INFO0940-1: Operating Systems (Groups of 2)

Project 2: Implementing a kernel module

Prof. L. Mathy - G. Gain - B. Knott

Submission before Friday, 17th May 2024 (23:59:59 CET)

1 Introduction

Virtual memory is an abstraction that creates the illusion of a very large main memory. To manage memory efficiently, the kernel divides the virtual address space of a process into various blocks of fixed size (by default 4KB), called pages. When manipulating memory, the kernel first needs to consult the page table which contains the mapping between virtual addresses and physical addresses.

In this assignment, you will implement a Linux kernel module able to track memory usage across several processes. Interaction with the kernel will occur through the pseudo file system.

2 Module Behaviour

The module has a straightforward behaviour. During its initialization (or when the module is reset), it will create and populate an **in-memory** data structure with the names of all currently running processes on the system. For each set of process(es) with the same name, the module will store the following memory information:

- the pid(s) of all process(es) of the set;
- the total number of pages used by all process(es) of the set;
- the number of valid pages (i.e., pages that are present in RAM) of all process(es) of the set;
- the number of invalid pages (i.e., pages that are not present in RAM) of all process(es) of the set;
- the number of **readable**[1] pages that may be shared (e.g., identical) between all process(es) of the set;
- the number of groups of identical **readable** pages. If there are no identical pages, this field should be 0.

When the module is unloaded, the persisted data structure and all its associated resources must be freed.

3 Module Interaction

The module will be accessible through the pseudo file system via a file named `memory_info` in the `/proc` directory. This file will serve for both writing commands to the module and reading its output.
Interacting with the module will be done through the following write commands:

- `RESET`: reset the in-memory data structure and populate it with the name of all currently running processes and their memory information;
- `ALL`: display the memory information of all processes in a specific format (see below).
- `FILTER|<name>`: display the memory information of all processes called `name` in a specific format (exact match).
- `DEL|<name>`: delete the memory information of all processes called `name` (exact match).

[1] Readable pages are also valid pages (and may also be executable/writable).

For the last two commands, the "|" character is used as a separator between the command and the name. Using a malformed or unknown command will be treated as an error. (see Section 4). The module will output the result of the previous write command in the `/proc/memory_info` file.

The output format of the `DEL` and `RESET` commands, is only "`[SUCCESS]`" if the command is successful. Otherwise, an error message will be displayed (see Section 4).
The output format of the `ALL` and `FILTER` commands is showing in Listing 1.

```
<name>, total: <nb_total_pages>, valid: <nb_valid_pages>, invalid: <nb_invalid_pages>,
    may_be_shared: <nb_shareable_pages>, nb_group: <nb_group>, pid(#pid): <pid1>; <pid2>; ... <
    pidN>
```

Listing 1: Required format for `ALL` and `FILTER` commands.

Listing 2 shows a sequence of commands illustrating how to interact with the module:

```
# Display the memory information of all processes
echo "ALL" > /proc/memory_info && cat /proc/memory_info
sshd, total: 7845, valid: 75, invalid: 7770, may_be_shared: 4, nb_group: 2, pid(3): 861; 867; 892
cron, total: 1396, valid: 11, invalid: 1385, may_be_shared: 0, nb_group: 0, pid(1): 1752
dhclient, total: 1505, valid: 21, invalid: 1484, may_be_shared: 0, nb_group: 0, pid(1): 1711
systemd, total: 3550, valid: 420, invalid: 3130, may_be_shared: 3, nb_group: 1, pid(2): 1; 1870
zsh, total: 2065, valid: 11, invalid: 2054, may_be_shared: 0, nb_group: 0, pid(1): 1893
[...]

# Search and display the memory information of "sshd" processes
echo "FILTER|sshd" > /proc/memory_info && cat /proc/memory_info
sshd, total: 7845, valid: 75, invalid: 7770, may_be_shared: 4, nb_group: 2, pid(3): 861; 867; 892

# Search and delete the memory information of "sshd" processes
echo "DEL|sshd" > /proc/memory_info && cat /proc/memory_info
[SUCCESS]

# Does not exist anymore, so we should have an error
echo "FILTER|sshd" > /proc/memory_info && cat /proc/memory_info
[ERROR]: No such process

# Reset the data structure and populate it with the name of
# all currently running processes and their memory information
echo "RESET" > /proc/memory_info && cat /proc/memory_info
[SUCCESS]
```

Listing 2: Example of a sequence of commands to interact with the module

4 Errors Handling and Error Messages

In the kernel, it is necessary to use very specific errors so that users can know what causes an operation to fail to complete. To do this, the variable `errno` is used. Depending on the error, your implementation must return an appropriate error code. In addition, you must also display an error message in the kernel log and in the `/proc/memory_info` file.

The error codes and the associated messages are described in Table 1.

Error	Message	Meaning
`ENOENT`	No such file or directory	The file or directory does not exist.
`EFAULT`	Bad address	The address is invalid (e.g., error during memory copy).
`ESRCH`	No such process	The process does not exist.
`EINVAL`	Invalid argument	An invalid argument was given (e.g., wrong command).
`ENOMEM`	Memory allocation error	There is not enough memory to complete the operation.

