**实验一 处理器调度**

设计一个按优先级调度算法实现处理器调度的程序。

代码：

#include"pch.h"

#include <iostream>

#include<string>

using namespace std;

struct PCB

{

string name;

int priority;

int time;

//char state = 'R'; // 我觉得不需要这个 直接拿time是否等于零判断进程状态

PCB \*next;

};

PCB\* CreatePCB(string n) {

PCB\* p = new PCB;

p->name=n;

cout << "输入"<<p->name<<"的优先级：";

cin >> p->priority;

cout << "输入预计运行时间：";

cin >> p->time;

return p;

}

//需要传入首结点和要插入的结点，插入的时候按优先级排好

void AddToQueue(PCB\* L,PCB\* p) {

//如果只有首结点，直接放在首结点后面

if (L->next == NULL) {

L->next = p;

p->next = NULL;

return;

}

else {

PCB\* r=L; //工作结点r

//扫描整条链

while (r->next != NULL) {

//如果工作结点r优先级小于要插入的结点p 就把p插入在r前面

if (r->next->priority < p->priority) {

p->next = r->next;

r->next = p;

return;

}

else {

r = r->next;

}

}

//到链尾还比p大，p插在最后

r->next = p;

p->next = NULL;

return;

}

}

void PrintPCB(PCB\* L) {

cout << "当前队列中的进程：" << endl;

PCB\* w = L->next;

if (w != NULL) {

cout << "进程名 优先级 预计运行时间" << endl;

while (w != NULL) {

cout << w->name << "\t " << w->priority << "\t " << w->time << endl;

w = w->next;

}

}

else {

cout << "队列空！" << endl;

}

}

void Run(PCB\* L) {

PCB\* r=L->next;

while(r!=NULL) {

cout << "------进程" << r->name << "运行一次------" << endl;

r->priority--;

r->time--;

//时间为0了退出队列不重新加入队列

if (r->time == 0) {

//L->next->state = 'E';

L->next = r->next;

}

//时间不为0先退出队列再重新加入队列

else {

L->next = r->next;

r->next = NULL;

AddToQueue(L, r);

}

PrintPCB(L);

r = L->next;

}

cout << "-----系统运行结束-----" << endl;

}

int main() {

PCB \*L;

L = new PCB;//初始化头结点

L->next = NULL;

AddToQueue(L,CreatePCB("p1"));

AddToQueue(L,CreatePCB("p2"));

AddToQueue(L,CreatePCB("p3"));

AddToQueue(L, CreatePCB("p4"));

AddToQueue(L, CreatePCB("p5"));

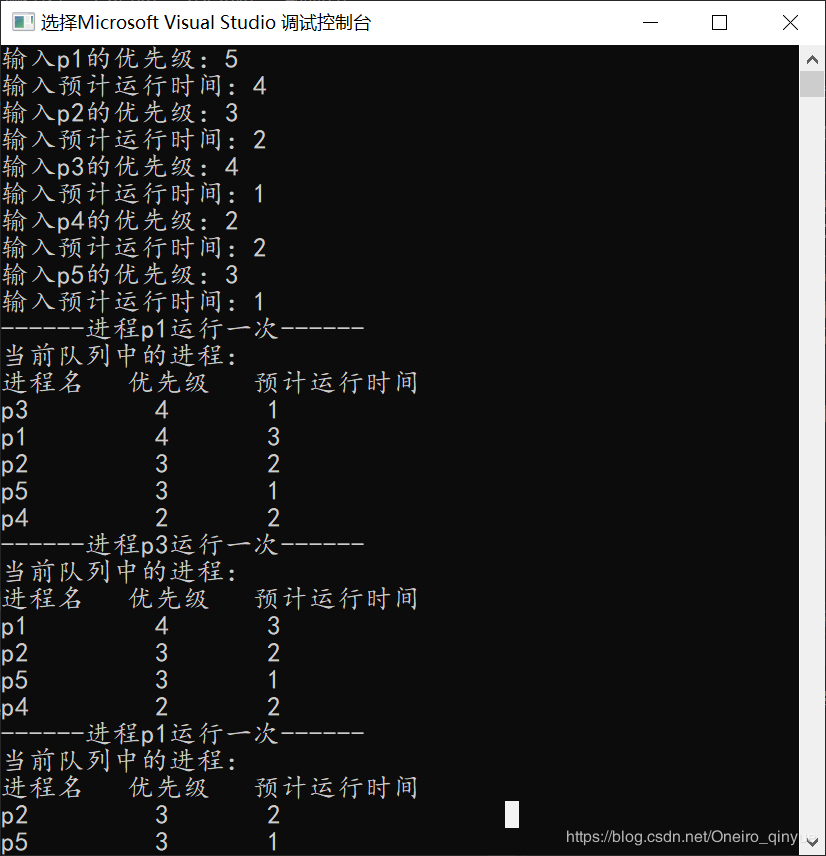
Run(L);

