操作系统课程设计

计算机科学与技术

姓名：李盛兰

学号：2015014378

目录

一、进程调度模拟程序设计（进程管理）

1.1 题目简介

1.2 所用数据结构

1.3 程序代码描述

1.4 运行结果

二、模拟实现动态分区存储管理（存储管理）

2.1 题目简介

2.2 所用数据结构

2.3 程序代码描述

2.4 运行结果

三、分页管理方式下存储分配情况模拟（存储管理）

3.1 题目简介

3.2 所用数据结构

3.3 程序代码描述

3.4 运行结果

四、页面置换模拟程序设计（存储管理）

4.1 题目简介

4.2 所用数据结构

4.3 程序代码描述

4.4 运行结果

五、个人总结

一、进程调度模拟程序设计

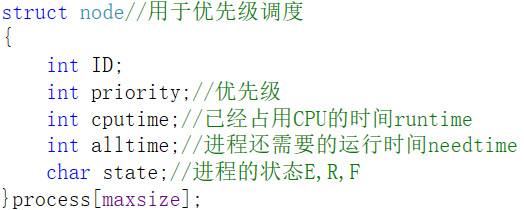
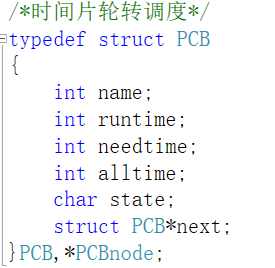
1.1 题目简介

用C语言设计一个对n个并发进程进行调度的程序，每个进程由一个进程控制块（PCB）结构表示，调度程序应当包含2种不同的调度算法，运行时可以任选一种，以利于各种方法的分析和比较，程序应能显示或打印各种进程状态和参数变化情况，便于观察。即要显示每个时间片内各进程的情况，并且指出运行进程及就绪和阻塞队列中的内容。

1.2 所用数据结构

调度程度包含两种不同的调度算法，所以定义了两种不同的PCB节点，一个用于动态优先级调度，一个用于时间片轮转调度。定义如下：

其中时间片轮转调度使用了链表，动态优先级调度使用了线性表。



1.3 程序代码描述

时间片轮转调度所用函数：

void creatpcb()//用来创建进程

void showpcb()//用来显示进程情况

void runpcb()//运行进程

void RRschedule()//进程调度

核心程序

void runpcb()

{

head->runtime++;//每次运行头结点的进程，然后把此进程放到队尾，更换头结点

head->needtime--;

head->state = 'E';

if (head->runtime == head->alltime)

{

cnt--;

cout << "进程" << head->name << "已经运行完成!" << endl;

cout << endl;

system("pause");

head = head->next;

if (head == NULL)

{

cout << "所有进程都运行完成！" << endl;

cout << endl;

system("pause");

return;

}

head->state = 'E';

}

system("cls");

PCBnode x;//指向头指针的下一个节点

x = head->next;

cout << "进程" << head->name << "正在运行" << endl;

cout << "在就绪队列中的进程是：";

while (x != NULL)

{

cout << "进程" << x->name;

x = x->next;

}

cout << endl;

showpcb();

PCBnode y;

y = head;

while (y->next != NULL)

{

y = y->next;

}

head->state = 'R';

y->next = head;

head = head->next;//把刚运行完的头结点放到队尾

y->next->next = NULL;

system("pause");

}

动态优先级调度所用函数

注：在写优先级调度时忽视了一个问题，就是当优先级小于所需运行时间时，此时优先级已经减为负值了，此时就要去查看needtime。

void output()//创建进程

void input()//显示进程当前状态

void runprocess()//运行进程

void DPschedule()//进程调度

核心程序：

void DPschedule()

{

input();

cout << "进程调度开始!" << endl;

cout << endl;

int cnt = 0;

int i;

char ch;

while (cnt <= n)

{

ch = getchar();

if (process[0].alltime > 0)

{

runprocess();

}

else

{

process[0].priority = -1;//撤销该进程

process[0].state = 'F';

cout << "进程" << process[0].ID << "已经运行完成" << endl;

sort(process, process + n, cmp);//按优先级排序

runprocess();

