# 《操作系统》实验

## 实验一：进程和线程的创建

### 实验要求

1． 在linux下编写一个应用程序，命名为an\_ch2\_1b。这个程序不断地输出如下行：

Those output come from child,[系统时间]

另外写一个应用程序，命名为an\_ch2\_1a。这个程序创建一个子进程，执行an\_ch2\_1b。这个程序不断地输出如下行：

Those output come from child,[系统时间]

观察程序运行的结果，并对你看到的现象进行解释。

2。在linux环境下编写一个控制台应用程序，程序中有一个共享的整型变量shared\_var，初始值为0；创建一个线程并使其立即与主线程并发执行。新创建的线程与主线程均不断地循环，并输出shared\_var 的值。主线程在循环中不断地对shared\_var 进行加1操作，即每次循环shared\_var 被加1；而新创建的线程则不断地对shared\_var 进行减1 操作，即每次循环shared\_var 被减1。观察程序运行的结果，并对你看到的现象进行解释。

### 实验过程

#### 进程

1. 用一个while循环，不断获取系统时间并打印，为了便于观察，加入睡眠时间。
2. 用gcc编译an\_ch2\_1b.c，生成an\_ch2\_1b.out，准备在an\_ch2\_1a中执行。
3. fork()创建子进程，调用execl函数执行an\_ch2\_1b。
4. 代码见an\_ch2\_1b.c,an\_ch2\_1a.c，可执行程序an\_ch2\_1b.out,an\_ch2\_1a.out

