**大连理工大学**

**本科实验报告**

课程名称： 操作系统实验

学院（系）：电子信息与电气工程

专 业： 计算机科学与技术

班 级： 电计1601

学 号： 201685085

学生姓名： 张世奇

2019年 6 月 11 日

**1.进程管理（20 分）**

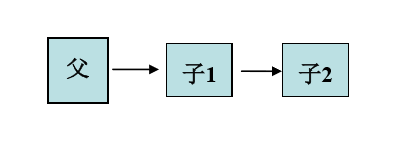
1.1 实验目的

加深对于进程并发执行概念的理解。实践并发进程的创建和控制方法。观察和体验进程的动态特性。进一步理解进程生命期期间创建、变换、撤销状态变换的过程。掌握进程控制的方法，了解父子进程间的控制和协作关系。练习 Linux 系统中进程创建与控制有关的系统调用的编程和调试技术。

1.2 实验说明

* 每个进程都执行自己独立的程序，打印自己的pid，每个父进程打

印其子进程的pid



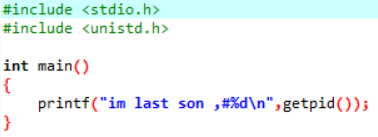
a.c：

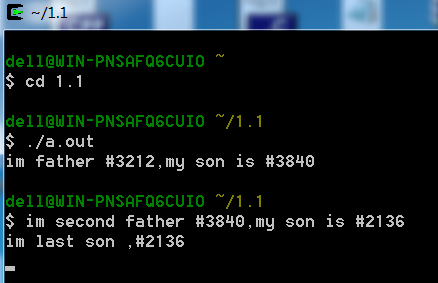


b.c:

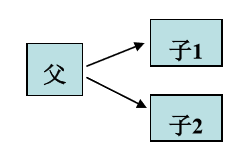


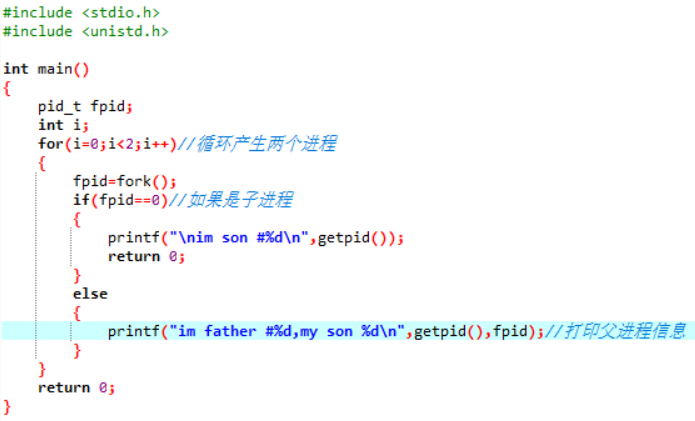
c.c:

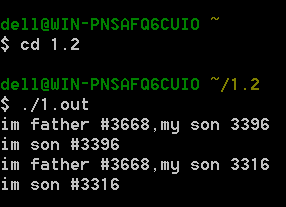




* 每个进程都执行自己独立的程序，打印自己的pid，父进程打印其子进程的pid

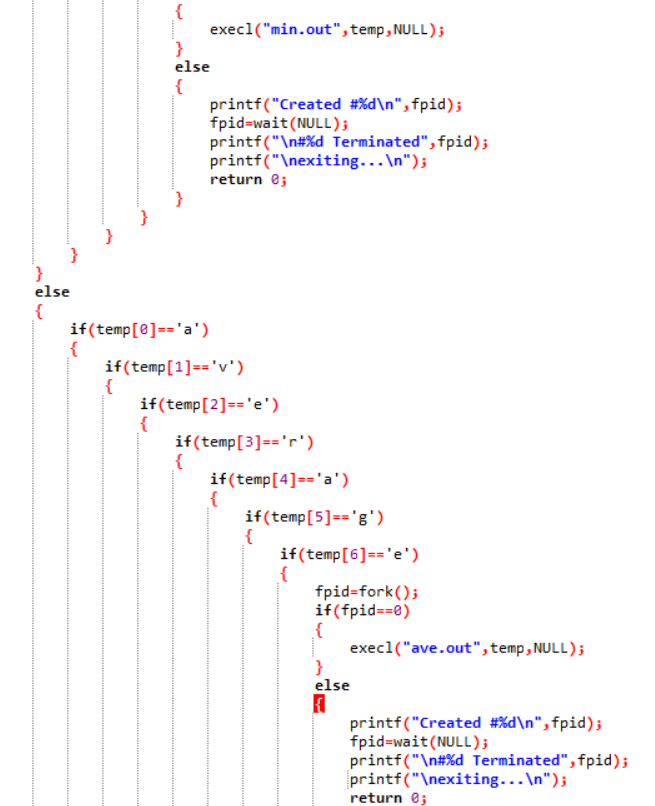


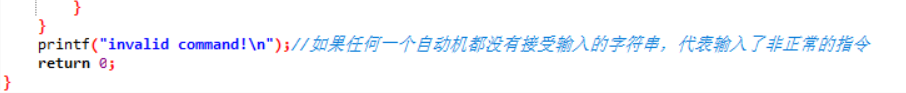




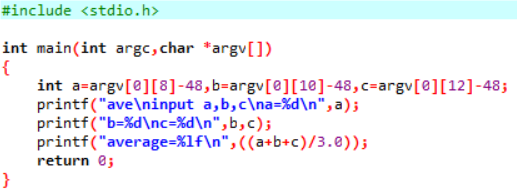
* 编写一个命令处理程序，能 处 理 max(m,n), min(m,n)和average(m,n,l)这几个命令。



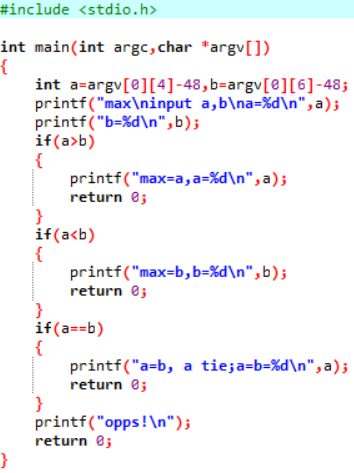




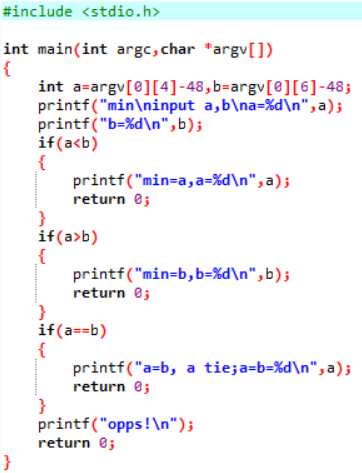
avg.c:

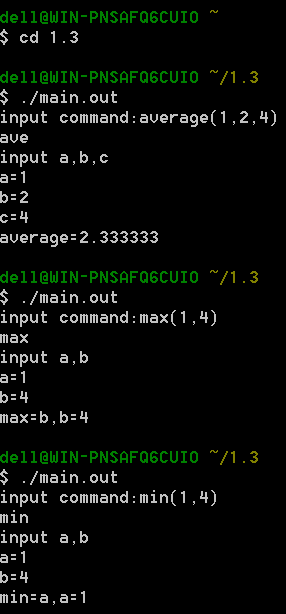


max.c:



min.c:





**2.** **处理器调度（20 分）**

1.1 实验目的

熟悉使用各种单处理器调度的各种算法，加深对于处理机调度机制的理解。练习模拟算法的编程技巧，锻炼分析试验数据的能力。

1.2 实验说明

随机给出一个进程调度实例，如：

进程 到达时间 服务时间

A 0 3

B 2 6

C 4 4

D 6 5

E 8 2

模拟进程调度，给出按照算法先来先服务 FCFS、轮转 RR（q=1）、最短进程优先 SJF、最高响应比优先 HRN 进行调度各进程的完成时间、周转时间、带权周转时间。

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <fstream>

#define PNUM 10

using namespace std;

ofstream fout("log\_Process.txt");

class list

{

private:

int data[3];//0:pid 1:arrive time 2:service time

list \*next;

list \*pre;

public:

list()

{

next=NULL;

pre=NULL;

}

void set(int pid,int \*a,list \*pre\_,list \*next\_)

{

data[0]=pid;

data[1]=a[0];

data[2]=a[1];

pre=pre\_;

next=next\_;

}

void append(list \*next\_)//append a list

{

next=next\_;

}

void cut\_in(list \*pre\_)//put a list in the front

{

pre=pre\_;

}

int \*get()

{

return data;

}

list \*g\_next()

{

return next;

}

list \*g\_pre()

{

return pre;

}

};

class link//a round link contains process info

{

private:

list \*front;

list \*rear;

int total;

public:

link(int a[][2])

{

list \*now;

total=PNUM;

front=new list;//initialize

front->set(0,\*a,front,front);

rear = front;

for(int i=1;i<total;i++)

{

now=new list;//create a new list

now->set(i,\*(a+i),rear,front);

rear->append(now);//append the new list after the rear list

front->cut\_in(now);//let the forefront list know

rear=now;//move on to the next insertion

}

/\* list \*temp=front;

for(int j=0;j<50;j++)

{

cout<<temp->get()[0]<<" ";

temp=temp->g\_next();

}\*/ // this is test code for a double-direction-round-linklist

}

list \*get()

{

return front;

}

int g\_total()

{

return total;

}

void set\_front(list \*new\_front)

{

front=new\_front;

}

void del\_list(list \*del)

{

list \*pre,\*next;

pre=del->g\_pre();

next=del->g\_next();

pre->append(next);

next->cut\_in(pre);

delete del;

}

list \*g\_front()

{

return front;

}

};

class process

{

private:

int attribute[PNUM][2];//start time ont the front followed by service time

public:

process()

{

srand(int(time(0)));//get seed ready

attribute[0][0]=0;//1st start time

attribute[0][1]=(rand()%10+1);//1st service time

for(int i=1,start\_time=0;i<PNUM;i++)

{

attribute[i][0]=start\_time+(rand()%10);//get a start time that other than 0

start\_time=attribute[i][0];//set the new start\_time

attribute[i][1]=(rand()%10+1);//get a service time

}

cout<<"process arrive\_time service\_time\n";

for(int i=0;i<PNUM;i++)

{

cout<<setw(4)<<i/\*+1\*/<<setw(12)<<attribute[i][0]<<setw(15)<<attribute[i][1]<<endl;

fout<<setw(4)<<i/\*+1\*/<<setw(12)<<attribute[i][0]<<setw(15)<<attribute[i][1]<<endl;//print to the log.txt

}

}

int (\*get())[2]

