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、实验过程

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。

修改usr/src/servers/pm中的alloc.c和break.c.

在usr/src/servers中make image并make install.

在usr/src/tools中make hdboot并make install.

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/3.1.2ar23;boot;}

键入save,键入menu，选择1在原内核里编译测试文件。

然后shutdown,键入menu，选择5在新内核里运行测试文件。

错误的sysnewmap参数会导致：

```
# ./test2_5
incremented by 1, total 1 , result + inc 765
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
PM panic (break.c): couldn't sys_newmap in adjust in alloc_new_mem: -22
```

正确的传参为：

```
if ((e = sys_newmap(rmp->mp_endpoint, rmp->mp_seg)) !=
OK)
panic(__FILE__, "couldn't sys_newmap in alloc_new_mem",
e);
```

在测试文件2的运行中，出现了如图错误。

```
# ./test2
incremented by 1, total 1 , result + inc 765
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
Memory fault - core dumped
```

解决方法是：

把src源码下载下来，用全局搜索找到含memory fault的文件

```
#define MAXNBLOCKS 10   /* Limit the number of blocks that yap will use
                * the input in core.
                * This was needed to let yap run on an IBM XT
                * running PC/IX. The problem is that malloc can
                * allocate almost all available space, leaving no
                * space for the stack, which causes a memory fault.
                * Internal yap blocks are 2K, but there is a lot of
                * additional information that yap keeps around. You
                * can also use it if you want to limit yap's maximum
                * size. If defined, it should be at least 3.
                * 10 is probably a reasonable number.
                */
```

```
If disaster is called some of the system parameters imported into ps are
probably wrong.  This tends to result in memory faults.
*/
```

通过printf打印发现。测试程序2的时候

```
free the old memory space
incremented by 262144, total 524287 , result + inc 528382
change stack segment
free the old memory space
incremented by 524288, total 1048575 , result + inc 1052670
change stack segment
free the old memory space
incremented by 1048576, total 2097151 , result + inc 2101246
change stack segment
free the old memory space
incremented by 2097152, total 4194303 , result + inc 4198398
change stack segment
free the old memory space
incremented by 4194304, total 8388607 , result + inc 8392702
change stack segment
free the old memory space
incremented by 8388608, total 16777215 , result + inc 16781310
change stack segment
free the old memory space
incremented by 16777216, total 33554431 , result + inc 33558526
change stack segment
free the old memory space
incremented by 33554432, total 67108863 , result + inc 67112958
allocate ---error!allocate ---error!Memory fault - core dumped
* _
```

修改测试代码test2.c

首先明确sbrk的用途。

> sbrk不是系统调用，是C库函数。系统调用通常提供一种最小功能，而库函数通常提供比较复杂的功能。sbrk/brk是从堆中分配空间，本质是移动一个位置，向后移就是分配空间，向前移就是释放空间，sbrk用相对的整数值确定位置，如果这个整数是正数，会从当前位置向后移若干字节，如果为负数就向前若干字节。在任何情况下，返回值永远是移动之前的位置

修改方法一：将return 0;改成exit(0);

原来出现返回两次的原因是:数据段顶继续赋值，把上面栈段的内容修改。main中的return(0);返回栈段的内容。而栈段内容已经被修改，所以返回时会出现coredump。即越界访问。

另一种修改方法，新的数据段顶初始化不初始化到栈段的内容，只初始化一部分。也不会出现coredump.

三、内容与设计思想

3.1 alloc.c

在alloc_mem中，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。用temp标记第一个符合条件的块，后面的通过块的大小是否更合适进行比较，更新temp。

初始时temp、temp的前一个以及hp都在链表头部。

```
temp_ptr = hole_head;
      prev_ptr=NIL_HOLE;
      temp=hole_head;
  hp = hole_head;
```

在elseif中判断是否有一个更小的符合条件的块，如果有就更新空闲块。

```
while (hp != NIL_HOLE && hp->h_base < swap_base)
        {
            /*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;
        }
        /*以第一个的长度作为参照，找到比原来更合适的就记录下来*/
        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;
            }
```

通过遍历链表temp中储存最优的内存块

```
if(temp!=hole_head){
        /*用hp存最合适的，同时保存最合适的前一个*/
        hp=temp;
        prev_ptr=temp_ptr;
           /* We found a hole that is big enough.  Use
it. */
           old_base = hp->h_base;    /* remember where
it started */
           hp->h_base += clicks;    /* bite a piece off
*/
           hp->h_len -= clicks;    /* ditto */
           /* Remember new high watermark of used
memory. */
           if(hp->h_base > high_watermark)
               high_watermark = temp->h_base;
               /*完全用完了，就删除*/
            if(hp->h_len==0)
            del_slot(prev_ptr,hp);
       /* Return the start address of the acquired
block. */
           return(old_base);
       }
```

