苏州大学实验报告

院、系	计算机学院	年级专业	20计科	姓名	柯骅	学号	2027405033
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2023.03.16	

实 验 名 称 实 验 3 内 存 管 理

一、 实验目的

1. 掌握动态分区分配方式使用的数据结构和分配算法
2. 进一步加深对动态分区分配管理方式及其实现过程的理解
3. 理解虚拟内存管理的原理和技术
4. 掌握请求分页存储管理的常用理论——页面置换算法
5. 理解请求分页中的按需调页机制

二、 实验内容

1. （动态分区分配方式的模拟）

编写C语言程序，模拟实现首次/最佳/最坏适应算法的内存块分配和回收，要求每次分配和回收后显示出空闲分区和已分配分区的情况。假设初始状态下，可用的内存空间为640KB。

2. （页面置换算法的模拟）

设计一个虚拟存储区和内存工作区，并使用下述常用页面置换算法计算访问命中率。

(1) 先进先出算法（FIFO）
(2) 最近最少使用算法（LRU）
(3) 最优置换算法（OPT）

要求如下：

(1) 通过随机数产生一个指令序列，共320条指令。
(2) 将指令序列转换成页面序列。
①页面大小为1KB；
②用户内存容量为4~32页；
③用户虚存容量为32KB。
④在用户虚存中，按每页存放10条指令排列虚存地址，即320条指令存在32个页面中。
(3) 计算并输出不同置换算法在不同内存容量下的命中率。命中率计算公式为：

命中率 = 1 - 页面失效次数/页面总数

三、 实验步骤

1. （动态分区分配方式的模拟）

本实验的主要目的是模拟实现动态分区分配方式下内存的分配与回收，而设计的分配与回收算法涉及首次适应算法、最佳适应算法和最坏适应算法。根据动态分区分配的原理，主要需要建立两个数据结构——空闲分区表和已分配分区表，它们都需要包含分区的起始地址、长度等信息。

当有新作业请求装入主存时，须查找空闲分区表，从中找出一个合适的空闲分区并将其分配给作业。然后按照作业需要的内存大小将其装入主存，剩下的部分仍为空闲分区，将其登记到空闲分区表中，作业占用的分区则登记到已分配分区表中。作业执行完毕后，应回收作业占用的分区，具体操作为:删除已分配分区表中的相关项，然后修改空闲分区表，并根据情况增加或合并空闲分区。

Linux 内存操作函数：

(1) malloc()

```
1. void *malloc(size_t size);
```

memeory allocation,动态内存分配,malloc 分配的内存是位于堆中的，并且没有初始化内存的内容；malloc()函数有一个参数,即要分配的内存空间的大小。

malloc()函数的实质体现在它会将可用内存块连接为一个列表的空闲链表。调用 malloc()函数时，它会沿这个链表寻址，找一个大到足以满足用户请求所需的内存块。然后将该内存块一分为二，其中一块的大小与用户请求 size 相符,malloc()函数会将其分配给用户;剩下的另-块则继续放到链表中。

(2) calloc()

```
1. void *calloc(size_t numElements,size_t sizeOfElement);
```

函数有两个参数,分别为元素的数目和每个元素的大小,这两个参数的乘积就是要分配的内存空间的大小。如果调用成功,函数 calloc()将返回所分配的内存空间的首地址。

calloc()函数用来配置指定个数的相邻内存单元(每个内存单元的大小都是指定的),并返回指向第一个元素的指针。虽然与使用 malloc()函数的效果相同，但在利用 calloc()配置内存时，会将内存中的内容初始化为。

(3) alloca()

```
1. void *alloca(size_t);
```

malloc()函数的实质体现在它会将可用内存块连接为一个列表的空闲链表。调用 malloc()函数时，它会沿这个链表寻址，找一个大到足以满足用户请求所需的内存块。然后将该内存块一分为二，其中一块的大小与用户请求 size 相符， malloc()函数会将其分配给用户;剩下的另-块则继续放到链表中。

