实验1　 **Linux基本环境**

学号： 201970504109 姓名： 边明豪 班级： 计201-1

时间： 2022-4-20 地点： 线上 指导老师：\_\_翟一鸣\_\_\_\_\_

一、实验目的

（1）熟悉Linux下的基本操作，学会使用各种Shell命令去操作Linux，对Linux有一个感性认识。

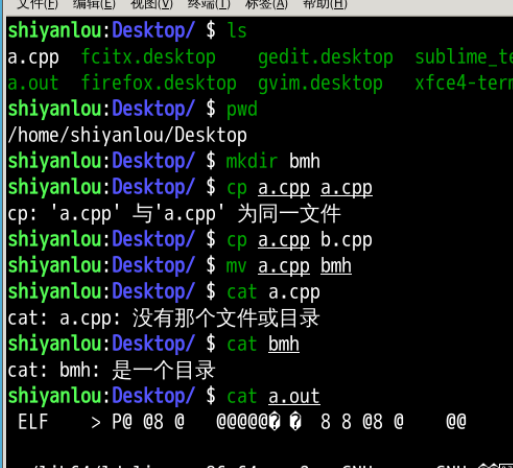
（2）学会使用vi编辑器编辑简单的C语言程序，并能对其编译和调试。

二、实验内容1

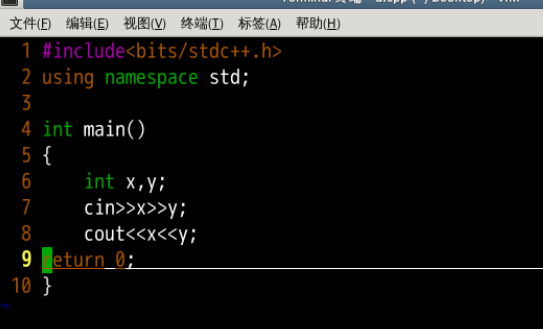
1. 问题描述

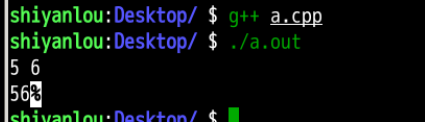
（1） 以root用户身份登陆，并使用“ls”,“cat”“cd”等命令来实现基本的文件操作并观察Linux文件系统的特点；





（2） 使用vi编辑器编写一C程序，并用gcc命令进行编译和链接，并用a.out来进行输出结果。





总结

使用命令行的方法使用了Linux系统，体验到了除了windows之前其他系统的功能，非常强大。Linux的基本命令还需要更加熟悉，以便于以后在linux行走更远。

实验2　进程管理

学号： 201970504109 姓名： 边明豪 班级： 计201-1

时间： 2022-4-27 地点： 线上 指导老师：\_\_翟一鸣\_\_\_\_\_

一、实验目的

（1）加深对进程概念的理解，明确进程和程序的区别。

（2）进一步认识并发执行的实质。

（3）分析进程竞争资源现象，学习解决进程互斥的方法。

（4）了解Linux系统中进程通信的基本原理。

二、实验内容1

1. 问题描述

进程的创建

编写一段程序，使用系统调用fork( )创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符；父进程显示字符“a”，子进程分别显示字符“b”和“c”。试观察记录屏幕上的显示结果，并分析原因。

2. 数据结构及算法设计

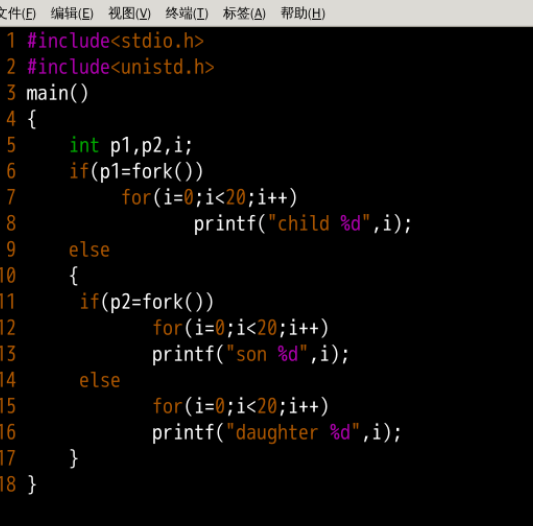
2.1

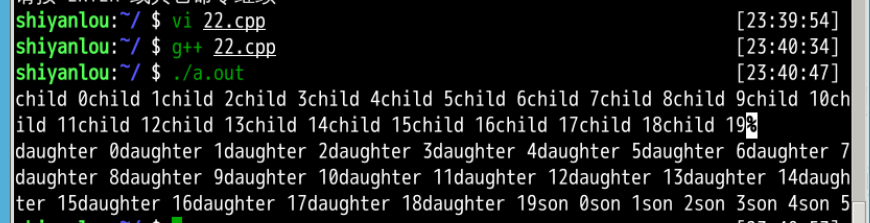


运行结果： 多次运行，总是bca，因为父进程，子进程，孙子进程依次创建。所以 依次输出



2.2



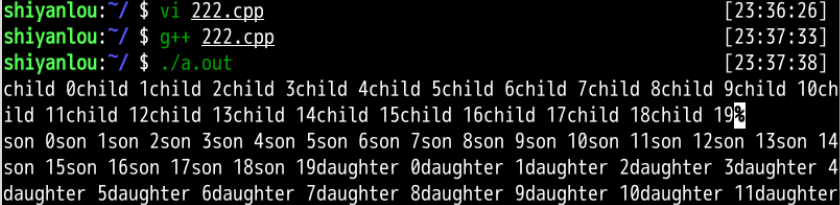


由于函数printf()输出的字符串之间不会被中断，因此，字符串内部的字符顺序输出时不变。但是 , 由于进程并发执行时的调度顺序和父子进程的抢占处理机问题，输出字符串的顺序和先后随着执行的不同而发生变化。这与打印单字符的结果相同。