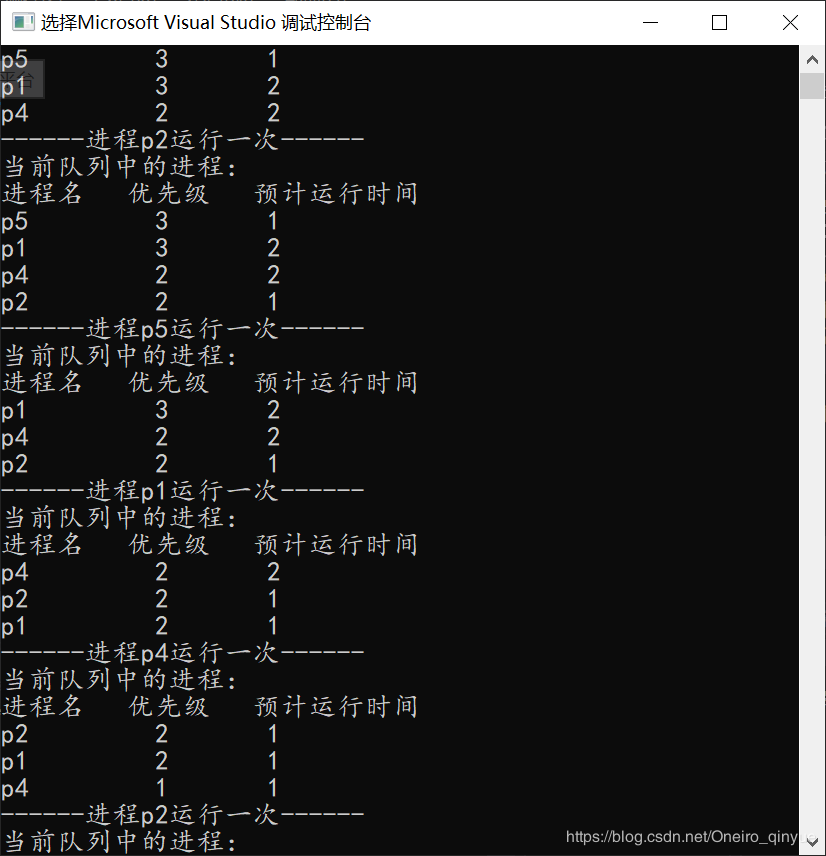
}

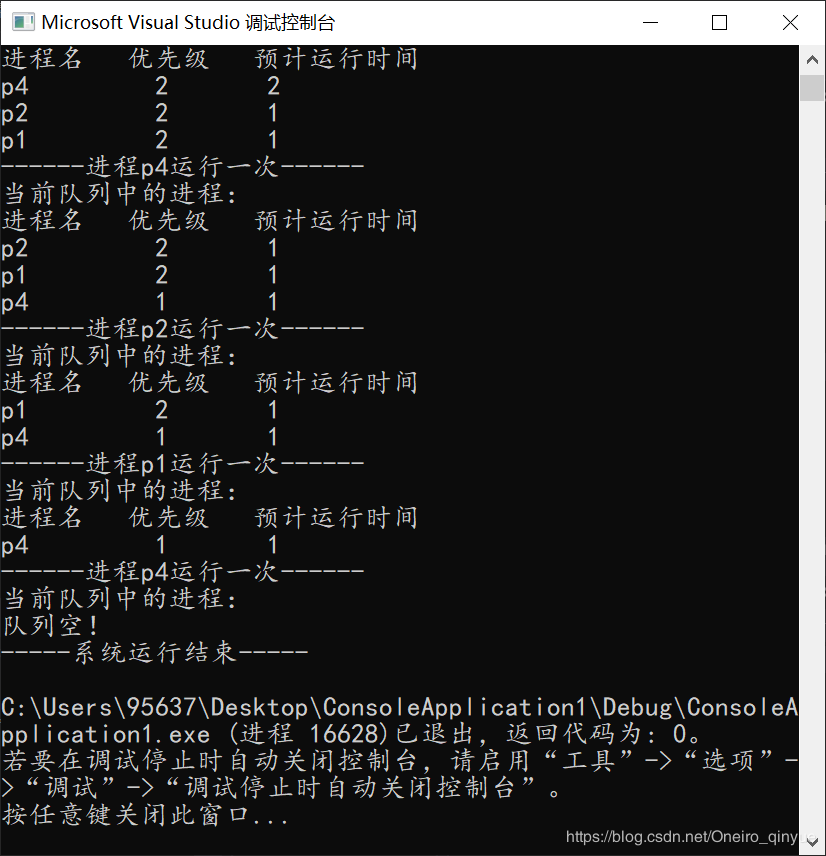
小结：这个程序没显示状态。显示的进程都是就绪进程，运行完直接退出队列了没有打印。