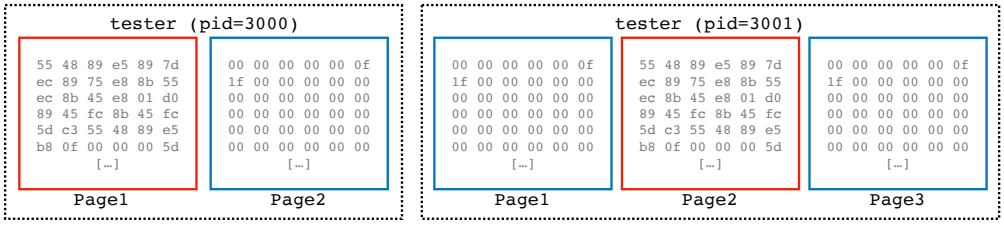
Table 1: Error codes and the associated messages.

It is up to you to know which one is the most suitable according to the error encountered. Note that, in some cases, you may only display the error message without returning an error code.

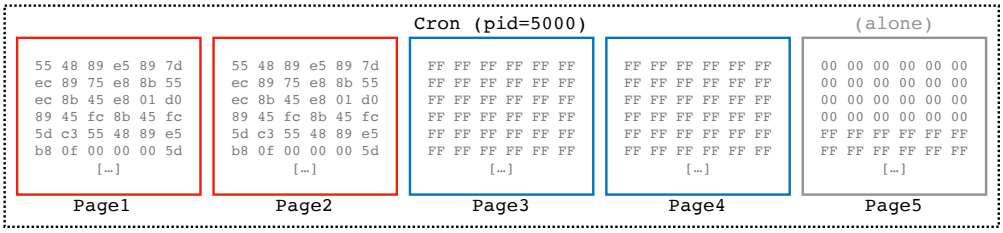
5 Managing Identical Pages (6 points)

In the Linux kernel, when multiple processes are running concurrently, it is common for certain pages of memory to be identical across these processes. This phenomenon occurs because many processes execute the same code or access the same shared data. As a result, the kernel can optimize memory usage by mapping these identical pages to a single physical memory location. This saves memory and improves performance by reducing the need to duplicate (physically) identical pages for each process. To detect and merge identical pages, the kernel can use a memory-deduplication daemon (KSM [2] - Kernel Same-page Merging).

In this assignment, you are asked to implement a mechanism to detect identical readable pages between processes which have the same name and to count the number of groups of identical readable pages. Note that a single process can also have identical readable pages. Figure 1 illustrates the concept of identical pages according to two different scenarios: Scenario 1a shows a group of two processes having identical readable pages, while Scenario 1b shows only one process having identical readable pages.



(a) The "`may_be_shared`" field is set to 5 and the "`nb_group`" field is set to 2.



(b) The "`may_be_shared`" field is set to 4 and the "`nb_group`" field is set to 2.

Figure 1: Identical pages according to two different scenarios.

We provide a user-space program that can be used to see if your module can correctly detect identical pages[3]. The program is available on eCampus.

6 Other Specification

- For simplicity, the maximum length of a process name is limited at 16 characters (this limit can be hardcoded);
- You may use static allocation only when appropriate, i.e. for small data structures whose length is known at compile time. For any other data structure, you **must** use dynamic allocation (e.g., `kmalloc`, `vmalloc`, etc.);
- Your implementation must display error messages with `KERN_ERR` in the kernel logging system. It is necessary to trace **all** errors by prefixing the error message with `[ERROR]`;
- You can use any type of data structure such as lists, trees and hash tables, but choose widely according to the context (do not reinvent the wheel!);
- You must only consider the x86 architecture (32bits) and version 4.15.0 of the Linux kernel.
- You must only populate the data structure with the name of the processes and their memory information when the module is loaded or when the module is reset.

[2] `https://elixir.bootlin.com/linux/v4.15/source/mm/ksm.c`

[3] To test the number of total/(in)valid/shared pages, we only consider processes that have been statically-linked. Note that we will NOT test your module (the memory stats) with dynamically-linked processes.

- Do not consider huge pages, consider only 4KB pages.
- The pseudo file must be named `memory_info`.
- Process(es) with no page (e.g., `NULL task->mm struct`) must be ignored.
- Your code must be readable. Use common naming conventions for variable names and **comments**.
- Your code must be robust and must not crash. In addition, errors handling and cleaning must be managed in a clean way.
- You can use kedr to track memory leaks.

7 Report

You are asked to write a very short report (**max 1 page**). A template of the report is provided on eCampus. The report should contain the following information:

- The names and the student IDs of the authors.
- A short overview of your structure/code. You can draw diagrams to illustrate your explanations but note that these ones **will not** be considered as appendices.
- A feedback on the assignment. In particular:
  - Was it too easy or too difficult compared to other projects?
  - Was it too short or too long compared to other projects?
  - If you have some remarks (positive or negative), you are free to write them.

8 Evaluation and Tests

Your program can be tested on the submission platform. A set of automatic tests will allow you to check if your program satisfies the requirements. Depending on the tests, a **temporary** mark will be attributed to your work. Note that this mark does not represent the final mark. Indeed, other criteria such as the structure of your code, the memory management, etc, will also be considered. You are however **reminded** that the platform is a **submission** platform, not a test platform.

9 Submission

Projects must be submitted before the deadline. After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of $2^N - 1$ marks (where N is the number of started days after the deadline).

Your submission will include your code (all the required {`.c`|`.h`} files in a directory named `src`) along with a Makefile which produces the kernel module file `module_project_os.ko`. Please also add your report (`.pdf`) in the `src` directory.

Submissions will be made as a `group_{your_group_ID}.tar.gz` archive (for example `group_3.tar.gz`), on the submission system. Failure to compile will result in an awarded mark of 0.
Bon travail...