实验四 内存管理

10215501422高宇菲

安装配置Minix3.1.2

1. 下载镜像文件

IDE-3.1.2a (1).iso.bz2
http://download.minix3.org/iso/IDE-3.1.2a.iso.bz2
在文件夹中显示

2. 新建虚拟机
   - 解压上面下载的.bz2文件，并将新虚拟机镜像改为解压出来的iso文件。

   - 修改VMware兼容性，选择5.x版本。
   - 操作系统改为其他，否则之后无法正常显示ip地址。

   - 启动虚拟机，选择网卡6，其余默认回车。
   - shutdown关闭，boot d0p0重启。
   - 执行mv /etc/rc.daemons.dist /etc/rc.daemons
   - packman命令，选1,等待下载安装。
   - 输入passwd root创建密码

```
* passwd root
Changing the shadow password of root
New password:
```

- 重启，可以看到配置成功：

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

192.168.153.134 login: root
Password:
```

> 完成以下实验之前，阅读《操作系统设计与实现》第四章内存管理

修改内存分配

修改/usr/src/servers/pm/alloc.c中的alloc_mem函数，把first-fit修改成 best-fit。

首先，看懂原代码。原代码中的alloc_mem接收一个参数clicks, clicks指定了请求的内存大小。minix3.1.2中，1 click = 1024 B。hp是指向空闲内存块hole的指针，prev_ptr是指向前一个空闲内存块的指针。do_while循环中找第一个满足的内存块，如果没找到，就执行swap_out将其他进程换出内存以释放空间。

循环中，遍历空闲内存块，找到一个剩余空间足够大的内存块，就分配并返回。

原代码中，hp->h_len代表了一个内存块中剩余可用的空闲内存。

修改源码时，要引入指向空闲内存块的指针best_fit和min_left来分别记录当前最合适的块和相应的剩余内存大小。遍历空闲块，如果遇到更小的剩余空间，就更新best-fit 和 min_left。

```
        while (hp != NIL_HOLE && hp->h_base < swap_base) {
          /* look for best_fit */
          if (hp->h_len >= clicks){
              left = hp->len-clicks;
              if(left < min_left){
                min_left = left;
                best_fit = hp;
                best_fit_prev = prev_ptr;
              }
          }
          prev_ptr = hp;
          hp = hp->h_next;
        }
```

找到之后，分配。只要将源码中的变量名修改位best_fit就可以。当然还需要仿照源码引入best_fit_prev变量。

```
        /* allocate */
        if (best_fit != NIL_HOLE)
        {
          /* We found a hole that is big enough.  Use it. */
          old_base = best_fit->h_base; /* remember where it started */
          best_fit->h_base += clicks;  /* bite a piece off */
          best_fit->h_len -= clicks;   /* ditto */

          /* Remember new high watermark of used memory. */
```

```
        if (best_fit->h_base > high_watermark)
            high_watermark = best_fit->h_base;

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

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

修改/usr/src/servers/pm/break.c中的adjust函数，增加一个allocate_new_mem局部函数在adjust函数中调用。

观察原代码，原代码如果发现数据段和栈段重叠，会立即返回，表示内存不足。

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

现修改该处代码，发现内存不足则分配新的更大的内存。

```
if (lower < gap_base){
  return allocate_new_mem(rmp, data_clicks, sp);
}
```

allocate_new_mem实现思路：

1. 调用alloc_mem函数申请一块新空间，新空间大小等于**现在程序所占内存大小+数据段和栈段重叠大小*2。**
2. 调用sys_abscopy函数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。
3. 修改进程表中记录的数据段和栈段的物理地址、虚拟地址。
4. 调用sys_newmap函数通知内核，注册内存段。
5. 调用free_mem函数释放原来的空间。