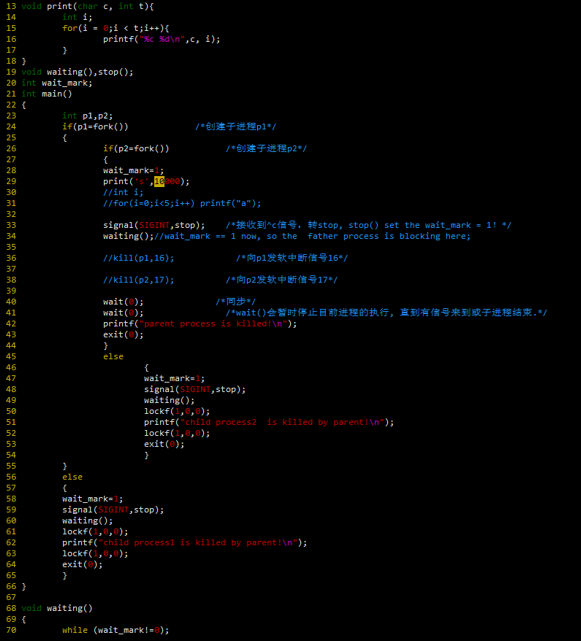
2.3加上lockf()之后的效果

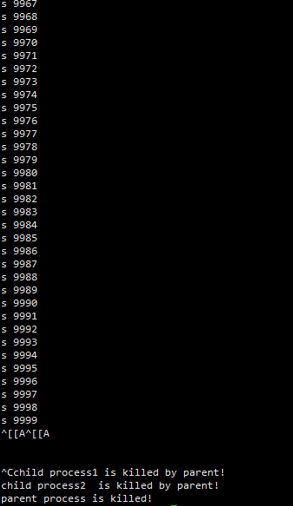


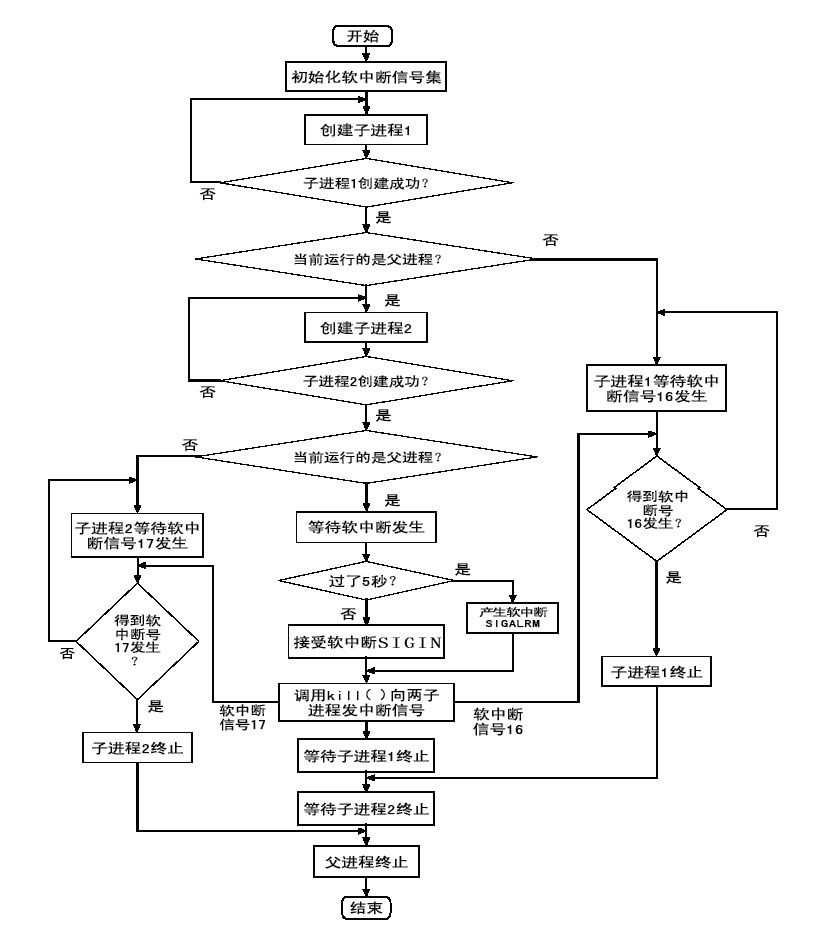
与加之前效果相同，猜测是服务器执行速度快导致的。



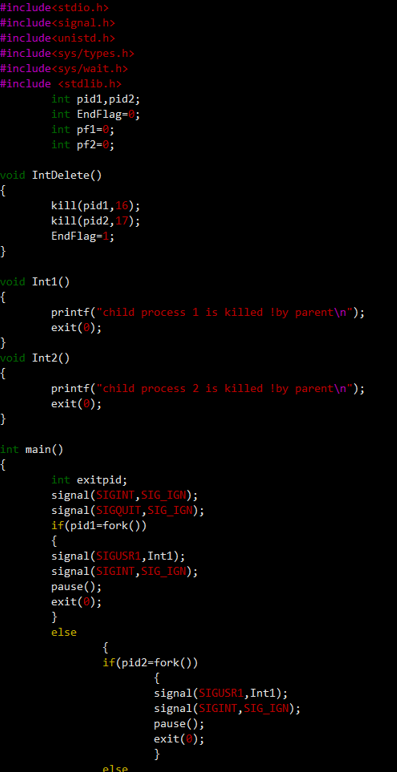
2.4加了中断调用，程序监听Ctrl+C中断。



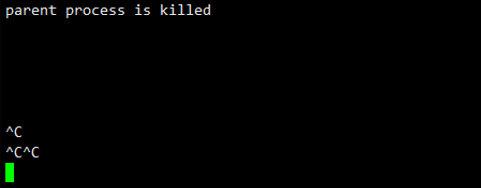




2.5 在上面的任务1中，增加语句signal(SIGINT,SIG\_IGN)和语句signal(SIGQUIT,SIG\_IGN)，观察执行结果，并分析原因。这里，signal(SIGINT,SIG\_IGN)和signal(SIGQUIT,SIG\_IGN)分别为忽略键信号以及忽略中断信号。



运行结果如下：由于忽略了中断与退出信号，程序会一直保持阻塞状态而无法退出。

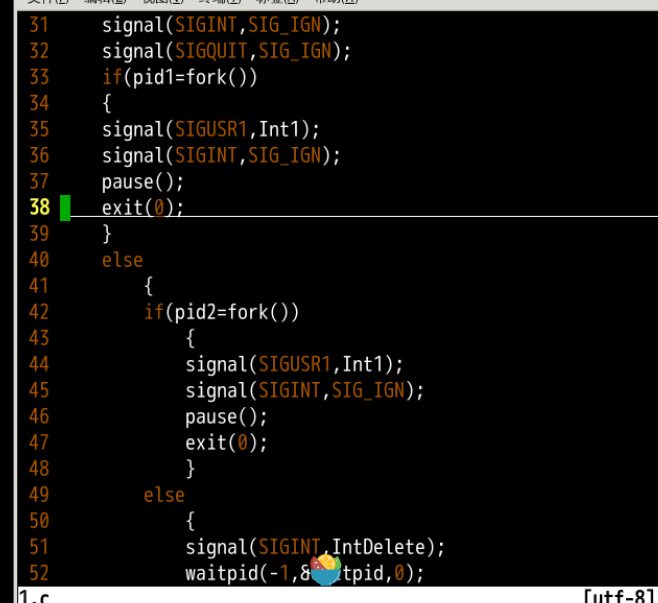


2.6使用系统调用pipe()建立一条管道线。两个子进程p1和p2分别向通道个写一句话：

child1 process is sending message!

child2 process is sending message!