{

return attribute;

}

};

class HRN

{

private:

list \*l;

int log[PNUM][3];//[0]:total time [1]:service time [2]:turnaround time

float pri[PNUM];//priority of processes

float TTR[PNUM];//Turnaround Time with Rights

int total;//the time that plugged in

int p\_total;//the number of process that exist right now

int service;//the service time of current process

int i;//a temp container for pid

float max;//the maximum value of priority

int pid\_max;//the pid of maximum priority process

list \*p\_max;//the ptr of MPP

public:

void p\_finish(int current,int total,int service,int turnaround,float ttr,float pri)//print the finish info

{

cout<<"\n#"<<current<<" process finished after "<<total<<" cycle(s) ";

cout<<"service time is "<<service<<" cycle(s)\n";

cout<<"priority is "<<pri;

cout<<" turnaround time is "<<turnaround<<" turnaround time with right is "<<ttr<<endl;

}

void set(int pid,int total\_time,int service,int arrive)

{

log[pid][0]=total\_time;

log[pid][1]=service;

log[pid][2]=(total\_time - arrive);

}

void up\_pri(int pid,int arrive,int serve)//update the priority

{

pri[pid]=((float)((total-arrive)+serve)/(float)serve);

}

HRN(link li)

{

int flag=0;//whether a new process appeared or not

int flag\_compared=0;//to detect if the front ptr was compared(avoid all process arrived and dead-loop in comparison)

service=0;

max=0.0;//init the maximum

p\_max=0;

list \*front;

l=li.g\_front();

front=l;

p\_max=l;

p\_total=PNUM;

total=0;

list \*temp;//update ptr

for(int j=0;j < p\_total;j++)//init the priority

{

pri[j]=1.0;

}

for(int j=0;j<p\_total;j++)//init the log

{

for(int k=0;k<3;k++)

{

log[j][k]=0;

}

}

while(p\_total != 0)//if there still some processes left

{

max=0.0;//init the maximum

service=0;

flag=0;

flag\_compared=0;

for(int j=0;p\_total != 0;j++)//search for the pri max process

{

if(l->get()[1] > total || flag\_compared >= 2)//arrive time > total || the front ptr was compared twice(that means all process was arrived)

{

if(flag == 0)//if no process was chose in this loop

{

// cout<<"time="<<total<<" idle"<<endl;

fout<<"time="<<total<<" idle"<<endl;

total++;

continue;

}

i=p\_max->get()[0];

l=p\_max;//choose the pri max process

break;

}

i=l->get()[0];

if(pri[i] > max)//if pri > max

{

flag=1;//a pri-greater process that appeared

p\_max=l;

pid\_max=i;

max=pri[i];

}

if(l == front)

{

flag\_compared++;

}

l=l->g\_next();

}

for(;service < l->get()[2] ;service++)//serve the process

{

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<endl;

total++;

}

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<" finished\n"<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<" finished\n"<<endl;

set(i,total,service,l->get()[1]);//save the scene

if(p\_max == front)//if the process that need to be deleted is the front

{

front=front->g\_next();

}

l=front;//point the current process to the head

li.del\_list(p\_max);//delete the finished process

p\_total--;

temp=front;

for(int j=0;j < p\_total;j++)//update all the pri of processes which meet the request

{

if(temp->get()[1] <= total)

{

up\_pri(temp->get()[0],temp->get()[1],temp->get()[2]);

}

if(temp->get()[1] > total)//if update procedure is finished

{

break;

}

temp=temp->g\_next();

}

}

for(int j=0;j<PNUM;j++)//calc the turnaround time with right

{

TTR[j]=((float)log[j][1]/(float)log[j][2]);

}

for(int j=0;j<PNUM;j++)//print the result

{

p\_finish(j,log[j][0],log[j][1],log[j][2],TTR[j],pri[j]);

}

}

};

class SJF

{

private:

list \*l;

int log[PNUM][3];//[0]:total time [1]:service time [3]:turnaround time

float TTR[PNUM];//Turnaround Time with Rights

int total;//the time that plugged in

int p\_total;//the number of process that exist right now

int service;//the service time of current process

int i;//a temp container for pid

int min;//the minimum value of service time

int pid\_min;//the pid of minimum service time process

list \*p\_min;//the ptr of MSTP

public:

void p\_finish(int current,int total,int service,int turnaround,float ttr)//print the finish info

{

cout<<"\n#"<<current<<" process finished after "<<total<<" cycle(s) ";

cout<<"service time is "<<service<<" cycle(s)\n";

cout<<"turnaround time is "<<turnaround<<" turnaround time with right is "<<ttr<<endl;

}

void set(int pid,int total\_time,int service,int arrive)

{

log[pid][0]=total\_time;

log[pid][1]=service;

log[pid][2]=(total\_time - arrive);

}

SJF()

{

l=NULL;

p\_min=NULL;

}

SJF(link li)

{

int flag=0;//whether a new process appeared or not

int flag\_compared=0;//to detect if the front ptr was compared(avoid all process arrived and dead-loop in comparison)

service=0;

min=9999;//init the minimum

p\_min=0;

list \*front;

l=li.g\_front();

front=l;

p\_min=l;

p\_total=PNUM;

total=0;

for(int j=0;j<p\_total;j++)

{

for(int k=0;k<3;k++)

{

log[j][k]=0;

}

}

while(p\_total != 0)//if there still some processes left

{

min=9999;//init the minimum

service=0;

flag=0;

flag\_compared=0;

for(int j=0;p\_total != 0;j++)//search for the shortest process

{

if(l->get()[1] > total || flag\_compared >= 2)//arrive time > total || the front ptr was compared twice(that means all process was arrived)

{

if(flag == 0)//if no process was chose in this loop

{

// cout<<"time="<<total<<" idle"<<endl;

fout<<"time="<<total<<" idle"<<endl;

total++;

continue;

}

i=p\_min->get()[0];

l=p\_min;//choose the min-service process

break;

}

i=l->get()[0];

if(l->get()[2] < min)//if service time < min

{

flag=1;//a smaller process that appeared

p\_min=l;

pid\_min=i;

min=l->get()[2];

}

if(l == front)

{

flag\_compared++;

}

l=l->g\_next();

}

for(;service < min ;service++)//serve the process

{

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<endl;

total++;

}

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<" finished\n"<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i<<" finished\n"<<endl;

set(i,total,service,l->get()[1]);//save the scene

if(p\_min == front)//if the process that need to be deleted is the front

{

front=front->g\_next();

}

l=front;//point the current process to the head

li.del\_list(p\_min);//delete the finished process

p\_total--;

}

for(int j=0;j<PNUM;j++)//calc the turnaround time with right

{

TTR[j]=((float)log[j][1]/(float)log[j][2]);

}

for(int j=0;j<PNUM;j++)//print the result

{

p\_finish(j,log[j][0],log[j][1],log[j][2],TTR[j]);

}

}

};

class RR

{

private:

list \*l;

int log[PNUM][3];//[0]:total time [1]:service time [3]:turnaround time

float TTR[PNUM];//Turnaround Time with Rights

int i;//the pid of current running process

int total;//the time that plugged in

int p\_total;

public:

void p\_finish(int current,int total,int service,int turnaround,float ttr)//print the finish info

{

cout<<"\n#"<<current<<" process finished after "<<total<<" cycle(s) ";

cout<<"service time is "<<service<<" cycle(s)\n";

cout<<"turnaround time is "<<turnaround<<" turnaround time with right is "<<ttr<<endl;

}

void set(int pid,int total\_time,int arrive)

{

log[pid][0]=total\_time;

log[pid][2]=(total\_time - arrive);

}

void inc(int pid)

{

log[pid][1]++;

}

RR()

{

l=NULL;

}

RR(link li)

{

int flag=0;//whether the current process was deleted or not

int flag\_processed=0;//to detect that is there a process which is processed in last cycle

list \*front;

l=li.g\_front();

front=l;

p\_total=PNUM;

total=0;

for(int j=0;j<p\_total;j++)//init the log

{

for(int k=0;k<3;k++)

{

log[j][k]=0;

}

}

while(p\_total!=0)//if there still some processes left

{

flag=0;

i=l->get()[0];

if(total < l->get()[1])//if no process arrive on this time

{

if(flag\_processed == 0)

{

total++;

// cout<<"time="<<total<<" idle"<<endl;

fout<<"time="<<total<<" idle"<<endl;

}

l=front;

flag\_processed=0;

}

else

{

flag\_processed=1;

// cout<<"time="<<total<<" serve time= "<<log[i][1]+1<<" process #"<<i<<endl;

fout<<"time="<<total<<" serve time= "<<log[i][1]+1<<" process #"<<i<<endl;

total++;

inc(l->get()[0]);//serve the process

if(log[i][1] == l->get()[2])//if the process finished

{

// cout<<"process #"<<i<<" finished\n\n";

fout<<"process #"<<i<<" finished\n\n";

flag=1;

list \*temp;

set(i,total,l->get()[1]);//save the scene

temp=l;

l=l->g\_next();

if(temp == front)//if the process that need to be deleted is the front

{

front=front->g\_next();

}

li.del\_list(temp);//delete the finished process

p\_total--;

}

if(flag != 1)

{

l=l->g\_next();

}

}

}

for(int j=0;j<PNUM;j++)//calc the turnaround time with right

{

TTR[j]=((float)log[j][1]/(float)log[j][2]);

}

for(int j=0;j<PNUM;j++)//print the result

{

p\_finish(j,log[j][0],log[j][1],log[j][2],TTR[j]);

}

}

};

class FCFS

{

private:

int log[PNUM][3];//[0]:total time [1]:service time [3]:turnaround time

float TTR[PNUM];//Turnaround Time with Rights

int i;//the pid of current running process

int total;//the time that plugged in

int service;//current process time

public:

void p\_finish(int current,int total,int service,int turnaround,float ttr)//print the finish info

{

cout<<"\n#"<<current+1<<" process finished after "<<total<<" cycle(s) \n";

cout<<"turnaround time is "<<turnaround<<" turnaround time with right is "<<ttr<<endl;

}

FCFS(int array[][2])

{

i=0;

total=0;

service=0;//initialize

for(i=0;i<PNUM;i++)//big loop that changes between processes

{

if(total < \*\*(array+i))//if the total cycle < arrive time, fast forward to it

{

total=\*\*(array+i);

// cout<<"\nfast forward to total="<<total<<endl;

fout<<"\nfast forward to total="<<total<<endl;

}

// cout<<"\nnow start to serve process #"<<i+1<<endl;

fout<<"\nnow start to serve process #"<<i+1<<endl;

for(service=0;service < \*(\*(array+i)+1);service++)//provide service

{

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i+1<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i+1<<endl;

total++;

