操作系统第六次实验——C语言模拟操作系统内存分配

1. 实验目的
2. 理解并掌握操作系统的内存分配。
3. 掌握首次适应算法、最佳适应算法和最坏适应算法。
4. 实验内容
5. 将老师给的原始代码进行修改和完善
6. 写出最佳适应算法
7. 写出最坏适应算法
8. 实验过程
9. 原始代码的修改和完善
10. 算法思想：

将内存块中的所有的块按照地址递增的顺序连接成一个链表，每次要将新的作业放入内存的时候就按顺序查找内存块链表，每次都是用找到的可以用的第一个内存块。

链表数据结构：

链表结点共有4个区域和一个下指针构成，四个区域分别记录该内存块的起始地址，该内存块长度，内存块的状态和内存块存放的作业编号（没有作业时存放的是0）。

1. 模拟实现的策略：

① 插入操作时依照地址递增的顺序检查可以装入的第一个内存块；

若找到，则装入内存中，并且把内部碎片分出来变成新的结点插入链表中；

若没找到则分配失败；

② 撤销操作时按照作业编号找到要撤销作业所在内存块；

若找到，则将状态置位未分配，此时假如前后有可以合并的空内存块则合并；

若没有找到，则返回没有找到该作业信息；

1. 原始代码如下：

#include <stdio.h>

#include <stdlib.h>

/\*

算法思想：

1.插入操作时依照地址递增的顺序检查可以装入的第一个内存块

若找到，则装入内存中，并且把内部碎片分出来变成新的结点插入链表中

若没找到则分配失败

2.撤销操作时按照作业编号找到要撤销作业所在内存块

若找到，则将状态置位未分配，此时假如前后有可以合并的空内存块则合并

若没有找到，则返回没有找到该作业信息

\*/

//使用链表的数据结构

typedef struct LNode{

int address;//表示该分区的起始地址

int length; //表示该分区的长度

int state; //0表示该分区空闲，1表示该分区已分配

int number; //作业的编号

struct LNode \*next;

}\*pLNode;

//链表的插入操作

//L是链表的头指针，addr是该内存块的首地址，len是该内存块的长度

//sta是内存块的状态，num是将占有内存块作业的编号

void InsertLNode(pLNode L,int addr,int len,int sta,int num) {

pLNode p = (pLNode)malloc(sizeof(LNode));

p->address = addr;

p->length = len;

p->state = sta;

p->number = num;

p->next = NULL;

L->next = p;

}

//初始化链表

//初始化链表的头指针，假设操作系统有8M的内存空间，且始址为0

//leng为内存空间的总大小，单位为M

void InitLNode(pLNode L,int leng){

//L = (pLNode)malloc(sizeof(LNode));

L->address = 0;//从首地址开始

L->length = 8;

L->state = 1;

L->number = 65535;

//插入剩余的空闲分区

InsertLNode(L, 8, leng - 8,0,0);

printf("初始化成功！\n");

}

//撤销作业

//L为链表头指针，num为待撤销的作业编号

//若找到该作业则把其占有的内存块state置0

void Revocation(pLNode L, int num) {

pLNode p = L->next;

if (p->number == num) {

//找到了这个作业

p->state = 0;

printf("成功撤销编号为%d的作业！\n",num);

return;

}

while (p->next != NULL) {

p = p->next;

p->state = 0;

printf("成功撤销编号为%d的作业！\n", num);

return;

}

printf("撤销作业失败，没有找到编号为%d的作业\n", num);

}

//检查链表中是否有连续的未分配内存区，如果有就合并

void Merge\_free(pLNode L) {

pLNode p = L,pre;

int num = 0;//用于检测连续空闲内存空间数量，等于2表示有连续的内存空间

while (p != NULL) {

if (p->state == 0) {

//表示为空闲区

num++;

if (num == 1)

pre = p;//为合并做准备

else if (num == 2) {

//表示有连续的空闲区，执行合并操作

//将p结点的长度加给前驱结点的长度

//前驱结点指向p结点的下结点

//释放p结点

//将p指向前驱结点，方便下次判断

//num置1

pre->length += p->length;

pre->next = p->next;

free(p);

p = pre;

num = 1;

}

}

else if (p->state == 1)

num = 0;

p = p->next;

