

**本科生实验报告**

**实验课程 操作系统与安全**

**学院名称 信息科学与技术学院**

**专业名称 网络空间安全**

**学生姓名 彭重严**

**学生学号 201817030115**

**指导教师 杨国城**

**实验地点 6A501**

**2020年11 月**

目录

[实验项目一：Linux编程基础实验 2](#_Toc61815726)

[实验项目二：进程创建与控制 12](#_Toc61815727)

[实验项目三：银行家算法实现 22](#_Toc61815728)

[实验项目四：磁盘调度实验 29](#_Toc61815729)

# 

# 实验项目一：Linux编程基础实验

一、实现目的

1）了解Linux编程环境和编程工具

2）掌握基本的Linux系统命令及执行过程

3）了解shell的作用及主要分类

4）掌握shell脚本程序运行原理及基础语法，学会便携简单的shell脚本程序

二、实验准备

1）安装虚拟机及Ubuntu操作系统

2）熟悉Linux下文本编辑工具

3）初步了解GCC和GDB的概念

三、实验基本知识及原理

**1.shell**

shell是Linux系统中重要的一层，是包裹在Linux内核上的“壳程序”，是用户和Linux内核之间的接口程序，用户在shell提示符（$或#）下输入的每一个命令都由shell先解释，然后传给内核执行（#表示该系统的root用户）。常见的shell有两类：一类是bourne shell，如sh、ksh、bash（bourne again shell）等；另一类是C shell，如csh、tcsh等。

在Ubuntu系统下，可以通过组合键Ctrl+Alt+T来启动bash，然后直接通过命令来执行相应的操作。

**2.Linux常用命令**

man：改命令提供一个在线帮助文档，用于查找Linux命令的使用方法，例如查看ls命令的使用方法时，可以在bash下输入man ls，同样，查看man命令更详细的使用方法可以通过输入man man来进行操作。

ls：列举当前文件目录下的所有内容，可以通过参数设置来选择要显示的内容。

grep：该命令提供一个搜索功能，打印包含参数中指定正则表达式字符串的一整行内容。

cat：该命令将文件连接起来并用标准输出方式进行打印。

more：该命令提供了翻页的文件查看方式，如果一个文件很大，使用cat来擦好看很不方便，而more命令则弥补了这方面的缺陷。

cd：切入特定的目录，在Linux下，当前目录可以用“.”来表示，上一目录可以用“..”来表示。

cp：拷贝文件，该命令在拷贝目录的时候，需要添加参数-r，表示递归的对目录下的内容进行操作。

mv：移动目录或文件。

rm：删除文件，如需删除目录，同样需要添加-r参数来完成。

which：用来查询一个命令的位置。

sudo：通过在命令前使用sudo，可以使普通用户获得root的权限，从而执行一些普通用户无权执行的操作。

**3.管道**

Linux下可以使用管道将不同命令的输入和输出连接起来，管道即为“|”，出现在其左边的命令执行的输出将作为其右边命令的输入。

**4，Linux系统的执行过程**

以ls为例，在shell命令行输入ls命令，键盘驱动程序识别出输入的内容，将它们传递给shell，由外壳程序来负责查找同名的可执行程序（ls），如果在/bin/ls目录中找到了ls，则调用核心服务将ls的可执行映像读入虚拟内存并开始执行。ls调用核心的文件子系统来寻找那些文件是可用的。文件系统使用缓冲过的文件系统信息，或者调用磁盘设备驱动从磁盘上读取信息。ls命令还可能引起网络驱动程序和远程机器交换信息锁定系统要访问的远程文件系统信息（文件系统可以通过网络文件系统或者NFS进行远程安装）。当得到这些信息后，ls命令通过调用视频驱动将这些信息写到显示器屏幕上。

**5.Linux下文件的编辑工具**

vim：文本编辑器的一种，它在vi的基础上增加了许多新的特性。vim同Emacs都是Linux文本编辑的常用工具软件，与Windows下文本编辑器（如notepad）相比，vim下整个文本编辑器都是由键盘命令完成而非鼠标完成，这就实现了在没有图形化界面的情况下对文件内容的编辑操作。

gredit：在很多Linux图形界面下都嵌有一个图形化的文本编辑工具，可通过gredit来使用这个工具，gredit工具与Windows下的文本编辑器使用方法很类似。

**6.GCC与GDB**

GCC（GNU Compiler Collection）是GNU编译器集合，是一套由GNU开发的编程语言编译器。它是一套以GPL及LGPL许可证所发行的自由软件，是GNU计划的关键部分，也是类UNIX以及苹果电脑Mac OS X操作系统的标准编译器。能处理的语言有：C、C++、Fortran、Pascal、Java等。

假设有一个写好的main.c文件，可以通过命令gcc main.c来对其进行编译链接等工作，默认生成名为a.out的可执行文件，使用./a.out来执行a.out文件。

如果想指定输出的可执行文件的文件名，可通过-o参数来设定输出文件的文件名，如gcc main.c -o test，则会生成test的可执行文件。

GDB（GNU Debugger）是GNU开源组织发布的UNIX下的程序调试工具。GDB是一个命令强大的命令行调试工具，相比Windows下的VC、VS的界面调试，GDB的优势在于命令行可以形成执行序列，形成脚本。

