实验 9 页面置换算法与动态内存分配

实验性质：验证+设计
建议学时：2 学时
任务数：3 个

一、 实验目的

- 掌握OPT、FIFO、LRU、LFU、Clock等页面置换算法；
- 掌握可用空间表及分配方法；
- 掌握边界标识法以及伙伴系统的内存分配方法和回收方法。

二、 预备知识

页面置换算法

请求分页虚存管理的实现原理是：把作业的所有分页副本存放在磁盘中，当它被调度投入运行时，首先把当前需要的页面装入内存，之后根据程序运行的需要，动态装入其他页面；当内存空间已满，而又需要装入新页面时，根据某种算法淘汰某个页面，以便装入新页面。因此，在页表中必须说明哪些页已在内存，存在什么位置；哪些页不在内存，它们的副本在磁盘中的什么位置。还可以设置页面是否被修改过，是否被访问过，是否被锁住等标志供淘汰页面算法使用。

在地址映射过程中，若页表中发现所要访问的页不在内存，则产生缺页异常，操作系统接到此信号后，就会调用缺页异常处理程序，根据页表中给出的磁盘地址，将该页面调入内存，使作业继续运行下去。如果内存中有空闲页，则分配一个页，将新调入页面装入，并修改页表中相应页表项的驻留位以及相应的内存块号；若此时内存中没有空闲页，则要淘汰某页面，若该页在此期间被修改过，还要将其写回磁盘，这个过程称为页面替换。

动态内存分配

在动态存储管理系统中，开始时，整个内存区是一个“空闲块”。随着作业进入系统，先后提出申请存储空间的请求，系统则依次进行分配。因此，在系统运行的初期，整个内存区基本上分隔成两大部分：低地址区包含若干占用块；高地址区是一个“空闲块”。经过一段时间以后，有的作业运行结束，它所占用的内存区在释放后变成空闲块，这就使整个内存区呈现出占用块和空闲块犬牙交错的状态。

如果又有新的作业进入系统请求分配内存，通常有两种做法：一种策略是系统继续从高地址的空闲块中进行分配，而不理会已分配给用户的内存区是否已空闲，直到分配无法进行时，系统才回收所有作业不再使用的空闲块，并且重新组织内存，将所有空闲的内存区连接在一起成为一个大的空闲块。另一种策略是作业一旦运行结束，便将它所占内存区释放成为空闲块，同时，每当新的作业请求分配内存时，系统需要巡视整个内存区中所有空闲块，并从中找出一个“合适”的空闲块分配之。由此，系统需建立一张记录所有空闲块的“可利用空间表”，此表的结构可以是“目录表”，也可以是“链表”。

根据系统运行的不同情况，可利用空间表可以有 3 种不同的结构形式：第一种情况是系统运行期间所有作业请求分配的存储块大小相同；第二种情况是系统运行期间作业请求分配的存储块有若干种大小的规格；第三种情况是系统在运行期间分配给作业的内存块的大小不固定，可以随请求而变。

由于可利用空间表中的结点大小不同，则在分配时有 3 种不同的分配策略:(1)首次拟合法；（2）最佳拟合法；（3）最差拟合法。

三、 实验内容

在 EOS 操作系统中还没有实现虚拟内存的管理，当然也就不存在页面置换算法，如果读者直接在 EOS 中添加虚拟内存管理，或者修改 EOS 的动态内存分配方法又会比较复杂，所以读者可以先根据本实验的内容，通过编写 Windows 控制台应用程序的方式来模拟实现页面置换算法和动态内存分配。待读者对相关内

容有一定的理解后，可以考虑完成本实验后面的思考与练习题目，加深对页面置换算法和动态内存分配的理解。

3.1 任务（一）：页面置换算法

3.1.1 准备实验

请读者按照下面方法之一在本地创建一个 Windows 控制台项目，用于完成本次任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，就需要使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将实验模板克隆到本地磁盘中，实验模板的 URL 为 https://www.codecode.net/engintime/os-lab/Project-Template/page-replace-algorithm.git。

建议教师按照下面的步骤在 CodeCode.net 平台上布置此次实验任务：

(1) 在操作系统实验课程中，新建一个实验任务。

(2) 在“新建任务”页面中，教师需要添写“任务名称”，推荐使用“实验 9 页面置换算法与动态内存分配-任务（一）”；还需要填写“模板项目URL”，应该使用“https://www.codecode.net/engintime/os-lab/Project-Template/page-replace-algorithm.git”。