(4) realloc()

```
1. realloc(void *__ptr, size_t __size)
```

reallc()用于更改原内存块的大小到指定字节。若更改的容量值比原来的内存空间小，内存内容将保持不变，且返回原来内存空间的起始地址;若更改的容量值大于原来的内存空间，且原有内存还有足够的剩余空间，则使用 realloc()更改后的内存等于原来的内存加上剩余内存，但仍返回原来内存空间的起始地址;否则，realloc()将申请新的内存，把原来的内存数据复制到新内存中，并释放原来的内存，返回新内存的地址。若没有指定原内存块的大小(即参数为 NULL),则 realloc()的作用相当于 malloc();若更改的大小为 0,则作用相当于 free(ptr)。

(5) mmap()

```
1.  void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

mmap()用来将某个文件的内容映射到内存中，对相应内存区域的存取即直接对该文件内容的读写。

- start 允许用户使用某个特定的地址作为这段内存的起始地址。如果被设为NULL，则系统自动分配一个地址。
- length 指定内存段的长度。
- prot 用来设置内存段的访问权限（注意是prot不是port）。可按位或取值:
  1）PROT_READ，内存段可读;
  2）PROT_WRITE，内存段可写;
  3）PROT_EXEC，内存段可执行;
  4）PROT_NONE，内存段不能被访问。

- flags 控制内存段内容被修改后程序的行为。可以被设置为（列举几个常用值）：
  1）MAP_SHARED 进程间共享这段内存，对该内存段的修改将反映到被映射的文件中。提供了进程间共享内存的POSIX方法。
  2）MAP_PRIVATE 内存段为调用进程所私有。对该内存段的修改不会反映到被映射的文件中。
  3）MAP_ANONYMOUS 这段内存不是从文件映射而来的。其内容被初始化为全0。这种情况下，mmap函数的最后2个参数将被忽略。
  4）MAP_FIXED 内存段必须位于start参数指定的地址处。start必须是内存页面大小（4096byte）的整数倍。
  5）MAP_HUGETLB 按照"大内存页面"来分配内存空间。"大内存页面"的大小可通过/proc/meminfo 文件来查看。
- fd 被映射文件对应的文件描述符。一般通过open系统调用获得。
- offset 设置从文件的何处开始映射，对于不需要读入整个文件的情况时，需要设置。

(6) munmap()

```
1. int munmap(void *addr, size_t length);
```

munmap()用来取消参数所指的映射内存的起始地址。当进程结束或利用 exec 函数族执行其他程序时，映射内存将自动解除，但关闭对应的文件描述符时不会解除映射。

两个参数同 mmap。

(7) free()

```
1. void free(void *ptr);
```

free()用于释放指针指向的内存空间，该指针必须是 malloc/calloc/realloc 调用的返回值。若所指的内存空间已被收回或指针指向未知的内存地址，则可能发生无法预知的情况。若指针为空，则不执行任何操作。

(8) getpageszie()

getpagesize(用于取得内存分页大小，单位为字节(Byte)。内存分页大小作为系统的分页大小，不一-定和硬件分页大小相同。

具体步骤：

首次适应算法：从空闲分区表的第一个表目起查找该表，把最先能够满足要求的空闲区分配给作业
最佳适应算法：从全部空闲区中找出能满足作业要求，且大小最小的空闲分区分配给作业
最坏适应算法：扫描整个空闲分区或链表，总是挑选一个最大的空闲分区分配给作业

(1) 定义分区、空闲分区表与已分配分区表

定义一个结构体 RECT 作为分区，其中包含此分区的首位地址（adress）与分区的大小（size），和指向下一个分区的指针（next）：