而父进程则从管道中读出来自两个进程的信息，显示在屏幕上。



运行结果如下



思考：

1. 程序中的sleep(5)起什么作用？

sleep把进程的运行状态改为睡眠，将其从系统可执行队列去掉，这样系统就不会调度到该进程，不会分配CPU时间片。同时根据该进程的睡眠时间，将进程挂入相应的定时器队列中。

2. 子进程1和2为什么也能对管道进行操作？

在linux的pipe管道下，在写端进行写数据时，不需要关闭读端的缓冲文件(即不需要读端的文件描述符计数为0)，但是在读端进行读数据时必须先关闭写端的缓冲文件(即写端的文件描述符计数为0)然后才能读取数据。

实验3 进程间通信

学号： 201970504109 姓名： 边明豪 班级： 计201-1

时间： 2022-5-05 地点： 线上 指导老师：\_\_翟一鸣\_\_\_\_\_

一、实验目的

Linux系统的进程通信机构 (IPC) 允许在任意进程间大批量地交换数据。本实验的目的是了解和熟悉Linux支持的消息通讯机制及信息量机制。

二、实验内容1

1. 问题描述

使用系统调用msgget( ), megsnd( ), msgrev( )及msgctl()编制一长度为1K的消息发送和接收的程序 。

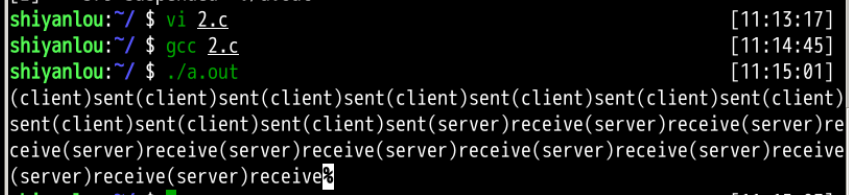
2. 数据结构及算法设计

1. 为了便于操作和观察结果，用一个 程序为“引子”，先后fork( )两个子进程，SERVER和CLIENT，进行通信。
2. SERVER端建立一个Key为75的消息队列，等待其他进程发来的消息。当遇到类型为1的消息，则作为结束信号，取消该队列，并退出SERVER 。SERVER每接收到一个消息后显示一句“(server)received”。
3. CLIENT端使用Key为75的消息队列，先后发送类型从10到1的消息，然后退出。最后的一个消息，既是 SERVER端需要的结束信号。CLIENT每发送一条消息后显示一句“(client)sent”。
4. 父进程在 SERVER和 CLIENT均退出后结束。

3. 程序实现



4. 测试



从理想的结果来说，应当是每当Client发送一个消息后，server接收该消息，Client再发送下一条。也就是说“(Client)sent”和“(server)received”的字样应该在屏幕上交替出现。实际的结果大多是，先由 Client 发送数十条消息，然后Server接收数十条消息。个人认为是由于计算机运行速度过快导致。

实验内容2

1. 问题描述

使用系统调用shmget(),sgmat(),smgdt(),shmctl()编制一个与上述功能相同的程序.

2. 数据结构及算法设计

(1)为了便于操作 和观察结果，用一个 程序为“引子”，先后fork( )两个子进程，SERVER 和 CLIENT，进行通信。

(2)SERVER端建立一个KEY为75的共享区,并将第一个字节置为-1.作为数据空的标志.等待其他进程发来的消息.当该字节的值发生变化时,表示收到了该消息,进行处理.然后再次把它 的值设为-1.如果遇到的值为0,则视为结束信号,取消该队列,并退出SERVER.SERVER每接 收到一次数据后显示”(server)received”.

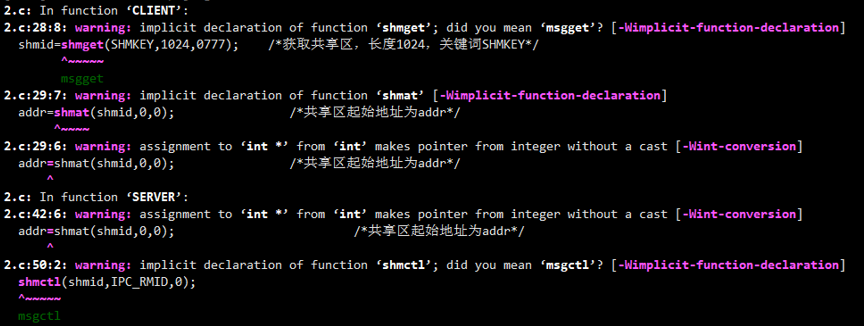
(3)CLIENT端建立一个为75的共享区,当共享取得第一个字节为-1时, Server端空闲,可发送 请求. CLIENT 随即填入9到0.期间等待Server端再次空闲.进行完这些操作后, CLIENT 退出. CLIENT每发送一次数据后显示”(client)sent”.

(4)父进程在SERVER和CLIENT均退出后结束.

3. 程序实现



4. 测试



程序运行段错误

总结

管道是一种半双工的通信方式，数据只能单向流动，上游进程往管道中写入数据，下游进程从管道中接收数据。如果想实现双方通信，那么需要建立两个管道。管道适合于传输大量信息。管道发送的内容是以字节为单位的，没有格式的字节流。

信号量在底层的实现是通过硬件提供的原子指令。共享内存允许两个或多个进程共享同一段物理内存。不同进程可以将同一段共享内存映射到自己的地址空间，然后像访问正常内存一样访问它。不同进程可以通过向共享内存端读写数据来交换信息。

实验4　存储管理

学号： 201970504109 姓名： 边明豪 班级： 计201-1

时间： 2022-5-12 地点： 线上 指导老师：\_\_翟一鸣\_\_\_\_\_

一、实验目的

本实验的目的是通过请求页式存储管理中页面置换算法模拟设计，了解虚拟存储技术的技术特点，掌握请求页式存储管理的页面置换算法。

二、实验内容1

1. 问题描述

存储管理的主要功能之一是合理地分配空间。请求页式管理是一种常用的虚拟存储管理技术。

本实验的目的是通过请求页式存储管理中页面置换算法模拟设计，了解虚拟存储技术的技术特点，掌握请求页式存储管理的页面置换算法。

2. 数据结构及算法设计

（1）通过随机数产生一个指令序列，共320条指令。指令的地址按下述原则生成：

①50%的指令是顺序执行的；

②50%的指令是均匀分布在前地址部分；

③50%的指令是均匀分布在后地址部分。

具体的实施方法是：

①在 [0，319] 的指令之间随即选取一起点m;