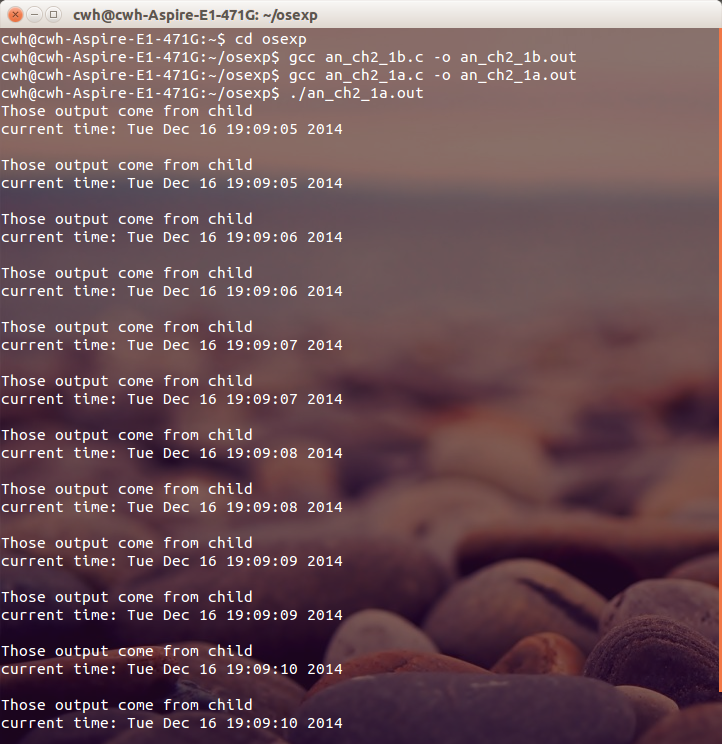
#### 线程

1. 主线程循环对share\_var自增，新线程循环对share\_var自减。
2. 代码见mythread.c，可执行程序mythread.out

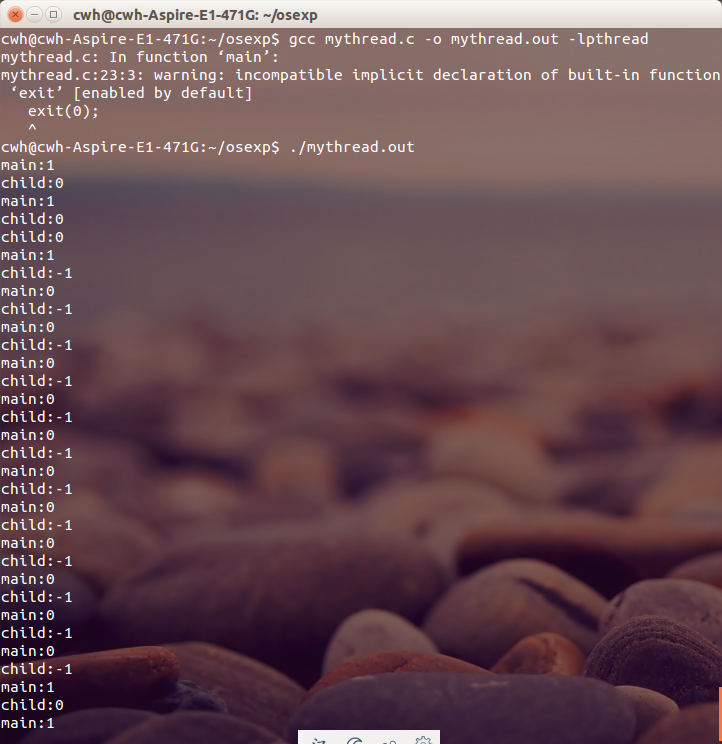
### 实验结果与分析

#### 进程

因为fork成功后返回两个值，一个为0一个为子进程pid，即if-else实际上产生3个分支，fork成功的两个分支都执行了an\_ch2\_1b.out，



#### 线程



主程序和线程同时对shared\_var做修改，所以shared\_var的值在0附近抖动。

## 实验二：进程和线程同步和互斥

### 实验要求

#### 1．生产者消费者问题（信号量）

参考教材中的生产者消费者算法，创建5个进程，其中两个进程为生产者进程，3个进程为消费者进程。一个生产者进程试图不断地在一个缓冲中写入大写字母，另一个生产者进程试图不断地在缓冲中写入小写字母。3个消费者不断地从缓冲中读取一个字符并输出。为了使得程序的输出易于看到结果，仿照的实例程序，分别在生产者和消费者进程的合适的位置加入一些随机睡眠时间。

可选的实验：在上面实验的基础上实现部分消费者有选择地消费某些产品。例如一个消费者只消费小写字符，一个消费者只消费大写字母，而另一个消费者则无选择地消费任何产品。消费者要消费的产品没有时，消费者进程被阻塞。注意缓冲的管理。