四、实验说明

1）本次实验是操作系统课程设计的入门实验，整体难度适中

2）本实验中的代码若无特殊说明均为GCC编译器编译

3）关于GDB调试和Linux下图形界面编程调试请参见第一部分第6章

五、实验内容

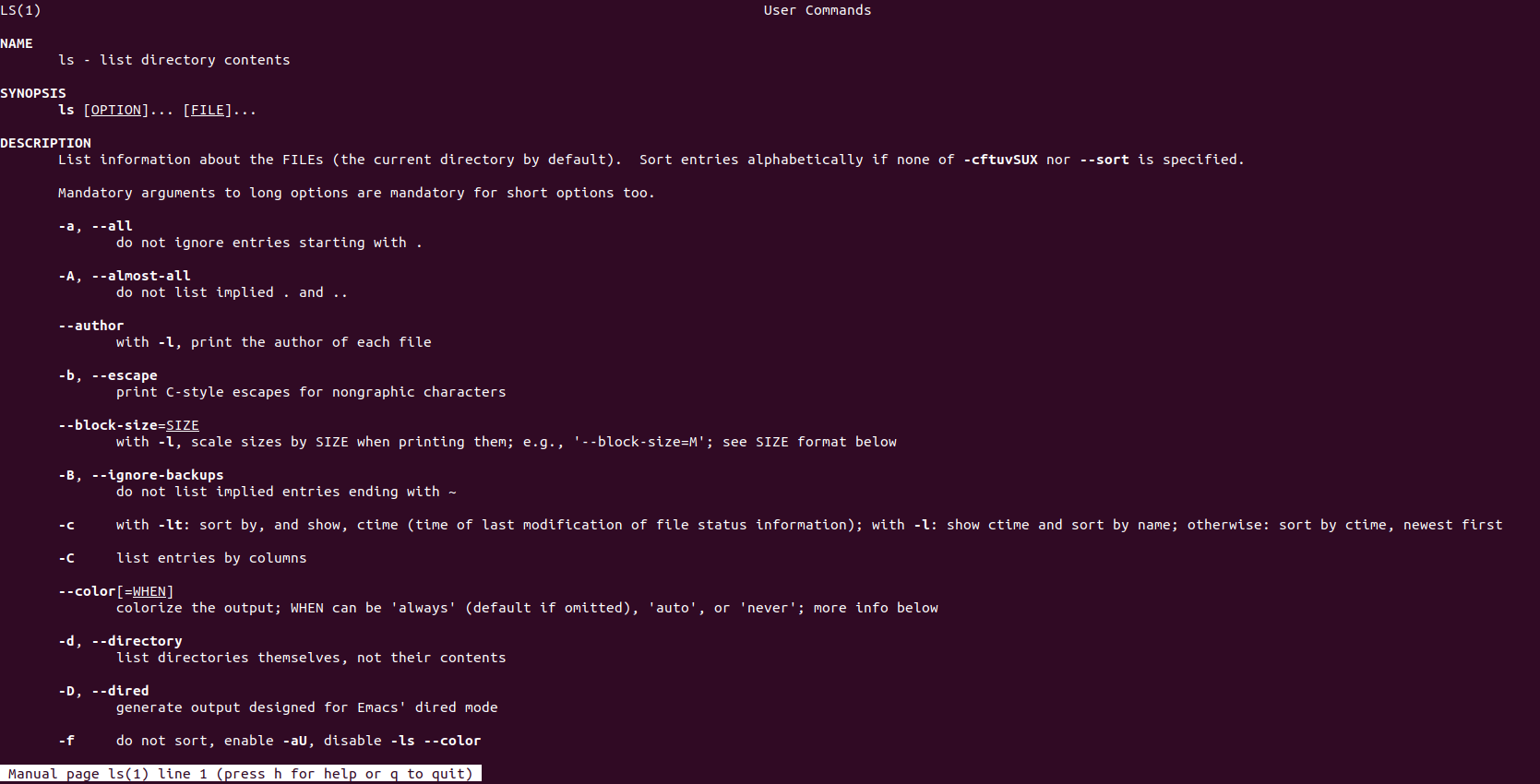
**实验一 Linux命令实验**

在Ubuntu下启动shell（按组合键Ctrl+Alt+T），使用man命令查看上述每个命令的具体使用方法及参数的作用。操作上面提到的命令，体会其作用，并按照实验报告要求完成实验报告中的相应内容。

重点学习的命令有：man、ls -al、cat、more、grep、which、who、rm、mv

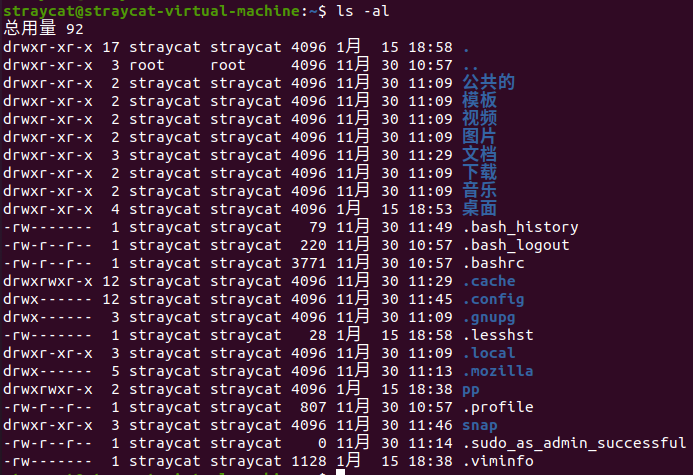
①man：

会以文本编辑器的形式显示出帮助内容



②ls -al

显示当前目录下所有内容



③cat

将文件内容打印到屏幕



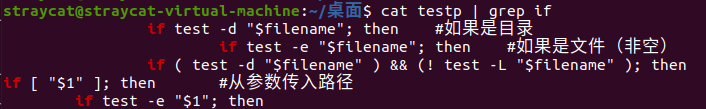
④more

分页显示



⑤grep

搜索指定文本



⑥which

查询命令所在位置



⑦who

查询当前用户身份



⑧rm

删除文件/文件夹



⑨mv

移动文件位置



**实验二 文本编辑工具、GCC以及GDB的使用**

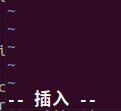
1）使用vim编辑C源文件

vim是vi的加强版，它共有三种形式：命令模式、插入模式、底行模式。通常将底行模式算入命令模式中。

🡪在shell终端中输入“vim 文件名”，若存在该文件名的文件，则会打开该文件，若无该文件，则会以该文件名创建一个文件。这时，进入vim模式为命令模式。



🡪按下1键可以进入插入模式，这是可以对文件进行插入操作。



🡪原文件写完以后，按下Esc键，返回到命令模式，使用“:wq”命令进行保存（若不保存直接退出可在命令模式下使用“:q!”执行）。



2）使用GCC编译

使用GCC对其进行编译，GCC是默认没有使用C99特性的，可以根据提示添加参数“-std=c99”。



3）使用GDB调试

使用GDB对程序进行调试，GDB的使用请参考第一部分第6章的内容。



4）完成实验报告相应内容

**实验三 shell脚本程序设计（小组实验）**

1.shell变量

shell变量包括环境变量和临时变量。其中shell脚本中的临时变量又分为用户定义的变量和位置参数两类。

用户定义的变量是最普遍的shell变量，变量名是以字母或下划线开头的字母、数字和下划线符序列，并且大小写字母意义不同，变量名的长度不受限制。定义变量并赋值的一般形式是：变量名=字符串，如MYFILE=/root/m1.c。

在程序中使用变量的值时，要在变量名前加上一个“$”符号，这个符号告诉shell要读取该变量的值。作为交互式输入手段，可以利用read命令由标准输入（即键盘）上读取数据，然后赋值给指定的变量，其一般格式是read 变量1 [变量2 …]。利用read命令交互地为变量赋值输入数据时，数据间以空格或者制表符作为分隔符，但是要注意以下情况：