②顺序执行一条指令，即执行地址为m+1的指令；

③在前地址[0，m+1]中随机选取一条指令并执行，该指令的地址为m′；

④顺序执行一条指令，其地址为 m′+ 1；

⑤在后地址[m′+ 2，319]中随机选取一条指令并执行；

⑥重复上述步骤①-⑤，直到执行320次指令。

（2）将指令序列变换为页地址流

设：①页面大小为1k；

②用户内存容量为4页到32页；

③用户虚存容量为32k。

在用户虚存中，按每k存放10条指令排在虚存地址，即320条指令在虚存中的存放方式为：

第0条-第9条指令为第0页（对应虚存地址为[0，9]）；

第10条-第19条指令为第一页（对应虚存地址为[10，19]）；

… …

第310条~第319条指令为第31页（对应虚地址为[310，319]）。

按以上方式，用户指令可组成32页。

（3）计算并输出下述各种算法在不同内存容量下的命中率。

①先进先出的算法（FIFO）；

②最近最少使用算法（LRR)；

③最佳淘汰算法（OPT）先淘汰最不常用的页地址；

④最少访问页面算法（LFR）；

⑤最近最不经常使用算法（NUR）。

其中③和④为选择内容。

命中率=1-页面失效次数/页地址流长度

在本实验中，页地址流长度为320，页面失效次数为每次访问相应指令时，该指令所对应的页不在内存的次数。

3、随机数产生办法，Linux或UNIX系统提供函数strand()和rand(),分别进行初始化和产生随机数。例如：

strand ();

语句可初始化一个随机数；

a[0]=10\*rand()/65535\*319+1;

a[1]=10\*rand()/65535\*a[0];

语句可用来产生a[0]与a[1]中的随机数。

3. 程序实现

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define TRUE 1

#define FALSE 0

#define INVALID -1

#define NUL 0

#define total\_instruction 320 /\*指令流长\*/

#define total\_vp 32 /\*虚页长\*/

#define clear\_period 50 /\*清零周期\*/

typedef struct{ /\*页面结构\*/

int pn,pfn,counter,time;

}pl\_type;

pl\_type pl[total\_vp]; /\*页面结构数组\*/

struct pfc\_struct{ /\*页面控制结构\*/

int pn,pfn;

struct pfc\_struct \*next;

};

typedef struct pfc\_struct pfc\_type;

pfc\_type pfc[total\_vp],\*freepf\_head,\*busypf\_head,\*busypf\_tail;

int diseffect,a[total\_instruction];

int page[total\_instruction], offset[total\_instruction];

void initialize();

void FIFO();

void LRU();

void OPT();

void LFU();

void NUR();

int main()

{

int S,i;

srand((int)getpid());

S=(int)rand()%390;

for(i=0;i<total\_instruction;i+=1) /\*产生指令队列\*/

{

a[i]=S; /\*任选一指令访问点\*/

a[i+1]=a[i]+1; /\*顺序执行一条指令\*/

a[i+2]=(int)rand()%390; /\*执行前地址指令m’\*/

a[i+3]=a[i+2]+1; /\*执行后地址指令\*/

S=(int)rand()%390;

}

for(i=0;i<total\_instruction;i++) /\*将指令序列变换成页地址流\*/

{

page[i]=a[i]/10;

offset[i]=a[i]%10;

}

for(i=4;i<=32;i++) /\*用户内存工作区从4个页面到32个页面\*/

{

printf("%2d page frames",i);

FIFO(i);

LRU(i);

OPT(i);

LFU(i);

NUR(i);

printf("\n");

}

return 0;

}

void FIFO(total\_pf) /\*FIFO(First in First out)ALGORITHM\*/

int total\_pf; /\*用户进程的内存页面数\*/

{

int i;

pfc\_type \*p, \*t;

initialize(total\_pf); /\*初始化相关页面控制用数据结构\*/

busypf\_head=busypf\_tail=NUL; /\*忙页面队列头，对列尾链接\*/

for(i=0;i<total\_instruction;i++)

{

if(pl[page[i]].pfn==INVALID) /\*页面失效\*/

{

diseffect+=1; /\*失效次数\*/

if(freepf\_head==NUL) /\*无空闲页面\*/

{

p=busypf\_head->next;

pl[busypf\_head->pn].pfn=INVALID; /\*释放忙页面队列中的第一个页面\*/

freepf\_head=busypf\_head;

freepf\_head->next=NUL;

busypf\_head=p;

}

p=freepf\_head->next; /\*按方式调新页面入内存页面\*/

freepf\_head->next=NUL;

freepf\_head->pn=page[i];

pl[page[i]].pfn=freepf\_head->pfn;

if(busypf\_tail==NUL)

busypf\_head=busypf\_tail=freepf\_head;

else

{

busypf\_tail->next=freepf\_head;

busypf\_tail=freepf\_head;

}

freepf\_head=p;

}

}

printf("FIFO:%6.4F",1-(float)diseffect/320);

}

void LRU(total\_pf)

int total\_pf;

{

int min,minj,i,j,present\_time;

initialize(total\_pf);present\_time=0;

for(i=0;i<total\_instruction;i++)

{

if(pl[page[i]].pfn==INVALID) /\*页面失效\*/

{

diseffect++;

if(freepf\_head==NUL) /\*无空闲页面\*/

{

min=32767;

for(j=0;j<total\_vp;j++)

if(min>pl[j].time&&pl[j].pfn!=INVALID)

{

min=pl[j].time;

minj=j;

}

freepf\_head=&pfc[pl[minj].pfn];

pl[minj].pfn=INVALID;

pl[minj].time=-1;

freepf\_head->next=NUL;

}

pl[page[i]].pfn=freepf\_head->pfn;

pl[page[i]].time=present\_time;

freepf\_head=freepf\_head->next;

}

else

pl[page[i]].time=present\_time;

present\_time++;

}

printf("LRU:%6.4f",1-(float)diseffect/320);

}

void NUR(total\_pf)

int total\_pf;

{

int i,j,dp,cont\_flag,old\_dp;

pfc\_type \*t;

initialize(total\_pf);

dp=0;

for(i=0;i<total\_instruction;i++)

{

if(pl[page[i]].pfn==INVALID) /\*页面失效\*/

{

diseffect++;

if(freepf\_head==NUL) /\*无空闲页面\*/

{

cont\_flag=TRUE;old\_dp=dp;

while(cont\_flag)

if(pl[dp].counter==0&&pl[dp].pfn!=INVALID)

cont\_flag=FALSE;

else

{

dp++;

if(dp==total\_vp)

dp=0;

if(dp==old\_dp)

for(j=0;j<total\_vp;j++)

pl[j].counter=0;

}

freepf\_head=&pfc[pl[dp].pfn];

pl[dp].pfn=INVALID;

freepf\_head->next=NUL;

}

pl[page[i]].pfn=freepf\_head->pfn;

freepf\_head=freepf\_head->next;

}

else

pl[page[i]].counter=1;

if(i%clear\_period==0)

for(j=0;j<total\_vp;j++)

pl[j].counter=0;