```
1.  struct node  //定义分区
2.  { int address, size;
3.    struct node *next; };
4.  typedef struct node RECT;
```

随后使用 malloc()函数分别建立空闲分区表（head）和已分配分区表（heada）的初始状态，同时将给定的初始区域作为一个空闲分区使用 RECT 定义出来，并插入到空闲分区表的首位：

```
1.  head=malloc(sizeof(RECT)); //建立空闲分区表的初始状态
2.  heada=malloc(sizeof(RECT)); //建立已分配分区表的初始状态
3.  head->next=p;
4.  p->size=MAX;  p->address=0;  p->next=NULL;
```

(2) 回收分区

之所以要从回收空间说起，是因为各种适应算法的主要区别在此，同时也是分配空间的必要铺垫。

回收空间的主要任务是将指定一块已分配分区从已分配分区表（heada）中删除，并添加进空闲分区表（head），随后需要按照不同的分配适应算法，来对空闲分区表进行处理，具体来说：

首次适应算法：只需要将回收的空闲分区与空闲分区表中相应的前后空闲分区进行合并即可，相对位置并不需要变化

最佳适应算法：将回收的空闲分区与空闲分区表中相应的前后空闲分区进行合并后，需要将合并后的空闲分区按照分区容量**从小到大**的顺序重新插入到空闲分区表中，便于分配分区时寻找使用。

最坏适应算法：将回收的空闲分区与空闲分区表中相应的前后空闲分区进行合并后，需要将合并后的空闲分区按照分区容量**从大到小**的顺序重新插入到空闲分区表中，便于分配分区时寻找使用。

(3) 分配分区

经过回收分区步骤后，空闲分区链表已经按照需要的顺序排列，我们只需要从链表首位向后依次枚举，找到第一个大于等于所需空间的分区结构，将它分配给新作业即可。理由如下：

首次适应算法：由于我们回收分区时仅对空闲分区进行合并，空闲分区表中的空闲分区永远按照地址顺序进行排列，于是实现了总能把地址最小的，符合条件的空闲分区分配给作业。

最佳适应算法：由于在回收时，每次总将新的空闲区域按照分区容量**从小到大**的顺序排列，所以按照上述方法找到的第一个容量足够的空闲分区一定是所有容量足够的空闲分区中容量最小的。

最坏适应算法：由于在回收时，每次总将新的空闲区域按照分区容量**从大到小**的顺序排列，所以按照上述方法找到的第一个容量足够的空闲分区一定是所有容量足够的空闲分区中容量最大的。.

排序关键代码如下：