#### 2．用线程实现睡觉的理发师问题

理发师问题的描述：一个理发店接待室有n张椅子，工作室有1张椅子；没有顾客时，理发师睡觉；第一个顾客来到时，必须将理发师唤醒；顾客来时如果还有空座的话，他就坐在一个座位上等待；如果顾客来时没有空座位了，他就离开，不理发了；当理发师处理完所有顾客，而又没有新顾客来时，他又开始睡觉。

#### 3．读者写者问题

教材中对读者写者问题算法均有描述，但这个算法在不断地有读者流的情况下，写者会被阻塞。编写一个写者优先解决读者写者问题的程序，其中读者和写者均是多个进程，用信号量作为同步互斥机制。

### 实验过程

#### 生产者消费者问题（信号量）

1. 重新封装semop函数，来实现down和up操作（buf.sem\_op的值不同）
2. 开辟一个共享内存空间存放字符串buffer，存放操作的字符串。
3. 用mutex来控制同一时刻只有一个进程能读写buffer，用full来标志buffer的“满”状态，使得buffer有字符存在时才能读出字符，用empty来标志buffer的“空”状态，使得buffer有空位时才能写入。（最大10个字符）
4. 父进程循环输出buffer内容，Fork()创建五个子进程，producer1插入大写字母，producer2插入小写字母，consumer1消费大写字母，consumer2消费小写字母，consumer3消费任意字符。为了便于观察运行结果，加入了睡眠时间。
5. 对输出进行颜色控制区分生产者与消费者。
6. 代码见osexp2\_1.c，可执行程序osexp2\_1.out

