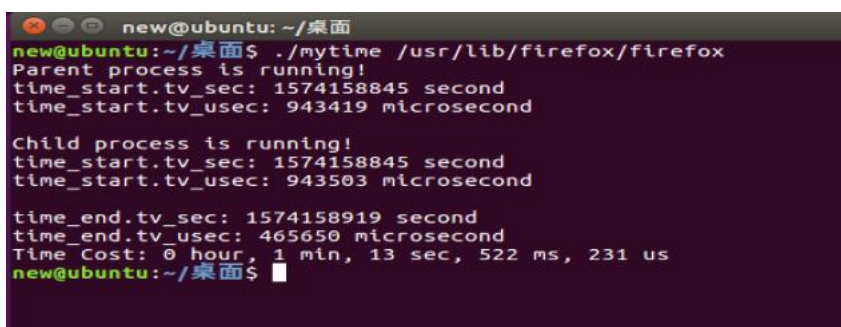
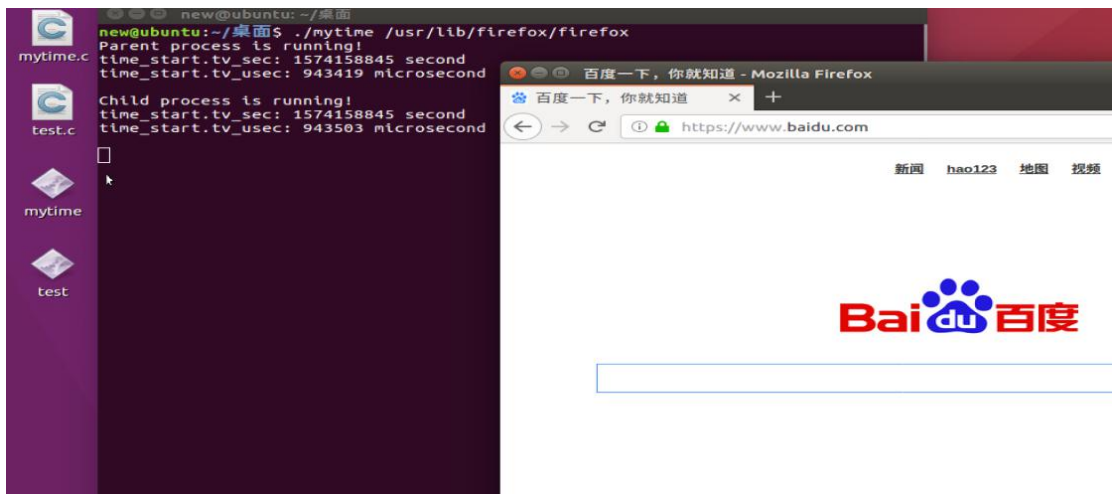
A B C D E F G H I J K L M N O P Q R S
A B C D E F G H I J K L M N O P Q R S
Z C D E F G H I J K L M N O P Q R S
Y D E F G H I J K L M N O P Q R S
X E F G H I J K L M N O P Q R S
W F G H I J K L M N O P Q R S
V G H I J K L M N O P Q R S
U H I J K L M N O P Q R S
T I J K L M N O P Q R S
S R Q P O N M L K J K L M N O P Q R S