1）若变量个数与给定数据个数相同，则依次给每个变量赋值

2）若变量个数少于数据个数，除最后一个变量外，每个变量依次赋值，剩下的数据全部赋予最后一个变量。

3）若变量个数多于数据个数，后面的变量没有输入数据与之对应时，后面的变量赋空值。

执行shell脚本时可以使用参数。将出现在命令行上位置确定的参数称作位置参数。在sh中总共有10个位置参数，其对应的名称依次是$0，$1，$2，…，$9。其中$0始终表示命令名或shell脚本名，对于一个命令行，必然有命令名，也就必定有$0，而其他位置参数依据实际需要，可有可无。

2.测试语句

测试语句有两种常用形式，一种是test命令（用来检测某个条件是否成立），另一种是用一对方括号将测试条件括起来。两种形式完全等价。例如，测试位置参数$1是否是已存在的普通文件，可写成test -f “$1”（-f表示某个文件是否存在，并且是否为普通文件），也可写成[-f $1]。

在格式上应该注意，如果在test语句中使用shell变量，为表示完整、避免造成歧义，最好用双引号将变量括起来。利用一对方括号表示测试条件时，在左方括号“[”之后、右方括号“]”之前应各有一个空格。

（1）if语句

if [condition]; then

… #if条件成立时

elif [condition]

… elif条件成立时

else

… #上面情况都不满足时

fi

注意：if之后需要有一个then，还需有一个fi表示if段结束，中国括号两边都需要加空格以分隔判断语句。

（2）循环语句

🡪for循环，条件中的每种情况都执行一次

for [var] in [con1,,con2,con3]; do

...

done

🡪while循环，当条件满足时一直执行下去

while [condition]; do

...

done

🡪until循环，与while相反，当条件满足时停止，否则一直执行

until [condition]; do

...

done

3.shell的函数功能

functionname()

{

command

…

command;

}

六、实验总结

实验还是很简单的，之前我也接触过一些Linux的命令，使用起来还是比较顺畅，但是我对shell编程不太了解，可能看起来还是有一些生晦。

通过本次实验，我能够更加熟悉Linux的命令行操作，了解到了基本的shell脚本语法，也了解到了编译型语言和解释性语言的区别。

七、程序代码

1.基本的Linux命令使用（见上文“五、实验内容”的截图）

2.小组任务：编写Linux bash脚本文件实现查看目录home中包含的文件数量和子目录数量，并以以下格式输出到文件file.ini中。

格式：

[文件夹1]

文件夹下文件（夹）1

文件夹下文件（夹）2

…

[文件夹2]

文件夹下文件（夹）1

文件夹下文件（夹）2

…

[Directories Count]

10

[File Count]

4

代码：

dir\_num=0 #目录数目记录变量

file\_nume=0 #文件数目记录变量

tree(){

echo '[' $(basename "$\*") ']' >> file.ini #输出父目录

for filename in "$\*"/\*; do #对于$\*目录下的每个文件

if test -d "$filename"; then #如果是目录

echo "$(basename "$filename")" >> file.ini

dir\_num=$( expr $dir\_num+1)

else

if test -e "$filename"; then #如果是文件（非空）

echo "$(basename "$filename")" >> file.ini

file\_num=$( expr $file\_num+1)

fi

fi

done

echo >> file.ini

for filename in "$\*"/\*; do #对于$\*目录下的每个文件夹进行递归

if ( test -d "$filename" ) && (! test -L "$filename" ); then

tree "$filename"

fi

done

}

rm -f file.ini #先删除目标文件再保存

if [ "$1" ]; then #从参数传入路径

if test -e "$1"; then

echo "Running......"

tree "$1"

else

echo "No such directory" "$1"

exit 1;

fi

else

echo "Running......"

tree "$(pwd)" #没有参数，以当前目录为目标

fi

echo '[Directories Count]' >> file.ini #输出目录个数

echo "$dir\_num" >> file.ini

echo >> file.ini

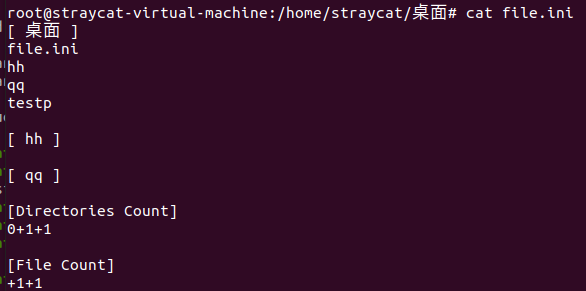
echo '[File Count]' >> file.ini #输出文件个数

echo "$file\_num" >> file.ini

echo "Success! The result has been saved in file.ini"

实验结果截图：





# 实验项目二：进程创建与控制

一、实现目的

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）分析进程争用资源的现象，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

**实验一 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。

**实验二 进程的控制**

修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话，再观察程序执行时屏幕上出现的现象，并分析原因。

如果在程序中使用系统调用lockf()来给每一个进程加锁，可以实现进程之间的互斥，观察并分析出现的现象。

**实验三 进程的软中断通信**

（1）编制一段程序，使其实现进程的软中断通信。

要求：使用系统调用fork()创建两个子进程，再用系统调用signal()让父进程捕捉键盘上来的中断信号（即按Del键）；当捕捉到中断信号后，父进程用系统调用kill()向两个子进程发出信号，子进程捕捉到信号后分别输出下列信息后终止：

child process1 is killed by parent!

child process2 is killed by parent!

父进程等待两个子进程终止后，输出如下的信息后终止：

parent process is killed!

（2）在上面的程序中增加语句signal(SIGINT, SIG\_IGN)和signal(SIGQUIT, SIG\_IGN)，观察执行结果，并分析原因。

**实验四 进程的管道通信**

编制一段程序，，实现进程的管道通信。

使用系统调用pipe()建立一条管道线；两个子进程P1和P2分别向管道各写一句话：

child1 is sending a message!

child2 is sending a message!

