华东师范大学数据学院上机实践报告

课程名称：操作系统　　**年级：**2019 级　　**上机实践成绩：**

指导教师：翁楚良　　**姓名：**龚敬洋

上机实践名称：内存管理　　**学号：**10195501436　　**上机实践日期：**2021/6/8

上机实践编号：

一、 目的

修改 Minix3.1.2a 的内存分配机制，使得当调用 brk 系统调用时，系统重新给进程分配一块更大的空间并将数据复制至新空间中。

二、 内容与设计思想

1. 将 Minix 系统中的内存分配机制由 First-Fit 修改为 Best-Fit。
2. 修改 brk 系统调用行为，使得当被调用时系统重新开辟空间并分配给进程。

三、 使用环境

开发环境：Clion 2020.3

宿主机系统环境：Mac OS Big Sur 11.2.3

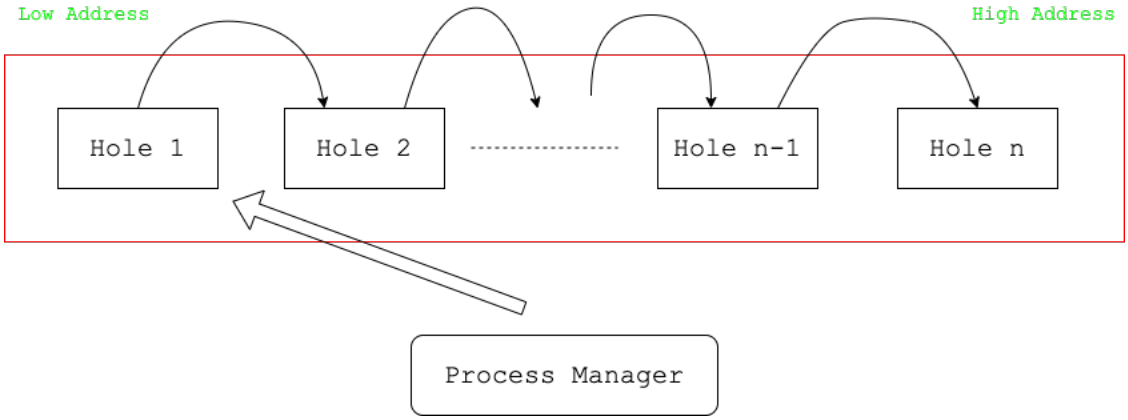
虚拟机应用：VirtualBox 6.1.18

虚拟机环境：Minix 3.1.2a

四、 实验过程

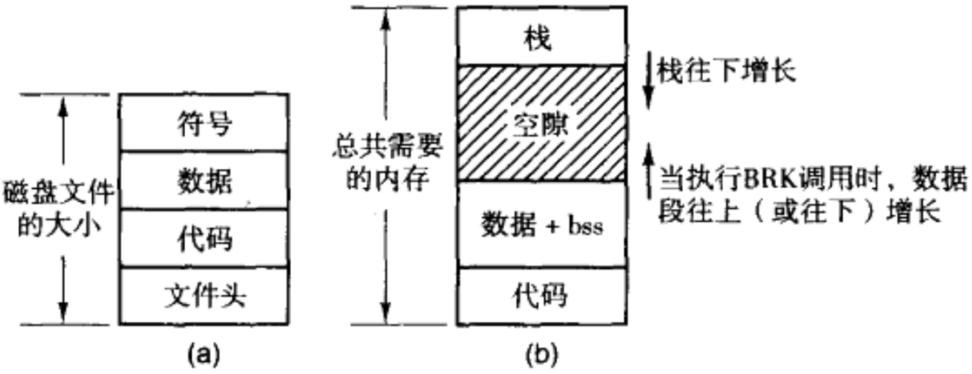
1. Minix3.1.2a 的内存管理策略

在较早的 Minix3 版本中，内存管理机制是十分固定和清晰的（Minix3.1.4 引入了页式存储管理，使得分配给进程的内存空间可能实际分布在内存地址的各个地方）。系统的进程管理器（Process Manager）维护一个空闲空间列表，根据内存地址从低到高排列：



当一个进程请求内存时，进程管理器会采用最先匹配法在空闲列表中找到第一个符合要求的空闲区，并将其分配给该进程。一旦进程被装入内存后，该片内存空间就被固定了下来，不会再被扩展。

