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：**2020 级　　**上机实践成绩：**

指导教师：翁楚良　　**姓名：**张熙翔

上机实践名称： 内存管理　　**学号：**10205501427　　**上机实践日期：**2022/6/10

一、 目的

1.熟悉 Minix 操作系统的进程管理
2.学习 Unix 风格的内存管理

二、 内容与设计思想

修改 Minix3.1.2a 的进程管理器，改进 brk 系统调用的实现，使得分配给进程的数据段+栈段空间耗尽时，brk 系统调用给该进程分配一个更大的内存空间，并将原来空间中的数据复制至新分配的内存空间，释放原来的内存空间，并通知内核映射新分配的内存段。

三、 使用环境

VMware Workstation Pro
Visual studio code
Xftp

四、 实验过程

1.修改内存分配：

1.1 修改/usr/src/servers/pm/alloc.c 中的 alloc_mem 函数，把 first-fit 修改成 best-fit，即分配内存之前，先遍历整个空闲内存块列表，找到最佳匹配的空闲块。

每个空闲区表有三个字段。地址和长度以 click 为单位，click 是 1024 字节。

```
PRIVATE struct hole {
    phys_clicks h_base;           /* 空洞的开始地址*/
    phys_clicks h_len;            /* 空洞的长度 */
    struct hole *h_next;          /* 指向下一个空洞 */
} hole[NR_HOLES];
```

在分配内存之前，先遍历整个空闲内存块列表，找到最佳匹配的空闲块，也就是说在 hp->h_len<clicks 的空闲块中，找到最小的。用 temp 标记第一个符合条件的块，后面通过块的大小是否更合适进行比较，更新 temp。

```
PUBLIC phys_clicks alloc_mem(clicks)
    phys_clicks clicks; /* 请求的内存 amount of memory requested */
{
    register struct hole *hp, *temp,*temp_ptr,*prev_ptr;
    /*hp 存放最合适的，prev_ptr 存放最合适前面的一个，temp 链表存储最有内存块*/
  phys_clicks old_base;
```

```
  do { /*单向链表从头找*/
    temp_ptr = hole_head;
    prev_ptr=NIL_HOLE; /*空的*/
    temp=hole_head;
    hp = hole_head;
    while (hp != NIL_HOLE && hp->h_base < swap_base) {
        /*先找到一个满足要求的空洞*/
        if (hp->h_len >= clicks&&temp==hole_head) {
            temp=hp;
            temp_ptr=prev_ptr;
        }
        /*找比第一个更小的更合适的*/
        if((hp->h_len >= clicks&&hp->h_len<temp->h_len)) {
            temp=hp;
            temp_ptr=prev_ptr;
        }
        /*遍历链表*/
        prev_ptr = hp;
        hp = hp->h_next;
    } if(temp!=hole_head){ /*如果找到了更合适的*/
        hp=temp;
        prev_ptr=temp_ptr;
           /* We found a hole that is big enough.  Use it. */
           old_base = hp->h_base;  /*返回值 块的起始位置*/
           hp->h_base += clicks;    /* 块的起始地址 */
           hp->h_len -= clicks;    /* 块的大小 */

           /*更新被使用内存的最高地址*/
           if(hp->h_base > high_watermark)
               high_watermark = temp->h_base;
               /*如果刚好用完了，就删除*/
            if(hp->h_len==0)
            del_slot(prev_ptr,hp);

           /* Return the start address of the acquired block. */
           return(old_base);
       }
  } while (swap_out());
   /* try to swap some other process out (wait,pause.....)*/
  return(NO_MEM);
}
```

如果 hp->base 达到了 swap_base，就需要用 while(swap_out())换出一些块。

swap机制其实就是将外存(如硬盘)当内存使用, 怎么可以把外存当内存使用呢? 原理就是当系统内存不够用的时候, 内核会选择某些进程, 把其使用较少的内存的内容交换(swap)到外存中,然后把内存让给需要的进程使用.