#### 用线程实现睡觉的理发师问题

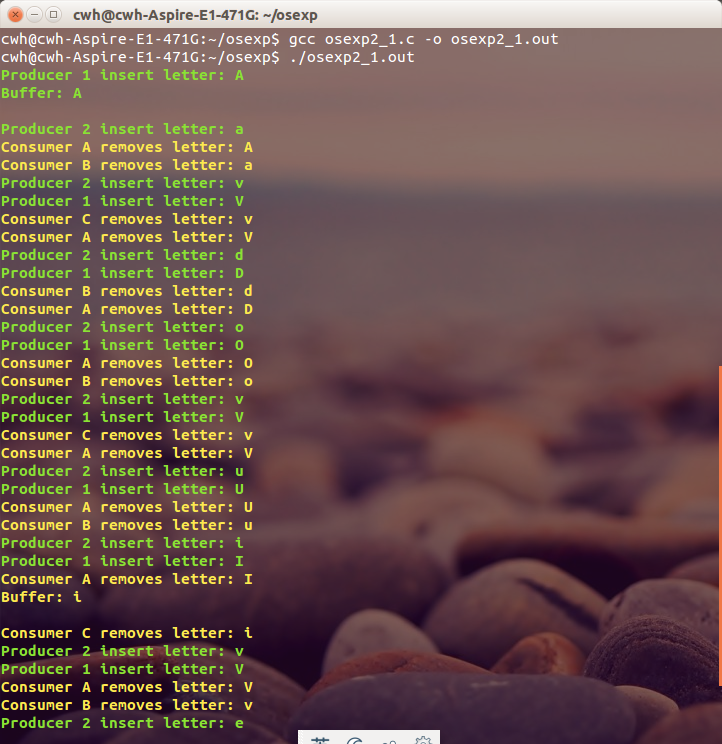
1. 创建3个信号量mutex，customers，barbers。mutex保证同一时刻只有一个线程对count（排队的客人数量）做操作；customers用于标志客人队列的状态，只有客人队列非空时，理发师才能执行理发操作；barbers用于标志理发师是否正在理发，理发师空闲时，客人才能占用理发师进行被理发操作。
2. 理发师3秒完成一次理发操作，每一秒进入一个客人，主进程创建一个理发师线程循环处理进入店里的客人，创建十个顾客线程。
3. 理发时，排队人数减少，输出正在理发以及排队人数
4. 客人进入理发店时，若还有空位，排队人数增加，等待理发师空闲，如果没有空位，输出客人离开的消息。
5. 对输出进行颜色控制区分理发师和客人的不同行为。
6. 代码见osexp2\_2.c，可执行程序osexp2\_2.out

#### 读者写者问题

1. 重新封装semop函数实现down和up操作（buf.sem\_op）的值不同
2. 写者策略：有一个读者在读时，等待读者读完，写者才开始写。如果有多个读者正在排队，写者只需跟第一个读者和其他的写者竞争，有不断的写者流时，第一个写者等待第一个读者读完后，就占用first\_read，直到没有其他写者才释放，避免读者插队。其他写者都在\_write上排队。
3. 读者策略：先检查是否有写进程，如果有，第一个读者会在first\_read排队，后续读者在write\_first排队，保证写者优先，读者进入后释放write\_first，使得下一个读者在first\_read排队等待写者，如果没有写者则进入，实现读者同时读。
4. 创建五个信号量，first\_read保证读写互斥，\_write保证写者互斥,writer\_first保证写者不需在读者后面排队，而读者需要在写者后面排队，提高写者优先级。
5. 对输出进行颜色控制区分读者写者。
6. 代码见osexp2\_3.c，可执行程序osexp2\_3.out