}

printf("NUR:%6.4f",1-(float)diseffect/320);

}

void OPT(total\_pf) /\*OPT(Optimal Replacement)ALGORITHM\*/

int total\_pf;

{

int i,j,max,maxpage,d,dist[total\_vp];

pfc\_type \*t;

initialize(total\_pf);

for(i=0;i<total\_instruction;i++)

{

if(pl[page[i]].pfn==INVALID)

{

diseffect++;

if(freepf\_head==NUL)

{

for(j=0;j<total\_vp;j++)

if(pl[j].pfn!=INVALID)

dist[j]=32767;

else

dist[j]=0;

d=1;

for(j=i+1;j<total\_instruction;j++)

{

if(pl[page[j]].pfn!=INVALID)

dist[page[j]]=d;

d++;

}

max=-1;

for(j=0;j<total\_vp;j++)

if(max<dist[j])

{

max=dist[j];maxpage=j;}

freepf\_head=&pfc[pl[maxpage].pfn];

freepf\_head->next=NUL;

pl[maxpage].pfn=INVALID;

}

pl[page[i]].pfn=freepf\_head->pfn;

freepf\_head=freepf\_head->next;

}

}

printf("OPT:%6.4f",1-(float)diseffect/320);

}

void LFU(total\_pf) /\*LFU(leat Frequently Used) ALGORITHM\*/

int total\_pf;

{

int i,j,min,minpage;

pfc\_type \*t;

initialize(total\_pf);

for(i=0;i<total\_instruction;i++)

{

if (pl[page[i]].pfn==INVALID)

{

diseffect++;

if(freepf\_head==NUL)

{

min=32767;

for(j=0;j<total\_vp;j++)

{

if(min>pl[j].counter&&pl[j].pfn!=INVALID)

{

min=pl[j].counter; minpage=j;

}

pl[j].counter=0;

}

freepf\_head=&pfc[pl[minpage].pfn];

pl[minpage].pfn=INVALID;

freepf\_head->next=NUL;

}

pl[page[i]].pfn=freepf\_head->pfn;

freepf\_head=freepf\_head->next;

}

else

pl[page[i]].counter++;

}

printf("LFU:%6.4f",1-(float)diseffect/320);

}

void initialize(total\_pf) /\*初始化相关数据结构\*/

int total\_pf; /\*用户进程的内存页面数\*/

{

int i;

diseffect=0;

for(i=0;i<total\_vp;i++)

{

pl[i].pn=i;pl[i].pfn=INVALID; /\*置页面控制结构中的页号，页面为空\*/

pl[i].counter=0;pl[i].time=-1; /\*页面控制结构中的访问次数为0，时间为-1\*/

}

for(i=1;i<total\_pf;i++)

{

pfc[i-1].next=&pfc[i];pfc[i-1].pfn=i-1;/\*建立pfc[i-1]和pfc[i]之间的连接\*/

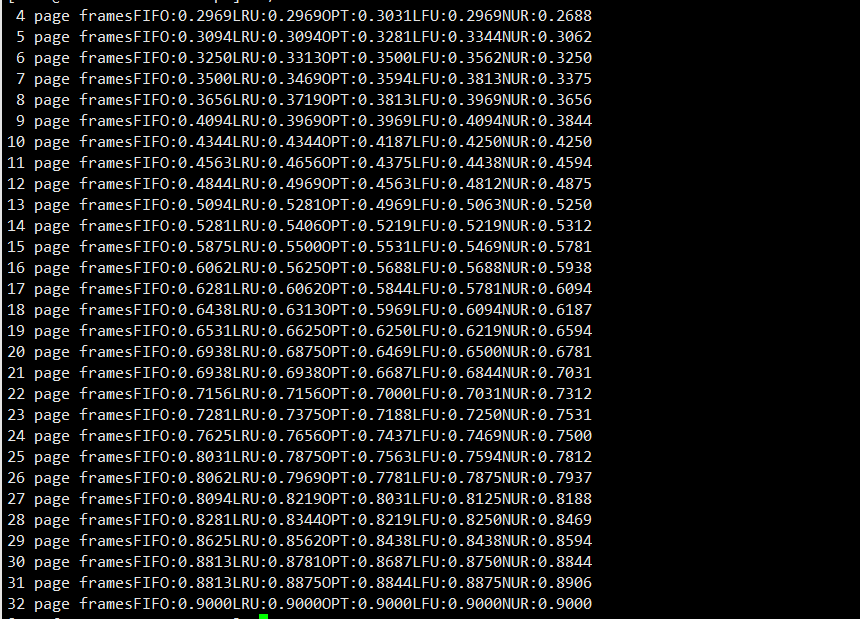
}

pfc[total\_pf-1].next=NUL;pfc[total\_pf-1].pfn=total\_pf-1;

freepf\_head=&pfc[0]; /\*页面队列的头指针为pfc[0]\*/

}

4. 测试



总结

（在总结部分需要全面梳理在知识、技能方面的收获、感受，可以提出个人对知识的独特、个性的理解。还可以自己提出对未来学习的改进意见。）

这次实验让我更加熟悉了OPT， FIFO，LRU，NUR算法，让我对这些页面置换算法的实际效率有了更加深刻的印象。同时自己的编程能力也得到了一定的提高，对调度算法的理解也更加深刻了。

FIFO算法有一些缺陷，因为有时候随着物理块个数的增加，其触发缺页中断的次数不降反升，这是由于FIFO置换的特点而决定的，这一现象叫做belady现象。产生的原因是较早调入的页往往是经常被访问的页，这些页在FIFO算法下被反复调入和调出，而FIFO是不看物理块的使用情况，而只关注第一次的调用时间。这就导致了FIFO是盲目的进出。所以产生了belady现象。

