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：**2018 级　　**上机实践成绩：**

指导教师：翁楚良　　**姓名：**郑佳辰

上机实践名称：内存管理　　**学号：**10182100359　　**上机实践日期：**2020/6/16 ~7/5

上机实践编号：

一、 目的

熟悉 Minix 操作系统的进程管理，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。当分配给进程的数据段和栈段空间耗尽时，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。然后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 15.5
Minix 3.1.2a
MobaXterm Personal Edition v20.1
FileZilla 3.49.2.1
Dev-C++ 5.11
SourceInsight 4.0

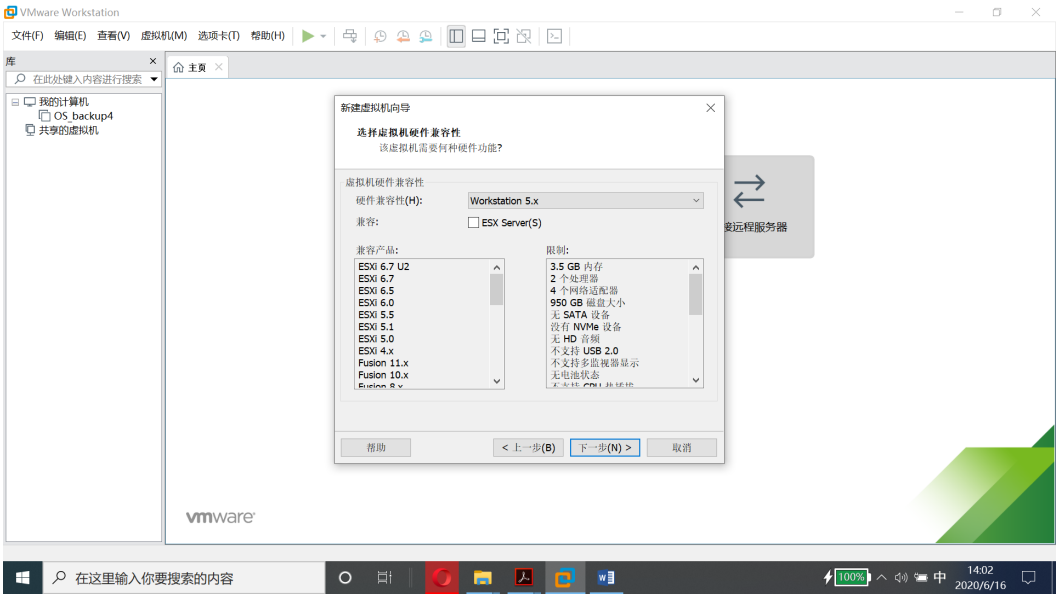
四、 实验过程

1.环境配置：安装 MINIX 3.1.2

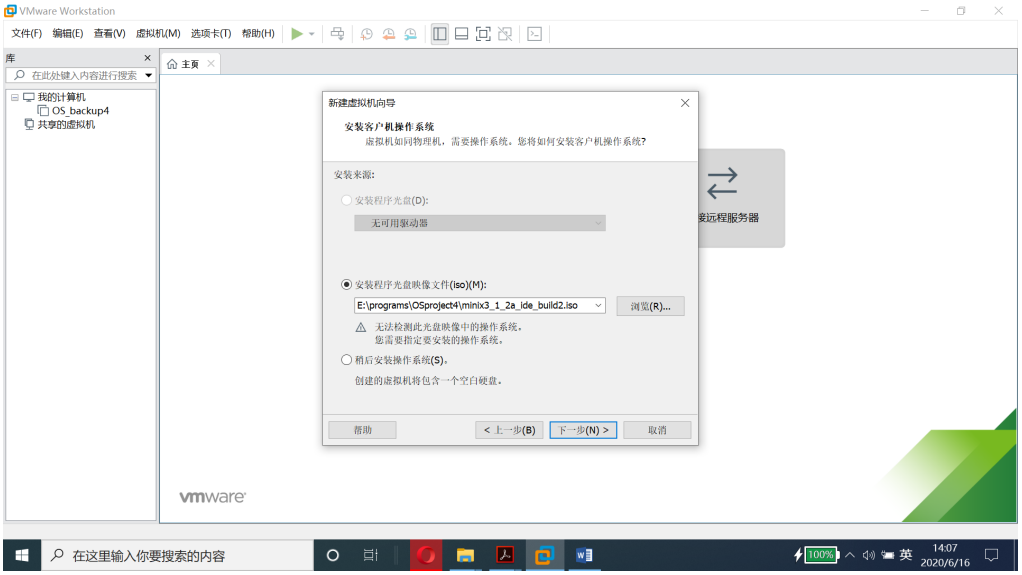
本实验所使用的 MINIX 版本不同，需要重新配置 Minix 3.1.2a 的实验环境。