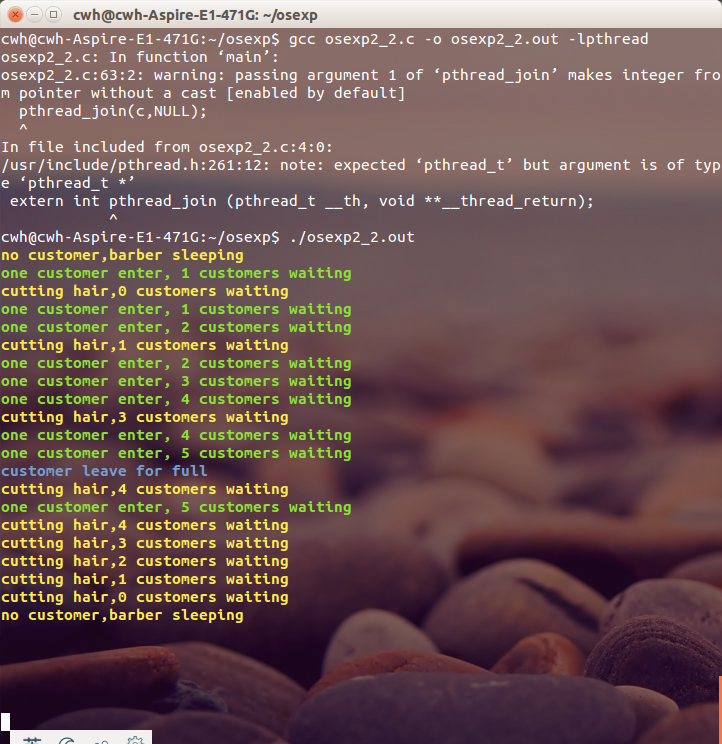
### 实验结果与分析

#### 1.生产者消费者问题



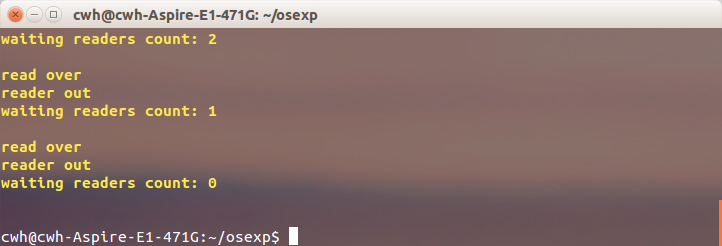
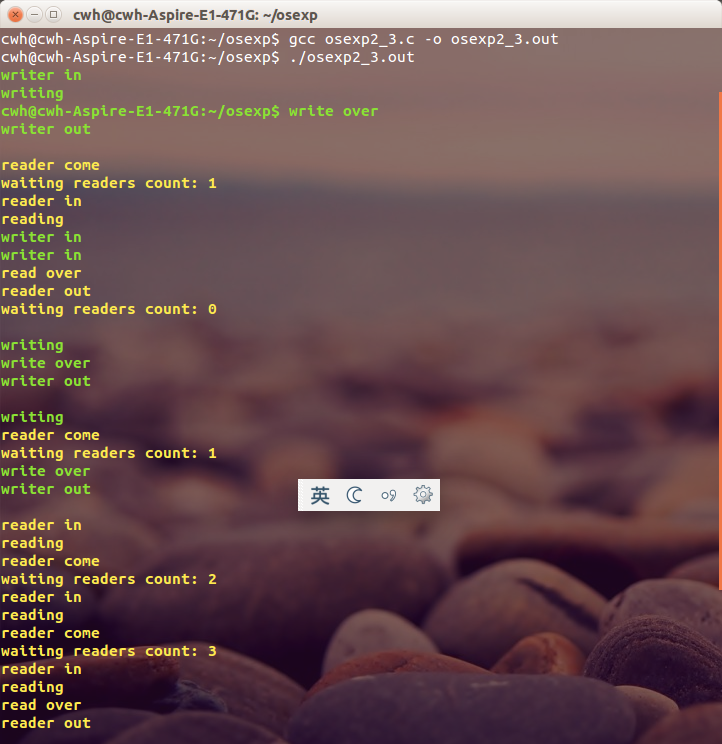
生产者在缓冲区尾部插入字符，消费者在缓冲区头部提取字符，并更新缓冲区，显示操作的字符信息。

#### 2.用线程实现睡觉的理发师问题



一开始没有客人，理发师睡觉，然后客人进入，理发师每两秒理一次发，也就是每秒有一个客人离开，又因为每秒一个客人进入，客人越来越多，店里只有五张椅子，第九个客人进来时没有座位，客人离开，第十个客人进入后，理发师对五个客人依次理发，直到没有客人，开始睡觉。