系统可以有的进程数量不固定。数据结构就是单向链表，每次插入都要全部遍历一遍。而且每运行一次都要重新插入。

运行截图：







**实验二 主存空间的分配和回收**

可变分区管理方式下采用首次适应算法（FF）实现主存分配和回收

代码：

#include "pch.h"

#include<iostream>

#include<string>

using namespace std;

int num = 3;//标记内存中载入了几个进程

//随便设内存大小为128好了

int Memory[128] = {

0,0,0,0,0,0,0,0,0,1,

1,1,1,1,1,1,2,2,2,2,

2,2,2,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,3,3,

3,3,3,3,3,3,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0

};

//空闲分区结点FreeBlock

struct FB {

int FBid=0;

int startAddr=0;

int len=0;

FB\* next=NULL;

FB\* before = NULL;

};

FB\* Init() {

FB\* w = new FB;

FB\* h = new FB;

int now=0;

int i=0;

while (now < 128) {

for (now; now < 128; now++) {

if (Memory[now] == 0) {

w->startAddr = now;

break;

}

}

for (now; now < 128; now++) {

if (Memory[now] != 0||now==127) {

w->len = now - (w->startAddr);

w->FBid=i++;

break;

}

}

if (i == 1) {

h = w;

}

FB\* m = new FB;

m->before = w;

w->next = m;

w = m;

now++;

}

return h;

}

void PrintMemory() {

cout << "\n当前内存状态：" << endl;

cout << "----------------------------------" << endl;

cout << "StartAddr Length Status" << endl;

cout << "----------------------------------" << endl;

int now = 0;

int end = 0;

int sAddr;

int status;

while (now < 128) {

if (Memory[now] == 0) {

status = 0;

}

else {

status = Memory[now];

}

now++;

sAddr = now - 1;

for (now; now < 128; now++) {

if (Memory[now] != status) {

end++;

break;

}

end++;

}

string s;

if (status == 0) {

s = "Free";

}

else {

s = "Allocated";

}

cout << sAddr << "\t\t" << end - sAddr << "\t " << s << endl;

}

}

void PrintFreeBlocks(FB\* h) {

cout << "\n空闲的内存块：" << endl;

FB\* w = h;

cout << "----------------------------------" << endl;

cout << "StartAddr Length FreeBlockID" << endl;

cout << "----------------------------------" << endl;

while (w->next != NULL) {

cout << w->startAddr << "\t\t" << w->len << "\t\t" << w->FBid << endl;

w = w->next;

}

}

FB\* Load(int len,FB\* h) {

FB\* w=h;//工作结点,从链首开始

num++;

while(w->next!=NULL){

if (len < w->len) {

for (int i = w->startAddr; i < w->startAddr + len; i++) {

Memory[i] = num;

}

w->startAddr = w->startAddr + len;

w->len = w->len - len;

return h;

}

if (len == w->len) {

for (int i = w->startAddr; i < w->startAddr + len; i++) {

Memory[i] = num;

}

w->next->before = w->before;//删去该空闲块

w = w->next;

h = w;

while (w->next!=NULL) {

w->FBid--;

w = w->next;

}

return h;

}

w = w->next;

}

return h;

}

FB\* Release(int start,FB\* h) {

FB\* w = h;

int i = start;

num--;

int PCBnum=Memory[start];

while (Memory[i] == PCBnum) {

Memory[i] = 0;

i++;//最后i会指向释放完以后的下一个地址

}

//如果释放的内存块前面是空的

if (Memory[start - 1] == 0&&Memory[i]!=0) {

//定位到前面的FB

while (w->next != NULL) {

if ((w->startAddr + w->len) == start ) {

w->len = i - w->startAddr;

return h;

}

w = w->next;

}

}

//如果释放的内存块后面是空的

if (Memory[i] == 0&&Memory[start-1]!=0) {

while (w->next != NULL) {

if (w->startAddr == i) {

w->startAddr = start;

w->len = w->len+i-start;

return h;

}

w = w->next;

}

}

//前后都是空的

if (Memory[start - 1] == 0 && Memory[i] == 0) {

while (w->next != NULL) {

if ((w->startAddr + w->len) == start) {

w->len = w->next->len + w->next->startAddr - w->startAddr;

w->next = w->next->next;//删除释放的内存块后面的那个空闲块

w->next->before = w;

w = w->next;

//更新之后的空闲块ID

while (w->next != NULL) {

w->FBid++;

w = w->next;

}

return h;

}

w = w->next;

}

}

//前后都不是空的

if (Memory[start - 1] != 0 && Memory[i] != 0) {

FB\* nfb = new FB;

nfb->startAddr = start;

nfb->len = i - start;

int temp = start-1;

while (Memory[temp] != 0&& temp!=0) {

temp--;

}

if (temp != 0) {

while (w->next != NULL) {

if ((w->startAddr + w->len) == temp+1) {

nfb->next = w->next;

w->next->before = nfb;

nfb->before = w;

w->next = nfb;

w = nfb;

