2.1 Memory management at kernel level

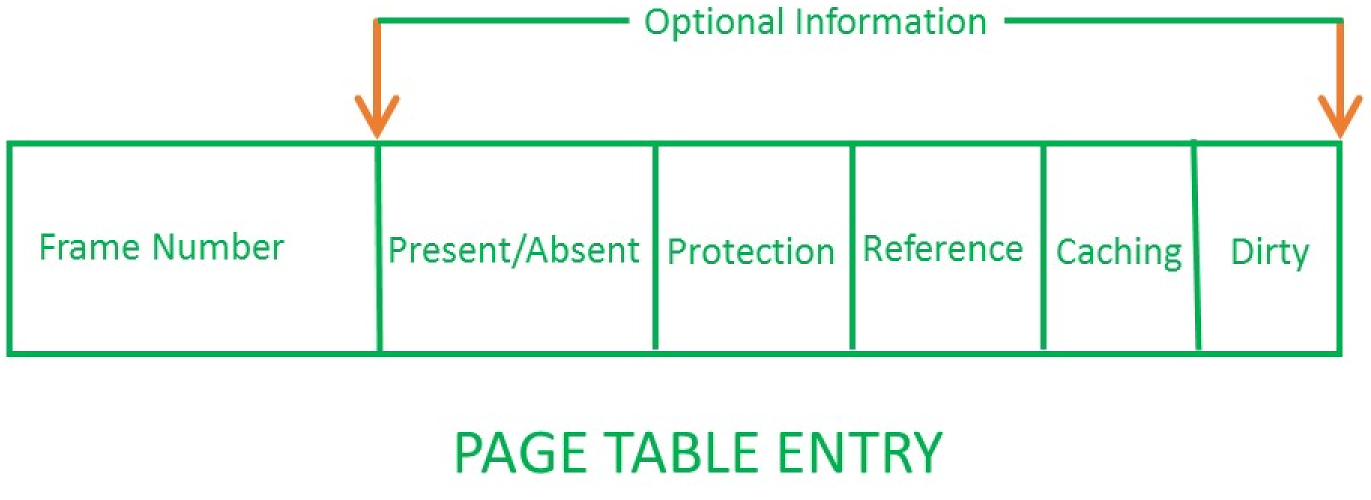
1. **What does `vm` stands for?**

   Virtual Memory

2. **Find the page table definition and search what fields each entry contain?** [1]

   A **page table** is the data structure used by a virtual memory system in a computer operation system to store the mapping between virtual addresses and physical addresses. Virtual addresses are used by the program executed by the accessing process, while physical addresses are used by the hardware, or more specifically, by the RAM subsystem. The page table is a key component of virtual address translation which is necessary to access data in memory.

   1. **Frame Number –** It gives the frame number in which the current page you are looking for is present. The number of bits required depends on the number of frames.Frame bit is also known as address translation bit.
   2. **Present/Absent bit –** Present or absent bit says whether a particular page you are looking for is present or absent. In case if it is not present, that is called Page Fault. It is set to 0 if the corresponding page is not in memory. Used to control page fault by the operating system to support virtual memory. Sometimes this bit is also known as **valid/invalid** bits.
   3. **Protection bit –** Protection bit says that what kind of protection you want on that page. So, these bit for the protection of the page frame (read, write etc).
   4. **Referenced bit –** Referenced bit will say whether this page has been referred in the last clock cycle or not. It is set to 1 by hardware when the page is accessed.
   5. **Caching enabled/disabled –** Some times we need the fresh data. Let us say the user is typing some information from the keyboard and your program should run according to the input given by the user. In that case, the information will come into the main memory. Therefore main memory contains the latest information which is typed by the user. Now if you try to put that page in the cache, that cache will show the old information. So whenever freshness is required, we don' t want to go for caching or many levels of the memory.The information present in the closest level to the CPU and the information present in the closest level to the user might be different. So we want the information has to be consistency, which means whatever information user has given, CPU should be able to see it as first as possible. That is the reason we want to disable caching. So, this bit **enables or disable** caching of the page.
   6. **Modified bit –** Modified bit says whether the page has been modified or not. Modified means sometimes you might try to write something on to the page. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk or it has to be written back or it has to be saved back. It is set to 1 by hardware on write-access to page which is used to avoid writing when swapped out. Sometimes this modified bit is also called as the **Dirty bit**.



