INFO0940-1: Operating Systems (Groups of 2)

Research Project 2: Adding System Calls

Prof. L. Mathy - G. Gain - J. Francart

Submission before Wednesday, May 12, 2021 (23:59:59)

1 Introduction

Virtual memory is an abstraction which allows to create the illusion of a very large main memory. To manage memory efficiently, the kernel divides the virtual addressing 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 is used by virtual memory to store the mapping between virtual addresses and physical addresses.

In this assignment, you will implement several syscalls in order to retrieve specific information about processes in the Linux kernel. You will mainly deal with statistics about the virtual memory of some specific processes.

2 Assignment

For this project, you will develop several features within the Linux kernel that implement the following system calls as an interface for user space applications.

```
long getnbpages(const char *process_name, const size_t name_len);
long getreadpages(char *bitmap, const char *process_name, const size_t
    name_len);
```

We provide a user space program (at the following github repository[1]) to test this interface. You can bring some slight modifications to the code but do not modify directly the syscalls number otherwise the program will not work anymore.

[1] https://github.com/gaulthiergain/INFO0940-1_project2

2.1 getnbpages

This system call has two different purposes:

1. First, it will allow you to retrieve the number of pages of one or several processes with a specific name (according to the `process_name` argument). For this part consider the whole virtual memory areas (VMA[2]) of the process(es).

2. While iterating through the page table to count the number of pages, it will also add all pages that are **present** in the main memory and in **read-only** into a kernel data structure which will be associated with the name of the corresponding process(es). If an entry already exists, it will override it.

Arguments and return value specification:

- `process_name`: pointer to a string that represents the name of the process(es) to analyze.
- `name_len`: length of the `process_name` string.
- *Return value:* On success, it returns the number of pages of one or several process(es) with a specific name. On failure, it returns a specific error code according to the error caused (see Section 3).

2.2 getreadpages

This system call will allow you to retrieve a bitmap structure of the size of a number of pages (obtained by the `getnbpages` system call) that will be set by the kernel. Basically, it will iterate through the previous data structure[3] associated to the specific process(es) and initialize the bitmap with 1 (page is present and in read-only) or 0 (other cases). For instance, if we have a process that has 8 pages, and that its first three pages are present in the main memory and in read-only, we will have the following bitmap: $\{1, 1, 1, 0, 0, 0, 0, 0\}$.

Once its work is done, it permanently deletes the data structure associated to the given process(es) (according to its name) and frees the corresponding kernel memory.

[2]The kernel uses virtual memory areas to keep track of the process's memory mappings; for example, a process has one VMA for its code, one for the heap, etc. A VMA consists of a number of pages, each of which has an entry in the page table.

[3]and not the page table

Arguments and return value specification:

- `process_name`: pointer to a string that represents the name of the process(es) to analyze.
- `name_len`: length of the `process_name` string.
- `bitmap`: pointer to a bitmap that represents an array of char which will be set by the kernel. If an index i is set to 1, it means that the corresponding page i of the process is present in the main memory and in read-only. Otherwise, the value is 0.
- *Return value:* On success, it returns 0. On failure, it returns a specific error code according to the error caused (see Section 3).

2.3 System Call Numbers

The system call number assignment is as follows (it is not allowed to change them):

```
385: getnbpages
386: getreadpages
```

For the last system call, you might have slightly different results at each run. This is due to the volatility of the main memory.

3 Generic Requirements

The following requirements also have to be satisfied:

- You can use any type of data structures such as list, tree, and hash tables;
- You must only consider **x86** architecture (32bits);
- Your code must be readable. Use common naming conventions for variable names and functions. In addition, your patches must also be clean;
- Your code must be robust and must not crash. Cleaning must be properly managed;
- You must manage errors in a clean way. For instance, a memory allocation error will return an *-ENOMEM* error code (out of memory)[4]. If it is not relevant to use an errno code, please return *-1*. In addition, you must display an error message with the following prefix "[INFO940][ERROR]" on the kernel log (`/var/sys/log`);

[4]Please refer to the *errno-base.h* header for kernel error codes.

- You can use the kernel log for debugging but do not forget to remove all `printk` that are not related to errors otherwise it will generate some overhead in the kernel;
- Do not consider huge pages, consider only 4KB pages.

4 Report

You are asked to write a very short report (**max 2 pages**) in which you explain your implementation. In addition, briefly explain how the memory of a process is managed by the kernel.

Note that, you can draw diagrams to illustrate your explanations but note that these ones **will not** be considered as appendices.

Finally, we would also appreciate an estimation of the time you passed on the assignment, and what were the main difficulties that you have encountered.

5 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, another criteria such as the structure of your code, the memory management, etc. You are however **reminded** that the platform is a **submission** platform, not a test platform.

6 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).

You must submit patches that contain diffs from linux-4.15. The patch file(s) has/have to be generated by the `git format-patch` command. Your submission will include your patch files in a directory named `patch`. In addition, add also your report in `.pdf` or `.txt` within the `patch` directory.

Submissions will be made as a `patch.tar.gz` archive, on the submission system. Failure to compile or to apply patches will result in an awarded mark of 0.

Bon travail...