而父进程则从管道中读取来自两个子进程的信息，显示在屏幕上。

要求父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

三、实验基本知识及原理

进程（Process）：它是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体。

对于进程，我们需要进行状态转换、控制、通信等基本操作，这里就需要涉及到一系列的调度算法来实现。

进程的状态：总共三种基本状态。就绪态、运行态和阻塞态。但是实际上还有两种状态，创建态和终止态。

进程的控制：指的是对系统中的所有进程进行状态转换，实现对进程的有效管理。

进程通信：指的是进程之间的信息交换。进程通信有三种方式，共享存储、管道通信和消息传递。

四、实验说明

1）运行环境：Ubuntu 20.04.1

2）所需软件：VMware 15.5.6

五、程序代码

**实验一 进程的创建**

#include<stdio.h>

main(){

int p1,p2;

//创建子进程p1（因为是没有执行体的while循环，所以如果创建失败，会一直创建，直到成功为止）

while((p1=fork())==-1);

//这里用if选择结构是为了让p1和p2的创建先后随机

if(p1==0) //子进程p1创建成功

putchar('b');

else

{

while((p2=fork())==-1); //创建子进程p2

if(p2==0)

putchar('c'); //子进程p2创建成功

else

putchar('a'); //父进程执行

}

}

**实验二 进程的控制**

**【代码1】**

#include<stdio.h>

main()

{

int p1,p2,i;

while((p1=fork)==-1); //创建子进程1

if(p1==0)

for(i=0;i<50;i++)

printf("Child %d\n",i); //输出50句Child x

else

{

while((p2=forl())==-1); //创建子进程2

if(p2==0)

for(i=0;i<50;i++)

printf("Son %d\n",i);

else

for(i=0;i<50;i++)

printf("Daughter %d\n",i);

}

}

**【代码2】**

#include<stdio.h>

#include<unistd.h>

main()

{

int p1,p2,i;

while((p1=fork())==-1);

if(p1==0)

{

lockf(1,1,0); //给进程上锁，让该进程的运行不被中断

for(i=0;i<50;i++)

printf("Child %d\n",i);

lockf(1,0,0); //进程解锁

}

else

{

while((p2=fork())==-1);

if(p2==0)

{

lockf(1,1,0);

for(i=0;i<50;i++)

printf("Son %d\n",i);

lockf(1,0,0);

}

else

{

lockf(1,1,0);

for(i=0;i<50;i++)

printf("Daughter %d\n",i);

lockf(1,0,0);

}

}

}

**实验三 进程的软中断通信**

**【代码1】**

#include<unistd.h>

#include<stdio.h>

#include<signal.h>

void waiting(),stop();

int wait\_mark;

main()

{

int p1,p2;

while((p1=fork())==-1);

if(p1>0)

{

while((p2=fork())==-1);

if(p2>0)

{

printf("Parent\n");

wait\_mark=1;

signal(SIGINT,stop); //接收Del信号，并转stop

waiting(0);

kill(p1,16); //向p1发送中断信号16

kill(p2,17); //向p2发送中断信号17

wait(0); //同步

wait(0);

printf("Parent process is killed!\n");

exit(0);

}

else

{

printf("p2\n");

wait\_mark=1;

signal(17,stop);

waiting();

lockf(stdout,1,0);

printf("Child process 2 is killed by parent!\n");

lockf(stdout,0,0);

exit(0);

}

}

else

{

printf("p1\n");

wait\_mark=1;

signal(16,stop);

waiting();

lockf(stdout,1,0);

printf("Child process 1 is killed by parent!\n");

lockf(stdout,0,0);

exit(0);

}

}

void waiting()

{

while(wait\_mark!=0);

}

void stop()

{

wait\_mark=0;

}

**【代码2】**

#include<unistd.h>

#include<stdio.h>

#include<signal.h>

int pid1,pid2;

int EndFlag=0,pf1=0,pf2=0;

void IntDelete()

{

kill(pid1,16); //向pid1发送中断信号16

kill(pid2,17); //向pid2发送中断信号17

EndFlag=1;

}

void Int1()

{

printf("Child process 1 is killed by parent!");

exit(0);

}

void Int2()

{

printf("Child process 2 is killed by parent!");

exit(0);

}

main()

{

int exitpid;

signal(SIGINT,SIG\_IGN); //忽略Del信号

signal(SIGQUIT,SIG\_IGN); //忽略中断信号

while((pid1=fork())==-1);

if(pid1==0)

{

printf("p1\n");

signal(SIGUSR1,Int1);

signal(16,SIG\_IGN);

pause();

exit(0);

}

else

{

while((pid2=fork())==-1);

if(pid2==0)

{

printf("p2\n");

signal(SIGUSR2,Int2);

signal(17,SIG\_IGN);

pause();

exit(0);

}

else

{

printf("parent\n");

signal(SIGINT,IntDelete);

waitpid(-1,&exitpid,0);

printf("Parent process is killed!\n");

exit(0);

}

}

}

**实验四 进程的管道通信**

#include<unistd.h>

#include<stdio.h>

#include<signal.h>

int pid1,pid2;

main()

{

int fd[3];

char OutPipe[100],InPipe[100]; //建立管道

pipe(fd);

while((pid1=fork())==-1);

if(pid1==0)

{

printf("p1\n");

lockf(fd[1],1,0);

sprintf(OutPipe,"Child 1 process is sending a message!"");

write(fd[1],OutPipe,50); //向管道中写入内容

sleep(1);

lockf(fd[1],0,0);

exit(0);

}

else

{

while((pid2=fork())==-1);

if(pid2==0)

{

printf("p2\n");

lockf(fd[1],1,0);

sprintf(OutPipe,"Child 2 process is sending a message!");

write(fd[1],OutPipe,50);

sleep(1);

lockf(fd[1],0,0);

exit(0);

}

else

{

printf("parent\n");

wait(0);

read(fd[0],InPipe,50); //从管道中读取内容

printf("%s\n",InPipe);

wait(0);

read(fd[0],InPipe,50);

printf("%s\n",InPipe);

exit(0);