}

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

{

if (process[i].alltime <= 0)

{

process[i].priority = -1;

process[i].state = 'F';

cout << "进程" << process[i].ID << "已经运行完成" << endl;

cnt++;

}

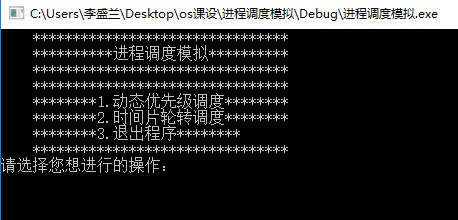
}

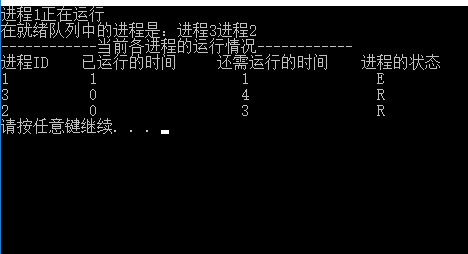
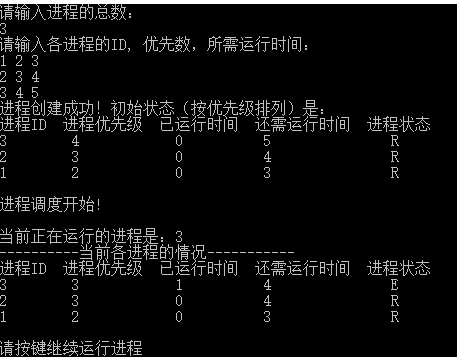
cout << "请按键继续运行进程" << endl;

}

}

1.4 运行结果

主菜单：

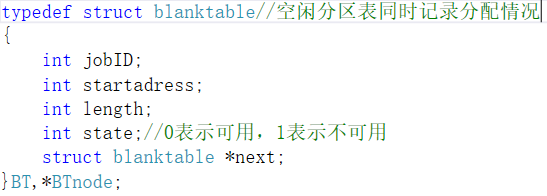
时间片轮转调度和动态优先级调度：

**二、模拟实现动态分区存储管理（存储管理）**

2.1 题目简介

初始状态：动态分区管理方式预先不将主存划分区域。而是把主存除操作系统占用区域外的空间看作一个大的空闲区。当作业要求装入主存时，根据作业的大小查询主存内各空闲区。并按照特定的算法选择一合适的空闲区，按作业大小的要求画出一个分区并装入该作业。剩下的区域作为新的空闲区。

当作业执行完毕后，所占用的主存空间将被回收，成为一个空闲区。注意如果该空闲区的相邻分区也是空闲区，则需要将相邻空闲区合并成一个空闲区。

2.2 所用数据结构

2.3 程序代码描述

void init()//初始化空闲分区表

void firstfit()//首次适应算法分配

void memoryrecycle()//回收内存

void showBT()//展示分配表

核心代码

void firstfit()//首次适应算法

{

BTnode newtask=new BT;

cout << "请输入作业的名称：" << endl;

cin >> newtask->jobID;

cout << "请输入作业所需的内存大小：" << endl;

cin >> newtask->length;

BTnode p = new BT;

p = head;

while (p->length < newtask->length || p->state == 1)

{

if (p->next != NULL)

p = p->next;//一直找到满足的空闲分区

else

{

p = p->next;

break;

}

}

if (p == NULL)//一直找到了最后

cout << "没有适合的空闲分区，分配内存失败！" << endl;

else

{

BTnode q;//找到了空闲分区，然后申请一个新的结点，分配空间和空闲区

q = new BT;

q->jobID = 0;

q->startadress = p->startadress + newtask->length;

q->length = p->length-newtask->length;

q->state = 0;

q->next = p->next;

p->state = 1;

p->jobID = newtask->jobID;

p->length = newtask->length;

p->next = q;

cout << "分配内存成功！" << endl;

}

}

回收算法：

if (rear != NULL&&front != NULL)

{

if (front->state == 0 && rear->state == 0)//前后相邻的两个分区都得空闲的

{

front->length = front->length + present->length + rear->length;

front->next = rear->next;

delete present;

delete rear;

