华东师范大学数据学院上机实践报告

课程名称： 操作系统　　**年级：**大二　　**上机实践成绩：**

指导教师：　　**姓名：**沈小奇

上机实践名称：　　**学号：** 10185501401　　**上机实践日期：**

上机实践编号：

一、 **目的**

实现一个简易的 shell

二、 **内容与设计思想**

进程管理、文件管理

三、 **使用环境**

Minix，mobaxterm

四、 **实验过程**

总体：用一个 while 循环模拟 shell 的运作方式。Shell 里的命令分为内置命令与外部命令，内置命令自己写程序执行，外部命令用 exec 系列函数执行。

准备工作：

While 循环里包括：①输入，将命令行中命令读进去

②将命令拆解，去掉空格，每个字符串存储在 argv[][]里

```
while(*buf!='\n'){

        while(buf[i]!='\n'&&buf[i]!=' '){
            i++;
        }

        if(buf[i] == '\n'){
            buf[i] = '\0';
            argv[argc++] = buf;
            break;
        }
        buf[i] = '\0';
        argv[argc++] = buf;

        buf+=i+1;

        i = 0;
        while(*buf==' '){
            buf++;
        }
}
```

③写区分内置和外置命令的函数

该实验里内置的只有 cd,history,exit，因此编写一个函数执行这几条命令，执行

完直接返回给主函数，其余的外置函数另外再写，主要通过 exec 函数来执行。

Task1:cd /your/path
通过调用 chdir(argv[1])函数，打开 cd 后面跟的目录名。

Task2：ls –a –l
这是一个外部命令，通过调用 execvp(argv[1],argv)来执行

Task3：ls –a –l > result.txt
将左边内容重定向到右边的文件中，需要先打开一个文件（设置为如果没有该文件就自动创建一个新文件），将文件描述符对应于标准输出文件，这样就可以输出到文件中。

```
fd=open(file,O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
                dup2(fd,1);
                execvp(temp[0],temp);
               exit(0);
```

Task4：vi result.txt
调用 execvp()函数直接执行

Task5： grep a < result.txt
将 result.txt 作为内容输入给 grep a
关闭标准输入，将打开的文件对应到标准输入

```
if((fd = open(file,O_RDONLY,0644)) < 0)
    printf("openfile error\n");
close(0);//关闭标准输入
dup2(fd,0);//将打开的文件对应到标准输入
execvp(temp[0],temp);
    exit(0);
```

Task6:ls –a –l | grep a
分两步，子进程：先对|左边修改，关闭标准输出，设置一个临时文本文件作为标准输出。执行左边完以后关闭该文件描述符。
然后对|右边修改。父进程：等待子进程执行完以后，关闭标准输入，将临时文件作为标准输入，然后执行|右半边。

```
if(pid2 == 0)
                {
                    if((fd2 =
open("/tmp/1.txt",O_WRONLY|O_CREAT|O_TRUNC,0644)) < 0)
                        perror("open");
                    dup2(fd2,1);//打开的文本文件定位到标准输出
                    execvp(temp[0],temp);//执行左边
```

```
            exit(0);
        }
        waitpid(pid2,&satu,0);//等待子进程执行完
        close(fd2);//关闭输出
        fd2 = open("/tmp/1.txt",O_RDONLY);
         dup2(fd2,0);//定位到标准输入
        execvp(temp1[0],temp1);//执行右边
        if(remove("/tmp/1.txt") < 0)//把这个临时文件移走
            perror("remove error");
        exit(0);
        break;
    }
```

Task7:vi result.txt &

&是后台运行符号，表示后台执行，没有返回值

关闭文件描述符

重定向到/dev/null 文件中

```
execvp(temp[0],temp);
  int devnullfd;
  devnullfd = open("/dev/null", 0);
  dup2(devnullfd, 0);
  dup2(devnullfd, 1);
  dup2(devnullfd, 2);
// 处理 SIGCHLD 信号
  signal(SIGCHLD,SIG_IGN);

  return;
```

Task8：mytop

运行时间 = 内核时间 加 用户时间 减去 空闲时间

间隔时间 = 内核时间 加 用户时间

cpu 使用率% = 运行时间 / 间隔时间 ;