使用源码查看工具找到sys_abscopy定义和用法：

```
#define sys_abscopy(src_phys, dst_phys, bytes) \
    sys_physcopy(NONE, PHYS_SEG, src_phys, NONE, PHYS_SEG, dst_phys, bytes)
```

```
PUBLIC int sys_physcopy(src_proc, src_seg, src_vir,
    dst_proc, dst_seg, dst_vir, bytes)
int src_proc;           /* source process */
int src_seg;            /* source memory segment */
vir_bytes src_vir;      /* source virtual address */
int dst_proc;           /* destination process */
int dst_seg;            /* destination memory segment */
vir_bytes dst_vir;      /* destination virtual address */
phys_bytes bytes;       /* how many bytes */
```

servers/pm/forkexit.c (1 occurrence)

```
66    s = sys_abscopy(parent_abs, child_abs, prog_bytes);
```

sys_abscopy接收三个参数，第一个是旧物理地址，第二个是新物理地址，第三个是总字节个数(bytes)。

查看sys_newmap定义：

```
PUBLIC int sys_newmap(proc, ptr)
int proc;           /* process whose map is to be changed */
struct mem_map *ptr;        /* pointer to new map */
```

sys_newmap接收两个参数，第一个是要修改进程的进程号，在minix中用mp_endpoint表示；第二个是新的内存映射指针，这两个都是进程表中的表项。

```
EXTERN struct mproc {
  struct mem_map mp_seg[NR_LOCAL_SEGS]; /* points to text, data, stack */
  char mp_exitstatus;       /* storage for status when process exits */
  char mp_sigstatus;        /* storage for signal # for killed procs */
  pid_t mp_pid;          /* process id */
  int mp_endpoint;       /* kernel endpoint id */
```

而结构体mp_seg的定义如下，都是以click为单位：

```
struct mem_map {
    vir_clicks mem_vir; /* virtual address */
    phys_clicks mem_phys; /* physical address */
    vir_clicks mem_len; /* length */
};
```

free_mem接收两个参数，第一个是要释放的内存的起始地址，第二个是要释放的内存大小。

```
PUBLIC void free_mem(base, clicks)
phys_clicks base;       /* base address of block to free */
phys_clicks clicks;     /* number of clicks to free */
```

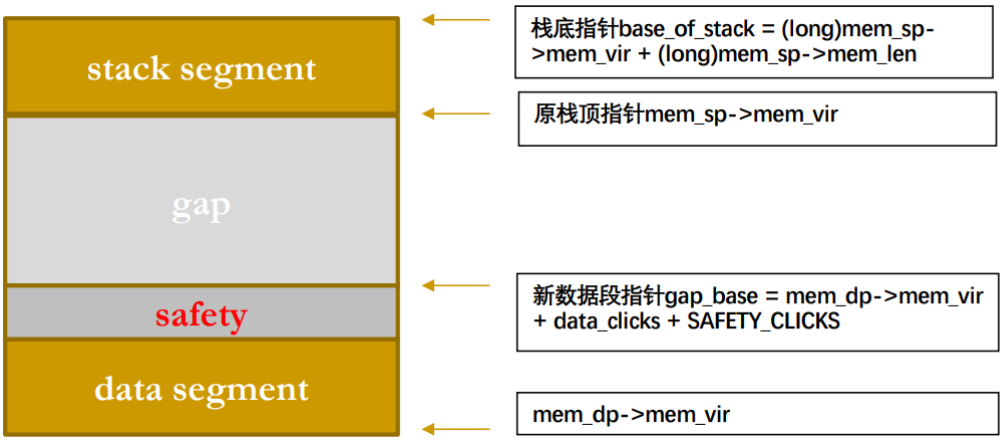
allocate_new_mem声明如下，其中delta是在adjust中计算的参数，gap_base_lower_delta是数据段和栈段重叠部分大小：

```
PUBLIC int allocate_new_mem(rmp, data_clicks, delta, gap_base_lower_delta)
register struct mproc *rmp; /* whose memory is being adjusted? */
phys_clicks data_clicks;     /* how big is data segment to become? */
long delta;   /*记录了新的栈顶和旧栈顶之间的差值*/
phys_clicks gap_base_lower_delta;  /*how big is collided space*/
```

首先仿照adjust 定义数据段和栈段地址信息的指针：

```
mem_dp = &rmp->mp_seg[D]; /* pointer to data segment map */
mem_sp = &rmp->mp_seg[S]; /* pointer to stack segment map */
change = 0;
```

准备alloc_mem的参数new_clicks:




```
  cur_total_click = (phys_clicks)(mem_sp->mem_vir - mem_dp->mem_vir + mem_sp-
>mem_len);
  new_clicks = cur_total_click + gap_base_lower_delta * 2;
```