```
/*前后合并*/
/*新空闲分区按照从小到大插入到合适位置*/
do{
    if(after==NULL||(after->size>back1->size)){
      before->next=back1;
      back1->next=after;
      insert=1;}
    else{
      before=before->next;
      after=after->next; }
  }while(!insert);
```

2. （页面置换算法的模拟）

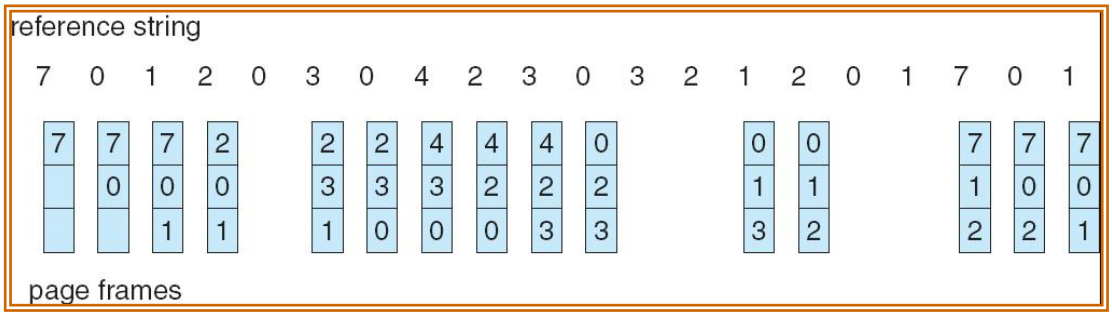
算法介绍：

(1) FIFO

置换在内存中驻留时间最长的页面

容易理解和实现、但性能不总是很好

实现：使用 FIFO 队列管理内存中的所有页



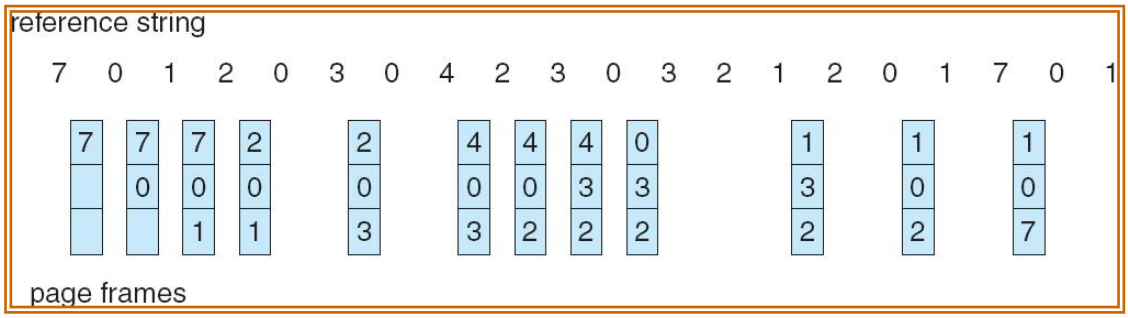
(2) LRU

置换最长时间没有使用的页

性能接近 OPT

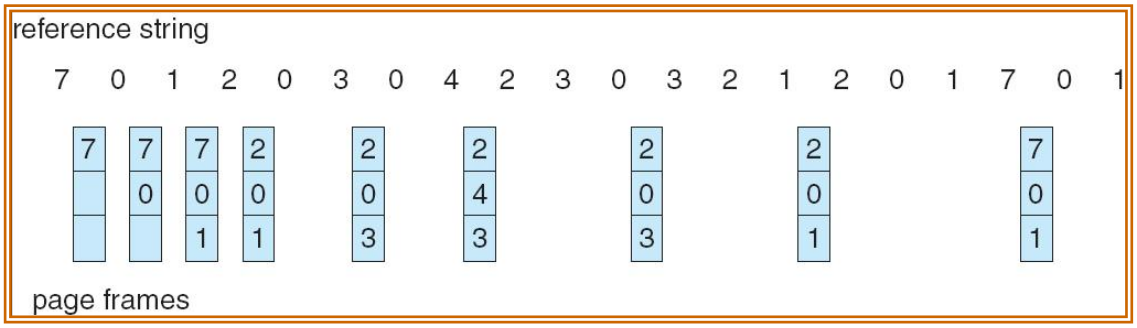
实现：计数器（时间戳）或栈

=> 开销大、需要硬件支持



(3) OPT

选择在将来不被使用，或者在最远的将来才被使用的老页面，也就是选择该页面后面较远才出现，甚至不出现的页面，将其淘汰，即被替换。



具体实现：

(1) 生成随即指令序列

使用随机函数 srand()和 rand()随机产生指令序列，然后将指令序列转换成相应的页面序列。

随机数的取值比较复杂，指令地址是按如下原则产生的:

①50%的指令是顺序执行的;

②25%的指令均匀地分布在前地址部分;

③25%的指令均匀地分布在后地址部分。

(2) 设计页面类型、页面控制结构等数据结构。

创建一个结构体（pl_type）作为页面结构，其中包含四个参数，pn（页号），pfn（内存块号），counter（一个周期内访问页面的次数），time（访问时间），同时创建一个该类型的数列（p1），用于保存各个页面的参数。

同时创建一个结构体 pfc_struct 作为页面控制结构，用于记录真正在内存中的页面，其中包含的参数有 pn（页号），pfn（内存块号），指向下一个 pfc_struct 的指针，以链表的形式进行存储。

(3) 计算使用指定页面置换算法时的访问命中率。

FIFO：

在程序中，当新页面等待读取时，先判断是否在内存中，如果在，那么记录下相应参数即可；如果不在且内存没有空位，那么就需要释放掉最先进入内存的页面，并将新页面加载进内存，记录下相应参数，直到所有页面读取列表处理完毕。

LRU：

在程序中，当一个新页面在内存中命中时，记录下相应参数，并更新留存在内存中的时间为 0，其余页面留存时间+1；当新页面没有命中且内存没有空位时，则需要找出在内存中停留时间最长的页面，将它置换出去，将新页面加载进内存，同时相应参数做出更改，其余页面停留时间+1，新页面停留时间为 0。

OPT：

在程序中，如果新页面没有命中且内存没有空位时，就创建一个 dist 数组记录内存中的每个进程距离未来下一次使用分别有多长的时间，如果未来不再用，就记为一个大数，表示无穷大，最后找到 dist 数组中最大数对应的页面，将它置换出去，同时相应参数做出更改即可。

四、 实验结果

1. （动态分区分配方式的模拟）

最佳适应算法：

依次输入右图数据，运行结果如下：