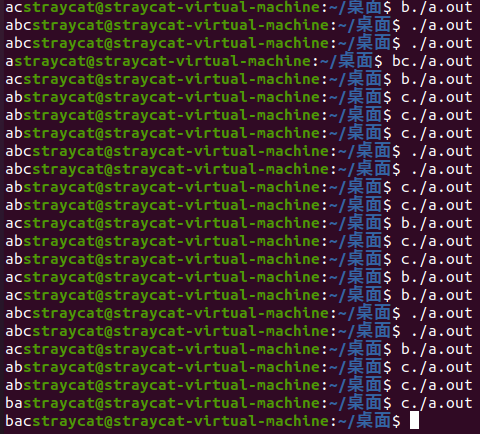
}

}

}

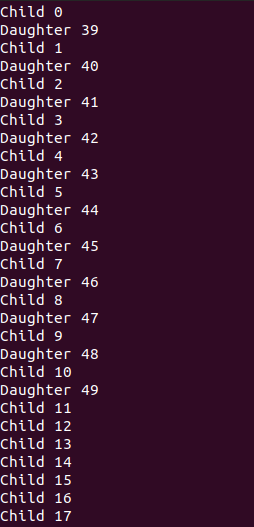
六、实验结果

**实验一 进程的创建**

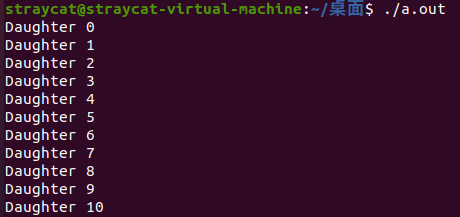


**实验二 进程的控制**

**【代码1】**

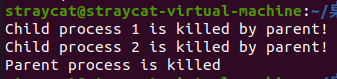
****

**【代码2】**

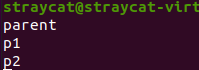
****

**实验三 进程的软中断通信**

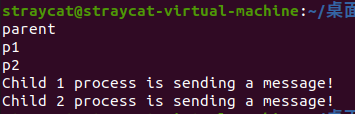
**【代码1】**

****

**【代码2】**

****

**实验四 进程的管道通信**



# 实验项目三：银行家算法实现

一、实现目的

1）理解死锁的概念，了解导致死锁的原因

2）掌握死锁的避免方法，理解安全状态和不安全状态的概念

3）理解银行家算法，并应用银行家算法避免死锁

二、实验准备

1）掌握VS下开发调试程序的方法

2）了解银行家算法的原理

3）了解如何利用银行家算法避免不安全状态的出现

4）了解输入输出重定向的概念和简单的使用方法

三、实验基本知识及原理

1.基本概念

**死锁：**多个进程在执行过程中，因为竞争资源会造成相互等待的局面。如果没有外力作用，这些进程将永远无法向前推进。此时称系统处于死锁状态或者系统产生了死锁。

**安全序列：**对于一个进程序列{P1,…,Pn}，如果对于每一个进程Pi以后尚需要的资源数量不超过系统当前剩余资源量和所有进程Pj（j＜i）当前占用资源之和，则称序列{P1,…,Pn}为一个安全序列。

**安全状态：**如果存在一个由系统中所有进程构成的安全序列P1,…,Pn，则系统处于安全状态，安全状态一定是没有死锁发生。

**不安全状态：**在当前形势下不存在安全序列，则系统处于不安全状态。

2.银行家算法

**银行家算法：**如果将操作系统的资源视为银行家管理的资金，进程向操作系统请求分配资源就好像用户向银行家贷款。操作系统可以像银行家一样，按照规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量，则按当前的申请量为其分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先判断本次申请资源数是否超过了剩余资源的总量，如果资源数未超过剩余资源总量，则进行分配，否则推迟分配。

**银行家算法的基本思想：**分配资源之前，先判断系统是否处于安全状态，若处于安全状态则分配资源，否则不进行分配。该算法是典型的避免死锁算法。

（1）银行家算法的数据结构

1）可利用资源向量Available：一个含有m个元素的数组，数组中每个元素代表一类可利用资源的数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。如果Available[j]=K，则表示系统中现有Rj类资源K个。

2）最大需求矩阵Max：一个n×m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max[i,j]=K，则表示进程i需要Rj类资源的最大数目为K。

3）分配矩阵Allocation ：一个n×m的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果Allocation[i,j]=K，则表示进程i当前已分得Rj类资源的数目为K

4）需求矩阵Need ：一个n×m的矩阵，用以表示每一个进程尚需的各类资源数。如果Need[i,j]=K，则表示进程i还需要Rj类资源K个，方能完成其任务。

（2）银行家算法的基本结构

设Requesti是进程Pi的请求向量，如果Requesti[j]=K，表示进程Pi需要K个Rj类型的资源。当Pi发出资源请求后，系统按下述步骤进行检查：

1）如果Requesti[j]≤Need[i.j]，便转向步骤B；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

2）如果Requesti[j]≤Available[j]，则便转向步骤C；否则，表示尚无足够资源，Pi须等待。

3）系统试探着将资源分配给进程Pi，并修改下面数据结构中的数值：

Avai1able[j]=Available[j]-Requesti[j];

Allocat1on[i,j]=Allocati1on[i,j]+Requesti[j];

Need[i,j]=Need[i,j]-Requesti[J];

4）系统在安全状态下执行，每次资源分配前都检查此次资源分配后系统是否仍处于安全状态。若处于安全状态，则将资源分配给进程Pi：否则，本次试探分配行为作废，恢复原来的资源分配状态，让进程Pi等待。