//更新FBid

while (w->next != NULL) {

w->FBid = w->before->FBid + 1;

w = w->next;

}

return h;

}

w = w->next;

}

}

//要改首结点了呢

else {

nfb->FBid = 0;

nfb->next = h;

h->before = nfb;

h = nfb;

while (w->next != NULL) {

w->FBid++;

w = w->next;

}

return h;

}

}

return h;

}

int main() {

FB\* h = Init();

int c;

while (true) {

PrintMemory();

PrintFreeBlocks(h);

cout << "\n\n分配(1) or 释放(0) or 退出(3)？" << endl;

cin >> c;

if (c == 1) {

cout << "大小？" << endl;

int s;

cin >> s;

h=Load(s,h);

}

else if(c==0) {

cout << "释放内存块的首地址？" << endl;

int s;

cin >> s;

h=Release(s,h);

}

else if (c == 3) {

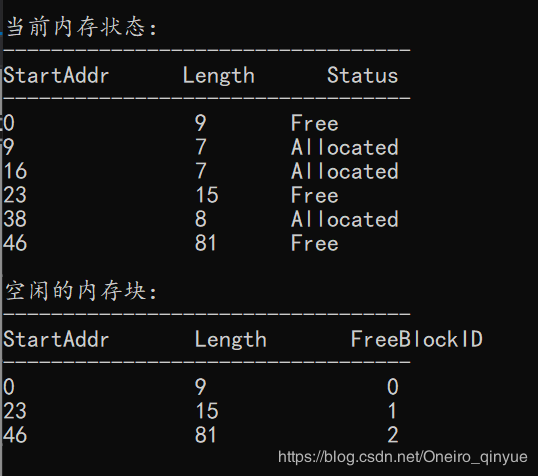
break;

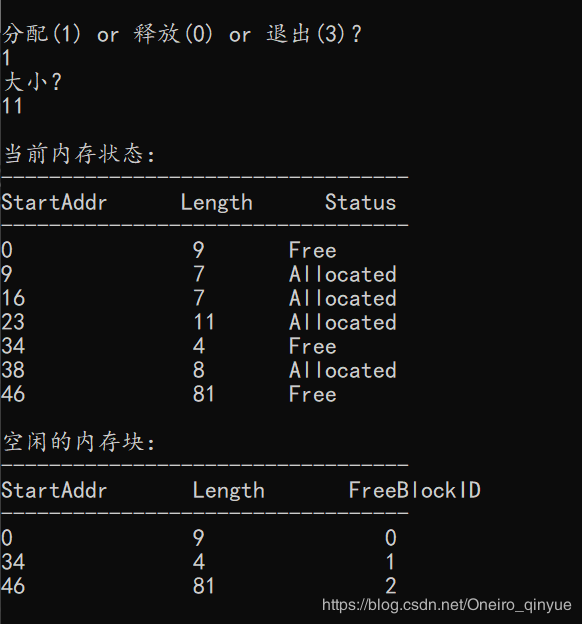
}

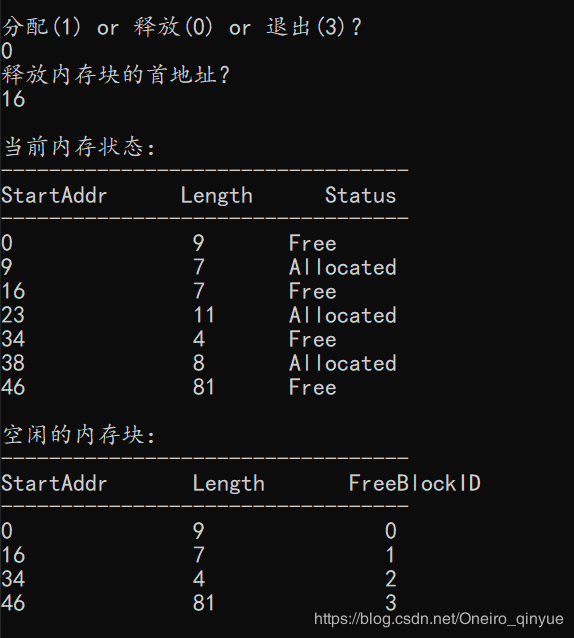
}

}

运行截图：







**实验三 磁盘存储空间的分配和回收**

用位示图管理磁盘存储空间

#include "pch.h"

#include<iostream>

using namespace std;

int BitMap[8][8];

void Print()

{

cout << "-------当前磁盘存储存储位示图-------" << endl;

cout << endl;

for (int i = 0; i < 8; i++)

{

cout << "\t ";

for (int j = 0; j < 8; j++)

{

cout << BitMap[i][j];

}

cout << endl;

}

cout << endl;

cout << endl;

}

void Assign()