#### 3.读者写者问题



Writer in或reader come即为排队，若不需排队则进入临界区，开始reading或writing，

一开始写者1进入完成写操作，写完离开，读者1进入，开始读，这时两个写者进入，写者2等待读者1读完离开，开始写，写者3排队，写者2写完，写者3,开始写，这时读者2来了，在写者3后排队，写者3写完，读者2开始读，同时其他后续读者也开始并行地读。

## 实验三：实现一个文件查找程序

### 实验要求

仿照unix操作系统中的find命令，在实现一个myfind命令。myfind命令从指定的目录下开始，递归地查找指定文件。其命令格式如下：

**myfind PATH -option 参数 [-print] [-exec 命令 {} \;]**

PATH：查找的起始目录。

-option 参数：用来控制搜索方式，具体如下（以下只给我常用的）：

    -name "文件"：指定要查找的文件名，可以有通配符?\*等。

    -prune 目录：指出搜索时不搜索该目录。

    -mtime +n或-n：按时间搜索，+n：表示n天之前修改过的文件；-n：表示今天到n天前之间修改过的文件。

    -ctime +n或-n：也是按时间查找，+n：表示n天之前创建的文件；-n：表示今天到n天前之间创建的文件。

   -print：将搜索结果输出到标准输出。

可选的实验1：实现-exe选项。

    -exec：对查找到的每一个结果实行指定的程序。格式如下：

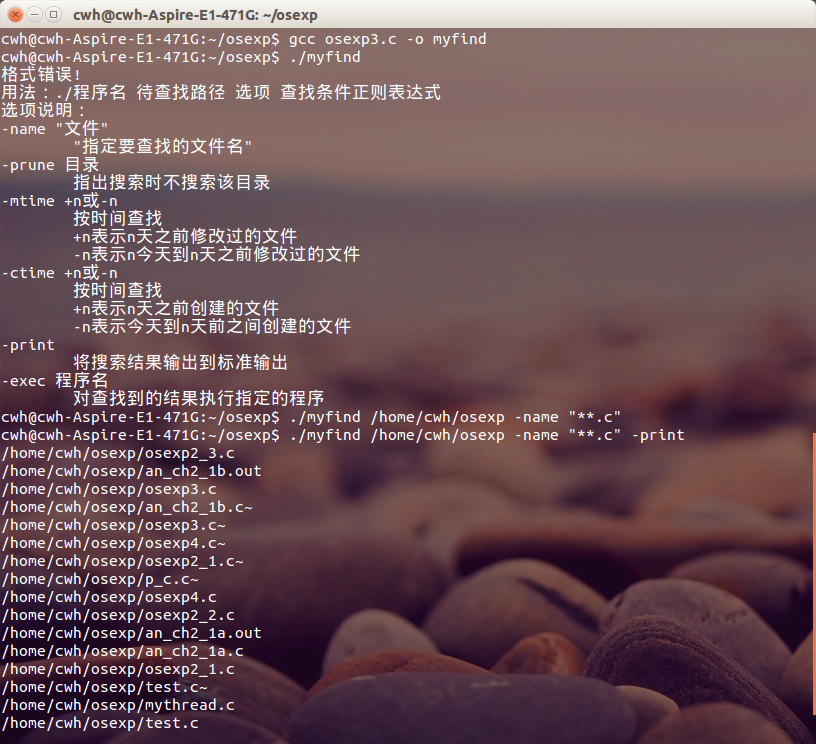
    -exec 程序名

提示：执行指定的程序可以创建一个进程执行

### 实验过程

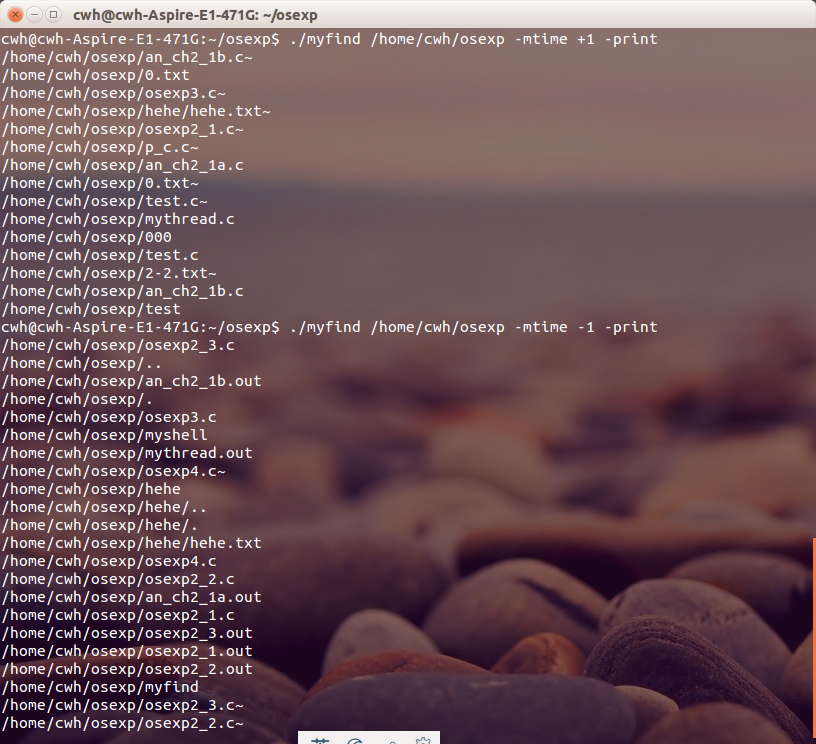
1. void init(int argc,char\*argv[]);获取myfind程序执行参数，确定要执行的命令；
2. 用DIR对象存放文件流，用dirent对象接收单个文件，如果读到的文件为一个文件夹（目录），递归执行myfind函数；
3. 用六个标志变量来作为开关控制执行选项。
4. 对每个函数进行各个选项的条件的匹配
5. 用正则表达式对文件名信息进行匹配查找
6. 用difftime函数来进行时间匹配查找
7. 用execvp函数控制是否执行额外的程序
8. 代码见osexp3.c，test.c,可执行程序myfind,test