}

}

//为作业分配内存空间

//L为链表头指针，len为作业需要的内存大小，num为作业编号

void Allocation(pLNode L, int len, int num) {

pLNode p = L->next;

while (p != NULL) {

if (p->state == 0) {

//表示内存块未分配

if (p->length > len) {

//表示可以分配内存空间

//但是需要把内部碎片分出来，构成新的空闲内存块

//新的内存块的起始地址为p->address + len，长度为p->length-len

pLNode l = (pLNode)malloc(sizeof(LNode));

l->length = p->length - len;

l->address = p->address + len;

l->state = 0;

l->number = 0;

l->next = p->next;

p->next = l;

p->length = len;

p->number = num;

p->state = 1;

printf("内存分配成功！\n");

return;

}

else if (p->length == len) {

//刚刚好够分配，只要修改作业编号和内存块状态即可

p->number = num;

p->state = 1;

printf("内存分配成功！\n");

return;

}

}

p = p->next;

}

printf("内存分配失败，没有找到合适的空闲内存块\n");

}

//打印

void print(pLNode L) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("内存分配情况如下：\n");

printf("起始地址\t内存块长度\t存放作业编号\t内存块状态\n");

pLNode p = L;

while (p != NULL) {

printf("%8d\t%10d\t", p->address, p->length);

if (p->number == 65535) {

printf(" 操作系统\t");

}

else if (p->number == 0) {

printf(" 无作业\t");

}

else {

printf("%12d\t", p->number);

}

if (p->state == 0) {

printf(" 空闲\n");

}

else {

printf(" 已分配\n");

}

p = p->next;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//服务选择菜单

int select() {

int a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择服务：\n");

printf("1.为新作业分配内存\t2.撤销作业\n3.查看内存分配情况\t4.退出\n");

printf("请输入编号以选择：");

scanf("%d", &a);

return a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

int main()

{

pLNode L = (pLNode)malloc(sizeof(LNode));

InitLNode(L, 512);

int a;

int len, num;

a = select();

getchar();

while (a != 4) {

switch (a) {

case 1:

printf("请输入要分配内存的作业的长度和编号：");

scanf("%d%d", &len, &num);

Allocation(L, len, num);

break;

case 2:

printf("请输入要撤销作业的编号：");

scanf("%d", &num);

Revocation(L, num);

break;

case 3:

//system("cls");

print(L);

break;

default:

break;

}

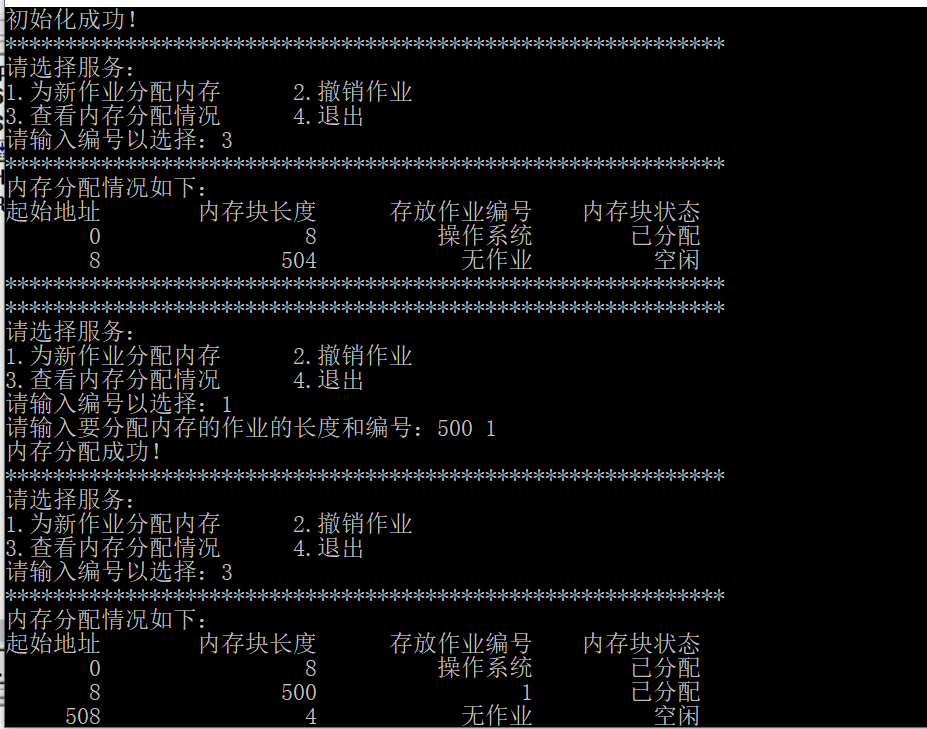
a = select();

}

return 0;