{

int request;//待分配的物理块数

int free = 0;

cout << "请输入你要分配的物理块数：";

cin >> request;

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

free++;

if (request <= free)

{

cout << "分配成功！" << endl;

cout << "分配的物理地址为：" << endl;

cout << "柱面号\t" << "磁道号\t" << "物理记录号" << endl;

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

if (request <= 0)

break;

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

{

BitMap[i][j] = 1;

cout << i << "\t" << j / 4 << "\t" << j % 4 << endl;

request--;

}

}

}

}

else

cout << "Sorry,没有足够的物理块进行分配!" << endl;

cout << endl;

Print();

}

void Recovery()

{

int cylinder, track, record;

cout << "请输入你想回收的物理地址：（柱面号、磁道号、物理记录号）" << endl;

cin >> cylinder >> track >> record;

int bytenum, position;

bytenum = cylinder;//字节号

position = track \* 4 + record;//位号

if (bytenum > 7 || position > 7)

{

cout << "输入的物理地址错误!" << endl;

cout << endl;

return;

}

BitMap[bytenum][position] = 0;

cout << "回收成功！" << endl;

cout << "回收字节号为:" << bytenum << " 回收的位号为:" << position << endl;

cout << endl;

Print();

}

int main()

{

for (int i = 0; i < 8; i++)//位示图初始化

{

for (int k = 0; k < 8; k++)

BitMap[i][k] = 0;

}

Print();

int choice;

while (true)

{

cout << "输入你要执行的操作： 1(分配) 2(回收) -1(退出)" << endl;

cin >> choice;

if (choice == 1)

Assign();

else if (choice == 2)

Recovery();

else if (choice == -1)

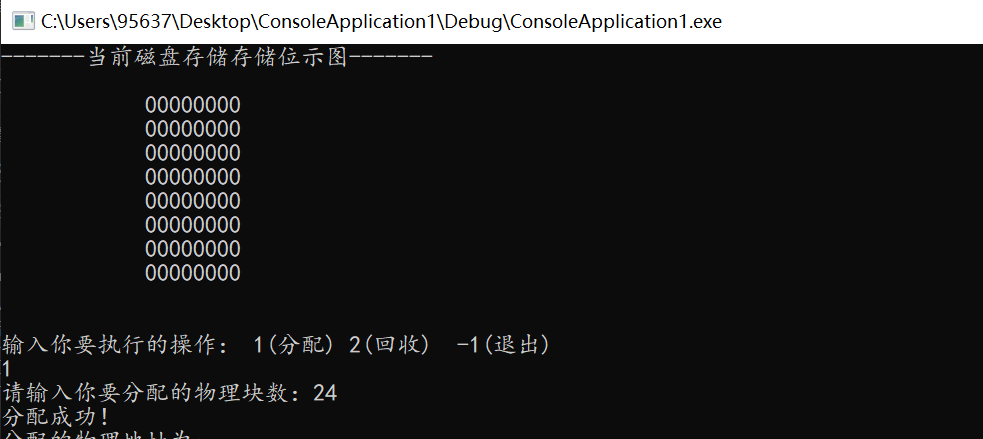
break;