Minix3 的程序大多被编译为进程的各个部分共用一个内存块的形式以方便作为一个整体进行加载，其中，栈和数据/代码段分别位于内存空间的顶部和底部，整体结构大体如下：



2. 内存分配原则修改

Minix 系统中的空闲块由一个链接来维护其元信息，其单元结构如下：（include/minix/type.h）

```
/* Memory allocation by PM. */
struct hole {
  struct hole *h_next;          /* pointer to next entry on the list */
  phys_clicks h_base;           /* where does the hole begin? */
  phys_clicks h_len;            /* how big is the hole? */
};
```

在 Minix3.1.2a 中，内存分配遵循首次适配原则（First-Fit），即在遍历到第一个能够容纳该进程的空闲块时，便将这一空闲块分配给进程。

现在我们来将这一分配机制修改为最优适配原则（Best-Fit）。Best-Fit 原则需要遍历整个空闲块链表，找出与进程所需空间最接近的空闲块。由于要保证空闲块大小大于进程所需内存大小，因此我们采用外部逼近的更新思路来实现。若在现有的空闲块中能够找到满足要求的空闲块，我们还需要更新空闲块的信息并将已被完全分配的空闲块从空闲链表中移出：（servers/pm/alloc.c）

```
PUBLIC phys_clicks alloc_mem(clicks)
phys_clicks clicks;     /* amount of memory requested */
{
    register struct hole *hp, *prev_ptr,*best,*prev_best;
    phys_clicks old_base,best_clicks;
    int flag= 0;

    do {
        prev_ptr = NIL_HOLE;
        hp = hole_head;
        //Procedure of finding the best-fit block
        while (hp != NIL_HOLE && hp->h_base < swap_base) {
            if (hp->h_len >= clicks) {
                if(!flag){
                    best = hp;
                    prev_best=prev_ptr;
                    flag=1;
                    best_clicks=best->h_len;
                }
                else if(flag && hp->h_len<best_clicks){
                    best=hp;
                    prev_best=prev_ptr;
                    best_clicks=best->h_len;
                }
            }
            prev_ptr = hp;
```

```
            hp = hp->h_next;
        }
    } while (swap_out());       /* try to swap some other process out */
    //Update the status of the hole
    if (flag){
        old_base = best->h_base;
        best->h_base += clicks;
        best->h_len -= clicks;

        if (best->h_base > high_watermark)
        high_watermark = best->h_base;

        if (best->h_len == 0) del_slot(prev_best,best);

        return(old_base);

    }
    return(NO_MEM);
}
```

3. 内存申请行为修改

在 Minix3.1.2 中，系统为进程分配的内存空间是不可变的。一旦进程使用完了分配的空间，程序便将报错退出，这可以在系统代码中直观的体现出来：（servers/pm/break.c）

```
PUBLIC int adjust(rmp, data_clicks, sp)
register struct mproc *rmp; /* whose memory is being adjusted? */
vir_clicks data_clicks;     /* how big is data segment to become? */
vir_bytes sp;           /* new value of sp */
{
  //Irrelevant code
  if (lower < gap_base) return(ENOMEM);  /* data and stack collided */
  //Irrelevant code
}
```

现在我们来修改这一行为。首先我们定义一个新的局部函数用于分配新的内存：（servers/pm/break.c）

```
PUBLIC int allocate_new_mem(rmp,old_clicks)
register struct mproc *rmp; //Pointer of target process
phys_clicks old_clicks; //Original space size
```

并在检测到程序空间不足时调用这一函数：（servers/pm/break.c）

```
#define ERROR 0

PUBLIC int adjust(rmp, data_clicks, sp)
register struct mproc *rmp; /* whose memory is being adjusted? */
vir_clicks data_clicks;     /* how big is data segment to become? */
vir_bytes sp;           /* new value of sp */
{
    //Irrelevant code
    if (lower < gap_base) {/* data and stack collided */
      if (allocate_new_mem(rmp, (phys_clicks)(mem_sp->mem_vir+mem_sp->mem_len-
mem_dp->mem_vir)) == ERROR)
            return(ENOMEM);
    }
    //Irrelevant code
}
```