实验5　文件系统设计

学号： 201970504109 姓名： 边明豪 班级： 计201-1

时间： 2022-5-24 地点： 线上 指导老师：\_\_翟一鸣\_\_\_\_\_

一、实验目的

通过一个简单多用户文件系统的设计，加深理解文件系统的内部功能及内部实现。

二、实验内容

1. 问题描述

为linux系统设计一个简单的二级文件系统。要求做到以下几点：

（1）可以实现下列几条命令（至少4条）；

login 用户登陆

dir 列文件目录

create 创建文件

delete 删除文件

open 打开文件

close 关闭文件

read 读文件

write 写文件

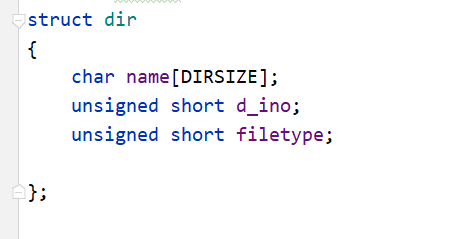
（2）列目录时要列出文件名、物理地址、保护码和文件长度；

（3）源文件可以进行读写保护。

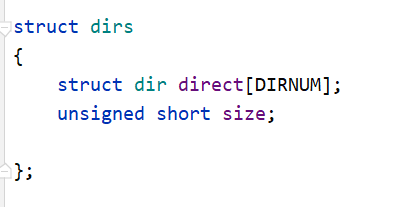
2. 数据结构及算法设计

目录结构体：

Name为目录名，d\_ino为目录id，filetype为文件类型



父目录结构体：包含一个目录结构体数组，和该目录的大小。



Inode结构体：i\_flag代表文件标识。

I\_ino代表文件的标识码

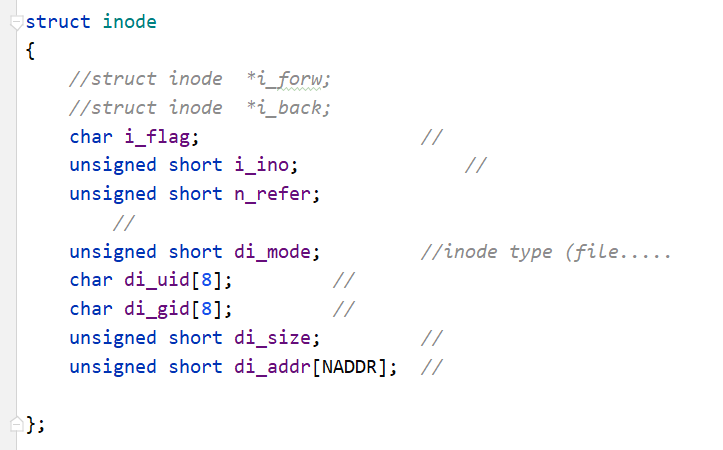
Di\_mode代表文件的rwx权限

Di\_uid代表所属用户

Di\_gid代表所属用户组。

Di\_size代表文件大小

Di\_addr代表文件的物理地址



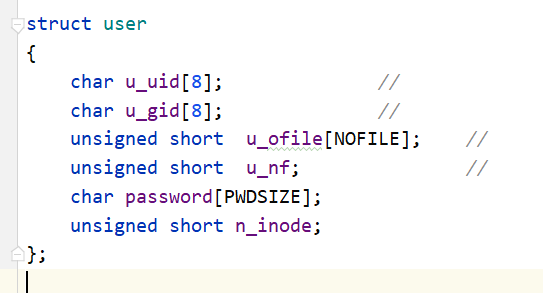
用户结构体：

U\_uid代表用户的用户名

U\_gid代表用户所在的用户组名

U\_nf代表用户的文件数目

Password代表用户的密码



Filesys结构体

S\_isize代表文件系统的总大小。

S\_fsize代表文件系统的闲置大小

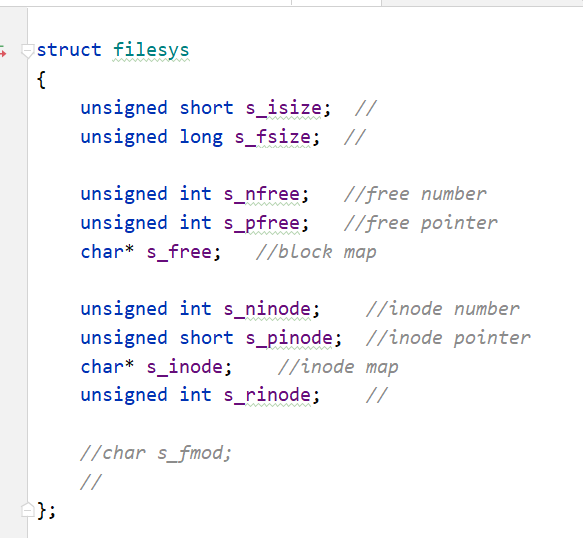
S\_nfree代表闲置的内存块数

S\_pfree代表闲置的指针数

S\_ninode代表iNode的个数

S\_pinode代表inode指针

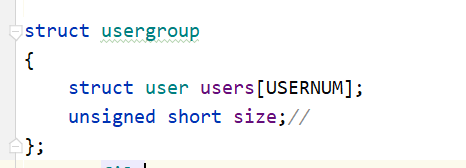
S\_free代表物理块闲置分布



用户组结构体：

含有一个用户结构体数组

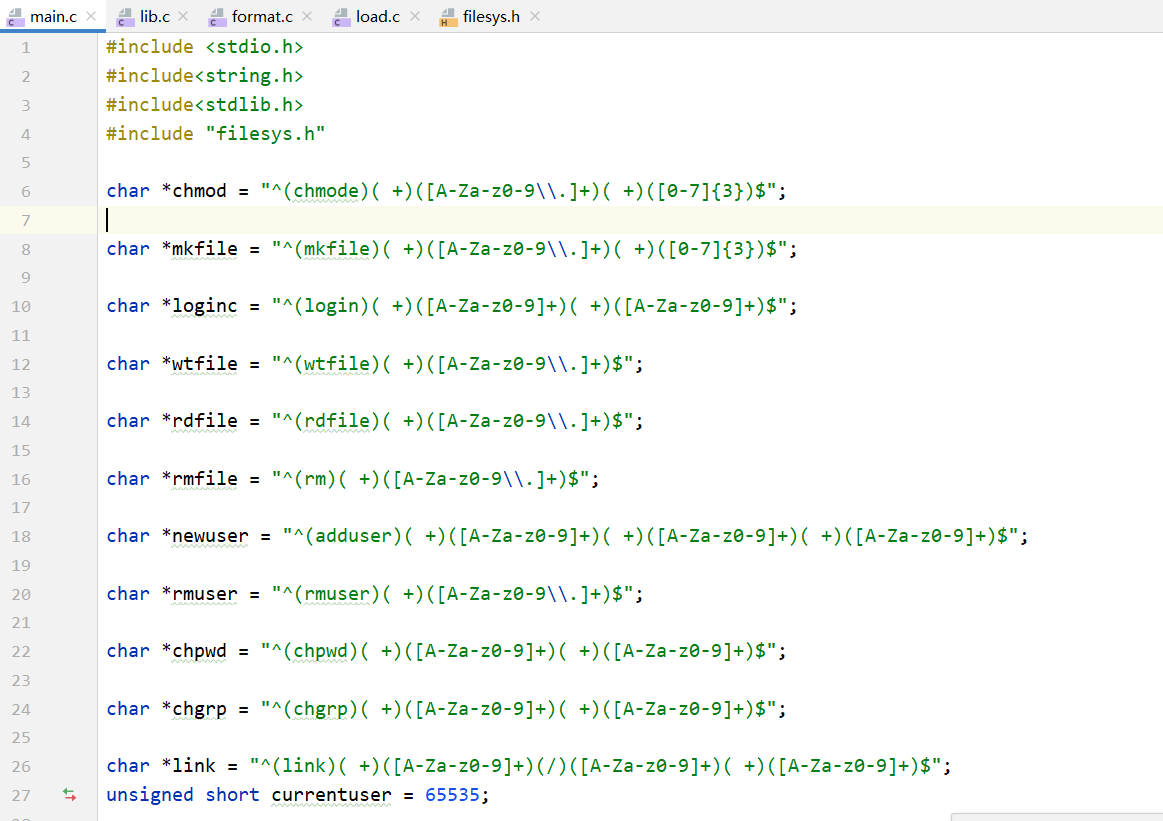
Size代表该用户组的用户个数



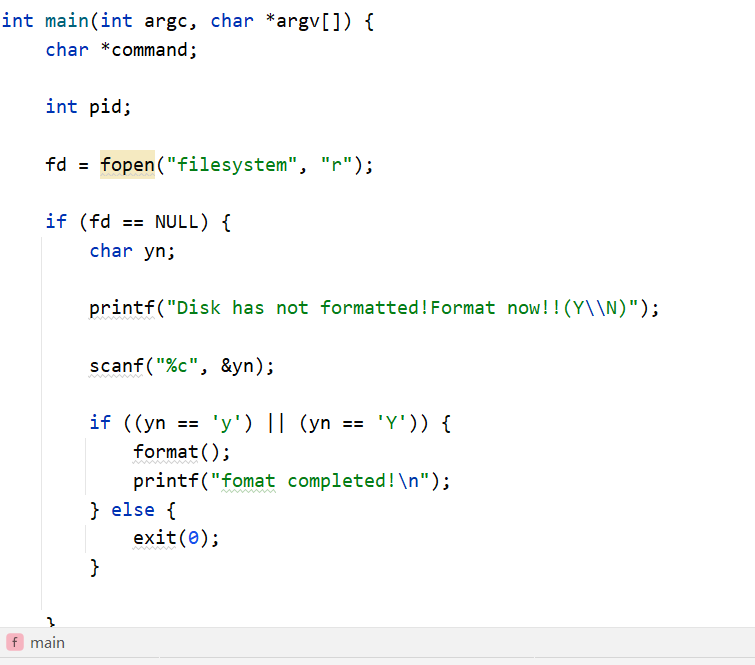