参考了 minix 源码,照着助教给的方法执行的。计算进程和任务总数 nr_total，读取/proc/kinfo

读取每个进程的信息，遍历/proc/pid/psinfo

如果 type==task，则进程标记 p_flags |=istask

如果 type==system，则进程标记 p_flags |=issystem

如果 state！=state_run,则标记 p_flags |=blocked
PUTIMENAMES 次 cycles_hi,cycle_lo（三个 cputimenames），然后拼接成 64 位，放在 p_cpucycles[]数组中。
计算每个进程 proc 的滴答，通过 proc 和当前进程 prev_proc 做比较，如果 endpoint 相等，则在循环中分别计算

```
for(i = 0; i < CPUTIMENAMES; i++) {
   if(!CPUTIME(timemode, i))
       continue;
   if(proc->p_endpoint == prev_proc->p_endpoint) {
       t = t + prev_proc->p_cpucycles[i] - proc->p_cpucycles[i];
   } else {
       t = t + prev_proc->p_cpucycles[i];
   }
}
```

计算总的 cpu 使用百分比，遍历所有的进程和任务，判断类型，计算 systemticks，userticks（由于 kernelticks 和 idleticks 为 0 不用计算）。

```
if(!(proc2[p].p_flags & IS_TASK)) {
          if(proc2[p].p_flags & IS_SYSTEM)
              systemticks = systemticks + tick_procs[nprocs].ticks;
          else
              userticks = userticks + tick_procs[nprocs].ticks;
   }
```

在showtop()函数中，打印top上显示的信息。本实验需要的是print_memory()和print_procs(),打印出内存和CPU使用情况。print_memory()需读取/proc/meminfo文件信息，print_procs()则要获取到每个进程和任务的信息，通过get_procs()函数将所有需要的信息放在结构体数组proc[]中，每个元素都是一个进程结构体。

get_procs()函数，首先记录当前进程，赋值给prev_proc，然后通过parse_dir()函数获取到/proc/下的所有进程pid,再通过parse_file()函数获取每一个进程信息，即读取/proc/pid/psinfo文件。

在parse_file()函数中读取的信息需要判断是否可用。比如version是否为1，如果不是该进程不需要记录。在判断slot时，需要用到SLOT_NR(endpt)函数，不过该函数有些问题，判断出来的slot大于nr_total,所以自己修改一下slot的赋值。在源码下有一句p = &proc[slot];所以了解到slot就是该进程结构体在数组中的位置，可以简单通过其他赋值，比如slot++，只要在数组中不会重复即可。该函数会给进程结构体变量赋值，看源码即可。也可按照源码一样全部变量都赋值，供后面使用。

再创建一个tp结构体，这个结构体包含了进程指针p和ticks，对应的就是某个进程和滴答。

在cputicks()函数中，计算每个进程的滴答。滴答并不是简单的结构体中的滴答，因为在写文件的时候需要更新。需要通过当前进程来和该进程一起计算，这里需要用到p_cpucycles,在前面赋值的时候已经写进进程结构体了。

print_procs()函数中就输出 CPU 使用时间，这是最后也是最重要的。这里创建了一个 tp 结构体的数组 tick_procs。对所有的进程和任务(即上面读出来的 nr_total)计算 ticks，具体看源码。把 kernelticks，userticks，systemticks 相加就可以得到 CPU 的使用百分比。在计算 idleticks 时因为 IDLE 已经大于 nr_total,所以计算出的 idleticks 恒为 0.

Task9:history n

设置一个二维数组 history[][]来储存每一条命令

strcpy(history[his_count],cmdstring);

his_count++;（命令数自加）

当调用 history n 命令时，输出倒数 n 条执行的命令

```
if(strcmp(argv[0],"history")==0){
        for(int i=atoi(argv[1]);i>=0;i--){
        printf("%s",history[i]);
        }
        return 1;
    }
```

Task10：exit

Exit 是内置命令，放在内置测试函数里。

```
if(strcmp(argv[0],"exit")==0)
exit（0）;
```

直接退出

五、总结

该实验主要是对内置命令和外置命令的 C 语言实现，一开始不太懂实验的要求，走了不少弯路。后来方向清楚了做起来轻松一些。明白了内置命令需要自己通过调用一些函数来实现。外置命令主要通过 exec 系列函数执行，但是外置命令中的重定向、管道等符号需要通过操作文件描述符来执行。Mytop 函数比较复杂，需要对进程的结构有所掌握。需要计算三种状态的时间，来计算 cpu 的利用率。将进程写作结构体，找到所有的 pid 号，计算所有进程的滴答数，分别计算 userticks 和 systemticks,最终算出百分比。