现在我们来实现内存空间的更新。allocate_new_mem 中需要完成以下几个任务：

1） 分配一块比原先更大的内存空间
2） 将远数据段和栈段分别复制至新空间的对应位置
3） 释放原内存空间
4） 通知系统映射新内存段

受**动态表（Dynamic Table）**的启发，我们在每次需要扩展空间时将空间大小扩展至原来的两倍。在 Minix 中，进程的栈段和数据段基地址均被存放在其进程管理表（Process Management Table）中：（servers/pm/mproc.h）

```
EXTERN struct mproc {
  struct mem_map mp_seg[NR_LOCAL_SEGS]; /* points to text, data, stack */
  //Irrelevant codes
}
```

其中 mp_seg[1]为数据段基地址，mp_seg[2]为栈段基地址，因此我们可以直接读取这一地址并根据新分配的空间大小计算出新的基地址。对于内存内容拷贝，Minix 提供了一个现成的 sys_abscopy 函数可供我们使用：（include/minix/syslib.h）

```
#define sys_abscopy(src_phys, dst_phys, bytes) \
    sys_physcopy(NONE, PHYS_SEG, src_phys, NONE, PHYS_SEG, dst_phys, bytes)
_PROTOTYPE(int sys_physcopy, (int src_proc, int src_seg, vir_bytes src_vir,
    int dst_proc, int dst_seg, vir_bytes dst_vir, phys_bytes bytes));
```

同样，对于内存释放，系统也封装了相应的函数：（servers/pm/proto.h）

```
_PROTOTYPE( void free_mem, (phys_clicks base, phys_clicks clicks)   );
```

于是我们便可以快速实现这一完整逻辑：

```
PUBLIC int allocate_new_mem(rmp,old_clicks)
register struct mproc *rmp;
phys_clicks old_clicks;
{
    register struct mem_map *mem_sp, *mem_dp;
    phys_clicks new_clicks, old_base,new_base;
    phys_clicks  old_stack_base,new_stack_base;


    phys_bytes data_bytes,stak_bytes;
    phys_bytes old_base_bytes,new_base_bytes;
    phys_bytes old_stack_base_bytes,new_stack_base_bytes;

    int x;

    mem_dp = &rmp->mp_seg[D];    /* Pointer to data segment */
    mem_sp = &rmp->mp_seg[S];  /* Pointer to stack segment */

    old_base=mem_dp->mem_phys;
    old_stack_base=mem_sp->mem_phys;

    data_bytes=(phys_bytes) mem_dp->mem_len << CLICK_SHIFT;
    stak_bytes=(phys_bytes) mem_sp->mem_len << CLICK_SHIFT;
    old_base_bytes=old_base << CLICK_SHIFT;
    old_stack_base_bytes=old_stack_base << CLICK_SHIFT;

    new_clicks=2*old_clicks;
    new_base=alloc_mem(new_clicks);
    if(new_base==NO_MEM){
      return(ERROR);
```

```
31.     }
32.
33.     new_base_bytes = (phys_bytes) new_base << CLICK_SHIFT;
34.
35.     if ((x=sys_memset(0,new_base_bytes,(new_clicks<<CLICK_SHIFT)))!=OK){
36.       panic(__FILE__,"new mem can't be zero",x);
37.     }
38.
39.
40.     new_stack_base=new_base+new_clicks-mem_sp->mem_len;
41.     new_stack_base_bytes=new_stack_base << CLICK_SHIFT;
42.
43.
44.
45.     x = sys_abscopy(old_base_bytes,new_base_bytes,data_bytes);
46.     if (x < 0 ) panic(__FILE__,"allocate_new_mem can't copy",x);
47.     x = sys_abscopy( old_stack_base_bytes,new_stack_base_bytes,stak_bytes);
48.     if ( x < 0 ) panic(__FILE__,"allocate_new_mem can't copy",x);
49.
50.
51.     rmp->mp_seg[D].mem_phys = new_base;
52.     rmp->mp_seg[S].mem_phys = new_stack_base;
53.     rmp->mp_seg[S].mem_vir = mem_dp->mem_vir+new_clicks-mem_sp->mem_len;
54.     free_mem(old_base,old_clicks);
55.   return (1);
56. }
```

4. 功能测试

现在我们可以来重编译系统并测试新实现的内存扩展功能了。由于 Minix3.1.2a 开发时间较早，需要手动安装新内核并将其加入开机菜单中：

```
1.  /usr/src/servers> make image
2.  /usr/src/tools> make hdboot
3.  /usr/src/tools> make install
4.  d0p0s0> newminix(5,start new kernel) {image=/boot/image/3.1.2ar1;boot;}
```

