华东师范大学数据科学与工程学院实验报告

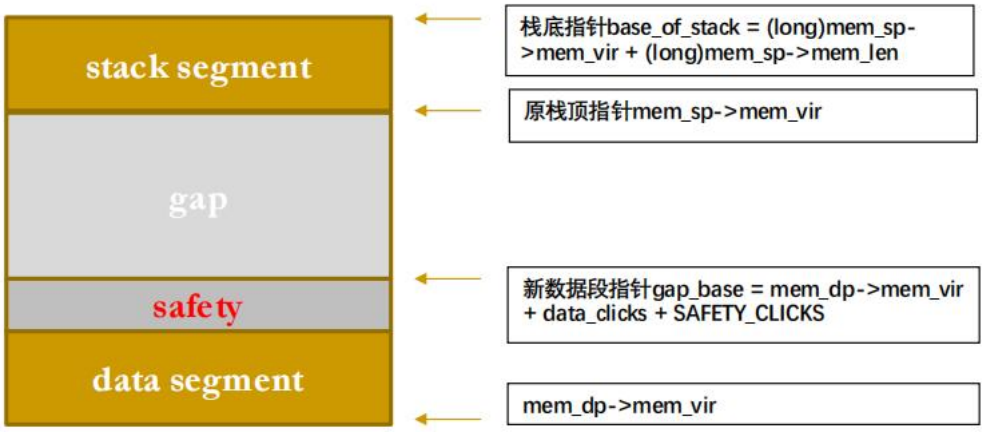
课程名称：操作系统	年级：2021	上机实践成绩：
指导教师：翁楚良	姓名：唐小卉	学号：10215501437
上机实践名称： project4		上机实践日期：2023.5.25
上机实践编号：4	组号：	上机实践时间：2023.5.25

一、 实验目的

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、 实验任务

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

修改内存分配：

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

```
PUBLIC phys_clicks alloc_mem(clicks)
phys_clicks clicks;  /* 请求的内存 */
{
  /*hole 空闲链表，按照内存地址的递增顺序列出了内存的各个空闲区域
  不过数据段和栈段的空隙不认为是空闲区，因为它们已经被分配给了进程
  每个空闲区表项有3个字段：空闲区的起始地址，空闲区的长度，一个指针指向下一个链表结点
  地址和长度都是以click为单位。click是1024个字节。该链表是一个单向链表，所以best-fit要
  头开始找。
  先遍历整个空闲内存块列表，找到最佳匹配的空闲块。一块分配给进程，一块
  给剩余的空闲区，仍然留在空闲链表上，但长度要发生变化。
  如果块的大小正好合适，那么就要调用del_slot，把这个结点从空闲链表中删除
  */
    register struct hole *hp, *prev_ptr, *best, *prev_best;
    phys_clicks old_base,best_clicks;
    int find=0; /*find 用来标识是否找到需要的内存块*/
    do {
      //从头开始找
        prev_ptr = NIL_HOLE;
        hp = hole_head;
        while (hp != NIL_HOLE && hp->h_base < swap_base){
            if (hp->h_len >= clicks){
                if (find == 0){/* 第一次找到合适的块 */
                    best = hp;                          /* 记录当前指针 */
                    prev_best = prev_ptr;               /* 记录当前指针 */
                    best_clicks = hp->h_len;            /* 更新最适块大小 */
                    find = 1;                           /* 第一次找到合适的块 */
                }else if (hp->h_len < best_clicks){ /*找到了一个更小的内存块*/
                    best = hp;                          /* 记录当前指针 */
                    prev_best = prev_ptr;               /* 记录当前指针 */
                    best_clicks = hp->h_len;            /* 更新最适块大小 */
                }}
            prev_ptr = hp;
            hp = hp->h_next;
        }
    } while (swap_out());
    //把一些像wait(), pause() or sigsuspend()这样的进程swap out
    if (find == 1)
    {
        //要将找到的空闲块一分为二了，前面一部分clicks单位的是要分配的，后面剩下来的b
>h_len要减去clicks，
        //然后其实地址也要往前移clicks
        //最后只要返回old_base
        old_base = best->h_base;
        best->h_base += clicks;                         //改变块的起始地址
        best->h_len -= clicks;                          //改变块的长度
        //high_watermark是用来记录被使用的内存的最高的地址，如果超了要更新
        if (best->h_base > high_watermark)
            high_watermark = best->h_base;
        /* 如果块的大小正好合适，即空洞的大小为0 h_len=0 */
        if (best->h_len == 0)
            del_slot(prev_best,best);
            //把best对应的块删了，pre_best的下一位指向原来best的下一位
        return(old_base);
    }
    return(NO_MEM);//否则就返回没有内存可以分配
}
```