(3) 点击“新建任务”按钮，完成新建任务操作。

3.1.2 查看最佳页面置换算法(OPT)和先进先出页面置换算法(FIFO)的执行过程

最佳页面置换算法

当需要换出一个页面时，淘汰那个以后不再需要的或最远的将来才会用到的页面。

先进先出页面置换算法

当需要换出一个页面时，淘汰那个最先调入主存的页面，或者说在主存中驻留时间最长的那一页。

在新建的项目中，仔细阅读文件中的源代码和注释，查看 OPT 和 FIFO 页面置换算法是如何实现的。

其中，在 main 函数中定义了一个数组 PageNumofR[]来存放页面号引用串，定义了一个指针 *BlockofMemory 指向一块存放页面号的内存块。依次将数组 PageNumofR[]中的页面装入内存，根据不同的页面置换算法淘汰内存中的页面。

按照下面的步骤查看 OPT 和 FIFO 的执行过程：

1. 按 F7 键生成修改后的控制台应用程序项目，确保没有语法上的错误。
2. 按 Crtl+F5 执行此程序，查看 OPT 和 FIFO 的执行过程，其中，“#”表示未引用的内存块。执行结果如下图所示：

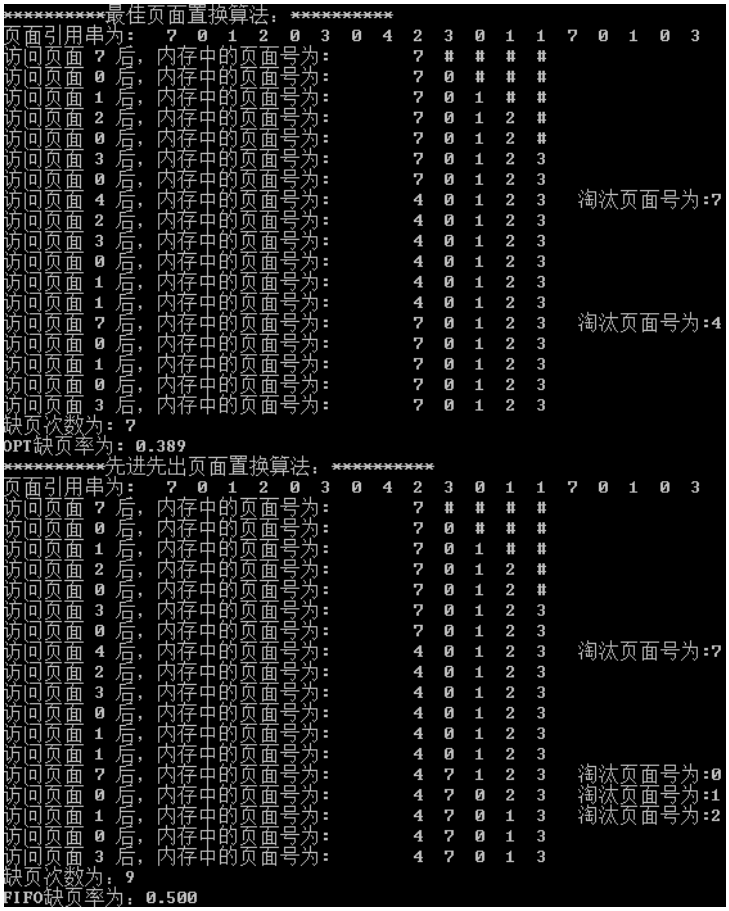

图 17-1：OPT 和 FIFO 页面置换算法的执行结果

3.1.3 完成最近最久未使用页面置换算法(LRU)

最近最久未使用页面置换算法

当需要换出一个页面时，淘汰那个在最近一段时间里较久未被访问的页面。它是根据程序执行时所具有的局部性来考虑的，即那些刚被使用过的页面可能马上还要被使用，而那些在较长时间里未被使用的页面一般可能不会马上使用。

在新建的项目中，仔细阅读其中的源代码，并仿照已经实现的 OPT 和 FIFO 算法来实现最近最久未使用页面置换算法(LRU)。正确实现 LRU 算法后的执行结果应该如下图所示。

