#include <stdio.h>

#include <stdlib.h>

typedef struct LNode{

int address;//起始地址

int length; //长度

int state; //0表示空闲，1表示已分配

int number; //作业的编号

struct LNode \*next;

}\*pLNode;

//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;

}

//初始化链表的头指针，假设操作系统有8M的内存空间，且始址为0

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");

}

//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;

}