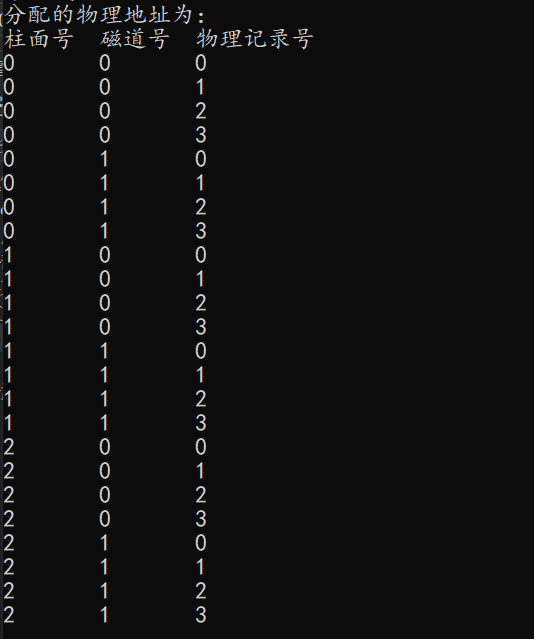
}

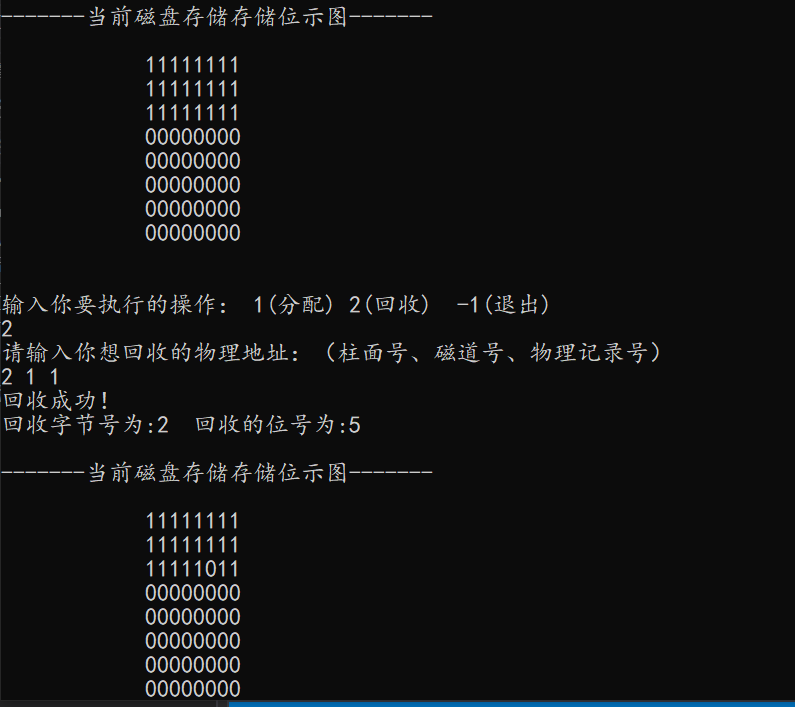
getchar();

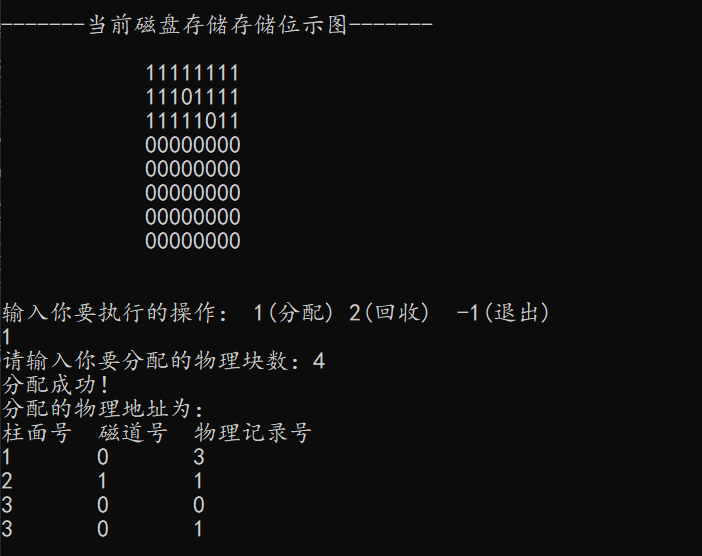
}

实现截图：









**实验四 进程创建**

代码：

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** pidB,pidC;

**if**((pidB=fork())<0){

printf("error in fork\n");

exit(-1);

}

**if**((pidB=fork())==0){

printf("B\n");

}

**if**((pidC=fork())<0){

printf("error in fork\n");

exit(-1);

}

**if**((pidC=fork())==0){

printf("C\n");