}

1. 原始代码的运行结果



1. 可见原始代码只能实现作业的连续分配，没有实现多个内存块的分配，而是将一整个内存切割，因此对此进行的改进如下：

新增的部分用蓝色标出

#include <stdio.h>

#include <stdlib.h>

/\*

算法思想：

1.插入操作时依照地址递增的顺序检查可以装入的第一个内存块

若找到，则装入内存中，并且把内部碎片分出来变成新的结点插入链表中

若没找到则分配失败

2.撤销操作时按照作业编号找到要撤销作业所在内存块

若找到，则将状态置位未分配，此时假如前后有可以合并的空内存块则合并

若没有找到，则返回没有找到该作业信息

\*/

//使用链表的数据结构

typedef struct LNode{

int address;//表示该分区的起始地址

int length; //表示该分区的长度

int state; //0表示该分区空闲，1表示该分区已分配

int number; //作业的编号

struct LNode \*next;

}\*pLNode;

//链表的插入操作

//L是链表的头指针，addr是该内存块的首地址，len是该内存块的长度

//sta是内存块的状态，num是将占有内存块作业的编号

void InsertLNode(pLNode L,int addr,int len,int sta,int num) {

pLNode p = (pLNode)malloc(sizeof(LNode));

p->address = addr;

p->length = len;

p->state = sta;

p->number = num;

p->next = NULL;

L->next = p;

}

//初始化链表

//初始化链表的头指针，假设操作系统有8M的内存空间，且始址为0

//leng为内存空间的总大小，单位为M

void InitLNode(pLNode L,int leng){

//L = (pLNode)malloc(sizeof(LNode));

L->address = 0;//从首地址开始

L->length = 8;

L->state = 1;

L->number = 65535;

//插入剩余的空闲分区

InsertLNode(L, 8, leng - 8,0,0);

printf("初始化成功！\n");

}

//撤销作业

//L为链表头指针，num为待撤销的作业编号

//若找到该作业则把其占有的内存块state置0

void Revocation(pLNode L, int num) {

pLNode p = L->next;

if (p->number == num) {

//找到了这个作业

p->state = 0;

printf("成功撤销编号为%d的作业！\n",num);

return;

}

while (p->next != NULL) {

p = p->next;

p->state = 0;

printf("成功撤销编号为%d的作业！\n", num);

return;

}

printf("撤销作业失败，没有找到编号为%d的作业\n", num);

}

//检查链表中是否有连续的未分配内存区，如果有就合并

void Merge\_free(pLNode L) {

pLNode p = L,pre;

int num = 0;//用于检测连续空闲内存空间数量，等于2表示有连续的内存空间

while (p != NULL) {

if (p->state == 0) {

//表示为空闲区

num++;

if (num == 1)

pre = p;//为合并做准备

else if (num == 2) {

//表示有连续的空闲区，执行合并操作

//将p结点的长度加给前驱结点的长度

//前驱结点指向p结点的下结点

//释放p结点

//将p指向前驱结点，方便下次判断

//num置1

pre->length += p->length;

pre->next = p->next;

free(p);

p = pre;

num = 1;

}

}

else if (p->state == 1)

num = 0;

p = p->next;

}

}

//为作业分配内存空间

//L为链表头指针，len为作业需要的内存大小，num为作业编号

void Allocation(pLNode L,int len, int num) {

pLNode p = L->next;

while (p != NULL) {

if (p->state == 0) {

//表示内存块未分配

if (p->length > len) {

//表示可以分配内存空间

//但是需要把内部碎片分出来，构成新的空闲内存块

//新的内存块的起始地址为p->address + len，长度为p->length-len

pLNode l = (pLNode)malloc(sizeof(LNode));

l->length = p->length - len;

l->address = p->address + len;

l->state = 0;

l->number = 0;

l->next = p->next;

p->next = l;

p->length = len;

p->number = num;

p->state = 1;

printf("内存分配成功！\n");

return;

}

else if (p->length == len) {

//刚刚好够分配，只要修改作业编号和内存块状态即可

p->number = num;

p->state = 1;

printf("内存分配成功！\n");

return;

}

/\* else if(p->length<len){

Merge\_free(p);

Allocation(p,len,num);

}\*/

}

p = p->next;

}

printf("内存分配失败，没有找到合适的空闲内存块\n");

}

//打印

void print(pLNode L) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("内存分配情况如下：\n");

printf("起始地址\t内存块长度\t存放作业编号\t内存块状态\n");

pLNode p = L;

while (p != NULL) {

printf("%8d\t%10d\t", p->address, p->length);

if (p->number == 65535) {

printf(" 操作系统\t");

