VE482 Lab Report

Lab 8 - Fall 2020

Wu Qinhang (518370910041)
Sun Zhimin (518370910219)

Table of Contents

Memory Management at Kernel Level

What does `vm` stands for?

- ~~VMware~~
- Virtual Memory

Find all the places where the vm used inside the kernel, Why does it appear in so many different places? [1]

- `find /usr/src -name "*.c" | xargs grep -l "vm" > vm_data`
- 

- I find 300 source files altogether that contain virtual memory, a very large number.
- Virtual memory is always being used, because all the process require its only virtual process, and even when the memory that is required by all running processes does not exceed the volume of RAM that is installed on the system, vm is still used.

How is memory allocated within the kernel? Why are not malloc and calloc used? [2]

- Instead of `malloc` or `calloc`, in the kernel, many new functions are used:

```
#include <linux/slab.h>

void *kmalloc(size_t size, int flags);
// kmalloc(): get any size of block
void *vmalloc(size_t size);
// vmalloc(): allocate contiguous virtual memory
```

It's quite similar to `malloc`, except that it allocates a block of **kernel memory** (or **virtual memory**).

- Since `malloc` and `calloc` are user library functions, the memory that allocated by those two functions is also accessible to users, and that results in the issue of kernel's security.
- IPC (Message passing) is used to send message between kernel space and user space. There is a structure defined in `/include/minix/ipc.h`:

```
typedef struct {
  endpoint_t m_source;    /* who sent the message */
  int m_type;     /* what kind of message is it */
  union {
  mess_1 m_m1;
  mess_2 m_m2;
  mess_3 m_m3;
  mess_4 m_m4;
  mess_5 m_m5;
  mess_7 m_m7;
  mess_8 m_m8;
  mess_6 m_m6;
  mess_9 m_m9;
  mess_10 m_m10;
  } m_u;
} message __aligned(16);
```

While allocating memory, how does the functions in kernel space switch back and forth between user and kernel spaces? How is that boundary crossed? How good or bad it is to put vm in userspace?

- When the user calls the function, e.g. `malloc` or `calloc`, take Minix3 as an example, the kernel calls the following functions one by one:
  1. `mmap()`
  2. `do_mmap()`
  3. `map_page_region()`
  4. `map_ph_writept()`
  5. `pt_writemap()`

  General idea is to ask the memory management unit for a block of memory, next establish a map connection of that block, then write that part onto the page table, finally return the pointer to the userspace, who can use it to have access to the memory block.
- By mapping user-space memory into the kernel (with get_user_pages).
- It depends. It's good for our usual PC, since it can prevent the user from changing the memory intentionally or accidently, thus protecting our computer. However, for real-time system or embedded system, the kernel is designed to serve, and two separate space (userspace and kernel-space are unnecessary).

How are pagefaults handled?

- When a pagefault occurred, the operating system first search to see if there is available page frame in the physical memory. If so, it set up a mapping relationship between the page and the page frame. If not, it locate a page frame in the physical memory that is not frequently used and write its content onto the hard drive. After that, it reads the page onto the page frame and set up a mapping relationship between them and redo the instruction.

MRU Implementation and Analysis [3]

What algorithm is used by default in Minix 3 to handle pagefault? Find its implementation and study it closely.

- Minix 3.2.1 is using LRU algorithm to handle pagefault. The detailed implementation is in `/servers/vm/pagefault.c`.
  - `char *pf_errstr(u32_t err)` is an error handling function.
  - `void do_pagefaults(message *m)` is the page fault handler.
  - `void do_memory(void)`: when `SIGKMEM` (kernel memory pending request) is asserted, `handle_memory` will be called to allocate the memory.
  - `int handle_memory(struct vmproc *vmp, vir_bytes mem, vir_bytes len, int wrflag)` call `map_handle_memory` to allocate the memory according to the page size.

Use the top command to keep track of your used memory and cache, then run `time grep -r "mum" /usr/src`. Run the command again. What do you notice? [4]

- This command will search for the pattern "mum" in Minix 3.2.1 source code.
- The first trial: `8.83 real / 0.28 user / 5.81 sys`
- The second trial: `3.03 real / 0.88 user / 2.08 sys`
- The third trial: `3.11 real / 0.95 user / 2.16 sys`
- We can conclude that calculation relevant with the searching process in the first trial is cached and perserved for a certain amount of time, so that in a neighbouring search process, the result is returned about two times faster. It matches LRU algorithm.