```
b
a
100
a
200
r
540 100
a
10
```

```
index****address****end*****size****
-------------------------------------
 0         0         639       640
-------------------------------------
Enter the way (best or first (b/f))
b
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
100
Allocation Success! ADDRESS=  540

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         539       540
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         540       639       100
-------------------------------------
```

（1）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
200
Allocation Success! ADDRESS=  340

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         339       340
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         540       639       100
-------------------------------------
-------------------------------------
 1         340       539       200
-------------------------------------
```

（2）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
r
Input address and Size:
540 100

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         540       639       100
-------------------------------------
-------------------------------------
 1         0         339       340
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         340       539       200
-------------------------------------
```

（3）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
10
Allocation Success! ADDRESS=  630

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         540       629       90
-------------------------------------
-------------------------------------
 1         0         339       340
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         340       539       200
-------------------------------------
-------------------------------------
 1         630       639       10
-------------------------------------
```

（4）

可以发现，空闲分区表一直按照从小到大的顺序进行排列，在同时出现 540~639（size=100）和 0~339（size=340）的时候，分配算法选择将 size=10 的新作业分配到 540~639（size=100），符合最佳适应算法。

教务处制

首先适应算法：

依次输入右图除模式（第一个字母）外，与上面**同样的**数据，运行结果如下：

```
f
a
100
a
200
r
540 100
a
10
```

```
index****address****end*****size****
-------------------------------------
 0         0         639       640
-------------------------------------
Enter the way (best or first (b/f))
f
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
100
Allocation Success! ADDRESS=  540

********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         539       540
-------------------------------------

********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         540       639       100
-------------------------------------
```

（1）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
200
Allocation Success! ADDRESS=  340

********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         339       340
-------------------------------------

********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         540       639       100
-------------------------------------
 1         340       539       200
-------------------------------------
```

（2）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
r
Input address and Size:
540 100

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         339       340
-------------------------------------
-------------------------------------
 1         540       639       100
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         340       539       200
-------------------------------------
```

（3）

```
Enter the allocate or reclaim (a/r),or press other key to exit.
a
Input application:
10
Allocation Success! ADDRESS=  330

*********Unallocated Table*************
index****address****end*****size****
-------------------------------------
 0         0         329       330
-------------------------------------
-------------------------------------
 1         540       639       100
-------------------------------------

*********Allocated Table***  **********
index****address****end*****size****
-------------------------------------
 0         340       539       200
-------------------------------------
-------------------------------------
 1         330       339       10
-------------------------------------
```