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：

do_brk 函数计算数据段新的边界，然后调用 **adjust** 函数，**adjust** 函数计算程序当前的空闲空间是否足够分配：

- 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。

```
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段
    //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;
    //数据段的以前的起始地址，内存以前的长度，新的起始地址，内存新的长度
    phys_clicks old_clicks, old_base; //地址和长度都是以click为单位。click是1024个字节。
    phys_clicks new_clicks, new_base;
    //栈段以前的起始地址，新的起始地址，长度
    phys_clicks new_s_base;
    phys_clicks old_s_base, stak_clicks;


    mem_dp = &rmp->mp_seg[D];//指向数据段
    mem_sp = &rmp->mp_seg[S];//指向栈段
    //内存以前的长度
    old_clicks = clicks;
    //新的2倍大小的内存空间
    new_clicks = clicks * 2;
    /* 调用alloc_mem获得新的2倍大小的内存空间 */
    /* 如果不成功，不释放原来的内存空间 */
    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;/
*/因为是字节，所以得到数据段的长度要左移CLICK_SHIFT
    stak_bytes = (phys_bytes) rmp->mp_seg[S].mem_len << CLICK_SHIFT;//同理原来的栈段的长度也是
    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;//要把上面获得的click为单位的化为以byte为
*的
    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; //计算得到新的栈段的地址，单位为clic
*  只需要已经开了2倍大小的内存空间的起始地址
    //+内存大小再减去栈的大小就可以得到
    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
    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);
}
```

原先的 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，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。

```
#define SAFETY_BYTES  (384 * sizeof(char *))
#define SAFETY_CLICKS ((SAFETY_BYTES + CLICK_SIZE - 1) / CLICK_SIZE)
  gap_base = mem_dp->mem_vir + data_clicks + SAFETY_CLICKS;
  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)))
        return(ENOMEM);
```

上面 alloc.c 是分配内存，现在是内存已经分配好了，就是数据段和栈段怎么分的问题，主要会调整数据段和栈段的空隙，要么增加，要么减少。

allocate_new_mem 函数首先需要申请新的足够大的内存空间，这里是申请为原来的 2 倍，然后将程序现有的数据段、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。

编译 MINIX

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。

```
exec cc -c -I/usr/include  timers.c
exec cc -c -I/usr/include  table.c
exec cc -o fs -i main.o open.o read.o write.o pipe.o dmap.o \
        device.o path.o mount.o link.o super.o inode.o \
        cache.o cache2.o filedes.o stadir.o protect.o time.o \
        lock.o misc.o utility.o select.o timers.o table.o -lsys -lsysuti
rs
install -S 512w fs
cd ./rs && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include main.c
exec cc -c -I/usr/include manager.c
exec cc -o rs -i main.o manager.o  -lsys -lsysutil
install -S 16k rs
exec cc -c -I/usr/include service.c
exec cc -o service -i service.o -lsys
cd ./ds && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include main.c
exec cc -c -I/usr/include store.c
exec cc -o ds -i main.o store.o  -lsys -lsysutil
install -S 16k ds
cd ./init && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include -O -D_MINIX -D_POSIX_SOURCE init.c
exec cc -I/usr/include -O -D_MINIX -D_POSIX_SOURCE -o init -i init.o -ls
install -S 192w init
```

```
cc  -I. -D_MINIX -o generic/ip_lib.o -c generic/ip_lib.c
cc  -I. -D_MINIX -o generic/ip_read.o -c generic/ip_read.c
cc  -I. -D_MINIX -o generic/ip_write.o -c generic/ip_write.c
cc  -I. -D_MINIX -o generic/ipr.o -c generic/ipr.c
cc  -I. -D_MINIX -o generic/rand256.o -c generic/rand256.c
cc  -I. -D_MINIX -o generic/tcp.o -c generic/tcp.c
cc  -I. -D_MINIX -o generic/tcp_lib.o -c generic/tcp_lib.c
cc  -I. -D_MINIX -o generic/tcp_recv.o -c generic/tcp_recv.c
cc  -I. -D_MINIX -o generic/tcp_send.o -c generic/tcp_send.c
cc  -I. -D_MINIX -o generic/ip_eth.o -c generic/ip_eth.c
cc  -I. -D_MINIX -o generic/ip_ps.o -c generic/ip_ps.c
cc  -I. -D_MINIX -o generic/psip.o -c generic/psip.c
cc  -I. -D_MINIX -o minix3/queryparam.o -c minix3/queryparam.c
cc  -I. -D_MINIX -o sha2.o -c sha2.c
cc -o inet  buf.o clock.o inet.o inet_config.o \
        mnx_eth.o mq.o qp.o sr.o stacktrace.o \
        generic/udp.o generic/arp.o generic/eth.o generic/event.o \
        generic/icmp.o generic/io.o generic/ip.o generic/ip_ioctl.o \
        generic/ip_lib.o generic/ip_read.o generic/ip_write.o \
        generic/ipr.o generic/rand256.o generic/tcp.o generic/tcp_lib.o \
        generic/tcp_recv.o generic/tcp_send.o generic/ip_eth.o \
        generic/ip_ps.o generic/psip.o \
        minix3/queryparam.o sha2.o version.c -lsys -lsysutil
install -c inet /usr/sbin/inet
#
```