### 实验结果与分析

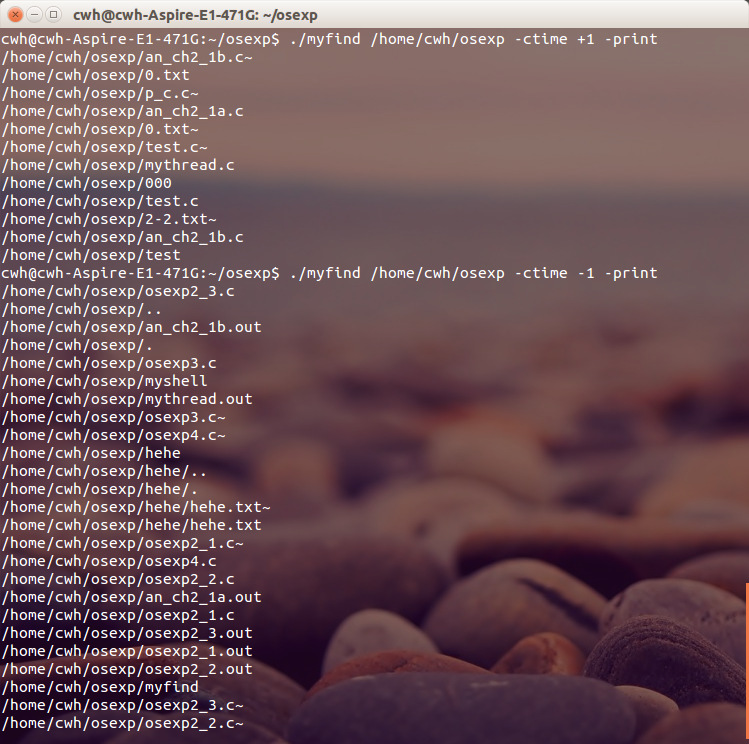


只输入./myfind或者参数格式错误，会显示帮助，

使用-name选项，引号内为名字格式，两个\*代表任意多个字符，没有加-print的时候只查找，不输出，加上-print后输出查找结果。



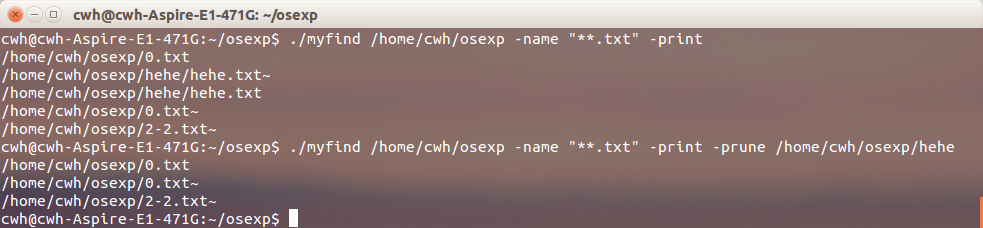
然后按修改时间查找，找到1天前修改的文件和1天前到今天修改过的文件。



然后按创建时间查找，找到1天前创建的文件和1天前到今天创建的文件。



对找到的文件执行test，test程序输出hello world。



-prune选项启动前，会查找hehe文件夹下的文件，启动后，不会查找。

## 实验四：实现一个unix命令解释程序

### 实验要求

1．在linux中实现一个命令执行程序doit，它执行命令行参数中的命令，之后统计

1）命令执行占用的CPU时间(包括用户态和系统态时间，以毫秒为单位)，

2）命令执行的时间，

3）进程被抢占的次数，

4）进程主动放弃CPU的次数，

5）进程执行过程中发生缺页的次数

2．在linux中实现一个简单的命令解释程序，功能要求：

1）同时支持内部命令和外部命令，内部命令支持两个（cd、exit）

2）支持后台命令

提示：实验中可能用到的系统调用如下：

• fork() — 创建一个新进程

• getrusage() — 取得进程的资源使用情况

• gettimeofday() — 取当前的时间

• execve() — 装入一个程序并执行

• wait() — 等待子进程结束

• chdir() — 改变进程的工作目录

• strtok() — 字符串解析

### 实验过程

**(内部命令除了cd和exit命令外，借用实验三的思路，实现ls命令)**

1. 获取进程执行前后系统时间，计算差值，可以得到进程运行时间
2. 通过getrusage()方法获取进程资源使用情况，包括用户态时间，系统态时间，进程被抢占次数，进程主动放弃CPU次数，缺页中断次数
3. 获取到进程cd的目的路径后，用chdir()方法切换进程的工作目录
4. 用实验三的思路实现LS命令（对找到的文件和文件夹的颜色进行区分）
5. 用守护进程的实现后台命令（用&标志一个进程是后台的）
6. exit命令：执行后，waitpid等待所有子进程（后台进程）结束
7. 外部命令，fork产生子进程后，用execve方法调用计算机bin目录下的程序。
8. 代码见osexp4.c，,可执行程序myshell，（后台程序）。