1.1 创建虚拟机

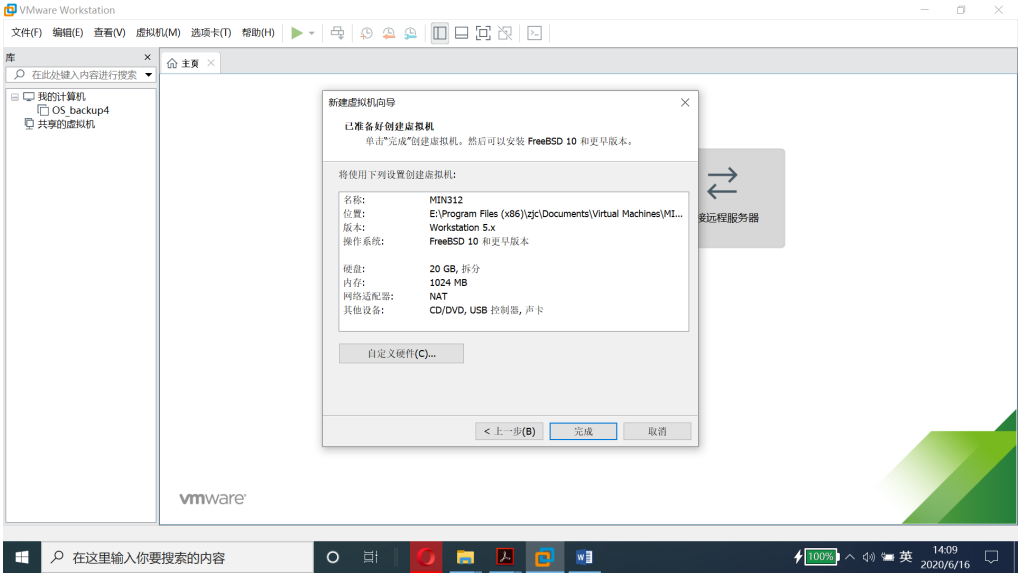
首先下载 Minix3.1.2 镜像文件，并将其解压。创建虚拟机时修改 VMware 兼容性。Windows 应选择 5.x 版本，如下图所示。若不修改会出现 no CD found 等兼容性问题。



然后在新建虚拟机时应选择 3.1.2 版本的镜像。



虚拟机配置如下图所示。完成创建。



1.2 启动虚拟机

启动虚拟机时，在 setup 步骤中，网卡选择 AMD LANCE，否则无法获得 IP 地址，其他选择默认设置。

```
What type of keyboard do you have?  You can choose one of:

    dvorak   italian    latin-america   scandinavian   uk
    french   japanese   olivetti        scandinavn     us-std
    german   latin-am   polish          spanish        us-swap

Keyboard type? [us-std]

 --- Step 2: Select your Ethernet chip ---------------------------------

MINIX 3 currently supports the following Ethernet cards. Please choose:

0. No Ethernet card (no networking)
1. Intel Pro/100
2. 3Com 501 or 3Com 509 based card
3. Realtek 8139 based card
4. Realtek 8029 based card (also emulated by Qemu)
   Note: If you want to use this in Qemu, set 'qemu_pci=1' in the boot monitor.
5. NE2000, 3com 503 or WD based card (also emulated by Bochs)
6. AMD LANCE (also emulated by VMWare)
7. Different Ethernet card (no networking)

You can always change your mind after the setup.

Ethernet card? [0] 6
```

安装完之后，输入 shutdown，然后输入 boot d0p0 重启，后续关机都需输入 shutdown，否则可能导致磁盘错误。

```
1+0 records out

Please type 'shutdown' to exit MINIX 3 and enter the boot monitor. At
the boot monitor prompt, type 'boot dXp0', where X is the bios drive
number of the drive you installed on, to try your new MINIX system.
Probably, the right command is "boot d0p0".

This ends the MINIX 3 setup script.  After booting your newly set up system,
you can run the test suites as indicated in the setup manual.  You also
may want to take care of local configuration, such as securing your system
with a password.  Please consult the usage manual for more information.


# shutdown


Broadcast message from root@minix (console)
Tue Jun 16 14:25:35 2020...

The system will shutdown NOW

Local packages (down):  done.
Sending SIGTERM to all processes ...
MINIX will now be shut down ...
fd0>boot d0p0
```