}

// cout<<"time="<<total<<" serve time= "<<service<<" process #"<<i+1<<" finished"<<endl;

fout<<"time="<<total<<" serve time= "<<service<<" process #"<<i+1<<" finished"<<endl;

log[i][0]=total;//save the scene

log[i][1]=service;

log[i][2]=(total-\*\*(array+i));

TTR[i]=((float)service/(float)(total-\*\*(array+i)));

}

for(int j=0;j<PNUM;j++)//print the result

{

p\_finish(j,log[j][0],log[j][1],log[j][2],TTR[j]);

}

}

};

int main()

{

process p1;

cout<<"\nFCFS:\n";

fout<<"\nFCFS:\n";

FCFS(p1.get());

link l1(p1.get()),l2(p1.get()),l3(p1.get());

cout<<"\nRR:\n";

fout<<"\nRR:\n";

RR r1(l1);

cout<<"\nSJF:\n";

fout<<"\nSJF:\n";

SJF s1(l2);

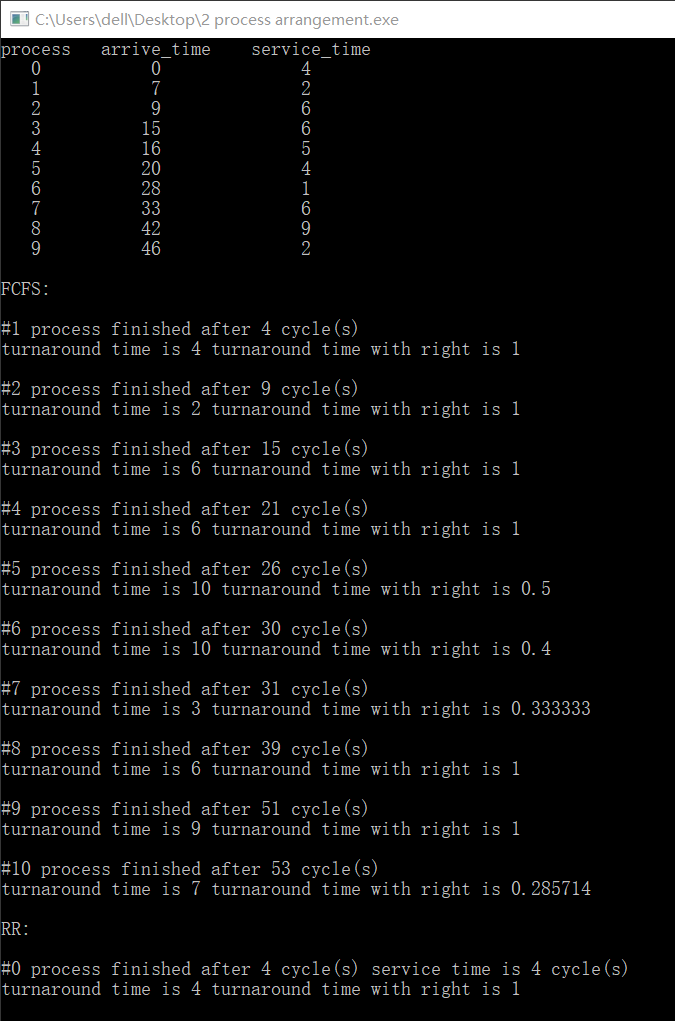
cout<<"\nHRN\n";

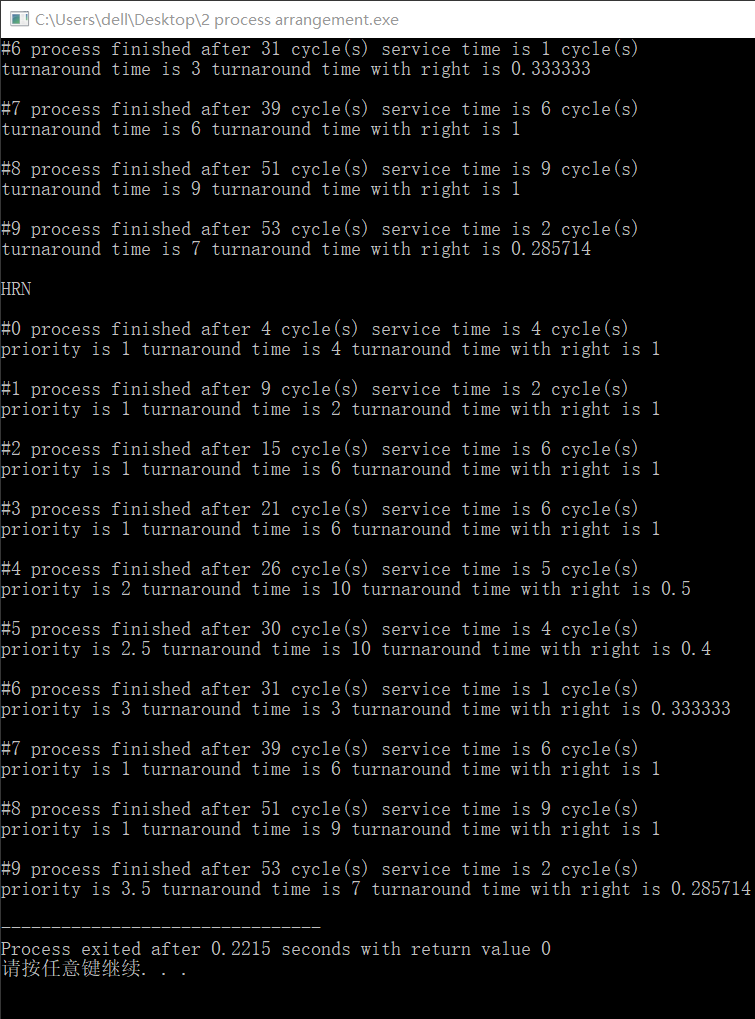
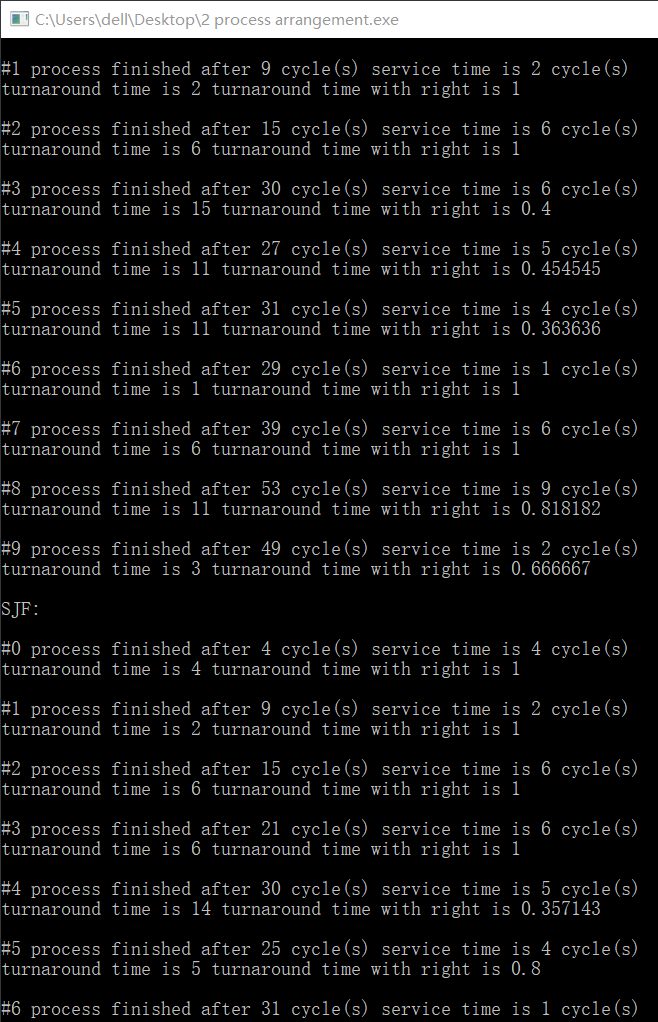
fout<<"\nHRN\n";

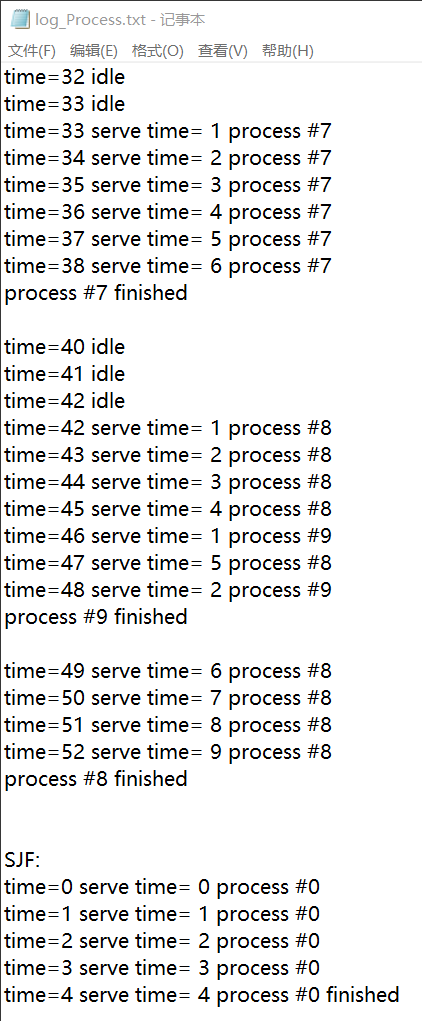
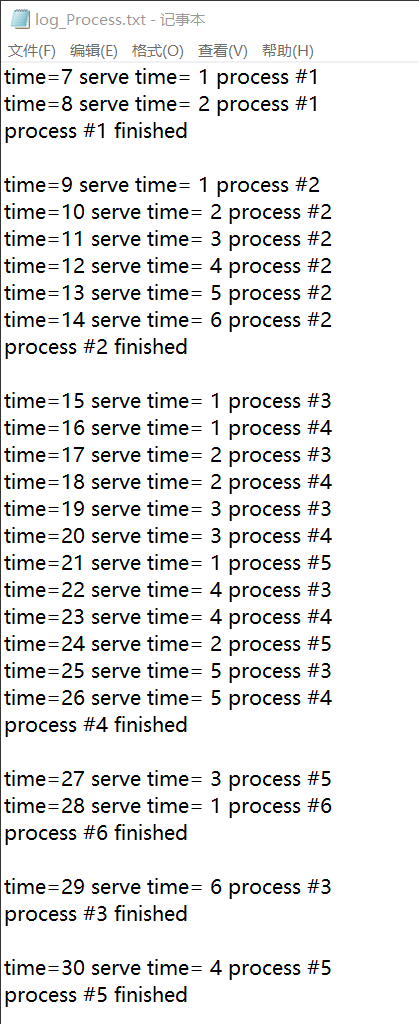
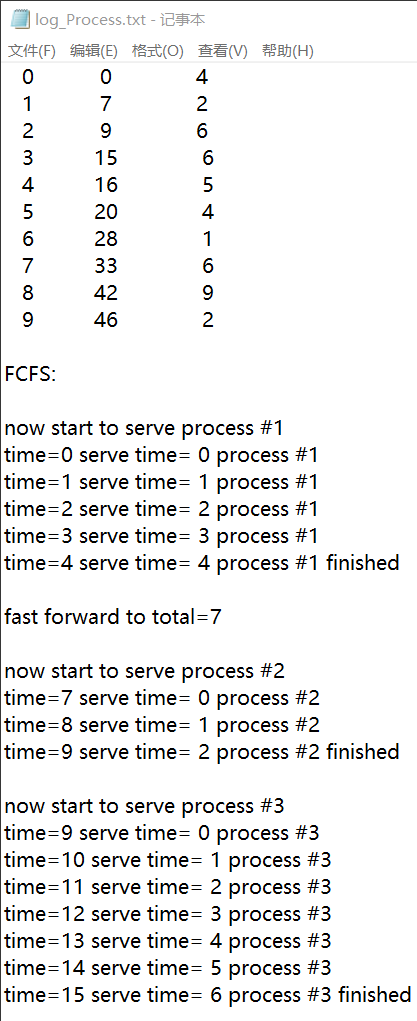
HRN h1(l3);