1.2 修改/usr/src/servers/pm/break.c 中的 adjust 函数，并增加了一个 allocate_new_mem 局部函数在 adjust 函数中调用。brk 系统调用流程：

若足够，则调整数据段指针，堆栈指针；通知内核程序的映像发生了变化，返回 do_brk 函数。

若不够，调用 allocate_new_mem 函数申请新的足够大的内存空间；将程序现有的数据

段和堆栈段的内容分别拷贝至新内存区域的底部(bottom)和顶部(top)；通知内核程序的映像发生了变化；返回 do_brk 函数。

lower 栈顶指针，gap_base 数据段的边界。原来的 adjust 函数中当栈段地址和数据段地址有重叠时直接返回 ENOMEM, do_brk 不能继续为数据段增加空间。

```
  if (lower < gap_base) return(ENOMEM); /* data and stack collided */
```

修改之后，当 lower < gap_base 时，调用新增的 allocate_new_mem 函数申请新的足够够大的空间，将进程复制到新的内存空间并通知内核程序的数据映像发生了改变。

```
if (lower < gap_base) /* data and stack collided */
  if(allocate_new_mem(rmp,(phys_clicks)(rmp->mp_seg[S].mem_vir -
rmp->mp_seg[D].mem_vir + rmp->mp_seg[S].mem_len))==0)
        return(ENOMEM);
```

简化 allocate_new_mem 函数，如果 if 条件判断不成立，继续进行 adjust，判断新的栈段、数据段大小是否适合地址空间，适合则调用 sys_newmap 函数更新虚拟地址到物理地址的映射；不合适则恢复栈段、数据段到旧的空间。

allocate_new_mem 函数：

1. 分配更大内存
2. 计算新分配的栈段和内存段的地址
3. sys_abscopy 将原来的数据段和栈段分别放入新内存块的底部和顶部。（源地址，目的地址，字节数）
4. free_mem 释放原来空间（起始位置，空间大小）
5. 更新进程表中地址

```
PUBLIC int allocate_new_mem(rmp,clicks)
register struct mproc *rmp;  /*pm 的进程表*/
phys_clicks clicks;
{
    register struct mem_map *mem_sp, *mem_dp; /*stack data*/

    /* 数据段旧起始地址，内存旧长度，新起始地址，新内存长度 */
    phys_clicks old_clicks, old_base;
    phys_clicks new_clicks, new_base;
    /* 栈段旧起始地址，新起始地址，长度 */
    phys_clicks new_s_base;
    phys_clicks old_s_base, stak_clicks;
    /* 地址和长度都是以 click 为单位。click 是 1024 个字节。
     字节版本*/
    phys_bytes old_bytes, data_bytes;
    phys_bytes stak_bytes;
    phys_bytes old_d_tran, new_d_tran;
    phys_bytes old_s_tran, new_s_tran;


    mem_dp = &rmp->mp_seg[D];/* 指向数据段 */
    mem_sp = &rmp->mp_seg[S];/* 指向栈段 */
    /* 申请为原来的 2 倍*/
    old_clicks = clicks;
```

```
    new_clicks = clicks * 2;
    if ((new_base = alloc_mem(new_clicks))==NO_MEM){
        return (ENOMEM);
    }
    /* 得到原来栈段和数据段的地址和大小 */
    data_bytes = (phys_bytes) rmp->mp_seg[D].mem_len << CLICK_SHIFT;/* 字节 click 要左
移 CLICK_SHIFT */
    stak_bytes = (phys_bytes) rmp->mp_seg[S].mem_len << CLICK_SHIFT;
    old_base = rmp->mp_seg[D].mem_phys;/* 以前的数据段的物理地址 */
    old_s_base = rmp->mp_seg[S].mem_phys;/* 以前的栈段的物理地址 */
    old_d_tran = (phys_bytes)old_base << CLICK_SHIFT;
    old_s_tran = (phys_bytes)old_s_base << CLICK_SHIFT;
    /* 计算得到新的栈端和数据段的地址 */
    new_s_base = new_base + new_clicks - mem_sp->mem_len;
    /* 数据段起始+内存大小再减去栈的大小 自下向上复制*/
    new_d_tran = (phys_bytes) new_base << CLICK_SHIFT; /* 数据段的起始地址化为 byte 单位
*/
    new_s_tran = (phys_bytes)new_s_base << CLICK_SHIFT;/*栈段的*/
    /* 之所以要化成字节因为后面在调用 sys_memset，复制的时候都需要以字节为单位，而不是 click
     调用 sys_memset 函数用 0 填充新获得的内存 */
    sys_memset(0,new_d_tran,(new_clicks<<CLICK_SHIFT));
    /* 将数据段和栈段分别复制到新的内存空间的底部和顶部 */
    d = sys_abscopy(old_d_tran,new_d_tran,data_bytes);
    if (d < 0)
        panic(__FILE__,"allocate_new_mem can't copy",d);
    s = sys_abscopy(old_s_tran,new_s_tran,stak_bytes);
    if (s < 0)
        panic(__FILE__,"allocate_new_mem can't copy",s);
    /* 更新进程数据段和栈段的内存地址以及栈段的虚拟地址 */
    rmp->mp_seg[D].mem_phys = new_base; /* 数据段的物理地址 虚拟为 0*/
    rmp->mp_seg[S].mem_phys = new_s_base;  /* 栈段的物理地址 */
    rmp->mp_seg[S].mem_vir = rmp->mp_seg[D].mem_vir + new_clicks - mem_sp->mem_len;
    /* 释放原来内存 */
    free_mem(old_base,old_clicks);
    return (OK);
}
```