（4）

可以发现，空闲分区表一直按照分区首地址顺序进行排序，在同时出现 540~639（size=100）和 0~339（size=340）的时候，分配算法选择将 size=10 的新作业分配到 0~339（size=340），符合首先适应算法。

2. （页面置换算法的模拟）

```
kh@ubuntu:~/test/test3$ ./replacement
 4 page frames FIFO:0.5188 LRU:0.5250 OPT:0.5125
 5 page frames FIFO:0.5344 LRU:0.5437 OPT:0.5281
 6 page frames FIFO:0.5531 LRU:0.5531 OPT:0.5375
 7 page frames FIFO:0.5625 LRU:0.5656 OPT:0.5531
 8 page frames FIFO:0.5844 LRU:0.5938 OPT:0.5688
 9 page frames FIFO:0.5969 LRU:0.6031 OPT:0.5906
10 page frames FIFO:0.6250 LRU:0.6062 OPT:0.6031
11 page frames FIFO:0.6375 LRU:0.6094 OPT:0.6187
12 page frames FIFO:0.6500 LRU:0.6344 OPT:0.6469
13 page frames FIFO:0.6719 LRU:0.6625 OPT:0.6531
14 page frames FIFO:0.6875 LRU:0.6812 OPT:0.6687
15 page frames FIFO:0.6969 LRU:0.6969 OPT:0.6781
16 page frames FIFO:0.7031 LRU:0.7156 OPT:0.6969
17 page frames FIFO:0.7125 LRU:0.7281 OPT:0.7219
18 page frames FIFO:0.7188 LRU:0.7469 OPT:0.7375
19 page frames FIFO:0.7469 LRU:0.7563 OPT:0.7406
20 page frames FIFO:0.7500 LRU:0.7594 OPT:0.7563
21 page frames FIFO:0.7688 LRU:0.7625 OPT:0.7688
22 page frames FIFO:0.7750 LRU:0.7750 OPT:0.7906
23 page frames FIFO:0.7844 LRU:0.7937 OPT:0.8062
24 page frames FIFO:0.8156 LRU:0.8062 OPT:0.8281
25 page frames FIFO:0.8250 LRU:0.8219 OPT:0.8375
26 page frames FIFO:0.8344 LRU:0.8281 OPT:0.8469
27 page frames FIFO:0.8500 LRU:0.8406 OPT:0.8531
28 page frames FIFO:0.8562 LRU:0.8500 OPT:0.8719
29 page frames FIFO:0.8656 LRU:0.8656 OPT:0.8813
30 page frames FIFO:0.8938 LRU:0.8719 OPT:0.8844
31 page frames FIFO:0.8938 LRU:0.8875 OPT:0.8969
32 page frames FIFO:0.9000 LRU:0.9000 OPT:0.9000
```

可以发现，当内存页面比较少的时候，命中率较低，随着内存页面的增多，三种算法的命中率都得到提高，OPT 算法在此数据集下效果略好。

五、 实验总结

1. 课后问题

（1）分析比较各种页面置换算法之间的差异。

FIFO实现方便，缺页率可以较高；OPT性能最佳，但在现实中无法实现；LRU实现时较复杂，且需要硬件支持，现实中常用近似算法如LFU等，但性能较靠近OPT算法。

2. 实验总结

本次是关于内存管理的实验，在动态分区分配方式的模拟实验中，学习了有关 Linux 内存分配、映射与取消映射的函数，同时，利用这些函数编模拟实现了首次/最佳/最坏适应算法的内存块分配和回收，对于分区分配机制有了更深的理解，最佳适应算法往往可以解决大作业的分配问题，但是容易产生不可利用的小空闲区，首先适应算法实现简单，但随着低端分区不断地划分而产生过多的小地址碎片,每次分配时查找时间开销会增大。

在模拟页面置换算法的实验中，回顾了三种置换算法（FIFO/OPT/LRU），利用结构体定义链表切实地解决了页面置换的问题，对于自定义链表和置换算法有了更深的理解。