四、实验说明

1）本实验将采用数组模型来模拟实现银行家算法

2）本程序使用输入重定向的方式进行数据输入

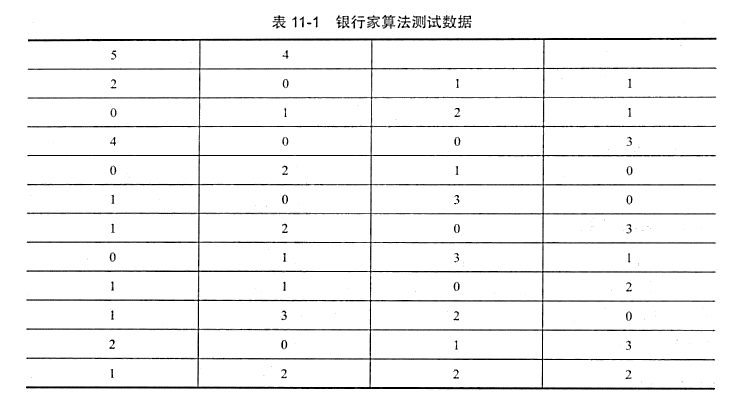
3）部分API说明：

void showdata(); //显示进程初始信息

void changedata(); //对i进程进行资源分配，修改相应的信息

void issafe(); //判断系统是否安全

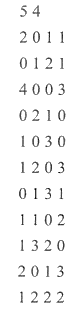
4）实验数据如表11-1所示



五、实验内容

1）实验步骤：创建VS工程banker，将代码2-5-1添加到该工程中

2）将测试数据写入到文本文件in中



3）阅读代码，根据银行家算法认真思考int issafe()函数的工作原理，以及该函数是如何生成一个安全序列的，是否有更高效的方法实现同样的功能。

4）函数“void changedata(int i)”表示第i次完成资源分配，请实现该函数。

5）执行程序：

①按下启动键+R，输入cmd进入命令行窗口

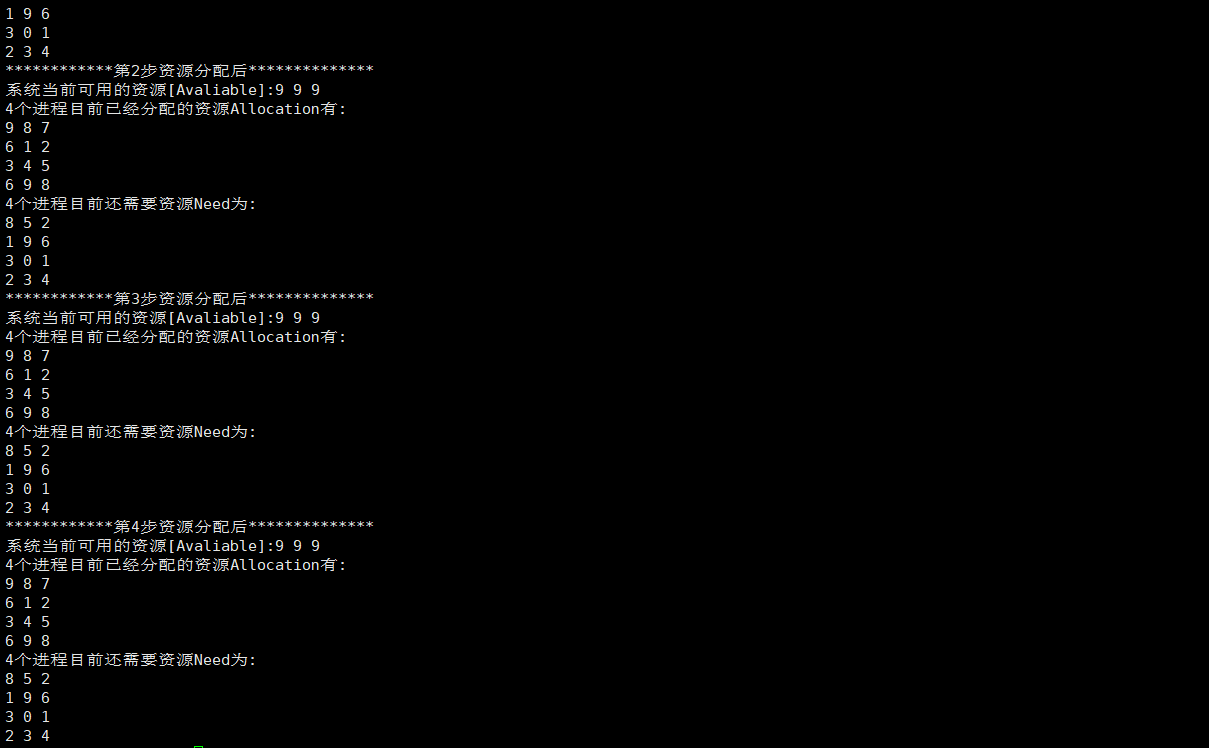
②切换到banker工程的debug目录下，将之前写好的in文件放到该目录下

③执行bank < （通过输入重定向实现数据的输入）

④查看结果

6）完成相应的实验报告

六、结果分析



七、程序代码

#include<stdio.h>

#include<stdbool.h>

#include<stdlib.h>

#define M 100

int Available[M] = {0}; // 可用资源数组

int MAXP[M][M] = {0}; // 最大需求矩阵

int Allocation[M][M] = {0}; // 分配矩阵

int Need[M][M] = {0}; // 需求矩阵

int tem[M] = {0}; // 存放安全序列

int Work[M] = {0}; // 存放系统可提供资源

bool Finish[M]; // 系统是否有足够资源分配

int m = 0; // 进程数

int n = 0; // 资源数

void showdata(); // 显示进程初始信息

void changedata(int i); // 进行资源分配

int issafe(); // 判断是否为安全

int main(){

int i, j;

// printf("m n: ");

// scanf("%d %d\n", &m, &n);

m = 4;

n = 3;

printf("allocation: \n");

for(i=0;i<m;i++)

for(j=0;j<n;j++)

scanf("%d", &Allocation[i][j]);

printf("Need: \n");

for(i=0;i<m;i++)

for(j=0;j<n;j++)

scanf("%d", &Need[i][j]);

printf("available: \n");

for(j=0;j<n;j++)

scanf("%d", &Available[j]);

showdata();

int a = issafe();

if(a==1){

for(i=0;i<m;i++)

{

int k = tem[i];

changedata(k);

printf("\*\*\*\*\*\*\*\*\*\*\*\*第%d步资源分配后\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", i+1);

showdata();

}

}

return 0;

}

void changedata(int i){

}

void showdata(){

int i, j;

printf("系统当前可用的资源[Avaliable]:");

for(i=0;i<n;i++)

printf("%d ", Available[i]);

printf("\n%d个进程目前已经分配的资源Allocation有:\n", m);

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

printf("%d ", Allocation[i][j]);

printf("\n");

}

printf("%d个进程目前还需要资源Need为: \n", m);

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

printf("%d ", Need[i][j]);

printf("\n");

}

}

int issafe(){

int num=0;

int i,j;

for(i=0;i<m;i++)

Work[i] = Available[i];

for(i=0;i<n;i++)

{

Finish[i] = false;

}

for(i=0;i<m;i++)

{

if(Finish[i] == true)

continue;

else{

for(j=0;j<n;j++)

{

if(Need[i][j]>Work[j])

break;

}

if(j==n)

{

Finish[i] = true;

int k=0;

for(k=0;k<n;k++)

Work[k] += Allocation[i][k];

tem[num] = i;

num ++;

i = -1;

}else continue;

}

}

if(num == m)

{

printf("YES \n");

printf("安全序列为: ");

for(i=0;i<num;i++)

{

printf("%d", tem[i]);

if(i < num-1)

printf("->");

}

printf("\n");

return 1;

}else{

printf("No \n");

return 0;

}

}

# 实验项目四：磁盘调度实验

一、实现目的

通过本章的实验，读者应达到如下要求：

1）了解磁盘结构。

2）了解磁盘上数据的组织方式。

3）掌握磁盘访问时间的计算方法。

4）掌握常用的磁盘调度算法以及算法的相关特性。

5）掌握界面程序的设计和编写方法。

6）巩固由算法到伪代码到代码再到可运行程序的转化能力。

二、实验准备

1）了解磁盘分类以及磁盘性能的评价标准

2）预习磁盘调度相关算法

三、实验基本知识及原理

1.磁盘结构

磁盘驱动器包含两个移动部件：一个是磁头组合，另一个是磁盘组合，磁盘组合由一个或多个圆盘组成，它们围绕着一根中心主轴旋转（轴心）。圆盘的上表面和下表面涂覆一薄层磁性材料，二进制位被存储在这些磁性材料上。其中， 0和1在磁材料中表现为不同的模式。磁盘的直径一般是3.5英寸。