如果hp->base达到了swap_base，就要换出一些块。用while (swap_out());

best的基地址要加上已分配的clicks,可以分配的长度要减去已分配的clicks.如果基地址超过了用掉的内存的high watermark，更新high_watermark的值。如果这个空闲块被完全用掉就del_slot.最后返回请求块的起始地址。

3.2 break.c

在do_brk中调用adjust并判断返回值。

```
r = adjust(rmp, new_clicks, new_sp);
 rmp->mp_reply.reply_ptr = (r == OK ? m_in.addr : (char
*)-1);
```

在adjust中当lower < gap_base，数据段和栈段冲突，这个时候调用alloc_new_mem并判断分配是否成功，如果返回值为0，分配不成功，返回ENOMEM表示无空间。

编写alloc_new_mem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；

函数头为

```
PUBLIC int allocate_new_mem(rmp, d_clicks, delta,
clicks) register struct mproc *rmp;
phys_clicks d_clicks;
long delta;
phys_clicks clicks;
```

rmp 指向进程的 mproc 数据结构；clicks 是原来的内存空间的大小。d_clicks为adjust中的data_clicks。新的数据段长度。delta为新旧栈指针的差值，表示栈向下增长了多少。

在mproc中的定义

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

```
mem_dp = &rmp->mp_seg[D];
mem_sp = &rmp->mp_seg[S];
```

初始化栈和数据段的指针。

分配为旧地址空间2倍大的新的内存。这里分配2倍内存，有更好的分配内存的方式。在do_brk中有用户需求的数据段大小，根据用户输入的大小分配合适的内存。

调用alloc_mem并检查返回值。

```
old_clicks = clicks;
new_clicks = clicks * 2;
 if ((new_base = alloc_mem(new_clicks)) == NO_MEM)
    {
        printf("allocate ---error!");
        return (ENOMEM);
    }
```

如果没有分配成功return

得到原来栈段和数据段的地址和大小;计算新的栈段基地址，由于复制是从下向上，所以栈顶地址。

```
 data_bytes = (phys_bytes)rmp->mp_seg[D].mem_len <<
CLICK_SHIFT;
 stak_bytes = (phys_bytes)rmp->mp_seg[S].mem_len <<
CLICK_SHIFT;
 old_base = rmp->mp_seg[D].mem_phys;
 old_s_base = rmp->mp_seg[S].mem_phys;
 /*the top of the stack*/
 new_s_base = new_base + new_clicks - mem_sp->mem_len;
```

将新旧栈段和数据段基地址都转化为phys_bytes

```
 new_d_tran = (phys_bytes)new_base << CLICK_SHIFT;
    old_d_tran = (phys_bytes)old_base << CLICK_SHIFT;
    new_s_tran = (phys_bytes)new_s_base << CLICK_SHIFT;
    old_s_tran = (phys_bytes)old_s_base << CLICK_SHIFT;
```

```
c = sys_memset(0, new_d_tran, (new_clicks <<
CLICK_SHIFT));
```

用0填充新获得的内存，否则访问新的内存时会出现page fault。

将数据段和栈段分别复制到新的内存空间的底部和顶部并通过sys_abscopy返回值判断是否复制成功

```
d = sys_abscopy(old_d_tran, new_d_tran, data_bytes);
    if (d < 0)
        panic(__FILE__, " can't copy data segment in
alloc_new_mem", d);
    s = sys_abscopy(old_s_tran, new_s_tran, stak_bytes);
    if (s < 0)
        panic(__FILE__, " can't copy stack segment in
alloc_new_mem", 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;
    rmp->mp_seg[S].mem_phys = new_s_base;
    rmp->mp_seg[S].mem_vir = rmp->mp_seg[D].mem_vir +
new_clicks - mem_sp->mem_len;
```

保存当前的数据段长度

```
cur_clicks = mem_dp->mem_len;
```

接下来类似adjust中记录栈和数据段的改变、更新栈段和数据段的长度。

```
if (d_clicks != mem_dp->mem_len)
    {
        mem_dp->mem_len = d_clicks;
        changed |= DATA_CHANGED;
    }
 /* Update stack length and origin due to change in
stack pointer. */
    if (delta > 0)
    {
        printf("change stack segment\n");
        mem_sp->mem_vir -= delta;
        mem_sp->mem_phys -= delta;
        mem_sp->mem_len += delta;
        changed |= STACK_CHANGED;
    }
```