需要注意的是，Minix3.1.2a 不支持 VirtualBox 的网卡配置，因此若使用 VirtualBox 进行调试，将无法通过主机使用 SSH 服务与 Minix 联通，需手动在虚拟机环境内修改代码。

首先我们对使用 sbrk 调用对内存分配进行基本的测试，测试代码如下：（test1.c）

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  int inc = 1;
4.  int total = 0;
5.  int i;
6.  char *sbrk(int incr);
7.  char *result;
8.  int main(int argc, int **argv)
9.  {
10.         while (((int)(result = sbrk(inc))) > 0)
11.         {
12.                 total += inc;
13.                 printf("incremented by %d, total %d\n", inc, total);
14.                 inc += inc;
15.         }
16.         return 0;
17. }
```

随后，我们实际访问新分配的内存，验证其分配空间是否能够正常使用：（test2.c）

```c
#include <stdio.h>
#include <unistd.h>
int inc = 1;
int total = 0;
char *sbrk(int incr);
char *result;
int i;
int main(int argc, int **argv)
{
    while (((int)(result = sbrk(inc))) > 0)
    {
        for (i = 0; i < inc; i++)
            result[i] = 0x12;
        total += inc;
        printf("incremented by: %d, total: %d , result: %d\n", inc, total,   (int)result);

        inc += inc;
    }
    exit (0);
}
```

经过测试可以发现，程序输出与预期相符，且两次分配内存大小相同，表明新实现的内存分配机制是有效的。程序输出结果如下：

```
incremented by 1, total 1
incremented by 2, total 3
incremented by 4, total 7
incremented by 8, total 15
incremented by 16, total 31
incremented by 32, total 63
incremented by 64, total 127
incremented by 128, total 255
incremented by 256, total 511
incremented by 512, total 1023
incremented by 1024, total 2047
incremented by 2048, total 4095
incremented by 4096, total 8191
incremented by 8192, total 16383
incremented by 16384, total 32767
incremented by 32768, total 65535
incremented by 65536, total 131071
incremented by 131072, total 262143
incremented by 262144, total 524287
incremented by 524288, total 1048575
incremented by 1048576, total 2097151
incremented by 2097152, total 4194303
incremented by 4194304, total 8388607
incremented by 8388608, total 16777215
incremented by 16777216, total 33554431
incremented by 33554432, total 67108863
```

```
incremented by: 1, total: 1 , result: 760
incremented by: 2, total: 3 , result: 4096
incremented by: 4, total: 7 , result: 4098
incremented by: 8, total: 15 , result: 4102
incremented by: 16, total: 31 , result: 4110
incremented by: 32, total: 63 , result: 4126
incremented by: 64, total: 127 , result: 4158
incremented by: 128, total: 255 , result: 4222
incremented by: 256, total: 511 , result: 4350
incremented by: 512, total: 1023 , result: 4606
incremented by: 1024, total: 2047 , result: 5118
incremented by: 2048, total: 4095 , result: 6142
incremented by: 4096, total: 8191 , result: 8190
incremented by: 8192, total: 16383 , result: 12286
incremented by: 16384, total: 32767 , result: 20478
incremented by: 32768, total: 65535 , result: 36862
incremented by: 65536, total: 131071 , result: 69630
incremented by: 131072, total: 262143 , result: 135166
incremented by: 262144, total: 524287 , result: 266238
incremented by: 524288, total: 1048575 , result: 528382
incremented by: 1048576, total: 2097151 , result: 1052670
incremented by: 2097152, total: 4194303 , result: 2101246
incremented by: 4194304, total: 8388607 , result: 4198398
incremented by: 8388608, total: 16777215 , result: 8392702
incremented by: 16777216, total: 33554431 , result: 16781310
incremented by: 33554432, total: 67108863 , result: 33558526
```

五、 总结

在本实验中，我们在 Minix3.1.2a 系统下对内存分配机制进行了修改。将 first-fit 内存分配策略修改为 best-fit 策略，可以有效的提高内存的综合利用率，减少内存碎片的产生。通过对 brk 系统调用实现的修改，得以让程序能够得到的内存空间随着需求动态扩展，极大的增强了系统的通用性和可扩展性。这一实验也使得我们对进程内存管理和内存空间调度的相关知识有了更深刻的了解。