}

else if(front->state==0&&rear->state==1)//前面相邻的分区是空闲的，后面相邻的分区不空闲

{

front->length = front->length + present->length;

front->next = present->next;

delete present;

}

else if (front->state == 1 && rear->state == 0)//前面相邻的分区不是空闲的，后面相邻的分区空闲

{

present->length = present->length + rear->length;

present->jobID = 0;

present->state = 0;

present->next = rear->next;

delete rear;

}

else if(front->state==1&&rear->state==1)//前后相邻的两个分区都是以分配的

{

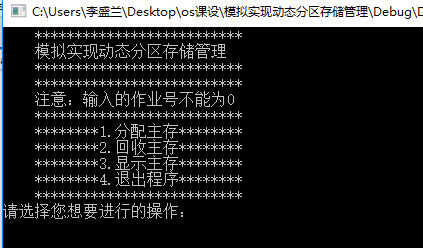
present->jobID = 0;

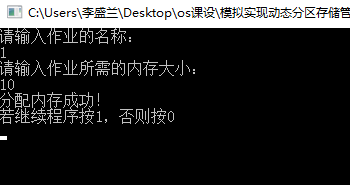
present->state = 0;

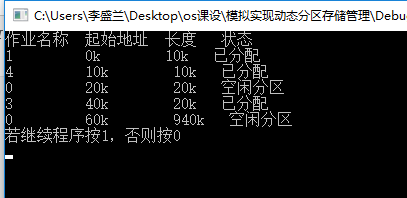
}

}

2.4 运行结果

主菜单：

分配内存：

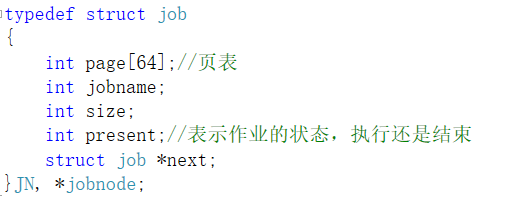
回收主存和显示主存：

**三、分页管理方式下存储分配情况模拟（存储管理）**

3.1题目简介

借用外存空闲分区管理中位示图的方法来表示主存分配情况，实现主存空间的分配和回收

要求能显示和打印分配或回收前后的位示图和当前空闲块数，对完成一次分配后还要显示或打印为作业建立的页表

3.2 所用数据结构

为要分配的作业建立数据结构

3.3 程序代码描述

void init()//初始化作业链

void showbitgraph()//显示位示图

void memoryallocate()//分配内存

void memoryrecycle()//回收内存

void showjob()//显示内存中还有多少作业

void showpagetable()//显示页表

核心代码：

void memoryallocate()//内存分配

{

int i, j, k = 0;

jobnode p = new JN;

jobnode q = head;

cout << "请输入作业的名称：" << endl;

cin >> p->jobname;

cout << "请输入作业所需的内存大小：" << endl;

cin >> p->size;

p->present = 1;

if (p->size > 64 - ::count)

cout << "作业所需的内存大小不能满足，分配失败" << endl;

else

{

int control = 1;

for (i = 0;i < 8 && control;i++)

{

for (j = 0;j < 8 && control;j++)

{

if (bitgraph[i][j] == 0)

{

p->page[k] = 8 \* i + j;

bitgraph[i][j] = 1;

::count++;

k++;

if (k == p->size)

control = 0;

}

}

}

while (q->next != NULL)

{

q = q->next;

}

q->next = p;

p->next = NULL;

}

cout << endl;

showbitgraph();

}

void memoryrecycle()//内存回收

{

int i;

int m, n, symbol = 0;

int id;

cout << "请输入执行结束作业的名称：" << endl;

cin >> id;

jobnode point = head->next;

while (point != NULL)

{

if (id == point->jobname)

{

for (i = 0;i < point->size;i++)

{

m = point->page[i] / 8;

n = point->page[i] % 8;

bitgraph[m][n] = 0;

::count--;

}

point->present = 0;

}

else

{

point->present = 1;

}

point = point->next;

}

jobnode r = head->next;

while (r != NULL)