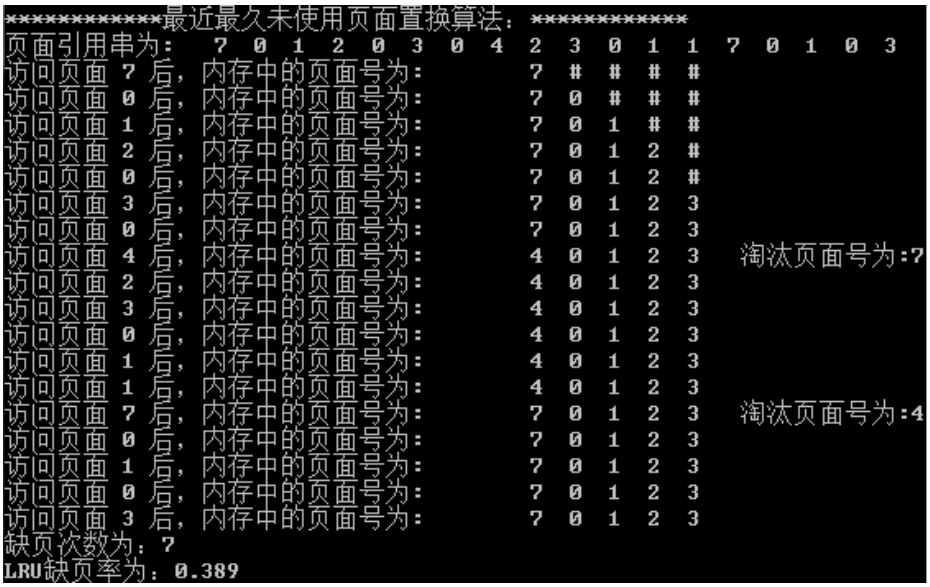

图 17-2：LRU 页面置换算法的执行结果

有兴趣的读者还可以尝试写出最不常用页面置换算法（LFU）和页面缓冲置换算法（PBA），以及 CLOCK

页面置换算法等。下面是这三种页面置换算法的实现原理。

最不常用页面置换算法

如果对应每个页面设置一个计数器，每当访问一页时，就使它对应的计数器加 1.过一定时间后，将所有计数器全部清 0.当发生缺页异常时，可选择计数值最小的对应页面淘汰，显然它是在最近一段时间里最不常用的页面。

页面缓冲置换算法

页面缓冲置换算法与先进先出 FIFO 算法有些相似，置换策略与 FIFO 一样，即将内存中最先进的页面置换出来。与 FIFO 不同的是，根据被置换出来的页面被放入两个链表中，分别是空闲链表和已修改链表。等下次需要访问页面时，如果在链表中有，则只需将链表中的页面调入内存即可。

CLOCK 页面置换算法

使用页表中的“引用位”，把进程已调入主存的页面链接成循环队列，用指针指向循环队列中下一个将被替换的页面，形成类似于钟表面的环形表，队列指针相当于钟表面上的表针，这就是时钟页面置换算法的得名。

3.1.4 提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。

3.2 任务（二）：动态内存分配 - 边界标识法

3.2.1 准备实验

请读者按照下面方法之一在本地创建一个 Windows 控制台项目，用于完成本次任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，就需要使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将实验模板克隆到本地磁盘中，实验模板的 URL 为 https://www.codecode.net/engintime/os-lab/Project-Template/boundary-identification.git。

建议教师按照下面的步骤在 CodeCode.net 平台上布置此次实验任务：

(1) 在操作系统实验课程中，新建一个实验任务。

(2) 在“新建任务”页面中，教师需要添写“任务名称”，推荐使用“实验 9 页面置换算法与动态内存分配-任务（二）”；还需要填写“模板项目URL”，应该使用“https://www.codecode.net/engintime/os-lab/Project-Template/boundary-identification.git”。

(3) 点击“新建任务”按钮，完成新建任务操作。

3.2.2 边界标识法的设计实现

边界标识法是操作系统中用以进行动态分区分配的一种存储管理方法。系统将所有的空闲块链接在一个双重循环链表结构的可利用空间表中；分配可按首次拟合进行，也可按最佳拟合进行。系统的特点在于：在每个内存区的头部和底部两个边界上分别设有标识，以标识该区域为占用块或空闲块，使得在回收用户释放的空闲块时易于判别在物理位置上与其相邻的内存区域是否为空闲块，以便将所有地址连续的空闲存储区组合成一个尽可能大的空闲块。