2. 编译 MINIX

1.1 进入/usr/src/servers 目录，输入 make image，等编译成功之后输入 make install 安装新的 PM 程序。

1.2 进入/usr/src/tools 目录，输入 make hdboot，成功之后再键入 make install 命令安装新的内核程序。

1.3 键入 shutdown 命令关闭虚拟机，进入 boot monitor 界面。设置启动新内核的选项，在提示符键入： newminix(5,start new kernel) {image=/boot/image/3.1.2ar1;boot;}

1.4 然后回车，键入 save 命令保存设置。 5 为启动菜单中的选择内核版本的键(数字键，可选其他数字键)，3.1.2ar1 为在/usr/src/tools 目录中输入 make install 之后生成的内核版本号，请记得在/usr/src/tools 中执行 make install 命令之后记录生成的新内核版本号。

1.5 输入 menu 命令，然后敲数字键（上一步骤中设置的数字）启动新内核，登录进 minix 3 中测试。

实验结果：

修改前：

```
# ./test1
incremented by 1, total 1 , result + inc 761
incremented by 2, total 3 , result + inc 4098
incremented by 4, total 7 , result + inc 4102
incremented by 8, total 15 , result + inc 4110
incremented by 16, total 31 , result + inc 4126
incremented by 32, total 63 , result + inc 4158
incremented by 64, total 127 , result + inc 4222
incremented by 128, total 255 , result + inc 4350
incremented by 256, total 511 , result + inc 4606
incremented by 512, total 1023 , result + inc 5118
incremented by 1024, total 2047 , result + inc 6142
incremented by 2048, total 4095 , result + inc 8190
incremented by 4096, total 8191 , result + inc 12286
incremented by 8192, total 16383 , result + inc 20478
incremented by 16384, total 32767 , result + inc 36862
incremented by 32768, total 65535 , result + inc 69630
#
```

```
# ./test2
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
#
```

修改后：

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

```
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
# _
```

测试程序最后的输出结果和分配给 minix 操作系统的物理内存数量有关。通过修改 alloc_mem(分配策略) 和 adjust(调整策略) 之后可以申请到更多的内存空间。