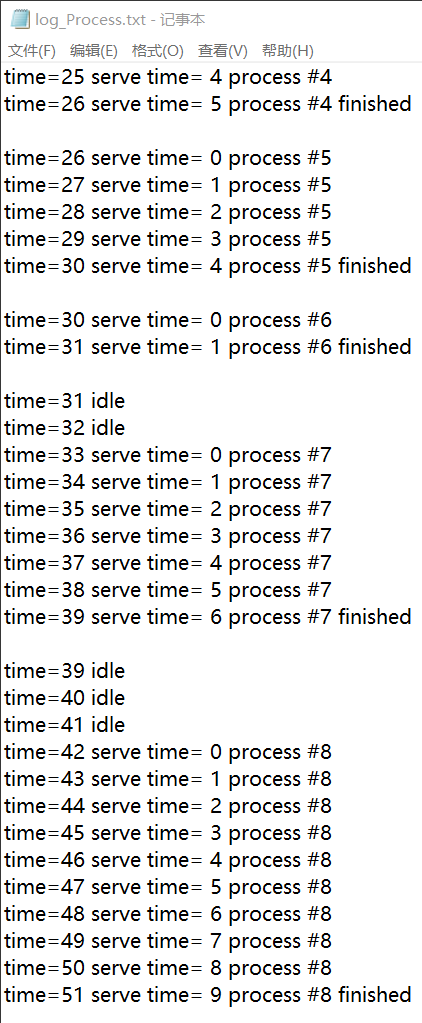
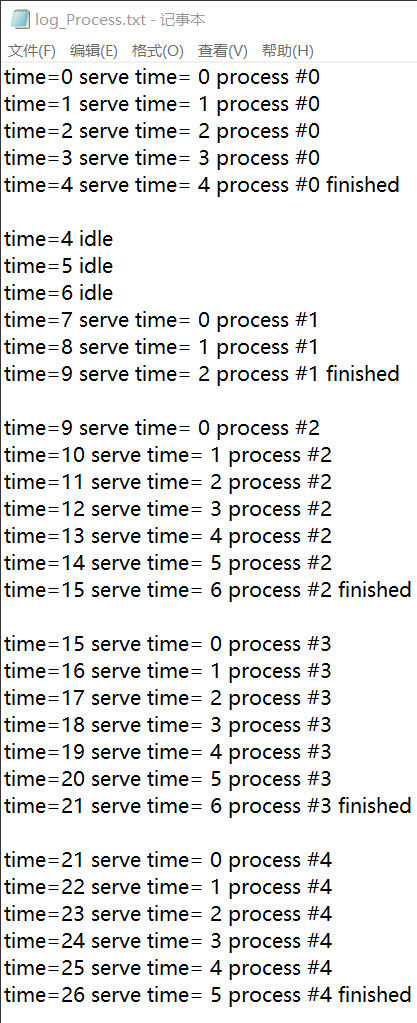
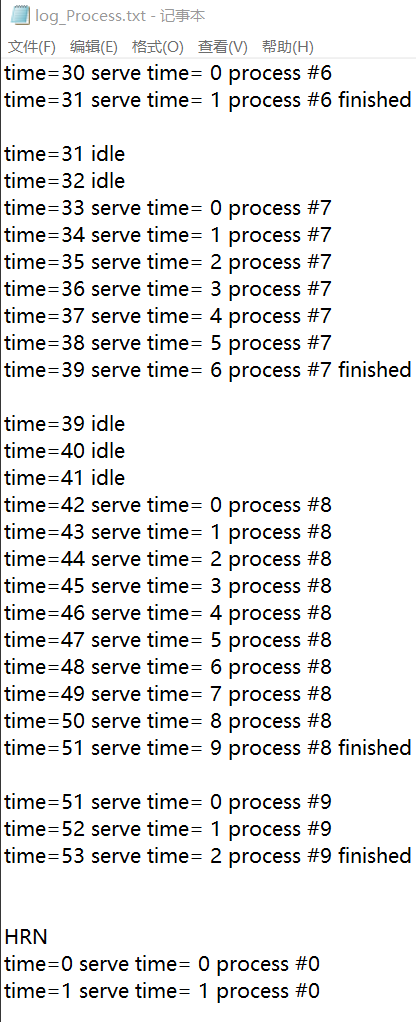
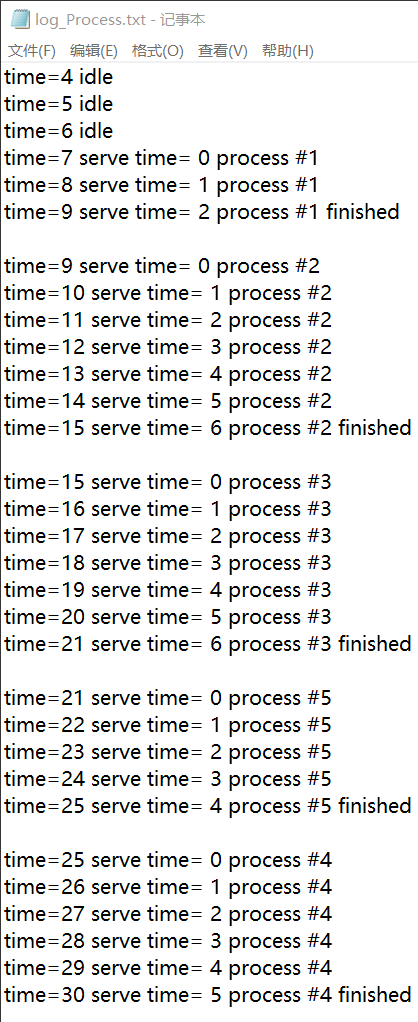
fout.close();

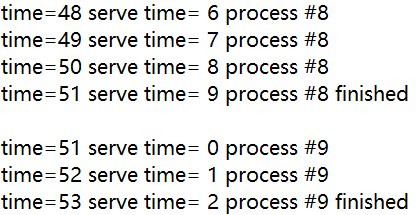
}











**3.** **存储管理上机作业（20 分）**

3.1 实验目的

加深对于存储管理的了解，掌握虚拟存储器的实现原理；观察和了解重要的页面置换算法和置换过程。练习模拟算法的编程技巧，锻炼分析试验数据的能力。

3.2 实验说明

1.示例实验程序中模拟两种置换算法：LRU 算法和 FIFO 算法。

2.给定任意序列不同的页面引用序列和任意分配页面数目，显示两种算法的页置换过程。

3.能统计和报告不同置换算法情况下依次淘汰的页号、缺页次数（页错误数）和缺页率。

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <iomanip>

#include <fstream>

#define PAGENUM 4 //硬件支持的块数

#define SEQLENGTH 20

using namespace std;

ofstream fout("log\_FIFO&LRU.txt");

class sequence

{

private:

int seq[SEQLENGTH];//请求页号序列

int totalpagenum;//请求序列长度

public:

sequence()

{

totalpagenum = 9;

srand(int(time(0)));//get the seed ready

for(int i = 0;i < SEQLENGTH;i++)

{

seq[i]=(rand() % totalpagenum + 1);

}

}

int \*g()

{

return seq;

}

int tpn()//返回请求序列长度

{

return totalpagenum;

}

};

class list

{

private:

int page;//储存的页号

list \*pre;//前一个块

list \*next;//n后一个块

public:

list()

{

page = 0;

}

void init\_set(list \*pre\_,list \*next\_)

{

pre = pre\_;

next = next\_;

}

void set(int i)

{

page = i;

}

void append(list \*block)//连接到尾指针

{

next = block;

}

void frontie(list \*block)//连接到头指针

{

pre = block;

}

int g()//g返回块中储存的页号

{

return page;

}

list \*g\_next()//返回下一个块的指针

{

return next;

}

};

class stack

{

private:

int s[PAGENUM];

public:

int top;

stack()

{

for(int i = 0;i < PAGENUM;i++)

{

s[i]=0;

}

top = -1;

}

int search(int page)//输入要找的页号，返回储存该页储存的位置，如果没有找到，返回 -1

{

for(int i =0 ;i < top + 1;i++)

{

if(s[i] == page)

{

return i;

}

}

return -1;

}

int move(int from)//将从栈顶到次=此位置储存的页全部往下移一个块

{

int temp = s[from];

for(int i = from + 1;i < PAGENUM;i++)

{

s[i - 1] = s[i];

}

return temp;

}

void push(int page)//将此页放入栈顶

{

if(top < PAGENUM - 1)

{

s[++top] = page;

}

else

{

s[top] = page;

}

}

int \*g()//get the stack

{

return s;

}

};

class LRU

{

private:

stack s;

int replaced;//换出的页信息

int miss\_cnt;//缺页计数器

public:

void print(int i,sequence seq,int flag)//flag==0 miss flag==1 hit

{

fout<<setw(6)<<seq.g()[i];

if(s.g()[0])

{

fout<<setw(11)<<s.g()[0];

}

else

{

fout<<setw(11)<<" ";

}

for(int j=1;j < PAGENUM;j++)

{

if(s.g()[j])

{

fout<<setw(9)<<s.g()[j];

}

else

{

fout<<setw(9)<<" ";

}

}

if(flag)//if hit(no missing page)

{

fout<<setw(13)<<"N";

}

else

{

fout<<setw(13)<<"Y";

fout<<setw(13)<<replaced;