}

else if (p->number == 0) {

printf(" 无作业\t");

}

else {

printf("%12d\t", p->number);

}

if (p->state == 0) {

printf(" 空闲\n");

}

else {

printf(" 已分配\n");

}

p = p->next;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//服务选择菜单

int select() {

int a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择服务：\n");

printf("1.为新作业分配内存\t2.撤销作业\n3.查看内存分配情况\t4.退出\t5.分配内存块\n");

printf("请输入编号以选择：");

scanf("%d", &a);

return a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

int main()

{

pLNode L = (pLNode)malloc(sizeof(LNode));

InitLNode(L, 512);

int a;

int len, num,addr,sta;

pLNode p=L->next ;

a = select();

getchar();

while (a != 4) {

switch (a) {

case 1:

printf("请输入要分配内存的作业的长度和编号：");

scanf("%d%d", &len, &num);

Allocation(L, len, num);

break;

case 2:

printf("请输入要撤销作业的编号：");

scanf("%d", &num);

Revocation(L, num);

break;

case 3:

//system("cls");

print(L);

break;

case 5:

printf("请输入要分配的内存块的起始地址、长度、状态、编号：");

scanf("%d%d%d%d",&addr,&len,&sta,&num);

InsertLNode(p,addr,len,sta,num);

p=p->next ;

default:

break;

}

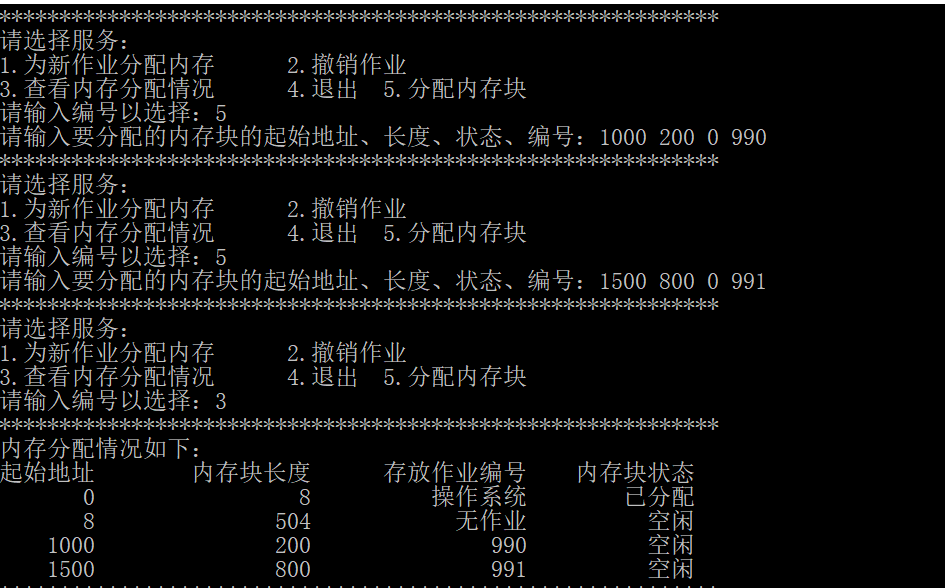
a = select();

}

return 0;