输入 mv /etc/rc.daemons.dist /etc/rc.daemons。这样在网络模式为 NAT 模式时，重启后可以看到 IP 地址。在虚拟机终端输入 packman 可以安装额外的软件包，选择全部安装。安装完之后，open-ssh，vim 等均已经安装。

```
# mv /etc/rc.daemons.dist /etc/rc.daemons
# packman
/dev/c0d2p2 is read-only mounted on /mnt
There are 46 CD packages.
Please choose:
 1  Install all 46 binary packages (408 MB uncompressed) from CD
 2  Install all 46 binary packages + sources from CD (1210 MB uncompressed)
 3  Display the list of packages on CD
 4  Let me select individual packages to install from CD or network.
 5  Exit.
Choice: [4] 1
```

输入 passwd root 为 root 用户创建密码，在重启后可在 NAT 模式下用 ftp 连接虚拟机。至此，MINIX3.1.2 实验环境配置完毕。

```
Minix  Release 3 Version 1.2a  (console)

192.168.101.135 login: root

To install X Windows, run 'packman' with the install CD still in the
drive.  To start X Windows after you have installed it, login as root
and type: 'xdm'. For more information about configuring X Windows, see
www.minix3.org.

If you do not have sufficient memory to run X Windows, standard MINIX 3
supports multiple virtual terminals. Just use ALT+F1, F2, F3 and F4 to
navigate among them.

To get rid of this message, edit /etc/motd.

# passwd root
Changing the shadow password of root
New password:
Retype password:
# _
```

2.修改内存分配

2.1 修改 alloc.c

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

在原始的文件中，hp 是用来遍历空闲列表的变量，当找到一个大于给定的 clicks 的空闲块时，就立即执行分割，分配等操作。修改之后，用 bhp 指针记录最优的块，prev_bptr 记录它的上一个块。当找到合适的块之后，就记录下来，继续寻找更优的块，直至所有块都被查找过一遍。用 best 指示是否找到了合适大小

的块，0 代表还没找到，1 代表已经找到。然后将 bhp 和 prev_bptr 赋给 hp 和 prev_ptr，继续执行使用该空闲块的流程。修改后的 alloc_mem 函数如下所示。

```
PUBLIC phys_clicks alloc_mem(clicks)
phys_clicks clicks;     /* amount of memory requested */
{
  register struct hole *hp, *prev_ptr;
  phys_clicks old_base;
  struct hole *bhp, *prev_bptr;
  int best;

  best=0;
  do {
        prev_ptr = NIL_HOLE;
    hp = hole_head;
    while (hp != NIL_HOLE && hp->h_base < swap_base) {
        if (hp->h_len >= clicks) {
                /* We found a hole that is big enough.  Save it. */
            if (best == 0 || best == 1 && hp->h_len < bhp->h_len) {
                  best = 1;
                bhp = hp;
                prev_bptr = prev_ptr;
            }
        }

        prev_ptr = hp;
        hp = hp->h_next;
    }
  } while (swap_out());     /* try to swap some other process out */

  hp = bhp;
  prev_ptr = prev_bptr;

  if (best == 1) {
    /* We found the best block.  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 = hp->h_base;

    /* Delete the hole if used up completely. */
    if (hp->h_len == 0) del_slot(prev_ptr, hp);

    /* Return the start address of the acquired block. */
    return(old_base);
  }

  return(NO_MEM);
}
```

2.2 修改 break.c

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。adjust 函数计算程序当前的空闲空间是否足够分配。若不够，调用 allocate_new_mem 函数申请新的足够大的内存空间。无法申请到足够大的内存空间则会返回 ENOMEM，其修改部分如下所示。

```
if (lower < gap_base) /* data and stack collided */
    if (allocate_new_mem(rmp,(phys_clicks)(mem_sp->mem_vir+mem_sp->mem_le
n)) == 0)
        return(ENOMEM);
```