}

fout<<endl;

}

LRU(sequence seq)

{

int temp;

miss\_cnt = 0;

fout<<setw(10)<<"wanted\_page ";

for(int j=0;j < PAGENUM;j++)

{

if( j < 9 )

{

fout<<setw(8)<<"block#"<<j+1;

}

else

{

fout<<setw(7)<<"block#"<<j+1;

}

}

fout<<setw(15)<<"page\_missing?";

fout<<setw(15)<<"replaced\_page\n";

for(int i = 0;i < SEQLENGTH; i++)

{

temp = s.search(seq.g()[i]);//保存要寻找的页的位置

if(temp != -1)//if hit

{

s.move(temp);

s.push(seq.g()[i]);

print(i,seq,1);

}

else//if miss

{

replaced = s.g()[0];//save the stack bottom

if(s.top != PAGENUM -1)

{

replaced = 0;

}

if(s.top == PAGENUM -1)

{

s.move(0);

}

s.push(seq.g()[i]);

print(i,seq,0);

miss\_cnt++;

}

}

fout<<"miss time ="<<miss\_cnt<<endl;

fout<<"miss rate ="<<(((float) miss\_cnt)/((float) SEQLENGTH))\*100<<"%\n";

cout<<"miss time ="<<miss\_cnt<<endl;

cout<<"miss rate ="<<(((float) miss\_cnt)/((float) SEQLENGTH))\*100<<"%\n";

}

};

class FIFO

{

private:

list \*front;

list \*rear;

int i;//目前请求的页号

// int log[SEQLENGTH];//page missing log

int miss\_cnt;//缺页信息

int r\_pno;//换出的页号

public:

void print(int flag,sequence s)

{

list \*temp;

temp = front;

fout<<setw(6)<<s.g()[i];

if(temp->g())

{

fout<<setw(11)<<temp->g();

}

else

{

fout<<setw(11)<<" ";

}

temp = temp->g\_next();

for(int j=0;j < PAGENUM - 1;j++)

{

if(temp->g())

{

fout<<setw(9)<<temp->g();

temp = temp->g\_next();

}

else

{

fout<<setw(9)<<" ";

temp = temp->g\_next();

}

}

if(!flag)//if hit(no missing page)

{

fout<<setw(13)<<"Y";

fout<<setw(13)<<r\_pno;

}

else

{

fout<<setw(13)<<"N";

}

fout<<endl;

}

FIFO(sequence s)

{

miss\_cnt = 0;//init the miss counter

i=0;//init the current seq\_no

fout<<setw(10)<<"wanted\_page ";

for(int j=0;j < PAGENUM;j++)

{

if( j < 9 )

{

fout<<setw(8)<<"block#"<<j+1;

}

else

{

fout<<setw(7)<<"block#"<<j+1;

}

}

fout<<setw(15)<<"page\_missing?";

fout<<setw(15)<<"replaced\_page\n";

int f\_hit=0;//hitted-or-not flag

list \*temp;

front = new list;

front->init\_set(front,front);//init the first list

rear = front;

for(int j = 1;j < PAGENUM;j++)//create a linklist

{

temp = new list;

front->frontie(temp);

rear->append(temp);

temp->init\_set(rear,front);

rear = temp;

}

for(i=0;i < SEQLENGTH;i++)//i == 目前进行到请求序列的第几个

{

temp = front;

f\_hit = 0;//0:miss

for(int j=0;j < PAGENUM;j++)//寻找请求的页的位置

{

if(temp->g() == s.g()[i])//if hit

{

f\_hit = 1;

break;

}

temp = temp->g\_next();

}

if(f\_hit == 1)///if hit

{

print(f\_hit,s);

}

else

{

miss\_cnt++;

r\_pno = front->g();

front->set(s.g()[i]);

front = front->g\_next();

rear = rear->g\_next();

print(f\_hit,s);

}

}

cout<<"miss time ="<<miss\_cnt<<endl;

cout<<"miss rate ="<<(((float) miss\_cnt)/((float) SEQLENGTH))\*100<<"%\n";

fout<<"miss time ="<<miss\_cnt<<endl;

fout<<"miss rate ="<<(((float) miss\_cnt)/((float) SEQLENGTH))\*100<<"%\n";

}

~FIFO()

{

list \*temp;

if(front->g\_next())

{

temp = front->g\_next();

delete front;

front = temp;

}

}

};

int main()

{

sequence s;

cout<<"FIFO:\n";

fout<<"FIFO:\n";

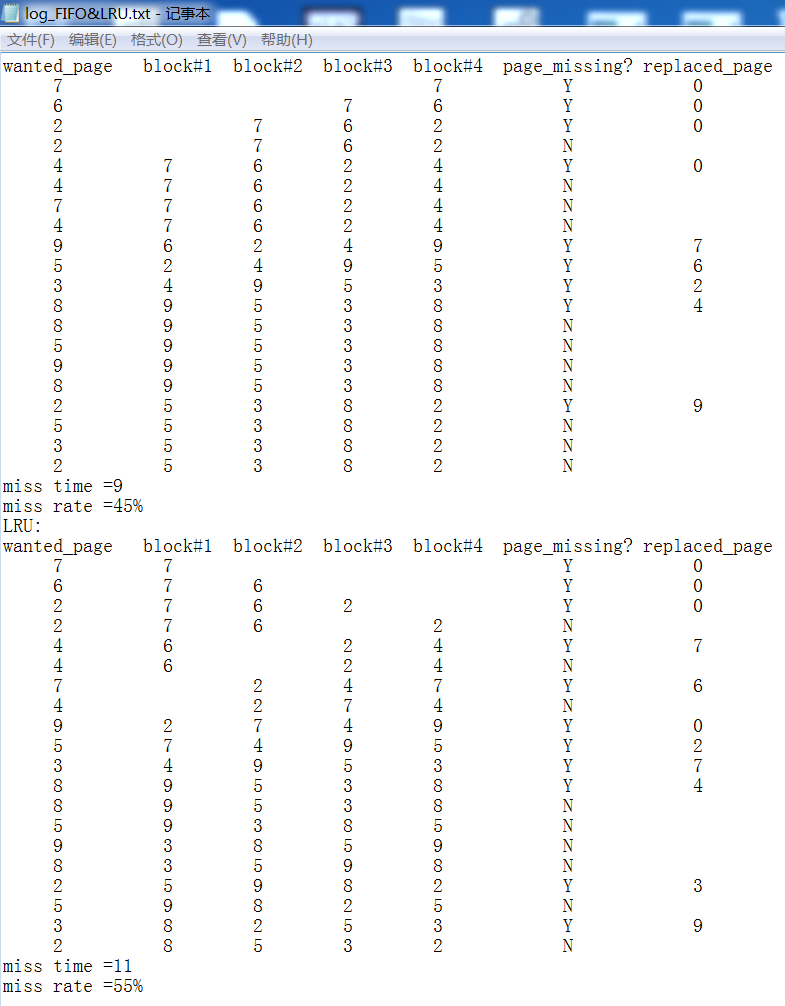
FIFO f(s);

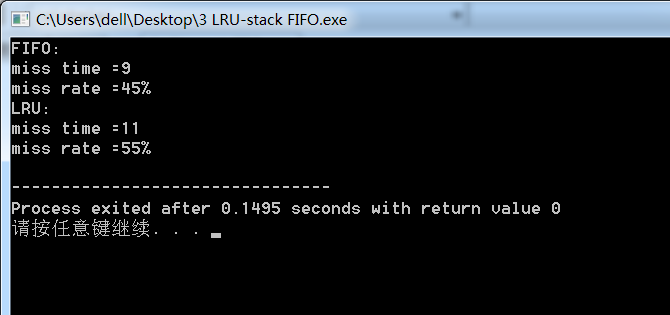
cout<<"LRU:\n";

fout<<"LRU:\n";

LRU l(s);

}





**4.** **磁盘移臂调度算法实验（20 分）**

4.1 实验目的

加深对于操作系统设备管理技术的了解，体验磁盘移臂调度算法的重要性；掌握几种重要的磁盘移臂调度算法，练习模拟算法的编程技巧，锻炼研究分析试验数据的能力。

4.2 实验说明

1. 示例实验程序中模拟两种磁盘移臂调度算法：SSTF 算法和 SCAN算法

2. 能对两种算法给定任意序列不同的磁盘请求序列，显示响应磁盘请求的过程。

3. 能统计和报告不同算法情况下响应请求的顺序、移臂的总量。

#include <iostream>

#include <graphics.h>

#include <ctime>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include <iomanip>

#define BNUM 70 //磁盘所拥有的磁道数

#define BLENGTH 30 //请求序列长度

#define PACE 20 //两个磁道之间的距离

using namespace std;

ofstream fout("log\_Hardrive.txt");

class sequence

{

private:

int seq[BLENGTH];

public:

int \*g()

{

return seq;

}

int g(int i)

{

return seq[i];

}

sequence()

{

srand(int(time(0)));//get the seed ready

for(int i = 0;i < BLENGTH;i++)

{

seq[i]=rand() % BNUM;

}

}

};

class SCAN

{

private:

int now;//目前磁头所在磁道位置

int close;//下一个要访问的磁道

int x\_n;//目前绘图位置x坐标

int x\_c;//下一个点的x坐标

int y;//y坐标

int log[BNUM];//在本序列中要被访问多少次

int seq[BLENGTH];//磁头访问顺序

int cnt;//磁头总移动距离

int dir;//磁头目前移动方向，0：向右 1：向左

public:

void draw()//画磁头运行轨迹

{

x\_n = now \* PACE + PACE;

x\_c = close \* PACE + PACE;

setfillcolor(LIGHTGRAY);

fillellipse(x\_n,y+2,3,3);

setlinewidth(10);

line(x\_n,y,x\_c,y + PACE);

fillellipse(x\_c,y + PACE+2,3,3);

y += PACE;

delay\_ms(100);//延迟

}

void search()//搜索并设定下一个要访问的磁道（找出距磁头最近并且在访问队列中存在的磁道）

{

for(int i = 0 ; i < BNUM; i++)//顺序扫描离磁头从近到远的磁道是否要被访问

{

if((now + i) == BNUM - 1 && log[now + i] == 0 && dir == 0)//if 检测到最后一个磁道 && 最后一个磁道都没有访问记录 && 磁头向右移动

{

dir = 1;//改变磁头移动方向

}

if((now - i) == 0 && log[now - i] == 0 && dir == 1)//if 检测到第一个磁道 && 第一个磁道都没有访问记录 && 磁头向左移动

{

dir = 0;//改变磁头移动方向

}

if(log[now + i] != 0 && (now + i) < BNUM && dir == 0)//如果磁头之后 i 个磁道要被访问 && 没有向右越界 && 磁头移动方向是向右，记录磁道位置并且结束循环