1） 分配算法

可以采用首次拟合法进行分配，即从表头指针所指节点起，在可利用空间表中进行查找，找到第一个容量不小于请求分配的存储量的空闲块时，即可进行分配。

2） 回收算法

一旦作业释放占用块，系统需立即回收以备新的请求产生时进行再分配。为了使地址相邻的空闲块结合成一个尽可能大的节点，则首先需要检查刚释放的占用块的左、右邻居是否为空闲块。若释放块的左、右邻居均为占用块，则处理最为简单，只要将此新的空闲块作为一个结点插入到可利用空闲表中即可；若只有左邻居是空闲块，则应与左邻区合并成一个结点；若只有右邻居是空闲块，则应与右邻区合并成一个结点；若左右邻居都是空闲块，则应将 3 块合起来成为一个结点留在可利用空间表中。

3） 实现

在新建的项目中，仔细阅读其中的源代码及注释，阅读时着重注意以下几点：

- 定义 ALLOC_MIN_SIZE 的目的是为了防止在多次分配以后链表中出现一些极小的、总也分配不出去的空闲块；
- 可利用空间表的节点结构

```
typedef struct Word{
    union {
        Word * preLink;//头部域前驱
        Word * upLink;//尾部域，指向结点头部
    };
    int tag;//0标示空闲,1表示占用
    int size;//仅仅表示可用空间，不包括头部和尾部空间
    Word * nextLink;//头部后继指针.
}*Space;
```

- 定义了 userSpace []数组来存放已分配区表；
- allocBoundTag 函数是采用首次适应算法分配内存的。
- 回收内存函数 reclaimBoundTag 在回收内存时要将回收的块与空闲分区中的左右邻居进行比较，在允许的情况下进行合并。
- SetColor 是设置字体的函数，可以使输出结果更加醒目。

按照下面的步骤查看边界标识法的实现过程：

1. 按 F7 生成项目，确保没有语法错误。
2. 按 Ctrl+F5 执行程序，查看运行结果。其提供的功能项如下图所示：

图 17-3：边界标识法的功能项

程序启动后会停留在此界面等待用户输入功能项序号 0-3 中的一个。系统会根据用户输入的序号执行相应的功能，然后继续停留在此界面等待用户的输入，直到用户按 0 退出应用程序。

例如，首先使用功能 3 查看当前内存的分配情况，然后使用功能 1，输入内存长度 20 来分配内存，接着分配长度为 30 和 500 的内存，然后使用功能 3 查看内存的分配情况，如图 17-4 所示。如果已分配的内存达到内存的最大值时，再次分配内存时，系统会提示“无法继续分配内存”。当然还可以使用功能 2 来回收内存（注意：在释放内存时，首地址的输入使用十六进制）。如果要回收的内存左右有空闲区，就会与之合并，否则就作为一个结点插入到可利用空间表中，如图 17-5 所示。读者可在使用这些功能的同时，加深对程序的理解。

图 17-4：分配内存

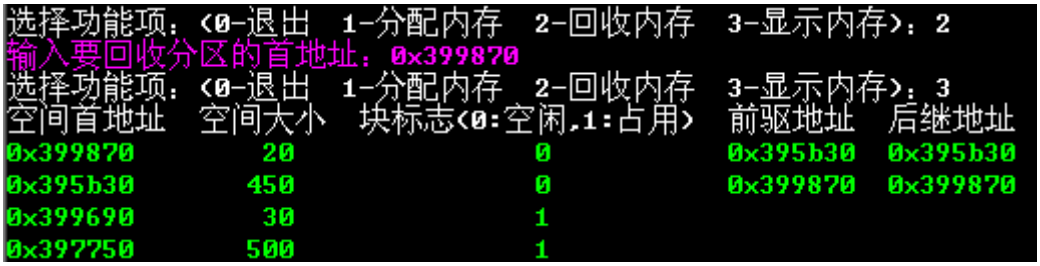

图 17-5：回收内存

4) 练习

a) 释放之前分配的大小为 500 和 30 的内存块，说明在释放这两个块时，分别属于哪种回收内存的情况。

b) 重新运行程序后尝试执行下面的操作：

1、 使用功能 1 先分配空间大小为 880 的内存块，再分配空间大小为 120 的内存块，然后使用功能 3 查看内存的情况，此时已将所有的内存空间都分配完。

2、 使用功能 2 回收内存，先回收空间大小为 120 的内存块，接着回收空间大小为 880 的内存块，这时会进行合并左块的操作，使用功能 3 查看内存的显示情况，可以看到内存已经正确回收。