{

if (r->present == 0)

{

cout << "内存回收成功！" << endl;

showbitgraph();

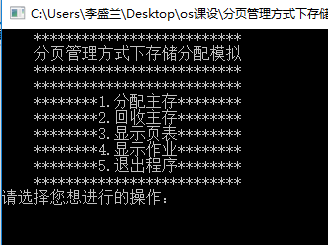
}

r = r->next;

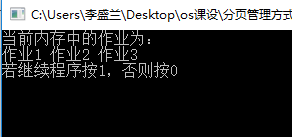
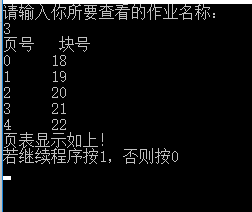
}

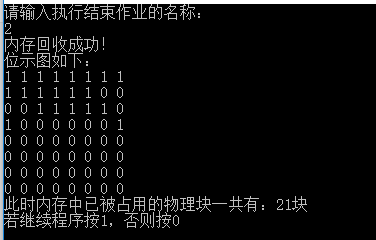
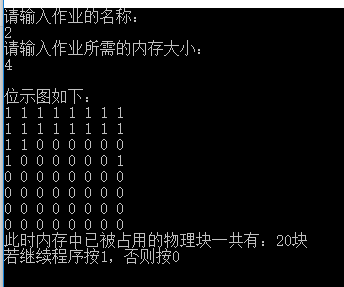
}

3.4 运行结果

主菜单：

分配主存：

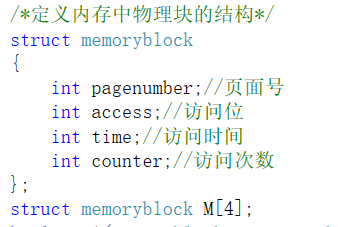
显示作业和页表：

回收主存：

**四、页面置换模拟程序设计（存储管理）**

4.1 题目简介

用C语言设计一个程序，模拟一作业的执行过程。设该作业共有320条指令，即它的地址空间为32页，目前它的所有页面都还未调入内存。在模拟过程中，如果所访问的指令已经在内存，则显示其物理地址，并转下一条指令。如果所访问的指令尚未装入内存，则发生缺页，此时需记录缺页的次数，并将相应页调入内存。如果4个内存块中均已装入该作业的虚页面，则需进行页面置换。最后显示其物理地址，并转下一条指令。在所有320条指令执行完毕后，请计算并显示作业运行过程中发生的缺页率。

4.2 所用数据结构

4.3 程序代码描述

void init()//初始化内存物理块

void random()//生成随机数

void FIFO()//先进先出置换算法

void CLOCK()//时钟置换算法

void LRU()//最近最久未使用置换算法

void LFU()//最近最少访问置换算法

核心代码：

void random()

{

srand((unsigned)time(NULL));

int cnt = 0;

command[cnt] = rand() % 320;

cnt++;

command[cnt] = command[cnt - 1] + 1;

cnt++;

while (cnt < 320)

{

command[cnt] = rand() % command[cnt - 2];

cnt++;

command[cnt] = command[cnt - 1] + 1;

cnt++;

command[cnt] = rand() % (320 - command[cnt - 1]) + command[cnt - 1];

cnt++;

command[cnt] = command[cnt - 1] + 1;

cnt++;

}

}

void oncefifo(int a)

{

int i;

int p=0;//指向队首的指令

int b = a / 10;

int flag=0;

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

{

if (M[i].pagenumber == b)//表示页面在内存中

{

M[i].access = 1;

flag = 1;

break;

}

if (M[i].pagenumber == -1)//表示页面不在内存中，且物理块为空

{

M[i].access = 1;

M[i].pagenumber = b;

p = (p + 1) % 4;

unfindpage++;

flag = 1;

break;

}

}

if (flag == 0)//进行页面置换

{

M[p].pagenumber = b;

M[p].access = 1;

unfindpage++;

}

}

void onceclock(int a)

{

int i;

int p=0;

int b = a / 10;

int flag = 0;