process 3545 has been killed

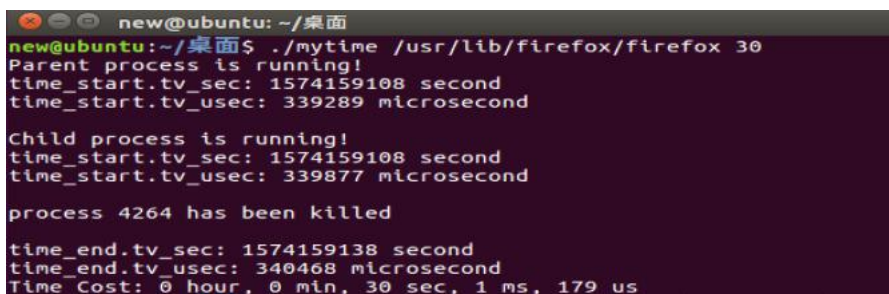
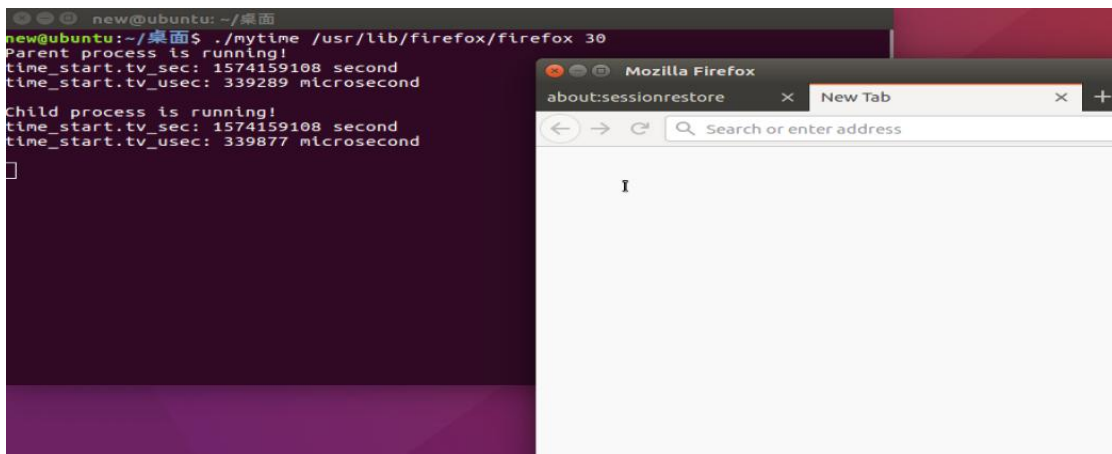
time_end.tv_sec: 1574158683 second
time_end.tv_usec: 570350 microsecond
Time Cost: 0 hour, 0 min, 3 sec, 124 ms, 404 us
new@ubuntu:~/桌面$
```

(四) 运行程序 firefox，不输入第三个参数，一段时间后关闭





(五) 运行程序 firefox，输入第三个参数，设为 30，单位为秒



## 五、 实验收获与体会

这个实验终于做完了，前后做了一下午加一晚上，总体上还算顺利。说实话，

这个实验网上有一些现成的资料，我找到了两份比较好的文档，写的比较详细。但是这两份文档都不完整，有一份是只在 Linux 上实现了两个功能，另外一份是在 Windows 和 Linux 上都只实现了功能一，没有实现输入三个参数的功能，因此我把这两份代码结合，经过反复测试，终于实现了实验所要求的全部功能，总共需要验证 8 次，两个操作系统，输入有没有第三个参数，运行程序是自己写的程序还是系统安装的应用程序，即  $2*2*2=8$  次。我给出这两份博客的链接，供以后回看。

Linux 版本: [https://blog.csdn.net/Crystal\\_ting/article/details/79500147](https://blog.csdn.net/Crystal_ting/article/details/79500147)

双系统功能一: <https://blog.csdn.net/yugemengjing/article/details/72790945>

其中第一份博客写的很详细，很清楚，包括用到的每个 API 接口，并且详细介绍了 argc 和 argv 如何接收输入的参数，我看了几遍，弄清楚了，就不赘述了。总的来说，在 Windows 和 Linux 中，都是调用相应的 API 创建进程、同步进程和获取时间，之后通过 argc 的值为 2 或为 3 判断进入哪个功能模块。如果 argc 为 2，等待进程自然结束；如果 argc 为 3，就让主进程休眠一段时间，然后杀死子进程。等子进程结束后，记录结束时间，通过开始时间和结束时间就能换算出应用程序的运行时间。但是 Windows 和 Linux 具体实现的函数接口还是有很大的不同，因此我花了很多时间适配接口，我在前面的实验设计版块中也在多处比较了两者的不同，此处不再赘述，下面列出几点我花了很长时间踩过的坑。

#### （一）不知道 Ubuntu 中系统应用程序的存放位置

由于 Ubuntu 中的文件没有后缀名，因此不好查找可执行程序所在位置，而在命令行中通过 top 查看所有进程 PID，通过 cd /proc/PID 切换到进程所在的文件夹，然后通过命令 ls -l exe 即可查看可执行程序的绝对路径。

参考链接: <https://blog.csdn.net/taoerchun/article/details/82851076>

#### （二）Windows 中不清楚如何结束 CreateProcess 创建的子进程

这个问题困扰了我好久好久，主要是因为另外一个问题（等下再说）。Linux 是通过 kill(pid, SIGKILL) 杀死 pid 对应的进程，而 Windows 中则是使用 TerminateProcess(pi.hProcess, exitCode) 结束 pi 对应的进程，实现代码为：

```
DWORD exitCode; //退出码
//.....
GetExitCodeProcess(pro_info.hProcess, &exitCode); //获取退出码
TerminateProcess(pro_info.hProcess, exitCode);
// 关闭句柄
CloseHandle(pro_info.hThread);
CloseHandle(pro_info.hProcess);
```

参考链接: <https://blog.csdn.net/duhuzhen/article/details/53738062>

#### （三）调用 TerminateProcess 不能结束某些应用程序

这个问题真是巨坑，由于在 Linux 中我是通过 mytime 运行 firefox 浏览器，因此在 Windows 中我是想尝试运行 chrome 浏览器，但是无论调用哪个 API 我都无法将其关闭，只能结束主进程，这导致我反复查询资料，耗费了两三

个小时。最后在看到某份博客时有人同样提到了用 `TerminateProge` 只结束主进程，没有结束子进程，有人回复可能是权限的问题。因此我就尝试运行其他应用程序，果然成功结束，问题终于解决了，所以有时候需要多想想问题可能出在哪。