分配内存，alloc_mem返回数据段底：

```
  if((new_address_data=alloc_mem(new_clicks))==NO_MEM){
    return(ENOMEM);
  }
```

对数据段和栈段分别拷贝，调用两次sys_abscopy。

```
sys_abscopy(old_address_data_byte, new_address_data_byte, databytes);
sys_abscopy(old_address_stack_byte, new_address_stack_byte, stackbytes);
```

调用之前分别准备他们的参数，因为传入的参数要求以字节为单位，所以要和click换算：

```
  databytes = (phys_bytes)mem_dp->mem_len << CLICK_SHIFT;
  stackbytes = (phys_bytes)mem_sp->mem_len << CLICK_SHIFT;

  old_address_data = mem_dp->mem_phys;
  new_address_stack = new_address_data + new_clicks - mem_sp->mem_len;
  old_address_stack = mem_sp->mem_phys;

  new_address_data_byte = (phys_bytes)new_address_data << CLICK_SHIFT;
  old_address_data_byte = old_address_data << CLICK_SHIFT;
  new_address_stack_byte = (phys_bytes)new_address_stack << CLICK_SHIFT;
  old_address_stack_byte = old_address_stack << CLICK_SHIFT;
```

复制之后，更新进程表项

```
  d = sys_abscopy(old_address_data_byte, new_address_data_byte, databytes);
  if (d < 0)
    panic(__FILE__, " couldn't copy data segment in alloc_new_mem", d);
  s = sys_abscopy(old_address_stack_byte, new_address_stack_byte, stackbytes);
  if (s < 0)
    panic(__FILE__, " couldn't copy stack segment in alloc_new_mem", s);

  mem_dp->mem_phys = new_address_data;
  mem_sp->mem_phys = new_address_stack;
  mem_sp->mem_vir = mem_dp->mem_vir + new_clicks - mem_sp->mem_len;
```

之后仿照adjust记录新的数据段，栈段长度；检查新的数据段，栈段大小在地址空间中是否合适，如果合适则调用sys_newmap通知内核，注册内存段，释放旧内存；如果不合适则恢复原来的大小。

```
  old_data_clicks = mem_dp->mem_len;

  if (data_clicks != mem_dp->mem_len)
  {
    mem_dp->mem_len = data_clicks;
    change |= DATA_CHANGED;
  }

  if (delta > 0)
  {
    mem_sp->mem_vir -= delta;
    mem_sp->mem_phys -= delta;
    mem_sp->mem_len += delta;
    change |= STACK_CHANGED;
  }
    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
  if (r == OK) {
    int r2;
    if (change && (r2=sys_newmap(rmp->mp_endpoint, rmp->mp_seg)) != OK)
      panic(__FILE__,"couldn't sys_newmap in adjust", r2);
    free_mem(old_address_data, old_clicks);
    return(OK);
  }

  /* New sizes don't fit or require too many page/segment registers. Restore.*/
  if (change & DATA_CHANGED) mem_dp->mem_len = old_data_clicks;
  if (change & STACK_CHANGED) {
  mem_sp->mem_vir += delta;
  mem_sp->mem_phys += delta;
  mem_sp->mem_len -= delta;
  }
```

编译新内核

进入/usr/src/servers目录：

```
# make image
```

报错如下，检查后发现可能是有些vir_clicks变量没有转化成phys_clicks变量。

```
cd ./pm && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include break.c
"break.c", line 66: (warning) 'allocate_new_mem' old-fashioned function
definition
"break.c", line 84: identifier not expected
make in /usr/src/servers/pm: Exit code 1
make in /usr/src/servers: Exit code 1
```