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

{

if (M[i].pagenumber == b)//表示页面在内存中

{

M[i].access = 1;

flag = 1;

break;

}

if (M[i].pagenumber == -1)//表示页面不在内存中，且物理块为空

{

M[i].access = 1;

M[i].pagenumber = b;

p = (p + 1) % 4;

unfindpage++;

flag = 1;

break;

}

}

if (flag == 0)

{

while (M[p].access != 0)

{

M[p].access = 0;

p = (p + 1) % 4;

}

M[p].pagenumber = b;

M[p].access = 1;

p = (p + 1) % 4;

unfindpage++;

}

}

void oncelru(int a)

{

int b = a / 10;

int flag = 0;

int i,j;

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

{

if (M[i].pagenumber == b)//页面在内存中，更新时间

{

M[i].time = 0;

M[i].access = 1;

flag = 1;

for (j = 0;j < 4;j++)

{

if (j != i&&M[j].pagenumber != -1)//其他未被访问的页面的时间加一

{

M[j].time++;

}

}

break;

}

if (M[i].pagenumber == -1)//页面不在内存中，且物理块为空

{

M[i].time = 0;

M[i].pagenumber = b;

M[i].access = 1;

unfindpage++;

flag = 1;

for (j = 0;j < 4;j++)

{

if (j != i&&M[j].pagenumber != -1)

{

M[j].time++;

}

}

break;

}

}

if (flag == 0)//页面不在内存中,进行置换

{

sort(M, M + 4, cmp1);

M[0].time=0;

M[0].access = 1;

M[0].pagenumber = b;

unfindpage++;

for (j = 1;j < 4;j++)

{

M[j].time++;

}

}

}

void oncelfu(int a)

{

int b = a / 10;

int flag = 1;

int i;

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

{

if (M[i].pagenumber == b)//页面在内存中

{

M[i].counter++;

flag = 1;

M[i].access = 1;

break;

}

if (M[i].pagenumber = -1)

{

M[i].counter++;

M[i].pagenumber = b;

M[i].access = 1;

flag = 1;

unfindpage++;

break;

}

}

if (flag == 0)

{

sort(M, M + 4, cmp2);

M[0].pagenumber = b;

M[0].counter=1;

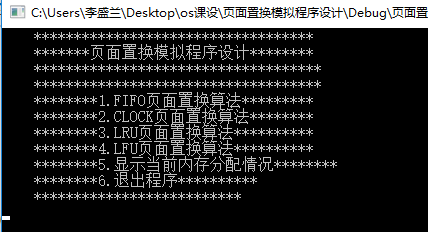
M[i].access = 1;

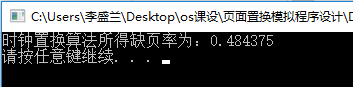
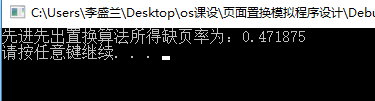
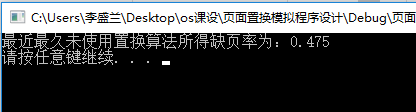
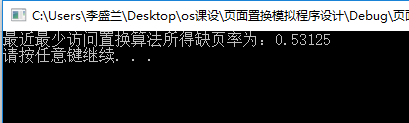
unfindpage++;

}

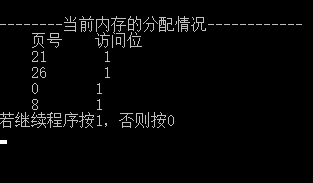
}

4.4 运行结果

主菜单：

选取算法计算缺页率：

显示当前内存分配情况：



五、个人总结

在做这次操作系统课程设计的过程中，主要运用的语言是C语言和C++，初看这些题目的时候，能够理解它们的过程，但是实际编写程序的时候还是无从下手，有的题连数据结构都很难构造，于是又去复习了下数据结构的线性表链表队列和操作系统，然后慢慢分析题目意思，分析程序运行过程，构造数据结构，先构造出主要的函数，然后在编写的过程中出现了很多小的问题，慢慢的去调试，修改，最终完成了后四个题目，感觉写完程序后对书上的理论知识的了解更加深入了。