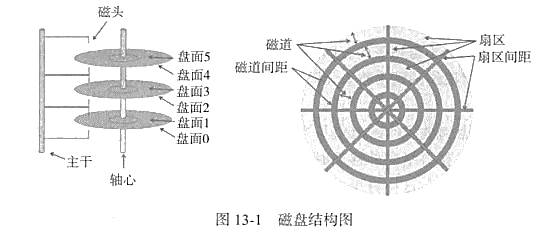
磁盘被组织成磁道，磁道是单个磁片上的同心圆。所有盘面上的半径相同的磁道构成柱面。磁道被组织成扇区，扇区是被间隙分割的圆的片段，间隙未被磁化为0和1，磁盘结构图如图13—1所示。

2.磁盘数据的组织

磁盘上每一条物理记录都有唯一的地址，该地址包括三部分：磁头号（盘面号）、柱面号（磁道号）和扇区号。给定这三个量就可以唯一地确定一个地址。

3.磁盘访问时间的计算方式

磁盘在工作时以恒定速率旋转。为保证读或写，磁头必须能移动到所要求的磁道上，当所要求的扇区的开始位置旋转到磁头下时，开始读或写数据。对磁盘的访问时间包括寻道时间、旋转时间和传输时间三部分。



(1)寻道时间Ts

磁臂（磁头）从当前位置移动到指定磁道上所经历的时间。该时间是启动磁臂的时间s与磁头移动n条磁道所花费的时间之和，即Ts =m+n+s

m是一个常数，它与磁盘驱动器的速度有关。对一般磁盘而言，m=0.3（或0.2）

(2)旋转延迟时间Tr

对于硬盘，典型的旋转速度为3600r/min （5400r/min或7200r/min），每转需16.6ms （11.1,s），平均旋转延迟时间T为8.3ms（5.55ms）。对于软盘，其旋转速度为300或600r/min，平均Tr为50~100ms

（3）传输时间T

Tt是指将数据从磁盘读出或向磁盘写入数据所经历的时间。Tt的大小与每次所读/写的字节数b及旋转速度有关：

Tt=b/(r+N)

r为磁盘以秒计的旋转速度；N为一条磁道上的字节数。当一次读/写的字节数相当于半条磁道上的字节数时，Tt与Tr相同。因此，可将访问时间Ta表示为：

Ta=Ts+1/2r+b/(гN)

（4）访问时间

对磁盘的访问时间即可表示为：Ts+Tr+Tt

4，磁盘调度算法

（1）先来先服务算法

该算法是一种简单的磁盘调度算法，它根据进程请求访问磁盘的先后次序进行调度。此算法的优点是公平、简单，且每个进程的请求都能依次得到处理，不会出现某一进程的请求长期得不到满足的情况。此算法由于未对寻道进行优化，在对磁盘的访问请求比较多的情况下，此算法将降低设备服务的吞吐量，平均寻道时间可能较长，但各进程得到服务的响应时间的变化幅度较小。

（2）最短寻道时间优先算法

该算法优先响应要求访问磁道与当前磁头所在磁道距离最近的进程，以使每次寻道时间最短。该算法可以得到较好的吞吐量、但却不能保证平均寻道时间最短。该算法的缺点是对用户服务请求的响应机会不是均等的，因而导致响应时间的变化幅度很大。在服务请求很多的情况下，对内外边缘磁道的请求将会无限期地被延迟，有些请求的响应时间将不可预期。

（3）扫描算法

扫描算法不仅考虑到欲访问的磁道与当前磁道的距离，而且优先考虑磁头的当前移动方向。例如，当磁头正在自里向外移动时，扫描算法所选择的下一个访问对象应是磁头移动方向向外且距离最近的。这样自里向外地访问，直到再无更外的磁道需要被访问才会将磁臂的移动方向变为自外向里移动。磁臂改变方向后，同样选择这样的进程来调度，即其要访问的磁道在当前磁道之内，从而避免了饥饿现象。由于这种算法中磁头移动的规律颇似电梯的运行，故称为电梯调度算法。此算法基本上克服了最短寻道时间优先算法的服务集中于中间磁道和响应时间变化比较大的缺点，而具有最短寻道时间优先算法的优点，即吞吐量较大，平均响应时间较小。但由于是摆动式的扫描方法，两侧磁道被访问的频率仍低于中间磁道。

（4）循环扫描调度算法

循环扫描算法是对扫描算法的改进。如果对磁道的访问请求是均匀分布的，当磁头到达磁盘的一端并反向运动时，落在磁头之后的访问请求相对较少。这是由于这些磁道刚被处理，而磁盘另一端的请求密度相当高，且这些访问请求等待的时间较长。为了解决这种情况，循环扫描算法规定磁头单向移动。例如，只自里向外移动，当磁头移到最外的被访问磁道时，磁头立即返回到最里的欲访问磁道，即将最小磁道号紧接着最大磁道号构成循环，进行扫描。

（5）LOOK调度算法

在SCAN算法中，磁头每次调转方向总是回到柱面顶端，增加了磁头的寻道长度，LOOK调度算法在SCAN算法的基础上，会判断该请求是否为沿着这个方向上的最后一个，如果是则立即掉头。

（6） CLOOK调度算法

CLOOK调度算法与LOOK相似，在CSCAN的基础上进行了改进。

四、实验说明

本实验为模拟实现磁盘调度算法。

1.测试数据

磁盘访问序列:98,181.35,112,24,144.45,57;

读写头起始位置：42

2，测试数据文件

job.txt数据组织方式：开始磁道/共需要访问的磁道数目/待访问磁道

3.程序要求

1）根据实验所给源码，描述实现SSTF算法过程并评价SSTF算法的性能。

2）实现SCAN，CSCAN，LOOK，CLOOK算法，根据测试数据给出测试结果；并根据测试结果比较FCFS，SSTF，CSCAN，CLOOK算法的性能。

3）比较SCAN算法和LOOK算法的异同

4.程序中涉及的函数名及变量

1）算法函数：

void FCFS();

void SSTF();

void SCAN();

void LOOK();

void CSCAN();

vo1d CLOOK();