3. **What basic functions are used to handle virtual memory?**

   In `minix-R3.2.1/servers/vm/main.c`, we can find functions related to virtual memory.

   The first three are initialization functions.

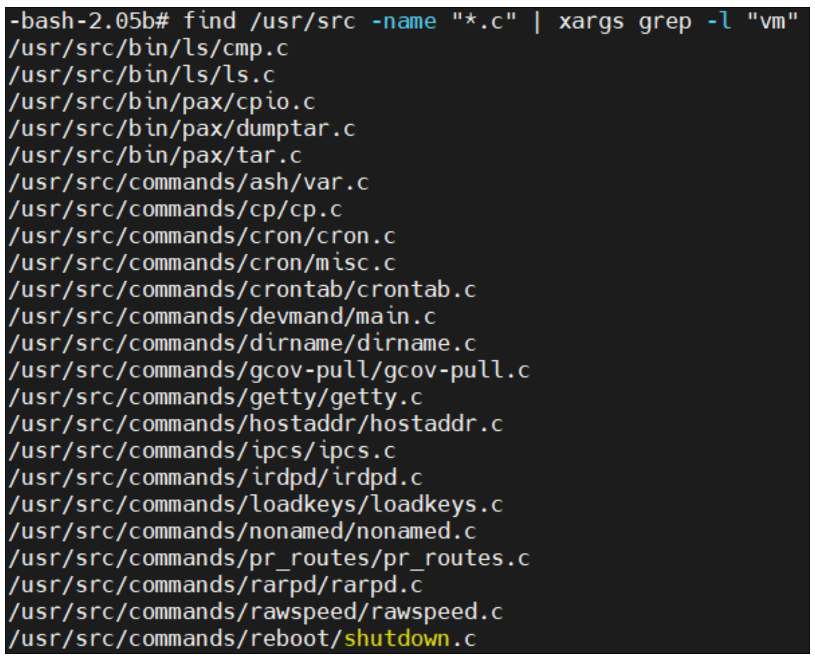
   - `init_vm()`: Initialize system so that all processes are runnable
   - `void sef_setcb_init_restart(sef_cb_init_t cb),void sef_setcb_init_response(sef_cb_init_response_t cb)`: Register init callbacks
   - `sef_startup()`: Let SEF perform startup.

   Then the virtual memory's main loop begins.

- `void pt_cycle(void)` : page table cycle
- `void do_pagefaults(message *m)` : deal with page faults
- `void pt_clearmapcache(void)` : Make sure kernel will invalidate table when using current pagetable to make new mappings

4. **Find all the places where the vm used inside the kernel, Why does it appear in so many different places?**

   By running `find /usr/src -name "*.c" | xargs grep -l "vm"`, we have the following result.



```
-bash-2.05b# find /usr/src -name "*.c" | xargs grep -l "vm"
/usr/src/bin/ls/cmp.c
/usr/src/bin/ls/ls.c
/usr/src/bin/pax/cpio.c
/usr/src/bin/pax/dumptar.c
/usr/src/bin/pax/tar.c
/usr/src/commands/ash/var.c
/usr/src/commands/cp/cp.c
/usr/src/commands/cron/cron.c
/usr/src/commands/cron/misc.c
/usr/src/commands/crontab/crontab.c
/usr/src/commands/devmand/main.c
/usr/src/commands/dirname/dirname.c
/usr/src/commands/gcov-pull/gcov-pull.c
/usr/src/commands/getty/getty.c
/usr/src/commands/hostaddr/hostaddr.c
/usr/src/commands/ipcs/ipcs.c
/usr/src/commands/irdpd/irdpd.c
/usr/src/commands/loadkeys/loadkeys.c
/usr/src/commands/nonamed/nonamed.c
/usr/src/commands/pr_routes/pr_routes.c
/usr/src/commands/rarpd/rarpd.c
/usr/src/commands/rawspeed/rawspeed.c
/usr/src/commands/reboot/shutdown.c
```