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。

```
installboot -image image ../kernel/kernel \
        ../servers/pm/pm \
        ../servers/fs/fs \
        ../servers/rs/rs \
        ../servers/ds/ds \
        ../drivers/tty/tty \
        ../drivers/memory/memory \
        ../drivers/log/log \
        ../servers/init/init
    text     data      bss      size
   24176     3384    44384     71944  ../kernel/kernel
   21728     3180    93940    118848  ../servers/pm/pm
   41536     5224  5019704   5066464  ../servers/fs/fs
    6848      840    20388     28076  ../servers/rs/rs
    3280      464     1808      5552  ../servers/ds/ds
   27072     5696    48104     80872  ../drivers/tty/tty
    6144   287784     3068    296996  ../drivers/memory/memory
    5968      572    63280     69820  ../drivers/log/log
    7056     2412     1356     10824  ../servers/init/init
  ------   ------   ------   -------
  143808   309556  5296032   5749396  total
exec sh mkboot hdboot
install image /dev/c0d0p0s0:/boot/image/3.1.2ar0
Done.
```

```
installboot -image image ../kernel/kernel \
        ../servers/pm/pm \
        ../servers/fs/fs \
        ../servers/rs/rs \
        ../servers/ds/ds \
        ../drivers/tty/tty \
        ../drivers/memory/memory \
        ../drivers/log/log \
        ../servers/init/init
    text     data      bss      size
   24176     3384    44384     71944  ../kernel/kernel
   21728     3180    93940    118848  ../servers/pm/pm
   41536     5224  5019704   5066464  ../servers/fs/fs
    6848      840    20388     28076  ../servers/rs/rs
    3280      464     1808      5552  ../servers/ds/ds
   27072     5696    48104     80872  ../drivers/tty/tty
    6144   287784     3068    296996  ../drivers/memory/memory
    5968      572    63280     69820  ../drivers/log/log
    7056     2412     1356     10824  ../servers/init/init
  ------   ------   ------   -------
  143808   309556  5296032   5749396  total
exec sh mkboot hdboot
install image /dev/c0d0p0s0:/boot/image/3.1.2ar1
Done.
#
```

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;}

```
Local packages (down): sshd  done.
Sending SIGTERM to all processes ...
MINIX will now be shut down ...
d0p0s0>newminix(5,start new kernel){image=/boot/image/3.1.2ar1;boot;}
d0p0s0>save
```

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。

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，

⑥ 登录进 minix 3 中测试

```
Hit a key as follows:

    1  Start MINIX 3 (requires at least 16 MB RAM)
    2  Start Small MINIX 3 (intended for 8 MB RAM systems)
    5  start new kernel
```

运行注意事项：

1. cc 编译器编译可执行文件的指令是 cc -o test1 test1.c。没有 clang 和 gcc。
2. 先在 kernel 1 里面编译可执行文件，然后执行 test1 和 test2，得出第一次的结果。再切换到 kernel 5，得出第二次的结果。（shutdown→menu→5）
3. 函数调用的参数和返回值请遵循原定义，请不要修改。
4. 程序运行错误会可能产生体积很大的 core 文件，建议在磁盘空间较大的 /home 下做实验，否则可能会占满磁盘，导致无法开机。
5. 如果 break 修改后导致程序的编译都无法正常进行（因为在程序编译的过程中可能需要 brk 系统调用）。在这种情况下，需从未修改过的内核中对程序进行编译。
6. CC 编译器版本很老，语法检查很严格，编写 C 代码时不要引入非英文字符，或者使用高级语法。这次实验我用了很多“//”、“/**/”的注释，还添加了中文的注释，报错无法识别 ASCII 码。最终的解决方法是将所有注释全部删除。
7. 本 project 使用的 Minix 版本号是 3.1.2。
8. CC 编译器的语法注意事项很多。比如：

   int d = sys_abscopy(old_d_tran,new_d_tran,data_bytes);　这段代码就会报错，必须先初始化定义 d（其他变量同理），然后再进行赋值。

运行结果：

Test1 第一次测试：

```
# cc -o test1 test1.c
# ./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
```

Test1 第二次测试：

```
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 33554432, total 67108863 , result + inc 67112958
incremented by 67108864, total 134217727 , result + inc 134221822
```

Test2 第一次测试：

```
# ./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
```

Test2 第二次测试：

```
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
incremented by: 67108864, total: 134217727 , result: 67112958
```

实验现象：两个测试程序可以分配比之前更多的内存。

五、总结

通过这次实验，我对分配策略有了更深刻的认识(下个匹配与最佳匹配等)。并更全面的从底层了解了堆调整的策略(比如 sys_abscopy,alloc_mem 函数，click 和 byte 之间的转化等)。同时，对虚拟机的内核编译、代码调试也有了更多的经验。