}

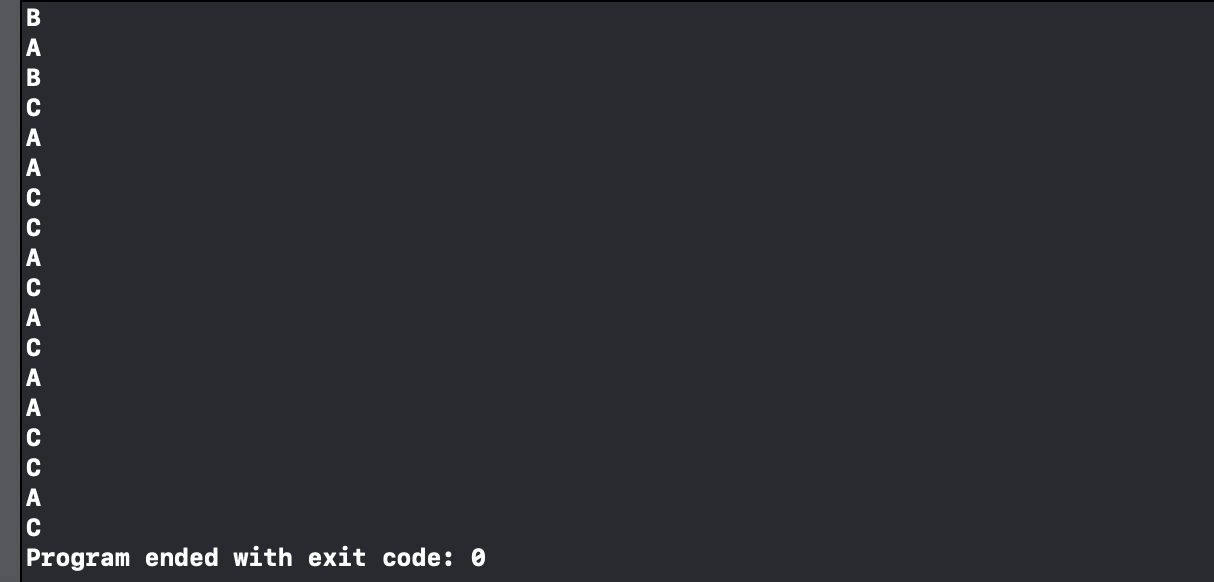
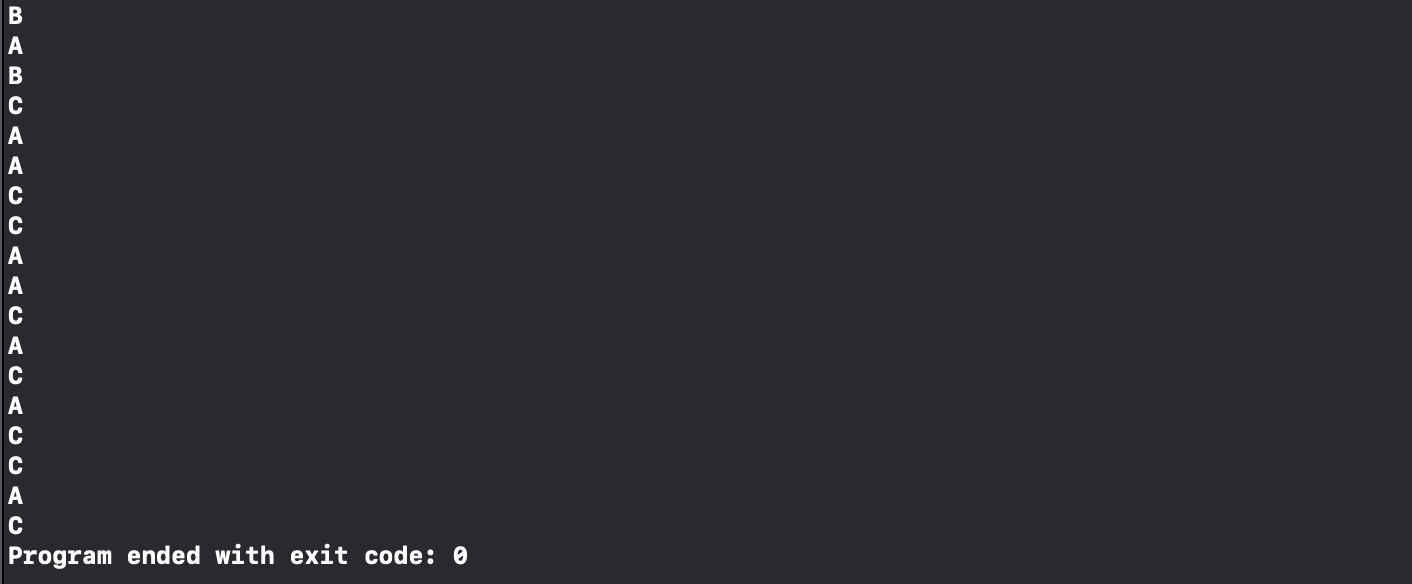
**else**{

printf("A\n");

}

**return** 0;

}

截图：

**实验五 模拟银行家算法**

代码：

#include "pch.h"

#include<stdio.h>

#define resourceNum 3

#define processNum 5

//系统可用（剩余）资源

int available[resourceNum] = { 3,3,2 };

//进程的最大需求

int maxRequest[processNum][resourceNum] = { {7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3} };

//进程已经占有（分配）资源

int allocation[processNum][resourceNum] = { {0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2} };

//进程还需要资源

int need[processNum][resourceNum] = { {7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1} };

//是否安全

bool Finish[processNum];

//安全序列号

int safeSeries[processNum] = { 0,0,0,0,0 };

//进程请求资源量

int request[resourceNum];

//资源数量计数

int num;

//打印输出系统信息

void showInfo()

{

printf("\n------------------------------------------------------------------------------------\n");

printf("当前系统各类资源剩余：");

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

{

printf("%d ", available[j]);

}

printf("\n\n当前系统资源情况：\n");

printf(" PID\t Max\t\tAllocation\t Need\n");

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

{

printf(" P%d\t", i);

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

{

printf("%2d", maxRequest[i][j]);

}

printf("\t\t");

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

{

printf("%2d", allocation[i][j]);

}

printf("\t\t");

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

{

printf("%2d", need[i][j]);

}

printf("\n");

}

}

//打印安全检查信息

void SafeInfo(int \*work, int i)

{

int j;

printf(" P%d\t", i);

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

{

printf("%2d", work[j]);

}

printf("\t\t");

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

{

printf("%2d", allocation[i][j]);

}

printf("\t\t");

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

{

printf("%2d", need[i][j]);

}

printf("\t\t");

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

{

printf("%2d", allocation[i][j] + work[j]);