{

close = now + i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

if(log[now - i] != 0 && (now - i) >= 0 && dir == 1)//如果磁头之前 i 个磁道~~~~~~~~ && 没有向左越界 && 磁头移动方向是向左，记录~~~~~~~~~~

{

close = now - i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

}

}

void print(int i)

{

fout<<setw(8)<<log[0];

for(int j=1;j < BNUM;j++)

{

fout<<setw(11)<<log[j];

}

fout<<setw(16)<<now;fout<<setw(12)<<close;

fout<<endl;

}

SCAN(sequence s,int init)

{

dir = 0;//初始时假设磁头向右移动

now = init;

//now = rand() % BNUM;//随机指派磁头位置

cout<<"head position: "<<now<<endl;

fout<<"head position: "<<now<<endl;

fout<<setw(10)<<"wanted\_block seq:";

cout<<setw(10)<<"wanted\_block seq:";

for(int i = 0; i < BLENGTH ; i++)

{

fout<<s.g(i)<<" ";

cout<<s.g(i)<<" ";

}

fout<<endl;

cout<<endl;

for(int j=0;j < BNUM;j++)

{

fout<<setw(10)<<"block#"<<j;

}

fout<<setw(17)<<"current\_block"<<setw(15)<<"nearest\_block\n";

for(int i = 0 ; i < BNUM ; i++)//init the log

{

log[i] = 0;

}

for(int i = 0;i < BLENGTH - 1;i++)

{

seq[i] = 0;

}

x\_c = 0;

x\_n = 0;

close = 0;

cnt = 0;

y = PACE;//init the var

for(int i = 0 ; i < BLENGTH ; i++)//计算每个磁道在这个序列中总共要被访问多少次

{

log[s.g(i)]++;

}

fout<<setw(8)<<log[0];//显示总的访问情况

for(int j=1;j < BNUM;j++)

{

fout<<setw(11)<<log[j];

}

fout<<endl;

for(int i = 0 ;i < BLENGTH ; i++)//start the SSTF

{

if(i == 0)//如果是第一次运行，查找最近的请求磁道并且设定磁头运动方向

{

for(int i = 0 ; i < BNUM; i++)//顺序扫描离磁头从近到远的磁道是否要被访问

{

if(log[now + i] != 0 && (now + i) < BNUM)//如果磁头之后 i 个磁道要被访问，记录磁道位置并且结束循环

{

dir = 0;

close = now + i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

if(log[now - i] != 0 && (now - i) >= 0)//如果磁头之前 i 个磁道~~~~~~~~，记录~~~~~~~~~~

{

dir = 1;

close = now - i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

}

}

else

{

search();//search for the next track

}

print(i);

draw();

now = close;

seq[i] = close;

}

fout<<"access sequence:";

cout<<"access sequence:";

for(int i = 0 ; i < BLENGTH; i++)

{

fout<<seq[i]<<" ";

cout<<seq[i]<<" ";

}

fout<<endl;

cout<<endl;

fout<<"total movement length = "<<cnt<<endl;

cout<<"total movement length = "<<cnt<<endl;

}

};

class SSTF

{

private:

int now;//目前磁头所在磁道位置

int close;//下一个要访问的磁道

int x\_n;//目前绘图位置x坐标

int x\_c;//下一个点的x坐标

int y;//y坐标

int log[BNUM];//在本序列中要被访问多少次

int seq[BLENGTH];//磁头访问顺序

int cnt;//磁头总移动距离

int init;//磁头开始位置

public:

void draw()//绘制磁头运行轨迹

{

x\_n = now \* PACE + PACE;

x\_c = close \* PACE + PACE;

setfillcolor(LIGHTGRAY);

fillellipse(x\_n,y+2,3,3);

setlinewidth(10);

line(x\_n,y,x\_c,y + PACE);

fillellipse(x\_c,y + PACE+2,3,3);

y += PACE;

delay\_ms(100);//延迟

}

int g\_init()//取得磁头开始位置的函数

{

return init;

}

void search()//搜索并设定下一个要访问的磁道（找出距磁头最近并且在访问队列中存在的磁道）

{

for(int i = 0 ; i < BNUM; i++)//顺序扫描离磁头从近到远的磁道是否要被访问

{

if(log[now + i] != 0 && (now + i) < BNUM)//如果磁头之后 i 个磁道要被访问，记录磁道位置并且结束循环

{

close = now + i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

if(log[now - i] != 0 && (now - i) >= 0)//如果磁头之前 i 个磁道~~~~~~~~，记录~~~~~~~~~~

{

close = now - i;

log[close]--;//减去一次访问次数（因为访问即将要被处理)

cnt += i;//累加磁头移动距离

break;

}

}

}

void print(int i)

{

fout<<setw(8)<<log[0];

for(int j=1;j < BNUM;j++)

{

fout<<setw(11)<<log[j];

}

fout<<setw(16)<<now;fout<<setw(12)<<close;

fout<<endl;

}

SSTF(sequence s)

{

now = rand() % BNUM;//随机指派磁头位置

init = now;

cout<<"head position: "<<now<<endl;

fout<<"head position: "<<now<<endl;

fout<<setw(10)<<"wanted\_block seq:";

cout<<setw(10)<<"wanted\_block seq:";

for(int i = 0; i < BLENGTH ; i++)

{

fout<<s.g(i)<<" ";

cout<<s.g(i)<<" ";

}

fout<<endl;

cout<<endl;

for(int j=0;j < BNUM;j++)

{

fout<<setw(10)<<"block#"<<j;

}

fout<<setw(17)<<"current\_block"<<setw(15)<<"nearest\_block\n";

for(int i = 0 ; i < BNUM ; i++)//init the log

{

log[i] = 0;

}

for(int i = 0;i < BLENGTH - 1;i++)

{

seq[i] = 0;

}

x\_c = 0;

x\_n = 0;

close = 0;

cnt = 0;

y = PACE;//init the var

for(int i = 0 ; i < BLENGTH ; i++)//计算每个磁道在这个序列中总共要被访问多少次

{

log[s.g(i)]++;

}

fout<<setw(8)<<log[0];//显示总的访问情况

for(int j=1;j < BNUM;j++)

{

fout<<setw(11)<<log[j];

}

fout<<endl;

for(int i = 0 ;i < BLENGTH ; i++)//start the SSTF

{

search();//search for the next block

print(i);

draw();

now = close;

seq[i] = close;

}

fout<<"access sequence:";

cout<<"access sequence:";

for(int i = 0 ; i < BLENGTH; i++)

{

fout<<seq[i]<<" ";

cout<<seq[i]<<" ";

}

fout<<endl;

cout<<endl;

fout<<"total movement length = "<<cnt<<endl;

cout<<"total movement length = "<<cnt<<endl;

}

};

int main()

{

sequence s;

cout<<"SSTF:\n";

fout<<"SSTF:\n";

initgraph(BNUM \* PACE + PACE \* 2, BLENGTH \* PACE + PACE \* 2);

SSTF s\_1(s);

getch();

closegraph();

cout<<"SCAN:\n";

fout<<"SCAN:\n";

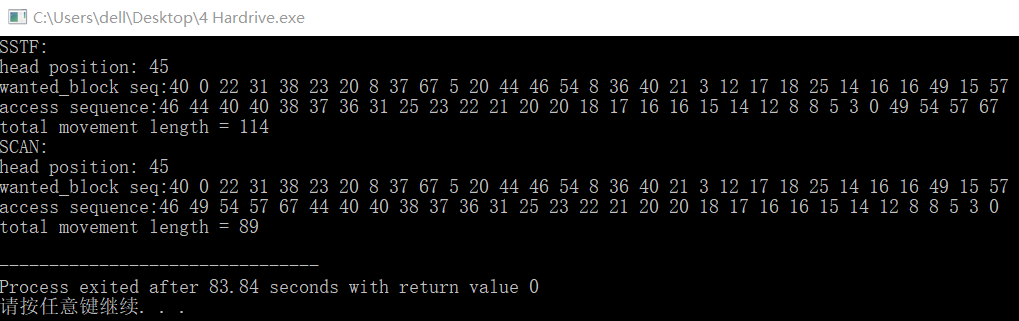
initgraph(BNUM \* PACE + PACE \* 2, BLENGTH \* PACE + PACE \* 2);

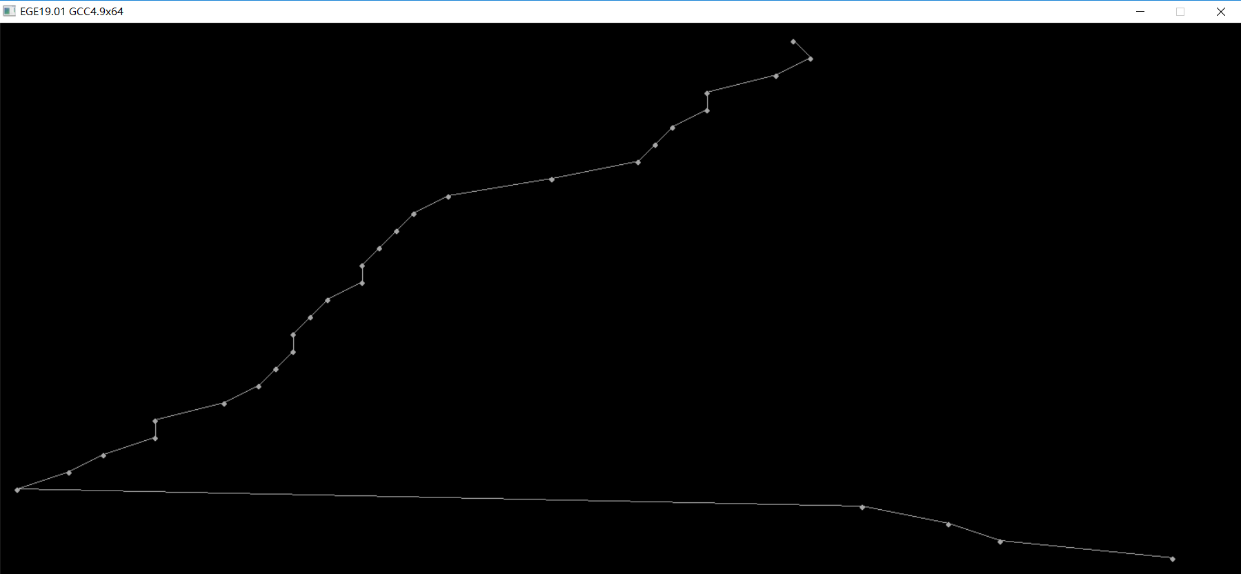
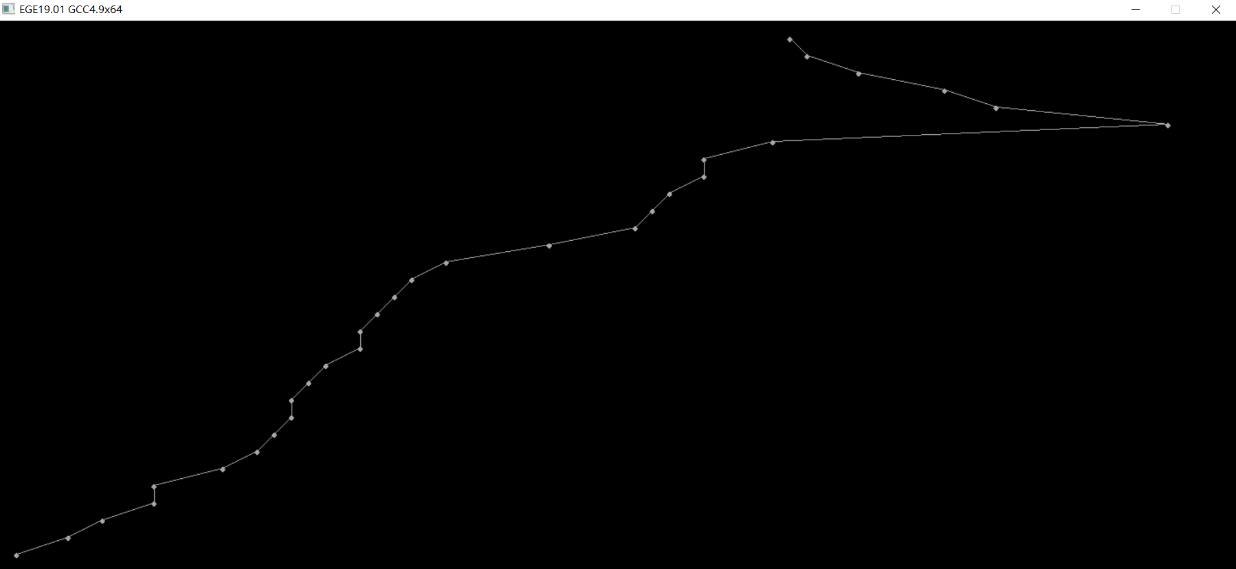
SCAN s\_2(s,s\_1.g\_init());