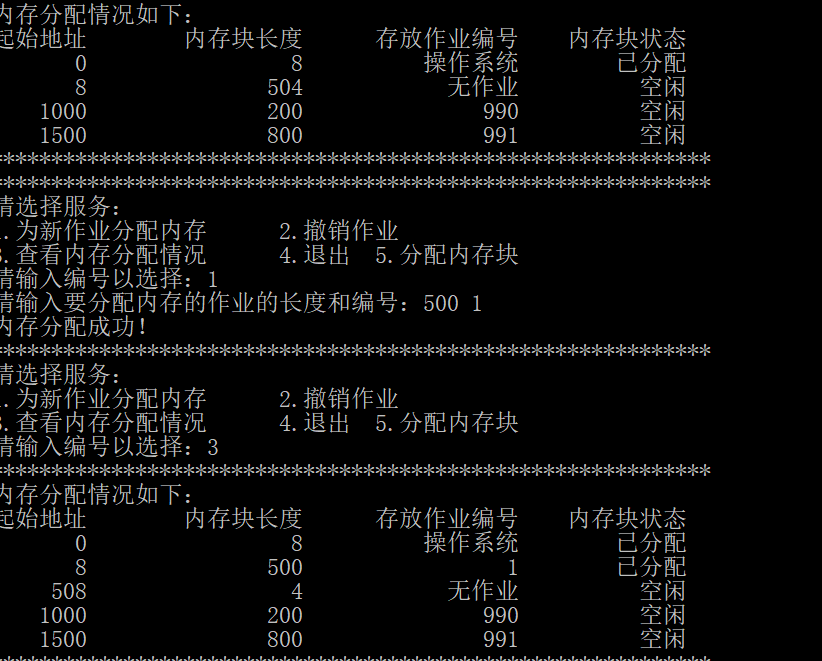
}

修改后的程序可以按照首次适应算法的规则，对不同长度的内存块分配作业，程序运行如下

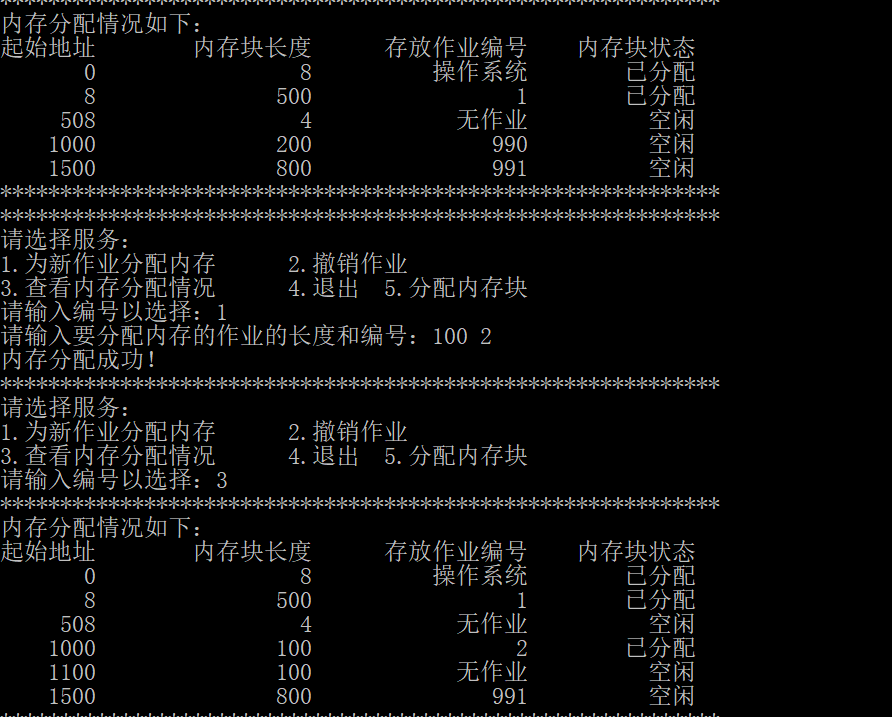


如上图所示，不同的内存块已经初始化完毕。现在对其添加作业：

对长度为500的作业分配内存，可见结果是我们想要的。



现在对长度为100的作业分配内存，我们想得到的结果是跳过内存块长度为4的内存块，对编号为990的内存块分配作业，如下图所示结果与我们预料的一致，说明首次适应算法的代码修改和完善成功。



1. 写出最佳适应算法
2. 最佳适应算法思想：首先，定义一个p指针，让p指针遍历空闲分区链表，当找到第一个满足进程请求空间大小的空闲区时，记录此位置，并且保存请求大小与空闲分区实际大小的差值记为a，然后让p指针继续遍历空闲分区链表，每当满足请求内存大小小于空闲区大小时，就记录两者的差值并且记录为b，比较a与b的大小关系，当a>b时，将b的值赋予a，并且修改记录位置为此空闲区的位置。若，a<=b,不做操作。继续遍历链表，重复上面的操作，直到p->next指向null为止。
3. 最佳适应算法代码：

**其中在原始代码基础上修改的部分已用紫色字体标出**

#include <stdio.h>

#include <stdlib.h>

//使用链表的数据结构

typedef struct LNode{

int address;//表示该分区的起始地址

int length; //表示该分区的长度

int state; //0表示该分区空闲，1表示该分区已分配

int number; //作业的编号

struct LNode \*next;

}\*pLNode;

//链表的插入操作

//L是链表的头指针，addr是该内存块的首地址，len是该内存块的长度

//sta是内存块的状态，num是将占有内存块作业的编号

void InsertLNode(pLNode L,int addr,int len,int sta,int num) {

pLNode p = (pLNode)malloc(sizeof(LNode));

p->address = addr;

p->length = len;

p->state = sta;

p->number = num;

p->next = NULL;