2）其他函数：

void show(int \*arr，int num); //显示结果

void readData（int \*arr）； //读取数据

3）程序中涉及的重要变量

int initPosition; //保存初始位置

int numTrack; //磁道数（不算初始位置）

int arrayTrack[65535]; //磁道序列

int arrayResult[65535]; //访问顺序序列

bool sign=0; //是否导入数据

int length=0; //寻道长度

五、实验内容

1）在Windows系统上，使用VS创建工程

2）在工程下导入实验源码

3）在工程源码的统计目录下创建job.txt文件，并导人测试数据：



4）编译源代码

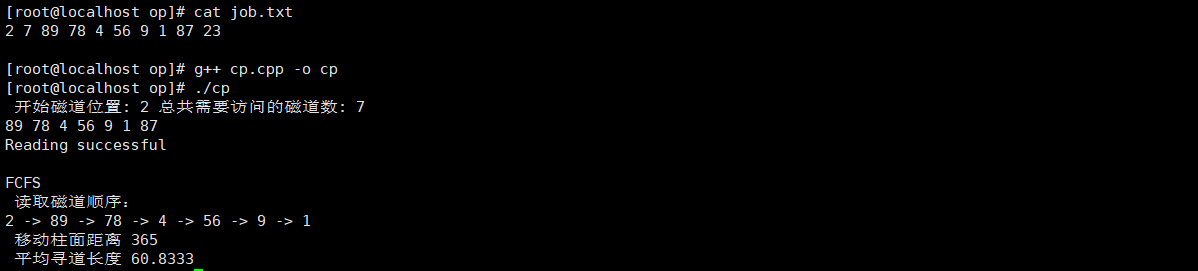
5）观察并结合运行结果研究实例代码中的FCFS算法和STTF算法

6)补充SCAN, LOOK, CSCAN, CLOOK代码,运行并分析运行结果

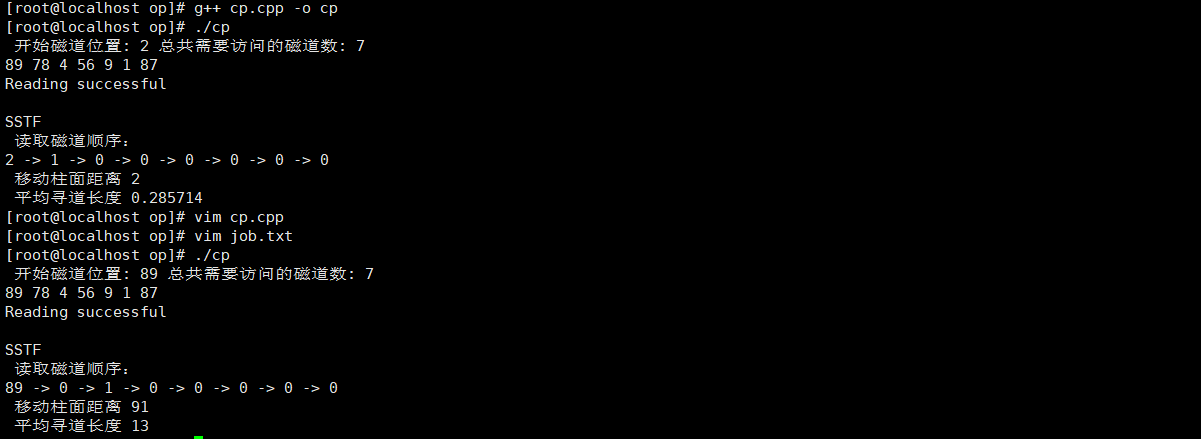
六、实验总结

这个实验比较简单，让我更加深刻理解了磁盘调度算法，更加直观看到了磁盘调度算法的运作方式。同时我也能够学会尝试着去评判调度算法的优劣。

FCFS算法的实验截图：



SSTF算法的实验截图：



七、程序代码

#include<iostream>

#include<fstream>

#include<math.h>

#include<cstdlib>

#include<algorithm>

#include<stdbool.h>

using namespace std;

int initPosition;

int numTrack;

int arrayTrack[65536];

int arrayResult[65536];

bool sign = 0;

int length = 0;

int i=0;

void show(int \*arr, int num);

void readData(int \*arr);

void FCFS();

void SSTF();

void SCAN();

void LOOK();

void CSCAN();

void CLOOK();

int main(){

readData(arrayTrack);

FCFS();

readData(arrayTrack);

SSTF();

return 0;

}

void FCFS()

{

if(sign == 0){

cout<< "未读入文件 "<<endl;

return;

}

cout <<"\nFCFS"<<endl;

length = 0;

arrayResult[0] = initPosition;

for(i=0;i<numTrack;i++)

{

arrayResult[i+1] = arrayTrack[i];

length += abs(arrayResult[i+1] - arrayResult[i]);

}

show(arrayResult,numTrack);

}

void SSTF(){

if(sign == 0){

cout<< "未读入文件 "<<endl;

return;

}

cout <<"\nSSTF"<<endl;

length = 0;

int index;

int min;

arrayResult[0] = initPosition;

for(i=0;i<numTrack;i++){

min = 65535;

index = i+1;

int t;

for(t=i+1;t<numTrack+1;t++)

{

if(abs(arrayResult[i]-arrayResult[t])<min){

min = abs(arrayResult[i]-arrayResult[t]);

index = t;

}

}

int tem = arrayResult[t+1];

arrayResult[i+1] = arrayResult[index];

arrayResult[index] = tem;

length += arrayResult[i+1]-arrayResult[i];

}

show(arrayResult, numTrack+1);

}

void show(int \*arr, int num){

cout << " 读取磁道顺序："<< endl;

for(i=0;i<num;i++){

if(i!=num-1)

cout<<arr[i]<<" -> ";

}

cout<<" 移动柱面距离 "<<length<<endl;

cout<<" 平均寻道长度 "<<((float)length)/((float)num-1)<<endl;

}

void readData(int \*arr){

int i=0;

ifstream fin("job.txt");

if(!fin)

{

cout <<"Wrong in open file"<<endl;

exit(1);

}

fin>>initPosition>>numTrack;

cout<<" 开始磁道位置: "<<initPosition<<" 总共需要访问的磁道数: "<<numTrack<<endl;

for(i=0;i<numTrack;i++)

{

fin>>arr[i];

cout<<arr[i]<< " ";

}

cout<<endl;

cout<<"Reading successful"<<endl;

sign = 1;

fin.close();

}

|  |  |
| --- | --- |
| **学生实验 心得** | 这次的操作系统课程，让我从更深的一个角度来了解到了计算机的内部工作原理，了解到了操作系统是如何组织硬件高效有序工作的。这让我能够很好的去解释平时碰到的一些计算机方面的问题，同时也为我今后的计算机学习道路奠下了坚实的基础。而这次的实验课程，是一次对理论知识的绝好拓展机会，四个课程：Linux编程基础、进程的创建与控制、银行家算法、磁盘调度。这四个实验加深了我对理论知识的理解和运用。  学生（签名）：彭重严  2020年11月23日 |
| **指导**  **教师**  **评语** | 成绩评定：  指导教师（签名）：  年 月 日 |

实验心得与教师评语