3. 程序实现

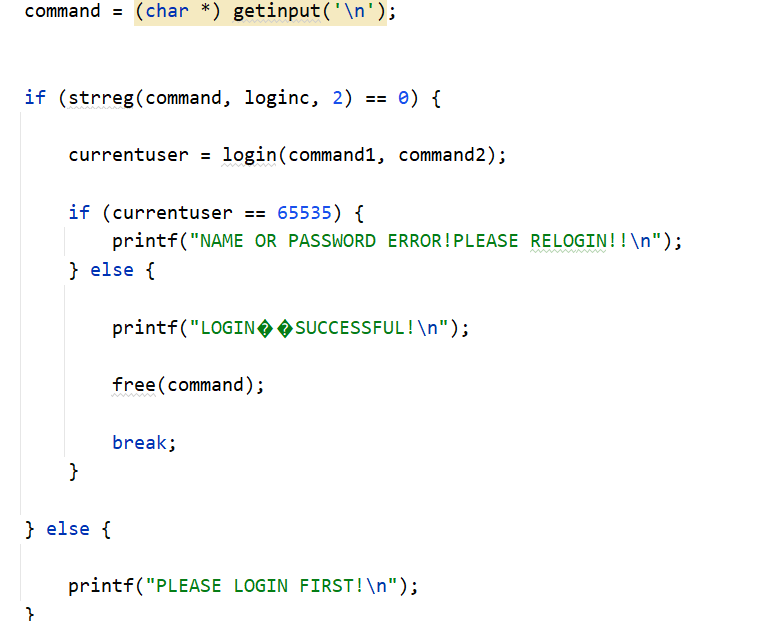
解读main.c文件：

变量名分别代表了其对应的指令，其字符串是该指令的正则匹配表达式。

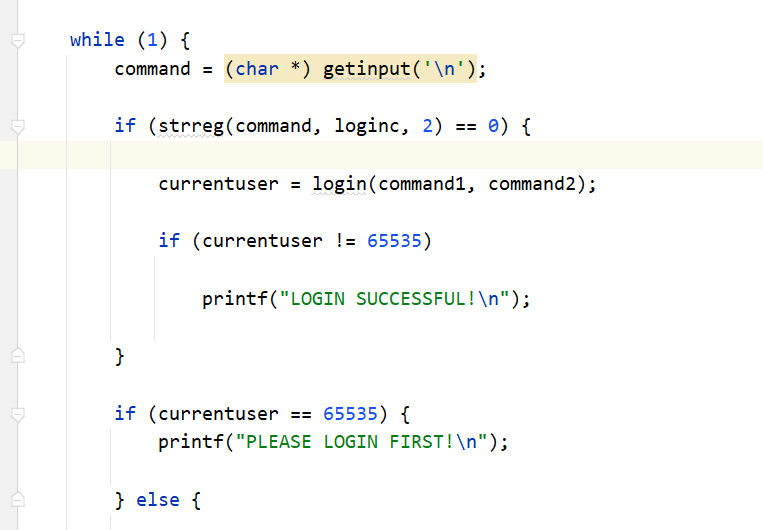


打开系统虚拟磁盘

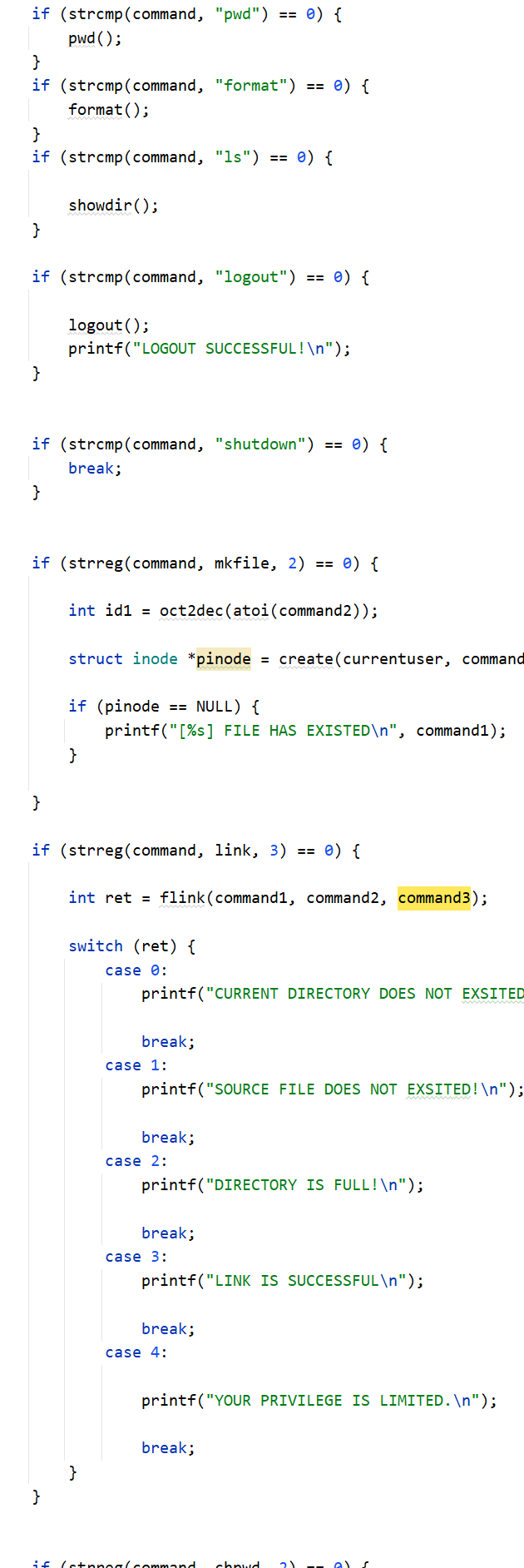
  
等待用户输入用户名密码:



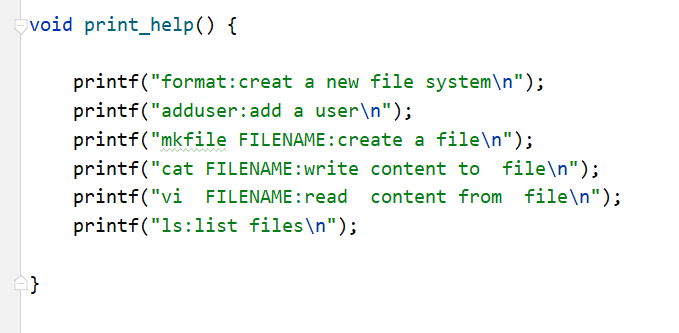
用户登陆成功：



等待用户输入指令，并执行相应的子程序：



查看帮助信息：



总结

尝试了好几种方法，各种调试，补库文件，但是很不幸最终还是没有成功运行出来这个文件系统。但是本人仔细研读了大部分代码，对该文件系统的架构有了一定的了解，理解了其中的代码结构和底层实现。

该程序对虚拟文件系统的实现相对完整，支持多用户，多文件，多权限等等，也支持很多命令。程序把“filesystem”作为整个程序的虚拟磁盘系统，并对其分块分页，所有的用户和文件都存放在对应的物理块中，非常有条理。而且该程序对用户指令的解析中使用了正则表达式，这使得处理用户的非法输入变得非常容易且优雅，这一点是值得学习的。

附：实验教学自评

实验1、2的评价量表

| **考核依据** | **分值** | **评价标准** | | | **打分** | **评论** |
| --- | --- | --- | --- | --- | --- | --- |
| **优良** | **合格** | **待改进** |
| 实验报告 | 10 | （9~10）报告结构合理，体现了规范的实验过程；各部分描述具体、清晰，图表规范合理，专业术语使用得当；版面边距、行距等方面处理恰当，整体可读性好。 | （6~8分）报告结构合理，体现了基本规范的实验过程；运用专业的图表、专业术语；版面整体可读性好。 | （1~5分）报告结构合理，体现了基本规范的实验过程，表达基本清楚。 | 8 | 报告结构合理，体现了实验过程，实验用语规范 |
| 程序验收 | 15 | （10~15分）程序完整实现了要求的内容；程序书写规范，一句一行，缩格排放，注释适当；在抽查验收中，准确回答提出的疑问。 | （5~10分）程序完整实现了要求的内容；程序书写基本规范，有适量的注释；在抽查验收中，基本准确回答提出的疑问。 | （1-5分）按时完成实验要求的主要任务；在抽查验收中，能回应提出的疑问。 | 14 | 程序完整实现了要求，有适当的注释 |

实验3的评价量表