L->next = p;

}

//初始化链表

//初始化链表的头指针，假设操作系统有8M的内存空间，且始址为0

//leng为内存空间的总大小，单位为M

void InitLNode(pLNode L,int leng){

//L = (pLNode)malloc(sizeof(LNode));

L->address = 0;//从首地址开始

L->length = 8;

L->state = 1;

L->number = 65535;

//插入剩余的空闲分区

InsertLNode(L, 8, leng - 8,0,0);

printf("初始化成功！\n");

}

//撤销作业

//L为链表头指针，num为待撤销的作业编号

//若找到该作业则把其占有的内存块state置0

void Revocation(pLNode L, int num) {

pLNode p = L->next;

if (p->number == num) {

//找到了这个作业

p->state = 0;

printf("成功撤销编号为%d的作业！\n",num);

return;

}

while (p->next != NULL) {

p = p->next;

p->state = 0;

printf("成功撤销编号为%d的作业！\n", num);

return;

}

printf("撤销作业失败，没有找到编号为%d的作业\n", num);

}

//检查链表中是否有连续的未分配内存区，如果有就合并

void Merge\_free(pLNode L) {

pLNode p = L,pre;

int num = 0;//用于检测连续空闲内存空间数量，等于2表示有连续的内存空间

while (p != NULL) {

if (p->state == 0) {

//表示为空闲区

num++;

if (num == 1)

pre = p;//为合并做准备

else if (num == 2) {

//表示有连续的空闲区，执行合并操作

//将p结点的长度加给前驱结点的长度

//前驱结点指向p结点的下结点

//释放p结点

//将p指向前驱结点，方便下次判断

//num置1

pre->length += p->length;

pre->next = p->next;

free(p);

p = pre;

num = 1;

}

}

else if (p->state == 1)

num = 0;

p = p->next;

}

}

//-----------------------分割线---------------------------

void best\_fit(pLNode L,int num,int len)//最佳适应算法

{

int surplus;//记录可用内存与需求内存的差值

pLNode t=(pLNode)malloc(sizeof(pLNode));

pLNode p=L->next;

t->number =num;

t->length =len;

t->state =1;

pLNode q=NULL;//记录最佳位置

while(p)//遍历链表，找到第一个可用的空闲区间将他给q

{

if (p->state ==0&&p->length >=len)

{

q=p;

surplus=p->length -len;

break;

}

p=p->next;

}

while(p)//继续遍历，找到更加合适的位置

{

if (p->state ==0&&p->length ==len)

{

p->state =1;

p->number =num;

// return;

break;

}

if (p->state ==0&&p->length >len)

{

if (surplus>p->length -len)

{

surplus=p->length -len;

q=p;

}

}

p=p->next;

}

if (q==NULL)//如果没有找到位置

{

printf("内存分配失败，没有找到合适的空闲内存块\n");;

}

else//找到了最佳位置

{

pLNode l = (pLNode)malloc(sizeof(LNode));

l->length = q->length - len;

l->address = q->address + len;

l->state = 0;

l->number = 0;

l->next = q->next;

q->next = l;

q->length = len;

q->number = num;

q->state = 1;

printf("内存分配成功！\n");

return ;

}

}

//--------------------------------------------------------

//打印

void print(pLNode L) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("内存分配情况如下：\n");

printf("起始地址\t内存块长度\t存放作业编号\t内存块状态\n");

pLNode p = L;

while (p != NULL) {

printf("%8d\t%10d\t", p->address, p->length);

if (p->number == 65535) {

printf(" 操作系统\t");

}

else if (p->number == 0) {

printf(" 无作业\t");

}

else {

printf("%12d\t", p->number);

}

if (p->state == 0) {

printf(" 空闲\n");

}

else {

printf(" 已分配\n");

}

p = p->next;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//服务选择菜单

int select() {

int a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择服务：\n");

printf("1.为新作业分配内存\t2.撤销作业\n3.查看内存分配情况\t4.退出\t5.分配内存块\n");

printf("请输入编号以选择：");

scanf("%d", &a);

return a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

int main()

{

pLNode L = (pLNode)malloc(sizeof(LNode));

InitLNode(L, 512);

int a;

int len, num,addr,sta;

pLNode p=L->next ;

a = select();

getchar();

while (a != 4) {

switch (a) {

case 1:

printf("请输入要分配内存的作业的长度和编号：");

scanf("%d%d", &len, &num);

best\_fit(L,num,len);

break;

case 2:

printf("请输入要撤销作业的编号：");

scanf("%d", &num);