函数 allocate_new_mem 的参数 rmp 指向进程的 mproc 数据结构，clicks 是原来的内存空间的大小。首先读取原来进程的栈段和数据段的地址和大小，之后调用 alloc_mem 函数分配新的内存空间，然后调用 sys_memset 函数将新的内存空间中的数据赋值为 0。然后再计算得到新的栈段和数据段的地址并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和顶部，通知内核程序的映像发生了变化并释放原有内存。最后返回 do_brk 函数。该函数的代码如下所示。

```
PUBLIC int allocate_new_mem(rmp,clicks)
register struct mproc *rmp;
phys_clicks clicks;
{
    register struct mem_map *mem_sp,*mem_dp;
    phys_bytes old_bytes,d_bytes,s_bytes,old_d_abs,new_d_abs,old_s_abs,new
_s_abs;
    phys_clicks old_clicks,old_d_base,new_clicks,new_d_base,d_clicks,new_s
_base,old_s_base,s_clicks;
    int s,r;
    mem_dp = &rmp->mp_seg[D];
    mem_sp = &rmp->mp_seg[S];
    old_clicks = clicks;
    new_clicks = clicks << 1;
    d_bytes = (phys_bytes) rmp->mp_seg[D].mem_len << CLICK_SHIFT;
    s_bytes = (phys_bytes) rmp->mp_seg[S].mem_len << CLICK_SHIFT;

    if ((new_d_base = alloc_mem(new_clicks))==NO_MEM)
        return (0);
    old_d_base = rmp->mp_seg[D].mem_phys;
    old_s_base = rmp->mp_seg[S].mem_phys;
    new_s_base = new_d_base + new_clicks - mem_sp->mem_len;
    new_d_abs = (phys_bytes) new_d_base << CLICK_SHIFT;
    old_d_abs = (phys_bytes) old_d_base << CLICK_SHIFT;
    new_s_abs = (phys_bytes) old_d_base << CLICK_SHIFT;

    s=sys_memset(0,new_d_abs,(new_clicks<<CLICK_SHIFT));
      if (s < 0){
       panic(__FILE__,"couldn't sys_memset in allocate_new_mem",s);
    }
    s = sys_abscopy(old_d_abs,new_d_abs,d_bytes);
    if (s < 0)
        panic(__FILE__,"couldn't sys_abscopy in allocate_new_mem",s);
    s = sys_abscopy( old_s_abs,new_s_abs,s_bytes);
```

```
    if (s < 0)
        panic(__FILE__,"couldn't sys_abscopy in allocate_new_mem",s);

    rmp->mp_seg[D].mem_phys = new_d_base;
    rmp->mp_seg[S].mem_phys = new_s_base;
    rmp->mp_seg[S].mem_vir = new_clicks-mem_sp->mem_len;
    if (!(rmp->mp_flags & SEPARATE))
        rmp->mp_seg[T].mem_phys = new_d_base;

    free_mem(old_d_base,old_clicks);
    return 1;
}
```

3.编译及测试

3.1 编译 MINIX3

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。然后进入/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。

```
   ------   ------   ------   -------
   143808   309588  5296032   5749428  total
exec sh mkboot hdboot
rm /dev/c0d0p0s0:/boot/image/3.1.2ar0
install image /dev/c0d0p0s0:/boot/image/3.1.2ar3
Done.
# shutdown
```

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。键入 save 命令保存设置。

```
     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
   ------   ------   ------   -------
   143840   309588  5296032   5749460  total
exec sh mkboot hdboot
install image /dev/c0d0p0s0:/boot/image/3.1.2ar1
Done.
# shutdown


Broadcast message from root@192.168.101.135 (console)
Wed Jun 24 15:39:35 2020...

The system will shutdown NOW

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
d0p0s0>_
```

输入 menu 命令，然后输入数字键 5 启动新内核，登录进 minix 3 中测试。

```
d0p0s0>save
d0p0s0>menu

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
_
```

3.2 测试

测试程序一只是简单测试 sbrk 调用，不断的调整数据段的上界，并未对新分

配的内存空间进行访问。实验截图如下。

```
# ./t1
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
```

测试程序二则对新分配的内存空间进行了访问。实验截图如下。

```
# ./t2
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
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
```

可以看到，修改后的代码可以成功通过两个测试程序。

五、 总结

这次实验的主要工作是在 MINIX3 系统中修改进程管理器并改进 brk 系统调用的实现。进程管理器负责处理与进程相关的系统调用，其中有些调用与存储管理密切相关，如 fork，exec 和 brk 等。它维护着一个空闲空间列表，当需要内存空间时，系统将采用最先匹配法。本次实验的任务之一就是将分配算法改成采用最佳匹配法，即遍历所有空闲块，找到大小大于所需空间的最小块。

库函数 brk 和 sbrk 是用来调整数据段上边界的。brk 的参数是绝对长度，以字节为单

位，并以此调用 brk 系统调用。sbrk 的参数是相对于当前长度的增量。brk 调用原本的实现是检查新的大小是否可行，然后更新相应的表格。它的大部分工作是 adjust 来完成的，这个函数负责检查数据段和栈段是否冲突。本次实验要将其修改成空间不够时调用 allocate_new_mem 函数申请足够大的内存。

通过这次内存管理的实验，我对进程管理器和空闲地址的分配有了更深入的理解，更加明确了基于链表的存储管理的算法及其实现。对存储管理系统中的重要概念有了更直观的印象，同时的借助 MINIX3 更好地理解存储管理器的实现。