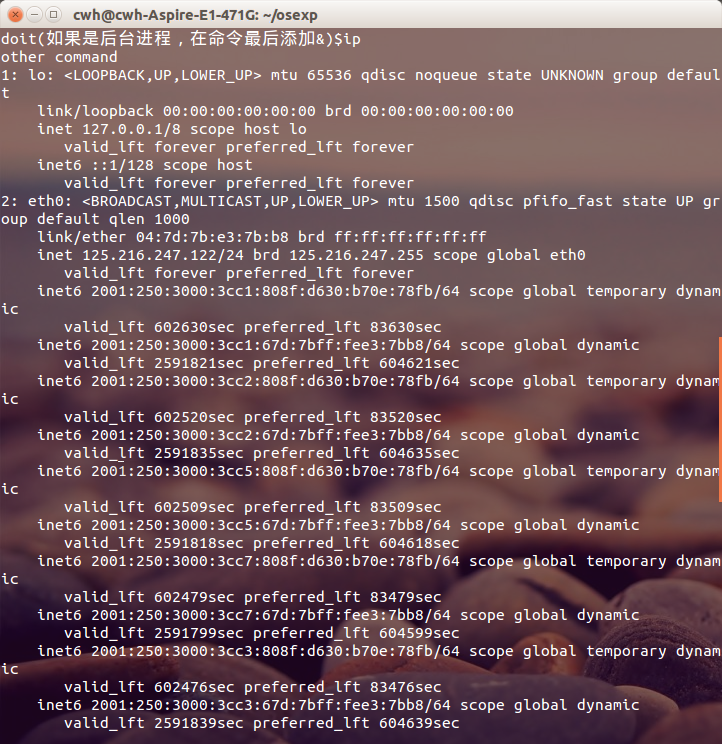
### 实验结果与分析



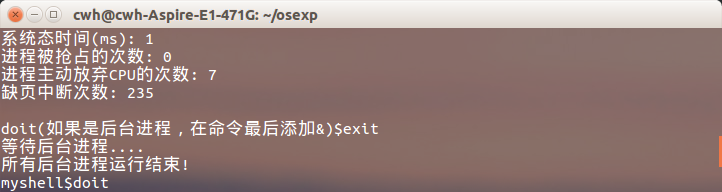
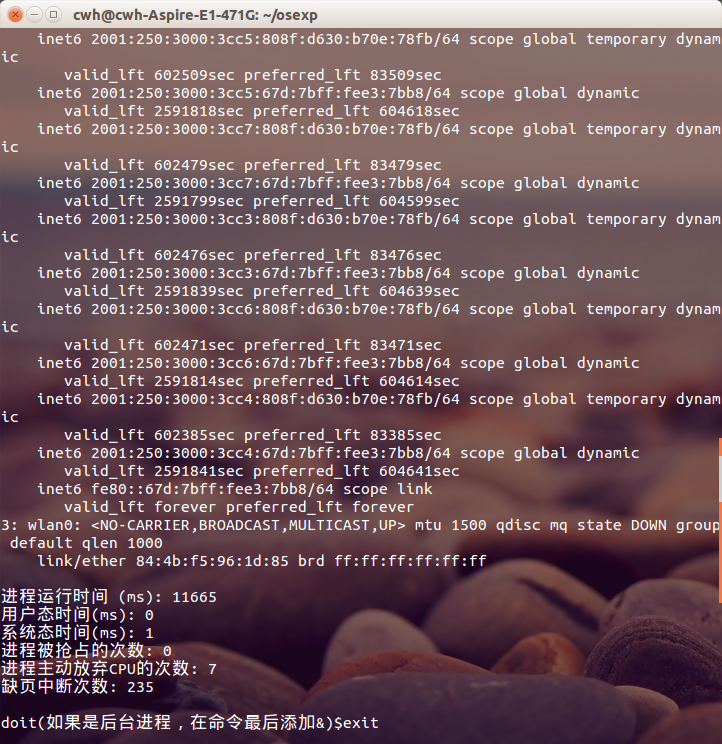
Ls命令查找当前文件夹下的目录和文件，并输出命令执行时间。



切换到hehe目录下，执行ls操作显示这个目录下的文件。

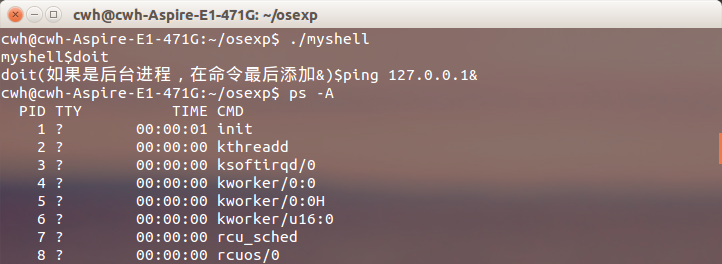


执行ip命令（外部命令）



执行exit命令退出doit。

然后执行后台命令ping（这个命令会在很长一段时间内一直执行）：



执行ping后，程序转入后台，回到终端。执行ps –A后可以看到myshell和ping在后台执行