3、 重复步骤 1，但是此次先回收空间大小为 880 的内存块，然后回收空间大小为 120 的内存块，这时会进行合并右块的操作，由于在合并右块的代码中存在缺陷，在回收空间大小为 120 的内存块时会导致程序崩溃，请读者修改合并右块的相关代码，使之可以正常合并右块。

c) 目前 allocBoundTag 函数在分配空间时，采用的是首次适应算法，请读者尝试采用循环首次适应算法和最佳适应算法分别实现边界标识法。

3.2.3 提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。

3.3 任务（三）：动态内存分配 -伙伴系统

3.3.1 准备实验

请读者按照下面方法之一在本地创建一个 Windows 控制台项目，用于完成本次任务：

方法一：从 CodeCode.net 平台领取任务

读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建个

人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，就需要使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将实验模板克隆到本地磁盘中，实验模板的 URL 为 https://www.codecode.net/engintime/os-lab/Project-Template/partner-system.git。

建议教师按照下面的步骤在 CodeCode.net 平台上布置此次实验任务：

(1) 在操作系统实验课程中，新建一个实验任务。

(2) 在“新建任务”页面中，教师需要添写“任务名称”，推荐使用“实验 9 页面置换算法与动态内存分配-任务（三）”；还需要填写“模板项目URL”，应该使用“https://www.codecode.net/engintime/os-lab/Project-Template/ partner-system.git”。

(3) 点击“新建任务”按钮，完成新建任务操作。

3.3.2 伙伴系统的设计实现

伙伴系统是操作系统中用到的另一种动态存储管理方法。它和边界标识法类似，当用户提出申请时，分配一块大小“恰当”的内存区给用户；反之，在用户释放内存区时即回收。所不同的是，在伙伴系统中，无论是占用块或空闲块，其大小均为 2 的 k 次幂（k 为某个正整数）。例如：当用户申请 n 个字节的内存区时，分配的占用块大小为 2^k 个字节（$2^{k-1}<n\leqslant 2^k$）。由此，在可利用空间表中的空闲块大小也只能是 2 的 k 次幂。

1) 分配算法

当用户提出大小为 n 个字节的内存请求时，首先在可利用表上寻找节点大小与 n 相匹配的子表，若此子表非空，则将子表中任意一个结点分配之即可；若此子表为空，则需从更大的非空子表中去查找，直至找到一个空闲块，则将其中一部分分配给用户，而将剩余部分插入相应的子表中。

2) 回收算法

在回收空闲块时，应首先判断其伙伴是否为空闲块，若否，则只要将释放的空闲块简单插入在相应子表中即可；若是，则需在相应子表中找到其伙伴并删除之，然后再判断合并后的空闲块的伙伴是否是空闲块。依此重复，直到归并所得空闲块的伙伴不是空闲块时，再插入到相应的子表中去。

3) 实现

在新建的项目中，仔细阅读其中的源代码及注释，并按照后面练习的要求实现回收算法。

阅读代码时请注意以下几点：

- 伙伴系统可利用空间表的结构体

```
typedef struct _Node{
    struct _Node *llink;    //指向前驱节点
    int bflag;              //块标志，0：空闲，1：占用
    int bsize;              //块大小，值为 2 的幂次 k
    struct _Node *rlink;    //指向后继节点
}Node;
```

可利用空间表的头结点的结构体

```
typedef struct HeadNode{
    int nodesize;       //链表的空闲块大小
    Node *first;        //链表的表头指针
}FreeList[M+1];
```

- AllocBuddy 函数的功能是实现分配算法，分配空闲块以后，还需要将剩余块插入相应的子表中。
- Reclaim 函数的功能是实现回收算法，在回收内存时，需要先查找其伙伴是否为空闲块。如果没有空闲块，则只要将释放的空闲块简单插入相应的子表中即可；如果有空闲块，则需在相应子表

中找到其伙伴并删除之，然后继续判断合并后的空闲块的伙伴是否是空闲块。依此重复，直到所得的空闲块的伙伴不是空闲块时为止，最后插入到相应的子表中去。

该程序提供的功能项如图 17-6 所示。例如，选择功能 1 分配内存，然后输入作业所需内存长度 16，执行完后再使用功能 3 查看内存分布情况，如图 17-7。

图 17-6：伙伴系统运行的结果

图 17-7：申请内存块后的内存分布情况

4） 练习

请读者编程实现伙伴系统的内存回收算法。完成算法后，可以按照下面的案例进行测试：

a） 先分配一个大小为 20 的内存块，然后使用功能 3 可以看到刚刚分配的内存块在已利用空间中的块号为 0，接下来回收块号为 0 的内存块，会导致所有的空闲块归并为一个空闲块。此时查看内存的显示情况，如图 17-8 所示，在可利用空间显示了一个内存块。