Adjust the implementation of LRU into MRU and recompile the kernel.

Change:

- in file `/lib/libminixfs/cache.c/lmfs_get_block(dev_t,block_t,int)`, the lower level implementation of LRU is changed into MRU
- in `/servers/vm/region.c/free_yielded(vir_bytes)`, the upper level implementation of LRU (freeing node) is changed into MRU
- for source files, see attachment.

Rebuild the kernel:

```
# use SCP to transfer files
scp -P 2222 ./minix-R3.2.1/servers/vm/region.c
root@192.168.227.129:/usr/src/servers/vm/region.c
scp -P 2222 ./minix-R3.2.1/lib/libminixfs/cache.c
root@192.168.227.129:/usr/src/lib/libminixfs/cache.c

# in Minix 3, rebuild the system
su
cd /usr/src
make build
reboot
```

Result:

```
         ptyfs is not in use
    postinstall checks passed: fontconfig motd mtree wscons x11 xkb varrwho tcpdumpc
    hroot catpages obsolete ptyfsoldnodes
    postinstall checks failed: bluetooth ddbonpanic defaults dhcpcd envsys gid gpio
    hosts iscsi makedev named pam periodic pf pwd_mkdb rc ssh uid atf
    To fix, run:
        /bin/sh /usr/src/usr.sbin/postinstall/postinstall -s '/usr/src' -d / fix blu
    etooth ddbonpanic defaults dhcpcd envsys gid gpio hosts iscsi makedev named pam
    periodic pf pwd_mkdb rc ssh uid atf
    Note that this may overwrite local changes.
       ================================
    do-obsolete ===> .
    install-obsolete-lists ===> etc
        install  /var/db/obsolete/minix
    install-etc-release ===> etc
         create  etc/etc-release
    hostname: not found
        install  etc/release
    do-hdboot ===> releasetools
    git: not found
    rm /dev/c0d0p0s0:/boot/minix/3.2.1r2
    Done.
    Build started at:  Sat Nov 21 20:45:24 GMT 2020
    Build finished at: Sat Nov 21 20:56:23 GMT 2020
    # _
```