判断新的栈段数据段大小是否合适地址空间。

```
 /* Do the new data and stack segment sizes fit in the
address space? */
    ft = (rmp->mp_flags & SEPARATE);
#if (CHIP == INTEL && _WORD_SIZE == 2)
    r = size_ok(ft, rmp->mp_seg[T].mem_len, rmp-
>mp_seg[D].mem_len,
                rmp->mp_seg[S].mem_len, rmp-
>mp_seg[D].mem_vir, rmp->mp_seg[S].mem_vir);
#else
    r = (rmp->mp_seg[D].mem_vir + rmp->mp_seg[D].mem_len
>
         rmp->mp_seg[S].mem_vir)
            ? ENOMEM
            : OK;
#endif
```

如果是大小是符合的就用sys_newmap映射，映射成功，释放旧的内存空间

映射不成功panic报错。

```
if (r == OK)
    {
        int r2;
        if (changed && (r2 = sys_newmap(rmp-
>mp_endpoint, rmp->mp_seg)) != OK)
            panic(__FILE__, "couldn't sys_newmap in
adjust", r2);
        free_mem(old_base, old_clicks);
        printf("free the old memory space\n");
        return (OK);
    }
```

如果不满足size_ok，就要恢复栈数据段地址到旧的空间。

```
 printf("size not ok!\n");
    /* New sizes don't fit or require too many
page/segment registers. Restore.*/
    if (changed & DATA_CHANGED)
    {
        mem_dp->mem_len = cur_clicks;
        mem_dp->mem_phys = cur_base;
    }
    if (changed & STACK_CHANGED)
    {
        printf("restore stack segment\n");
```

```
            mem_sp->mem_vir = cur_v_base;
            mem_sp->mem_phys = cur_s_base;
            mem_sp->mem_len = cur_s_click;
        }
        return (ENOMEM);
```

四、结果

最后的allocate_error是分配不成功的打印内容

```
if ((new_base = alloc_mem(new_clicks)) == NO_MEM)
{
    printf("allocate ---error!");
    return (ENOMEM);
}
```

test1测试结果

```
change stack segment
free the old memory space
incremented by 262144, total 524287
change stack segment
free the old memory space
incremented by 524288, total 1048575
change stack segment
free the old memory space
incremented by 1048576, total 2097151
change stack segment
free the old memory space
incremented by 2097152, total 4194303
change stack segment
free the old memory space
incremented by 4194304, total 8388607
change stack segment
free the old memory space
incremented by 8388608, total 16777215
change stack segment
free the old memory space
incremented by 16777216, total 33554431
change stack segment
free the old memory space
incremented by 33554432, total 67108863
allocate ---error!# _
```

test2测试结果

```
change stack segment
free the old memory space
incremented by: 262144, total: 524287 , result: 266238
change stack segment
free the old memory space
incremented by: 524288, total: 1048575 , result: 528382
change stack segment
free the old memory space
incremented by: 1048576, total: 2097151 , result: 1052670
change stack segment
free the old memory space
incremented by: 2097152, total: 4194303 , result: 2101246
change stack segment
free the old memory space
incremented by: 4194304, total: 8388607 , result: 4198398
change stack segment
free the old memory space
incremented by: 8388608, total: 16777215 , result: 8392702
change stack segment
free the old memory space
incremented by: 16777216, total: 33554431 , result: 16781310
change stack segment
free the old memory space
incremented by: 33554432, total: 67108863 , result: 33558526
allocate ---error!#
```

五、总结

本次实验通过修改alloc.c使我熟悉了最佳适配原则的代码实现，也复习了单向链表的部分相关知识。通过break.c中do_brk到adjust再到alloc_new_mem的层层调用，实现当分配给进程的数据段+栈段空间耗尽时，brk系统调用给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。熟悉了free_mem、sys_memset、sys_newmap、sys_abscopy、alloc_mem的调用及传参过程。明白了测试代码的工作原理、以及click和byte的转化。以及怎样通过设置打印内容调试代码错误。如果虚拟机新核无法开机，怎么修改代码（这样一般是比较重大的错误、需要将代码的逻辑进一步顺一遍）。也进一步熟悉掌握了minix3.1.2编译新内核的过程。