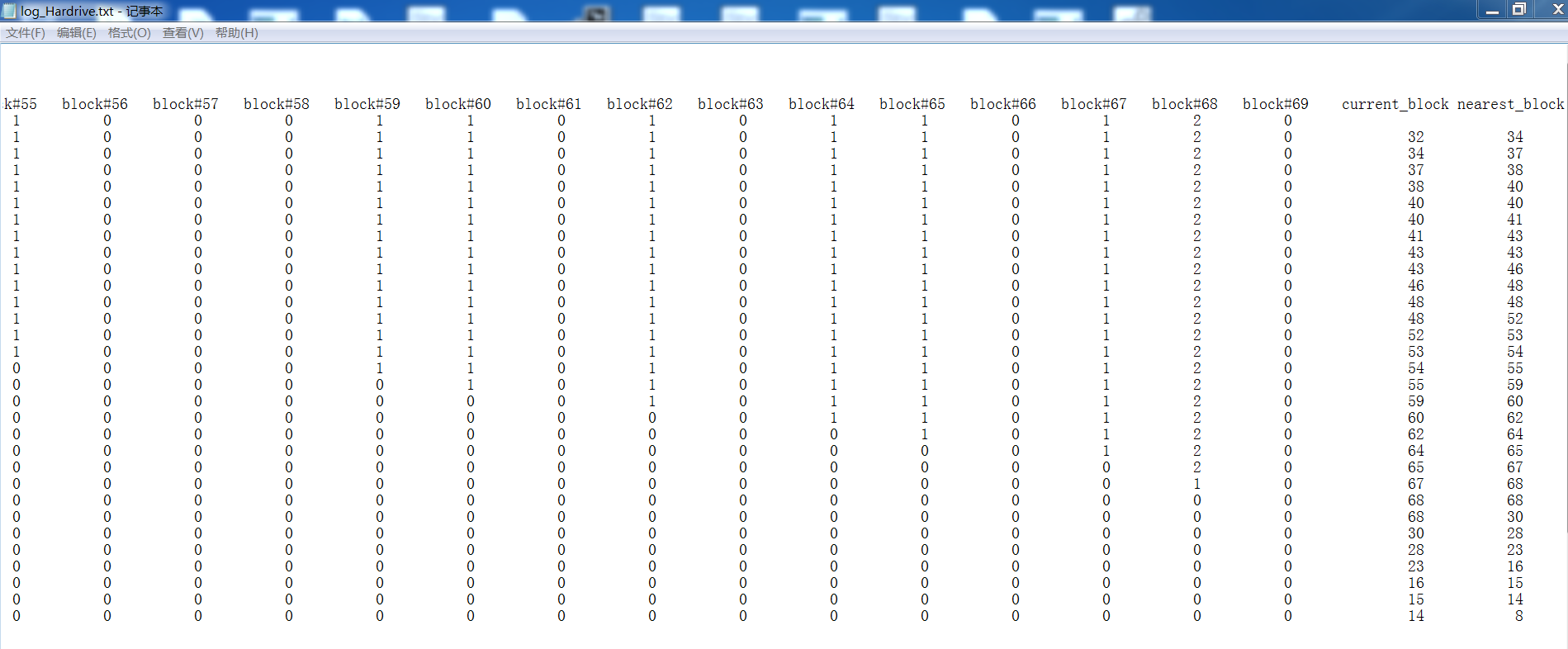
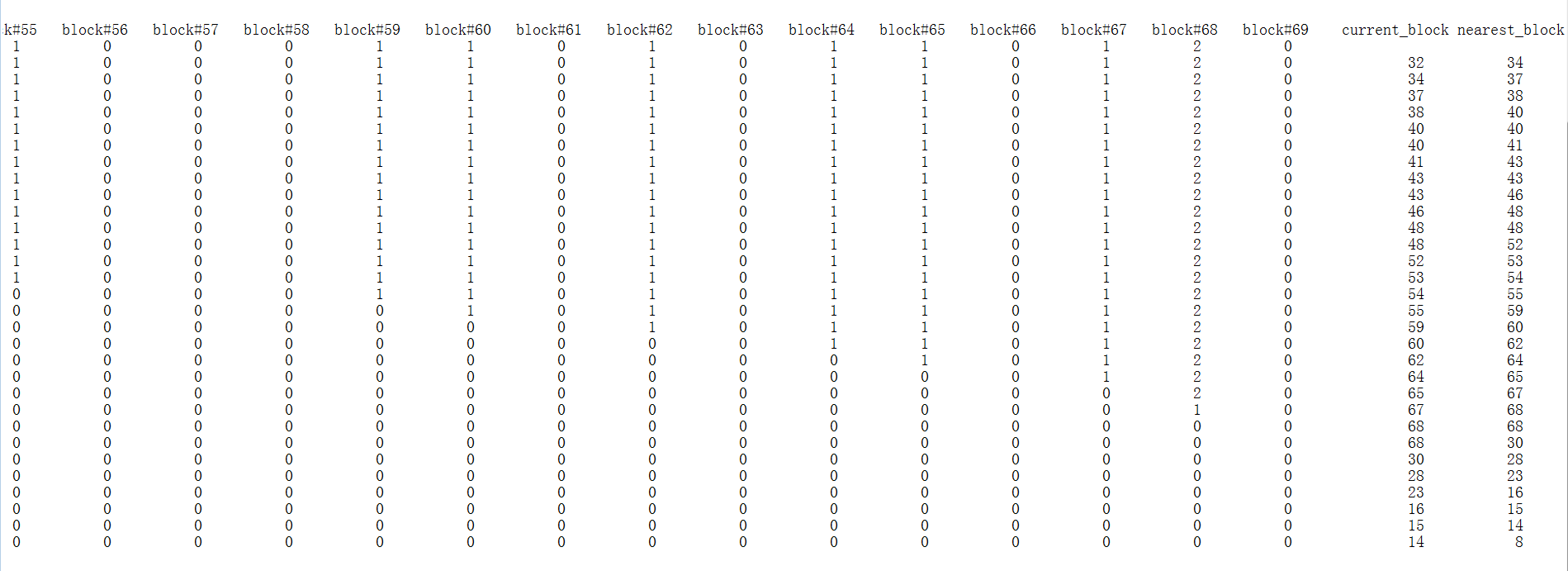
getch();

closegraph();

}







**5. 文件管理作业（20 分）**

5.1 实验目的

通过模拟文件的创建、删除操作，加深对于操作系统文件管理功能的了解，练习模拟算法的编程技巧，锻炼研究分析试验数据的能力。

5.2 实验说明

给出一个磁盘块序列：1、2、3、……、500，初始状态所有块为空的，每块的大小为 2k。选择使用空闲表、空闲盘区链、位示图三种算法之一来管理空闲块。对于基于块的索引分配执行以下步骤：

（1） 随机生成 2k-10k 的文件 50 个，文件名为 1.txt、2.txt、……、50.txt，按照上述算法存储到模拟磁盘中。

（2） 删除奇数.txt（1.txt、3.txt、……、49.txt）文件

（3） 新创建 5 个文件（A.txt、B.txt、C.txt、D.txt、E.txt），大小为：7k、5k、2k、9k、3.5k，按照与（1）相同的算法存储到模拟磁盘中。

（4） 给出文件 A.txt、B.txt、C.txt、D.txt、E.txt 的盘块存储状态和所有空闲区块的状态。

#include <iostream>

#include <windows.h>

#include <fstream>

#include <cstring>

#include <cstdio>

#include <ctime>

#define BLOCK\_SIZE 20 // 每个块2kB

using namespace std;

ofstream fout("log\_doc\_management.txt");

class blocks

{

private:

char path[13];//根目录路径

char temp[13];

char append[5];

char num[4];

bool blank[500];//空白块的位示图

public:

int create\_folder(int i)//创建第 i 号文件夹（模拟块）

{

int flag;

strcpy(temp,path);

itoa(i,num,10);

strcat(temp,num);

flag = CreateDirectory(temp,NULL);

return flag;

}

blocks()

{

CreateDirectory("E:\\Folder\\",NULL);//创建工程需要的根目录

char p[13] = "E:\\Folder\\";

char t[13] = "E:\\Folder\\";

char a[5] = ".txt";

char n[4] = "";

strcpy(path,p);

strcpy(temp,t);

strcpy(append,a);

strcpy(num,n);//初始化类的字符串数组

for(int i = 0 ; i < 500 ; i++)//init the blocks

{

blank[i] = true;//将每个块都设为可用状态

}

cout<<"creating 500 blocks, each volume = 2kB\n"<<endl;

for(int i = 1 ; i <= 500 ; i++ )//创建 500 个块

{

if(!create\_folder(i))//如果创建第 i 个文件夹失败

{

fout<<"creating "<<temp<<" folder error"<<endl;

cout<<"creating "<<temp<<" folder error"<<endl;

exit(1);