Revocation(L, num);

break;

case 3:

//system("cls");

print(L);

break;

case 5:

printf("请输入要分配的内存块的起始地址、长度、状态、编号：");

scanf("%d%d%d%d",&addr,&len,&sta,&num);

InsertLNode(p,addr,len,sta,num);

p=p->next ;

default:

break;

}

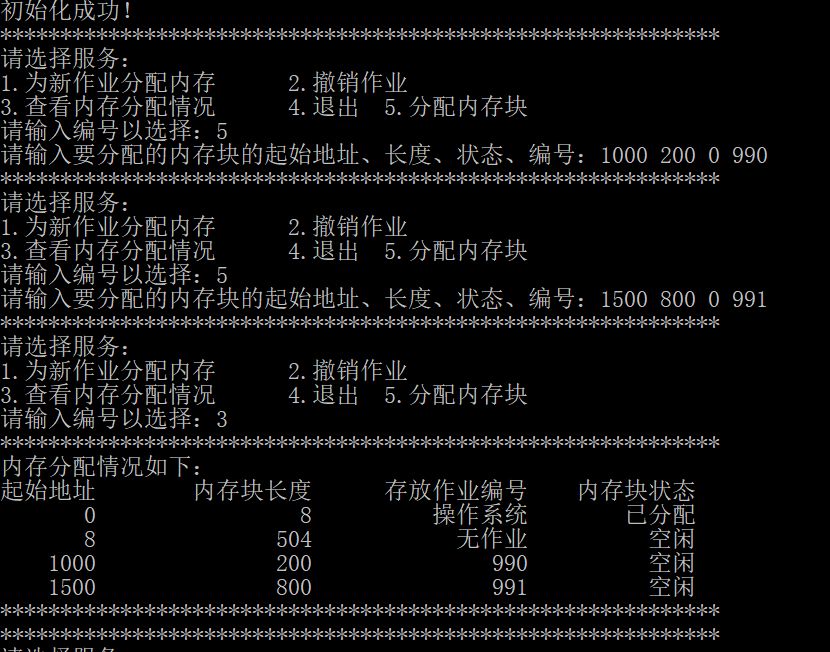
a = select();

}

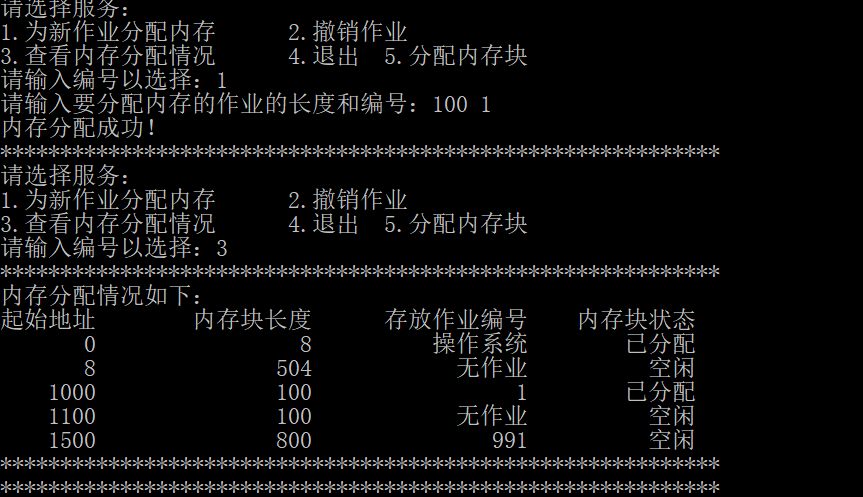
return 0;

}

1. 最佳适应法代码运行结果检验：

首先给系统分配如下所示四个内存块  


在四个内存块中，分配一个长度为100的作业。根据最佳适应算法的规则，应该分配给编号为990的内存，如下图所示，可知程序正确。



1. 最坏适应算法
2. 算法思想：当一个作业到达时，即从该表中检索出尺寸能满足要求且未分配的最大分区分配给它。该方法碎片最大，但剩余的空闲分区可再次使用。
3. 算法代码：

**修改部分用紫色标出**

#include <stdio.h>

#include <stdlib.h>

//使用链表的数据结构

typedef struct LNode{

int address;//表示该分区的起始地址

int length; //表示该分区的长度

int state; //0表示该分区空闲，1表示该分区已分配

int number; //作业的编号

struct LNode \*next;

}\*pLNode;