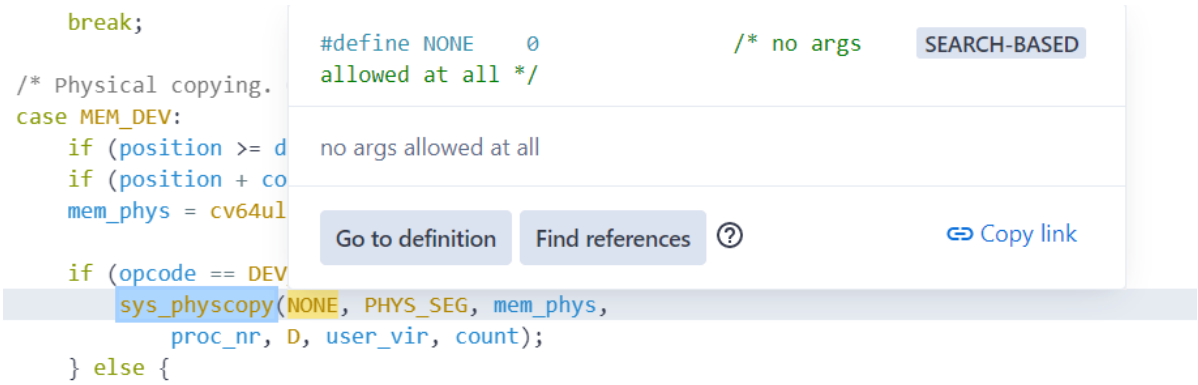
修改后编译，报错如下，继续检查，发现有一行语句结尾没有分号；另外有一处笔误。

```
* make image
cd ./pm && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include break.c
"break.c", line 66: (warning) 'allocate_new_mem' old-fashioned function
definition
"break.c", line 84: identifier not expected
make in /usr/src/servers/pm: Exit code 1
make in /usr/src/servers: Exit code 1
```

修改后报错如下，

```
* make image
cd ./pm && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include break.c
"break.c", line 23: cannot open include file "syslib.h"
"break.c", line 67: (warning) 'allocate_new_mem' old-fashioned function
definition
"break.c", line 109: NONE undefined
make in /usr/src/servers/pm: Exit code 1
make in /usr/src/servers: Exit code 1
```

109行是sys_abscopy复制数据段，而sys_abscopy将调用sys_physcopy，sys_phycopy的定义中用到了NONE。报错可能是NONE没有在包含的头文件中定义。用原码查看工具找到这里的NONE就是0。



在break.c中定义NONE：

```
#define NONE 0
```

修改后报错如下，alloc.c中有一处笔误。

```
* make image
cd ./pm && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include break.c
"break.c", line 67: (warning) 'allocate_new_mem' old-fashioned function
definition
exec cc -c -I/usr/include exec.c
exec cc -c -I/usr/include time.c
exec cc -c -I/usr/include timers.c
exec cc -c -I/usr/include signal.c
exec cc -c -I/usr/include alloc.c
"alloc.c", line 80: unknown selector len
make in /usr/src/servers/pm: Exit code 1
make in /usr/src/servers: Exit code 1
```

修改后，编译成功。

```
* make image
cd ./pm && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include alloc.c
exec cc -c -I/usr/include utility.c
exec cc -c -I/usr/include table.c
exec cc -c -I/usr/include trace.c
exec cc -c -I/usr/include getset.c
exec cc -c -I/usr/include misc.c
exec cc -o pm -i main.o forkexit.o break.o exec.o time.o timers.o \
        signal.o alloc.o utility.o table.o trace.o getset.o misc.o -lsys -
lsysutil -ltimers
install -S 256w pm
cd ./fs && exec make - EXTRA_OPTS= build
exec cc -c -I/usr/include  main.c
exec cc -c -I/usr/include  open.c
exec cc -c -I/usr/include  read.c
exec cc -c -I/usr/include  write.c
exec cc -c -I/usr/include  pipe.c
exec cc -c -I/usr/include  dmap.c
exec cc -c -I/usr/include  device.c
```

```
exec cc -c -I/usr/include  path.c
exec cc -c -I/usr/include  mount.c
exec cc -c -I/usr/include  link.c
exec cc -c -I/usr/include  super.c
exec cc -c -I/usr/include  inode.c
exec cc -c -I/usr/include  cache.c
exec cc -c -I/usr/include  cache2.c
exec cc -c -I/usr/include  filedes.c
exec cc -c -I/usr/include  stadir.c
exec cc -c -I/usr/include  protect.c
exec cc -c -I/usr/include  time.c
exec cc -c -I/usr/include  lock.c
exec cc -c -I/usr/include  misc.c
exec cc -c -I/usr/include  utility.c
exec cc -c -I/usr/include  select.c
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 -lsysutil -
ltimers
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 -lsysutil
install -S 192w init
```