}

else

{

fout<<"creating "<<temp<<" folder success"<<endl;

}

}

cout<<"creating 500 blocks complete\n"<<endl;

}

char \*g\_blank()//返回一个空闲块号，以字符串的形式

{

char \*dummy = new char[4];//存储暂时存放空闲块号字符串的字符数组

for(int i = 0 ; i < 4 ; i++)

{

dummy[i] = '\0';

}

bool flag = false;//将是否找到空 block 的标志置 false

int current = 0 ;//存储空闲块

for(current = 0 ; current < 500 ; current++ )//搜寻在存储空间中有没有空闲的块

{

if(blank[current])//如果 current 号块中空闲

{

flag = true;//出现空闲块

break;

}

}

if(flag)//如果有空闲块

{

blank[current] = false;//将 current 号块置为不可用

itoa(current+1,dummy,10);//块是从1开始标号

return dummy;

}

else//代表没有空闲的块了

{

cout<<"\n cannot find a blank block!\n"<<endl;

fout<<"\n cannot find a blank block!\n"<<endl;

exit(1);

}

}

bool clear(int i)

{

if(i == -1)

{

cout<<"error in file destruction\n"<<endl;

fout<<"error in file destruction\n"<<endl;

return false;

}

if(!blank[i])

{

blank[i] = true;

return true;

}

else

{

cout<<"\n#"<<i<<" block is already blank!\n"<<endl;

fout<<"\n#"<<i<<" block is already blank!\n"<<endl;

return false;

}

}

void dis()

{

for(int i = 0 ; i < 500 ; i++)

{

cout<<(int)blank[i]<<" ";

}

}

};

class file

{

private:

char \*name; //文件名

char \*path[5]; //文件存在位置，可能存在多个块内所以是指针数组

int size; //文件大小

public:

file(char \*n,int s,char \*p[5])

{

name = n;

size = s;

for(int i = 0 ; i < 5 ; i++)

{

path[i] = p[i];

}

for(int i = 0 ; (path[i] != NULL && i < 5) ; i++)//当路径不为空时继续创建

{

ofstream f(path[i]);//在 path[i] 上创建文件

f<<"this is file "<<name<<endl;

f<<"size = "<<(float) size/10<<"kB"<<endl;

if(i < 4)

{

if(path[i+1])//如果后面还有路径，证明整个文件在下一个块中还有

{

f<<"fraction = #"<<i<<endl;

f<<"remaining size = "<<(float) (size - 20)/10<<" kB"<<endl;

f<<"next fraction ="<<path[i+1];

size -= 20;//下一个文件将少 2kB

}

else

{

if(i == 0)//如果第一次创建文件时后面就没有目录，代表不分片

{

f<<"fraction = no";

}

else//如果不是第一次，代表分片结束

{

f<<"fraction = #"<<i<<endl;

f<<"last fraction"<<endl;

}

}

}

else

{

if(i == 0)//如果第一次创建文件时后面就没有目录，代表不分片

{

f<<"fraction = no";

}

else//如果不是第一次，代表分片结束

{

f<<"fraction = #"<<i<<endl;

f<<"last fraction"<<endl;

}

}

f.close();

}

}

void destruction(blocks &b)

{

char temp[4] = "\0";

int block\_no = -1;

for(int i = 0 ; i < 5 ; i++)

{

if(path[i])//如果有路径信息

{

if(path[i][10] >= '1' && path[i][10] <= '9')//使用自动机来判断字符，并生成此文件的块号

{

temp[0] = path[i][10];

if(path[i][11] >= '0' && path[i][11] <= '9')

{

temp[1] = path[i][11];

if(path[i][12] >= '0' && path[i][12] <= '9')

{

temp[2] = path[i][12];

temp[3] = '\0';

}

}

}

block\_no = atoi(temp);//将块号转换为 int

b.clear(block\_no-1);//将此块状态改写为空

DeleteFile(path[i]);//删除该块下的文件分片

}

}

}

void dis()

{

char temp[4] = "\0";

cout<<"\nname = "<<name<<endl;

cout<<"size = "<<(float)size/10<<"kB"<<endl;

cout<<"stored in block ";

for(int i = 0 ; i < 5 ; i++)

{

temp[0] = '\0';

if(path[i])//如果有路径信息

{

if(path[i][10] >= '1' && path[i][10] <= '9')//使用自动机来判断字符

{

temp[0] = path[i][10];

if(path[i][11] >= '0' && path[i][11] <= '9')

{

temp[1] = path[i][11];

if(path[i][12] >= '0' && path[i][12] <= '9')

{

temp[2] = path[i][12];

temp[3] = '\0';

}

else

{

temp[2] = '\0';

}

}

else

{

temp[1] = '\0';

}

}

cout<<"#"<<temp<<" ";

}

}

cout<<endl;

}

};

class index //所有 FileControlBlock 的索引

{

private:

char \*name[55];//文件名称索引

file \*ind[55];//控制块索引

char path[13];//文件的根目录路径

char append[5];//.txt后缀

char num[4]; //存储文件名的临时变量

char \*temp;//存放路径的临时字符串指针

public:

index()

{

char p[13] = "E:\\Folder\\";

char t[20] = "E:\\Folder\\";

char a[5] = ".txt";

char n[4] = "";

strcpy(path,p);

strcpy(append,a);

strcpy(num,n);//初始化所有路径变量

for(int i = 0 ; i < 55 ; i++)

{

name[i] = NULL;

}

for(int i = 0 ; i < 55 ; i++)

{

ind[i] = NULL;

}

}

int create\_file(char \*ch,blocks &b,int s = 0)

{

int size;

char \*temp\_path[5];//暂时存储文件存放路径

for(int j = 0 ; j < 5 ; j++)

{

temp\_path[j] = NULL;

}

if(s == 0)//如果传入的 size == 0，代表想让程序随机生成一个大小

{

size =rand()%80 + 20;//随机生成 2kB - 10kB 的文件体积

}

else

{

size = s;

}

int i = 0 ;//存放空白索引的位置

for(int j = 0 ; j < 4 ; j++)//将文件名暂时存储

{

num[j] = \*(ch+j);

}

for(i = 0 ; i < 55 ; i++)//寻找空白索引指针的位置

{

if(ind[i] == NULL)//如果找到空白索引指针，那么相对应的文件名也应该是空白的

{

break;

}

}

if(i != 55)//如果找到空白索引指针，创建文件

{

name[i] = ch;//赋文件名

int j = 0;

for(int t\_size = size ; t\_size > 0 ; t\_size-=20)//当文件剩余体积 > 0 时，继续寻找下一个路径

{

temp = new char[20];

\*temp = '\0';

strcpy(temp,path);//初始化中间数组中间变量

strcat(temp,b.g\_blank());//将空白块的路径赋给中间变量

strcat(temp,"\\");

strcat(temp,ch);//将文件名赋给中间变量

strcat(temp,append);

temp\_path[j] = temp;//将生成的单个路径交给存放路径的临时变量

j++;

}

file \*f= new file(ch,size,temp\_path);//创建文件 i节点

ind[i] = f;//存储文件 i节点至索引

fout<<"create file "<<ch<<".txt success"<<endl;

}

else

{

cout<<"\n cannot find a blank index!\n"<<endl;

cout<<"create file "<<ch<<".txt failed"<<endl;

fout<<"\n cannot find a blank index!\n"<<endl;

fout<<"create file "<<ch<<".txt failed"<<endl;

exit(1);

}

}

bool delete\_file(char \*s,blocks &b)

{

int i = 0;

for(i = 0 ; i < 55 ; i++)//循环查找与要删除的文件名所在的位置

{

if(name[i])//if(name[i] != NULL)

{

if(!strcmp(name[i],s))//if name[i] == s

{

ind[i]->destruction(b);//删除文件

delete[] ind[i];//删除文件 i\_node

ind[i] = NULL;//索引置空

name[i] = NULL;

break;

}

}

}

if(i == 55)//如果循环被完整执行 55 次，代表没有找到相应文件名的文件，抛出错误信息

{

cout<<"\ncannot find file "<<s<<endl<<endl;

fout<<"\ncannot find file "<<s<<endl<<endl;

return false;

}

else

{

fout<<"delete file "<<s<<".txt success"<<endl;

return true;

}

}

bool dis(char \*s)

{

int i = 0;

for(i = 0 ; i < 55 ; i++)//循环查找文件名所在的位置

{

if(name[i])//if(name[i] != NULL)

{

if(!strcmp(name[i],s))//if name[i] == s

{

ind[i]->dis();

break;

}

}

}

if(i == 55)//如果循环被完整执行 55 次，代表没有找到相应文件名的文件，抛出错误信息

{

cout<<"\ncannot find file "<<s<<endl<<endl;

fout<<"\ncannot find file "<<s<<endl<<endl;

return false;

}

else

{

return true;

}

}

~index()

{

for(int i = 0 ; i < 55 ; i++)

{

if(ind[i])

{

//delete[] name[i]; //这里为啥不行？？？

delete[] ind[i];

}

}

}

};

int main()

{

srand(int(time(0)));//get the seed ready

blocks b;

index j;

cout<<"\ncreating 50 files"<<endl;

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

{

char \*name = new char[3];

for(int k = 0 ; k < 3 ; k++)

{

\*(name + k) = '\0';

}

itoa (i+1,name,10);

j.create\_file(name,b);

}

cout<<"create complete\n"<<endl;

cout<<"delete the odd-number-file"<<endl;

for(int i = 1 ; i <= 49 ; i += 2)

{

char \*name = new char[3];

for(int k = 0 ; k < 3 ; k++)

{

\*(name + k) = '\0';

}

itoa (i,name,10);

j.delete\_file(name,b);

}

cout<<"delete complete\n"<<endl;

cout<<"memory status:"<<endl;

b.dis();

cout<<endl;

cout<<"create A.txt"<<endl;

j.create\_file("A",b,70);

cout<<"create B.txt"<<endl;

j.create\_file("B",b,50);

cout<<"create C.txt"<<endl;

j.create\_file("C",b,20);

cout<<"create D.txt"<<endl;

j.create\_file("D",b,90);

cout<<"create E.txt"<<endl;

j.create\_file("E",b,35);

j.dis("A");

j.dis("B");

j.dis("C");

j.dis("D");

j.dis("E");

cout<<"memory status:"<<endl;

b.dis();

}