图 17-8：回收内存块后的内存分布情况

b） 连续分配 5 次块大小为 20 的内存块，查看内存的显示情况，然后按照下面的顺序回收内存块，回收块号为 0 的内存块、回收块号为 2 的内存块、回收块号为 1 的内存块、回收块号为 3 的内存块、回收块号为 4 的内存块，并在每次回收之后查看内存的显示情况，如图 17-9 和图 17-10 所示，完成所有的内存块回收之后，可利用空间恢复为一个内存块。

图 17-9：回收块号为 2 的内存块后的分布情况

```
输入要回收块的块号：1
选择功能项：(0-退出  1-分配主存  2-回收主存  3-显示主存):3
可利用空间：
      块的大小     块的首地址     块标志(0:空闲 1:占用)  前驱地址   后继地址
        32          0x3e5090                  0          0x3e5690  0x3e5690
        32          0x3e5690                  0          0x3e5090  0x3e5090
        64          0x3e4c90                  0          0x3e5890  0x3e5890
        64          0x3e5890                  0          0x3e4c90  0x3e4c90
        256         0x3e5c90                  0          0x3e5c90  0x3e5c90
        512         0x3e6c90                  0          0x3e6c90  0x3e6c90

已利用空间：
     占用块块号 占用块的首地址  块大小  块标志(0:空闲 1:占用)
         3         0x3e5290      32        1
         4         0x3e5490      32        1

选择功能项：(0-退出  1-分配主存  2-回收主存  3-显示主存):2
输入要回收块的块号：3
选择功能项：(0-退出  1-分配主存  2-回收主存  3-显示主存):3
可利用空间：
      块的大小     块的首地址     块标志(0:空闲 1:占用)  前驱地址   后继地址
        32          0x3e5690                  0          0x3e5690  0x3e5690
        64          0x3e5890                  0          0x3e5890  0x3e5890
        128         0x3e4c90                  0          0x3e4c90  0x3e4c90
        256         0x3e5c90                  0          0x3e5c90  0x3e5c90
        512         0x3e6c90                  0          0x3e6c90  0x3e6c90

已利用空间：
     占用块块号 占用块的首地址  块大小  块标志(0:空闲 1:占用)
         4         0x3e5490      32        1

选择功能项：(0-退出  1-分配主存  2-回收主存  3-显示主存):2
输入要回收块的块号：4
选择功能项：(0-退出  1-分配主存  2-回收主存  3-显示主存):3
可利用空间：
      块的大小     块的首地址     块标志(0:空闲 1:占用)  前驱地址   后继地址
        1024        0x3e4c90                  0          0x3e4c90  0x3e4c90
```

图 17-10：回收块号为 1、3、4 的内存块后的分布情况

3.3.3 提交作业

如果读者是通过从 CodeCode.net 平台领取任务创建的项目，并将项目克隆到本地进行实验，实验结束后可以将本地已更改的项目再推送到 CodeCode.net 平台的个人项目中，方便教师通过 CodeCode.net 平台查看读者提交的作业。

四、 思考与练习

1、 新建一个 EOS kernel 的项目，打开 mm/mempool.c 文件，其中的 PoolAllocateMemory 和 PoolFreeMemory 函数实现了在系统内存池中动态分配内存和回收内存的功能。请读者通过查看这两个函数的调用情况，观察一下 EOS 内核中都有哪些内容是通过动态内存分配来获取内存的。然后再仔细阅读这两个函数中的源代码及注释，查看一下 EOS 是如何使用伙伴算法来实现动态内存分配和回收的。请读者尝试使用边界标识法修改系统内存池的动态分配内存和回收内存的方法。

2、 x86 平台为实现改进型的 Clock 页面置换算法提供了硬件支持，它实现了二级映射机制，并且将页表项的格式定义如下：

图17-11：页目录和页表中的表项的格式

其中位 5 是访问标志，当处理器访问页表项映射的页面时，页表表项的这个标志就会被置为 1。位 6 是页面修改标志，当处理器对一个页面执行写操作时，该项就会被置为 1。

继续改进 EOS 提供的“mm”命令，在其命令函数中除了将二级页表映射打印出来，还要实现一个算法来根据页表项中的访问位和修改位计算出应该置换出的页面。不必完全实现 Clock 页面置换算法，只要能判断出应该置换的页面即可。