| **考核依据** | **分值** | **评价标准** | | | **打分** | **评论** |
| --- | --- | --- | --- | --- | --- | --- |
| **优良** | **合格** | **待改进** |
| 实验报告 | 10 | （9~10）报告结构合理，体现了规范的实验过程；各部分描述具体、清晰，图表规范合理，专业术语使用得当；版面边距、行距等方面处理恰当，整体可读性好。 | （6~8分）报告结构合理，体现了基本规范的实验过程；运用专业的图表、专业术语；版面整体可读性好。 | （1~5分）报告结构合理，体现了基本规范的实验过程，表达基本清楚。 | 9 | 报告结构合理，体现了实验过程，结构合理 |
| 程序验收 | 15 | （11~15分）程序完整实现了要求的内容；程序书写规范，一句一行，缩格排放，注释适当；在抽查验收中，准确回答提出的疑问。 | （5~10分）程序完整实现了要求的内容；程序书写基本规范，有适量的注释；在抽查验收中，基本准确回答提出的疑问。 | （1-5分）按时完成实验要求的主要任务；在抽查验收中，能回应提出的疑问。 | 14 | 程序完整实现了要求，有适当的注释 |

实验4、5的评价量表

| **考核依据** | **分值** | **评价标准** | | | **打分** | **评论** |
| --- | --- | --- | --- | --- | --- | --- |
| **优良** | **合格** | **待改进** |
| 实验报告 | 20 | （16~20分）针对问题选用合理的数据逻辑结构和存储结构，合理选用或设计算法并清晰表达；运用适当的程序实现技术；实验总结深入、具体。 | （8~15分）使用的数据逻辑结构、存储结构、算法基本合理，并对选择的进行了讨论；程序实现采用技术基本适当；客观总结实验。 | （1~8分）有对数据结构、算法选择依据的讨论，对实验收获做了总结。 | 18 | 选用合理的数据逻辑结构和存储结构 |
| 程序验收 | 30 | （26~30分）程序完整实现了要求的内容；程序书写规范，一句一行，缩格排放，注释适当；在抽查验收中，准确回答提出的疑问。 | （15~25分）程序完整实现了要求的内容；程序书写基本规范，有适量的注释；在抽查验收中，基本准确回答提出的疑问。 | （1~15分）按时完成实验要求的主要任务；在抽查验收中，能回应提出的疑问。 | 28 | 程序完整实现了要求，有适当的注释 |

注：(1)在上面的评价标准中，低分标尺中提供但高分标尺中未提及的，打高分必须满足低分标准；（2）实验过程中提倡同学们相互帮助，但每位同学要为个人的实验成果和报告负责，凡发现抄袭者，本次实验成绩以零分计。