A total number of about 300 files altogether contains virtual memory. Virtual memory is always being used, because all the process require its only virtual process.

5. **How is memory allocated within the kernel? Why are not malloc and calloc used?**

   In minix kernel, `kmalloc` and `vmalloc` are used instead of `malloc` and `calloc`.

```
#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
```

`malloc` and `calloc` are userspace library functions, while `kmalloc` and `vmalloc` are used for kernel security cause.

6. **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()`

   What Minix is doing here is basically map a block memory assigned by the memory management unit. Then it insert this block into the page table and return the pointer of that block to the userspace.

   Minix3 use `get_user_pages` to map userspace memory to the kernel.

   It's good because it keep users from changing the memory accidentally, which increases security for our computer. However, this may generate larger time latency, which is bad for certain real-time systems.

7. **How are pagefaults handled?**

   The operating system first check whether available page frame exists in the physical memory. It maps the page and the page frame if so. If not, however, it looks for a page that is least frequently used. Then the OS writes its content back to the hard drive. Next, it reads the page onto the page frame and construct new mapping connection.

2.2 Mum’s Really Unfair!

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

   LRU algorithm, which is implemented in `/servers/vm/pagefault.c`. There are four functions in this file:

   - `char *pf_errstr(u32_t err)`: error handling
   - `void do_pagefaults(message *m)`: page fault handling
   - `void do_memory(void)`: If `SIGKMEM` is asserted, call `handle_memory` to allocate the memory.
   - `int handle_memory(struct vmproc *vmp, vir_bytes mem, vir_bytes len, int wrflag)`: allocate the memory according to the page size.

2. **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?**

   This command searches for the "mum" key word in Minix 3.2.1.

   The first time:

   ```
   real 0m4.833s
   user 0m0.000s
   sys 0m0.083s
   ```

   The second time:

   ```
   real 0m1.383s
   user 0m0.366s
   sys 00.650s
   ```

   We can conclude that after first time's command, relevant memory information has been cached. Therefore, in the second time's command, it costs much less time to complete the command, which corresponds to LRU.