Use the top command to keep track of your used memory and cache, then run `time grep -r "mum" /usr/src`. Run the command again. What do you notice?

```
allows */
/usr/src/usr.bin/stat/stat.1:An optional decimal digit string specifying the min
imum field width.
/usr/src/usr.bin/stat/stat.1:and a decimal digit string that indicates the maxim
um string length,
/usr/src/usr.bin/stat/stat.1:output, or the minimum number of digits to appear i
n numeric output.
/usr/src/usr.sbin/installboot/fstypes.c:                    "(calculated %u, max
imum %u)",
/usr/src/usr.sbin/mkfs.mfs/mfsdir.h:/* Maximum Minix MFS on-disk directory filen
ame.
/usr/src/usr.sbin/mkfs.mfs/super.h:  int32_t s_max_size;                /* maxim
um file size on this device */
/usr/src/usr.sbin/mkfs.mfs/super.h:  /* The block size in bytes. Minimum MIN_BLO
CK SIZE. SECTOR_SIZE
/usr/src/usr.sbin/pwd_mkdb/pwd_mkdb.8:the input file up to a maximum of 8 megaby
tes.
/usr/src/usr.sbin/user/useradd.8:(16 groups maximum.)
/usr/src/usr.sbin/user/usermod.8:(16 groups maximum.)
Binary file /usr/src/usr.sbin/zic/zic.o matches
Binary file /usr/src/usr.sbin/zic/zic matches
      14.46 real       0.20 user       9.35 sys
# echo "first trial"
first trial
#
```

```
allows */
/usr/src/usr.bin/stat/stat.1:An optional decimal digit string specifying the min
imum field width.
/usr/src/usr.bin/stat/stat.1:and a decimal digit string that indicates the maxim
um string length,
/usr/src/usr.bin/stat/stat.1:output, or the minimum number of digits to appear i
n numeric output.
/usr/src/usr.sbin/installboot/fstypes.c:                    "(calculated %u, max
imum %u)",
/usr/src/usr.sbin/mkfs.mfs/mfsdir.h:/* Maximum Minix MFS on-disk directory filen
ame.
/usr/src/usr.sbin/mkfs.mfs/super.h:  int32_t s_max_size;                /* maxim
um file size on this device */
/usr/src/usr.sbin/mkfs.mfs/super.h:  /* The block size in bytes. Minimum MIN_BLO
CK SIZE. SECTOR_SIZE
/usr/src/usr.sbin/pwd_mkdb/pwd_mkdb.8:the input file up to a maximum of 8 megaby
tes.
/usr/src/usr.sbin/user/useradd.8:(16 groups maximum.)
/usr/src/usr.sbin/user/usermod.8:(16 groups maximum.)
Binary file /usr/src/usr.sbin/zic/zic.o matches
Binary file /usr/src/usr.sbin/zic/zic matches
       9.90 real       0.51 user       7.18 sys
# echo "second trial"
second trial
#
```

```
allows */
/usr/src/usr.bin/stat/stat.1:An optional decimal digit string specifying the min
imum field width.
/usr/src/usr.bin/stat/stat.1:and a decimal digit string that indicates the maxim
um string length,
/usr/src/usr.bin/stat/stat.1:output, or the minimum number of digits to appear i
n numeric output.
/usr/src/usr.sbin/installboot/fstypes.c:                        "(calculated %u, max
imum %u)",
/usr/src/usr.sbin/mkfs.mfs/mfsdir.h:/* Maximum Minix MFS on-disk directory filen
ame.
/usr/src/usr.sbin/mkfs.mfs/super.h:  int32_t s_max_size;               /* maxim
um file size on this device */
/usr/src/usr.sbin/mkfs.mfs/super.h:  /* The block size in bytes. Minimum MIN_BLO
CK SIZE. SECTOR_SIZE
/usr/src/usr.sbin/pwd_mkdb/pwd_mkdb.8:the input file up to a maximum of 8 megaby
tes.
/usr/src/usr.sbin/user/useradd.8:(16 groups maximum.)
/usr/src/usr.sbin/user/usermod.8:(16 groups maximum.)
Binary file /usr/src/usr.sbin/zic/zic.o matches
Binary file /usr/src/usr.sbin/zic/zic matches
       9.83 real       0.55 user       6.95 sys
# echo "third trial"
third trial
#
```

- I see that the first trial runs about one time slower compared with the LRU implementation. The second trial only speed up about 25% than the first trial. It matches MRU algorithm.

Discuss the different behaviours of LRU and MRU as well as the consequences for the users. Can you think of any situation where MRU would be better than LRU?

- As far as page replacement algorithm is concerned, LRU will replace the page that is least recently used, while MRU will replace the page that is most recently used. In the situation when a repeated sequence of pages are continuously inquired, MRU will perform better than LRU since the most recently used page will be inquired after a long time.

(EXTRA) How do LRU and MRU compare when many processes allocate much static and dynamic memory, and then recall it?

- In this situation, MRU will probably perform better than LRU algorithm. Since many processes are allocating memory one by one, the most recently used page will not be retrieved in the near future, while some least recently used page may be taken advantaged of.

---

1. Minix 3 Virtual Address, https://condor.depaul.edu/~glancast/443class/docs/slides/Oct06/slide16.html (accessed Nov. 20, 2020). ↩

2. kingkai, "Why malloc+memset is slower than calloc?," Stack Overflow, Apr. 22, 2010. https://stackoverflow.com/questions/2688466/why-mallocmemset-is-slower-than-calloc (accessed Nov. 22, 2020). ↩

3. Weatherly, C. (2009). MinixVM: An Implementation of Virtual Memory in Minix 3 (Doctoral dissertation, College of William & Mary). ↩

4. "What do 'real', 'user' and 'sys' mean in the output of time(1)?," Stack Overflow, Feb. 17, 2009. https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1 (accessed Nov. 20, 2020). ↩