输入make install，成功。

```
# make install
cd ./pm && exec make - install
install -o root -cs pm /usr/sbin/pm
cd ./fs && exec make - install
make: 'install' is up to date
cd ./rs && exec make - install
install -c service /bin/service
install -o root -c rs /usr/sbin/rs
cd ./ds && exec make - install
install -o root -c ds /sbin/ds
cd ./is && exec make - install
exec cc -c -I/usr/include main.c
exec cc -c -I/usr/include dmp.c
```

```
exec cc -c -I/usr/include dmp_kernel.c
exec cc -c -I/usr/include dmp_pm.c
exec cc -c -I/usr/include dmp_fs.c
exec cc -c -I/usr/include dmp_rs.c
exec cc -c -I/usr/include dmp_ds.c
exec cc -o is -i main.o dmp.o dmp_kernel.o dmp_pm.o dmp_fs.o dmp_rs.o dmp_ds.o  -
lsys -lsysutil
install -o root -c is /sbin/is
cd ./init && exec make - install
install -o root -cs init /usr/sbin/init
cd ./inet && exec make - install
cc  -I. -D_MINIX -o buf.o -c buf.c
cc  -I. -D_MINIX -o clock.o -c clock.c
cc  -I. -D_MINIX -o inet.o -c inet.c
cc  -I. -D_MINIX -o inet_config.o -c inet_config.c
cc  -I. -D_MINIX -o mnx_eth.o -c mnx_eth.c
cc  -I. -D_MINIX -o mq.o -c mq.c
cc  -I. -D_MINIX -o qp.o -c qp.c
cc  -I. -D_MINIX -o sr.o -c sr.c
cc  -I. -D_MINIX -o stacktrace.o -c stacktrace.c
cc  -I. -D_MINIX -o generic/udp.o -c generic/udp.c
cc  -I. -D_MINIX -o generic/arp.o -c generic/arp.c
cc  -I. -D_MINIX -o generic/eth.o -c generic/eth.c
cc  -I. -D_MINIX -o generic/event.o -c generic/event.c
cc  -I. -D_MINIX -o generic/icmp.o -c generic/icmp.c
cc  -I. -D_MINIX -o generic/io.o -c generic/io.c
cc  -I. -D_MINIX -o generic/ip.o -c generic/ip.c
cc  -I. -D_MINIX -o generic/ip_ioctl.o -c generic/ip_ioctl.c
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
```

进入/usr/src/tools目录:

```
# make hdboot
# make install
```

均成功。

```
install image /dev/c0d0p0s0:/boot/image/3.1.2ar1
Done.
```

键入shutdown后，在boot monitor界面，无法继续键入内容

换了一种方法,在开机时按esc, 然后设置新核，编号给5。

```
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)
    3  Start Custom MINIX 3
[ESC]
d0p0s0>newminix(5,start new kernel){image=/boot/image/3.1.2ar1;boot;}
d0p0s0>save
d0p0s0>
```

切换到老内核(编号为1)，cc编译test1.c, test2.c。

切换到新内核(编号5)，运行test1:

```
# ./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
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
```

运行test2，报错如下：

```
# ./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
Memory fault - core dumped
*
```

错误修正

因为分配内存成功而访问失败，考虑分配 的内存未初始化：

加上以下初始化代码：

```
107    sys_memset(0, new_address_data_byte, (new_clicks << CLICK_SHIFT));
```

删除test 文件夹下的core文件，再次编译内核, 并编号为6：

```
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
[ESC]
d0p0s0>newminix(6,start new kernel){image=/boot/image/3.1.2ar3;boot;}
```

在新内核下，test2运行成功：

```
# ./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
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
```

遇到的问题

对原代码的理解，如对delta的理解，delta代表新栈顶和旧栈顶之差。

总结

本次实验深入分析了minix pm部分源码，修改了minix的内存分配方式。本次实验增强了源码阅读能力，加深了对内存管理的理解，为今后学习打下基础。