3. **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

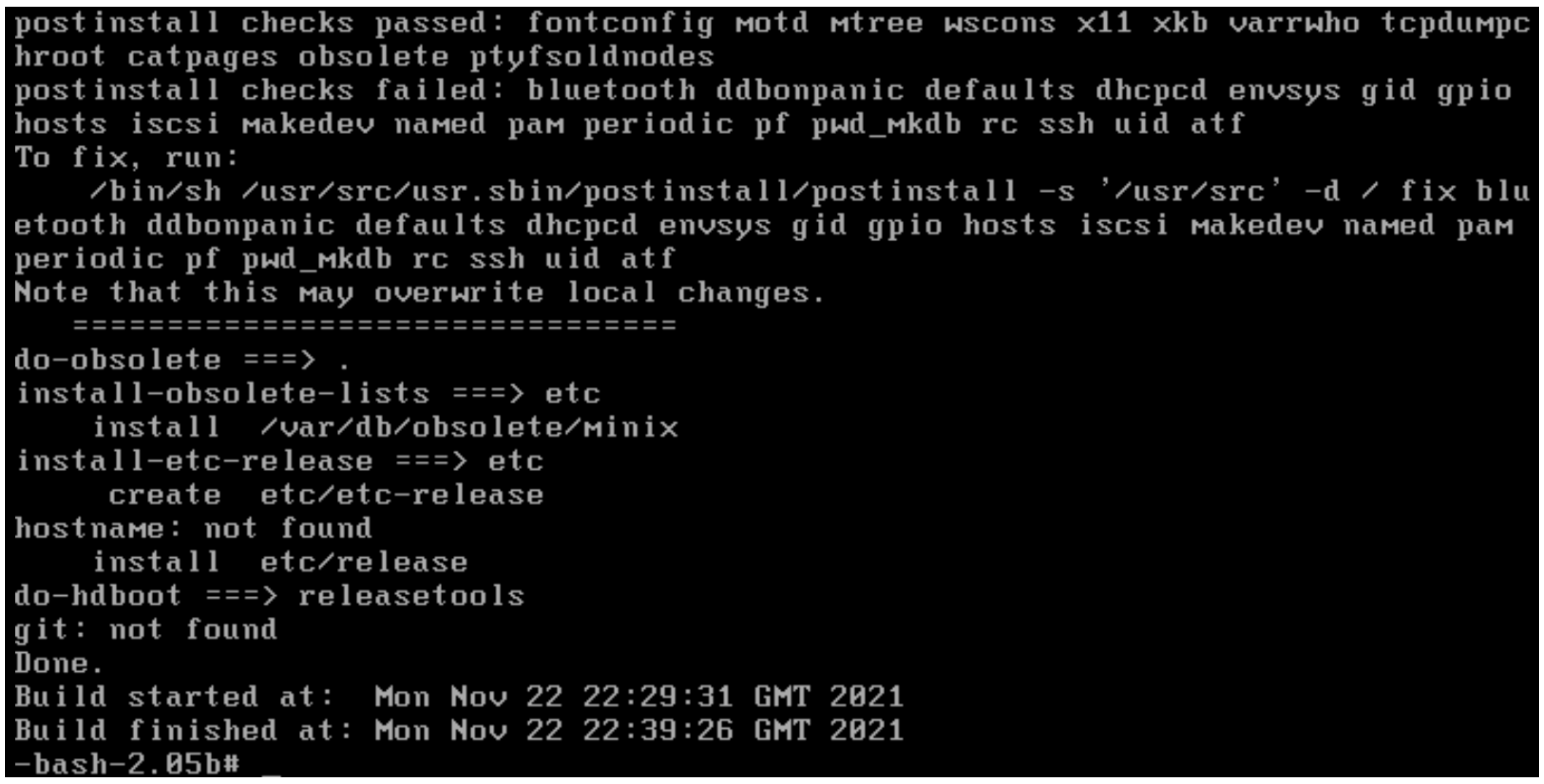
   Run the following commands to rebuild the kernel. First we transfer files to Minix3 through scp.

```
scp -P 2222 ./minix-R3.2.1/servers/vm/region.c root@192.168.65.129:/usr/src/servers/vm/region.c
scp -P 2222 ./minix-R3.2.1/lib/libminixfs/cache.c  root@192.168.65.129:/usr/src/lib/libminixfs/cache.c
```

   Then we recompile the kernel in Minix3.

```
# Assume you are the root
cd /usr/src
make build
reboot
```

Result:



```
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
Done.
Build started at:  Mon Nov 22 22:29:31 GMT 2021
Build finished at: Mon Nov 22 22:39:26 GMT 2021
-bash-2.05b# _
```

4. **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?**

   First time:

   ```
   real    0m5.833s
   user    0m0.016s
   sys     0m0.050s
   ```

   Second time:

   ```
   real    0m2.533s
   user    0m0.650s
   sys     0m1.416s
   ```

We can see that the first time runs about slitely slower compared with the LRU version. The second time only speeds up about 100%, which is a relatively small scale comparing with the LRU version.

5. **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?**

   On the occasion when a sequence of pages, repeating for several times are encountered, MRU may be better than LRU because the most recently used page will not be inquired again soon.

Reference

1. Page Table Entries in Page Table - GeeksforGeeks ↩