}

printf("\n");

}

//判断一个进程的资源是否全为零

bool isAllZero(int kang)

{

num = 0;

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

{

if (need[kang][i] == 0)

{

num++;

}

}

if (num == resourceNum)

{

return true;

}

else

{

return false;

}

}

//安全检查

bool isSafe()

{

//int resourceNumFinish = 0;

int safeIndex = 0;

int allFinish = 0;

int work[resourceNum] = { 0 };

int r = 0;

int temp = 0;

int pNum = 0;

//预分配为了保护available[]

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

{

work[i] = available[i];

}

//把未完成进程置为false

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

{

bool result = isAllZero(i);

if (result == true)

{

Finish[i] = true;

allFinish++;

}

else

{

Finish[i] = false;

}

}

//预分配开始

while (allFinish != processNum)

{

num = 0;

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

{

if (need[r][i] <= work[i] && Finish[r] == false)

{

num++;

}

}

if (num == resourceNum)

{

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

{

work[i] = work[i] + allocation[r][i];

}

allFinish++;

SafeInfo(work, r);

safeSeries[safeIndex] = r;

safeIndex++;

Finish[r] = true;

}

r++;//该式必须在此处

if (r >= processNum)

{

r = r % processNum;

if (temp == allFinish)

{

break;

}

temp = allFinish;

}

pNum = allFinish;

}

//判断系统是否安全

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

{

if (Finish[i] == false)

{

printf("\n当前系统不安全！\n\n");

return false;

}

}

//打印安全序列

printf("\n当前系统安全！\n\n安全序列为：");

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

{

bool result = isAllZero(i);

if (result == true)

{

pNum--;

}

}

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

{

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

}

return true;

}

//主函数

int main()

{

int curProcess = 0;

int a = -1;

showInfo();

printf("\n系统安全情况分析\n");

printf(" PID\t Work\t\tAllocation\t Need\t\tWork+Allocation\n");

bool isStart = isSafe();

//用户输入或者预设系统资源分配合理才能继续进行进程分配工作

while (isStart)

{

//限制用户输入，以防用户输入大于进程数量的数字，以及输入其他字符（乱输是不允许的）

do

{

if (curProcess >= processNum || a == 0)

{

printf("\n请不要输入超出进程数量的值或者其他字符：\n");

while (getchar() != '\n') {};//清空缓冲区

a = -1;

}

printf("\n------------------------------------------------------------------------------------\n");

printf("\n输入要分配的进程：");

a = scanf\_s("%d", &curProcess);

printf("\n");

} while (curProcess >= processNum || a == 0);

//限制用户输入，此处只接受数字，以防用户输入其他字符（乱输是不允许的）

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

{

do

{

if (a == 0)

{

printf("\n请不要输入除数字以外的其他字符，请重新输入：\n");

while (getchar() != '\n') {};//清空缓冲区

a = -1;

}

printf("请输入要分配给进程 P%d 的第 %d 类资源：", curProcess, i + 1);

a = scanf\_s("%d", &request[i]);

} while (a == 0);

}

//判断用户输入的分配是否合理，如果合理，开始进行预分配

num = 0;

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

{

if (request[i] <= need[curProcess][i] && request[i] <= available[i])

{

num++;

}

else

{

printf("\n发生错误！可能原因如下：\n(1)您请求分配的资源可能大于该进程的某些资源的最大需要！\n(2)系统所剩的资源已经不足了！\n");

break;

}

}

if (num == resourceNum)

{

num = 0;

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

{

//分配资源

available[j] = available[j] - request[j];

allocation[curProcess][j] = allocation[curProcess][j] + request[j];

need[curProcess][j] = need[curProcess][j] - request[j];

//记录分配以后，是否该进程需要值为0了

if (need[curProcess][j] == 0)

{

num++;

}

}

//如果分配以后出现该进程对所有资源的需求为0了，即刻释放该进程占用资源（视为完成）

if (num == resourceNum)

{

//释放已完成资源

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

{

available[i] = available[i] + allocation[curProcess][i];

}

printf("\n\n本次分配进程 P%d 完成,该进程占用资源全部释放完毕！\n", curProcess);

}

else

{

//资源分配可以不用一次性满足进程需求

printf("\n\n本次分配进程 P%d 未完成！\n", curProcess);

}

showInfo();

printf("\n系统安全情况分析\n");

printf(" PID\t Work\t\tAllocation\t Need\t\tWork+Allocation\n");

//预分配完成以后，判断该系统是否安全，若安全，则可继续进行分配，若不安全，将已经分配的资源换回来

if (!isSafe())

{

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

{

available[j] = available[j] + request[j];

allocation[curProcess][j] = allocation[curProcess][j] - request[j];

need[curProcess][j] = need[curProcess][j] + request[j];

}

printf("资源不足，等待中...\n\n分配失败！\n");

}

}

}

}

实现截图：