//链表的插入操作

//L是链表的头指针，addr是该内存块的首地址，len是该内存块的长度

//sta是内存块的状态，num是将占有内存块作业的编号

void InsertLNode(pLNode L,int addr,int len,int sta,int num) {

pLNode p = (pLNode)malloc(sizeof(LNode));

p->address = addr;

p->length = len;

p->state = sta;

p->number = num;

p->next = NULL;

L->next = p;

}

//初始化链表

//初始化链表的头指针，假设操作系统有8M的内存空间，且始址为0

//leng为内存空间的总大小，单位为M

void InitLNode(pLNode L,int leng){

//L = (pLNode)malloc(sizeof(LNode));

L->address = 0;//从首地址开始

L->length = 8;

L->state = 1;

L->number = 65535;

//插入剩余的空闲分区

InsertLNode(L, 8, leng - 8,0,0);

printf("初始化成功！\n");

}

//撤销作业

//L为链表头指针，num为待撤销的作业编号

//若找到该作业则把其占有的内存块state置0

void Revocation(pLNode L, int num) {

pLNode p = L->next;

if (p->number == num) {

//找到了这个作业

p->state = 0;

printf("成功撤销编号为%d的作业！\n",num);

return;

}

while (p->next != NULL) {

p = p->next;

p->state = 0;

printf("成功撤销编号为%d的作业！\n", num);

return;

}

printf("撤销作业失败，没有找到编号为%d的作业\n", num);

}

//-----------------------分割线---------------------------

void worse\_fit(pLNode L,int num,int len)//最坏适应算法

{

int surplus;//记录可用内存与需求内存的差值

pLNode t=(pLNode)malloc(sizeof(pLNode));

pLNode p=L->next;

t->number =num;

t->length =len;

t->state =1;

pLNode q=NULL;//记录最佳位置

while(p)//遍历链表，找到第一个可用的空闲区间将他给q

{

if (p->state ==0&&p->length >=len)

{

q=p;

surplus=p->length -len;

break;

}

p=p->next;

}

while(p)//继续遍历，找到更加合适的位置

{

if (p->state ==0&&p->length ==len)

{

p->state =1;

p->number =num;

// return;

break;

}

if (p->state ==0&&p->length >len)

{

if (surplus<p->length -len)

{

surplus=p->length -len;

q=p;

}

}

p=p->next;

}

if (q==NULL)//如果没有找到位置

{

printf("内存分配失败，没有找到合适的空闲内存块\n");;

}

else//找到了最坏位置

{

pLNode l = (pLNode)malloc(sizeof(LNode));

l->length = q->length - len;

l->address = q->address + len;

l->state = 0;

l->number = 0;

l->next = q->next;

q->next = l;

q->length = len;

q->number = num;

q->state = 1;

printf("内存分配成功！\n");

return ;

}

}

//--------------------------------------------------------

//打印

void print(pLNode L) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("内存分配情况如下：\n");

printf("起始地址\t内存块长度\t存放作业编号\t内存块状态\n");

pLNode p = L;

while (p != NULL) {

printf("%8d\t%10d\t", p->address, p->length);

if (p->number == 65535) {

printf(" 操作系统\t");

}

else if (p->number == 0) {

printf(" 无作业\t");

}

else {

printf("%12d\t", p->number);

}

if (p->state == 0) {

printf(" 空闲\n");

}

else {

printf(" 已分配\n");

}

p = p->next;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//服务选择菜单

int select() {

int a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("请选择服务：\n");

printf("1.为新作业分配内存\t2.撤销作业\n3.查看内存分配情况\t4.退出\t5.分配内存块\n");

printf("请输入编号以选择：");

scanf("%d", &a);

return a;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

int main()

{

pLNode L = (pLNode)malloc(sizeof(LNode));

InitLNode(L, 512);

int a;

int len, num,addr,sta;

pLNode p=L->next ;

a = select();

getchar();

while (a != 4) {

switch (a) {

case 1:

printf("请输入要分配内存的作业的长度和编号：");

scanf("%d%d", &len, &num);

worse\_fit(L,num,len);

break;

case 2:

printf("请输入要撤销作业的编号：");

scanf("%d", &num);

Revocation(L, num);

break;

case 3:

//system("cls");

print(L);

break;

case 5:

printf("请输入要分配的内存块的起始地址、长度、状态、编号：");

scanf("%d%d%d%d",&addr,&len,&sta,&num);

InsertLNode(p,addr,len,sta,num);

p=p->next ;

default:

break;

}

a = select();

}

return 0;

}

1. 程序运行结果：

