CS342 Operating Systems – Spring 2023
Project #4 – Physical and Virtual Memory Information

Assigned: May 15, 2023.
Due date: May 31, 23:59. Document version: 1.3

- This project will be done in groups of four students. A group may have less than four students, but not more. If you wish, you can do it individually as well. The group members can be from different sections.
- This project should be done in a **64-bit** machine with **x86-64** CPU architecture. You can run Linux over the bare hardware or over a virtual machine.
- Please start as soon as possible. Work incrementally, step by step.
- You will program in C. Programs will be tested in Ubuntu Linux.
- Goals of the project: Exercise with: paging, 4-level paging, virtual addresses, physical addresses, virtual memory layout and regions of a process, physical frames, page numbers, physical frame numbers, address translation, sharing pages, physical and virtual memory usage.

In this project you will implement a Linux program, in C, that will give **memory** related information for a **process** and for the physical memory of the system. The program will be called `pvm`. It will take the memory related information from the `/proc` virtual file system, which is an **interface** to the kernel for user-space programs (applications). In the `/proc` directory, there are a lot of files that can be read by a user program or viewed by a user to get various information about the **system state** and processes. The related information is retrieved from various kernel data structures and variables (from **kernel space**). Hence, the content of these files are derived from kernel memory, not from disk. In `/proc` directory, there is a sub-directory for each process, to get process specific information maintained by the kernel.

The project will be done on a **64** bit machine with **x86-64** architecture. The x86-64 architecture is using 4-level page tables for a process. Your program will use the following four `/proc` files to retrieve the requested information.

- `/proc/pid/maps`: A program can read this file to get information about the **virtual memory areas** (i.e., virtual memory regions) of the process with id `pid`. This is a text file. Therefore, you can view the content of it by typing `cat /proc/pid/maps`. By reading this file, your program will understand which parts of the virtual memory of the process `pid` are used (contains legal pages) and which parts are unused (undefined).

- `/proc/pid/pagemap`: This file lets a program get information about the **virtual pages** of a process with id `pid`. It is a binary file. It contains one entry (64 bits) per page. In the entry, the respective **frame number** is stored if the page is in memory (Present bit is 1), including some flags. It is indexed by virtual page number.

- /proc/kpagecount: This file lets a program to learn the **number of times** a page in a frame is mapped (i.e., how many processes are using the page at that moment). If there is **no page** in the frame (frame is empty), the mapping count is 0. This file is a binary file. It is indexed by physical frame number. Each entry is 64 bits long.

- /proc/kpageflags: This file lets a program to get some more information **about** each physical frame and **the page contained in it** (if any), i.e., some flags associated with each frame/page. Each entry is 64 bits long. The file is indexed by physical page number. This is a binary file.

You can learn more information about these files from the related kernel documentation [1]. In your program you can use open(), read(), close() **system calls** to access a **binary** file. You also need to use the lseek() call to jump to an entry and read it. In your program you can use fopen, fscanf, fclose functions to access a **text** file.

Your program will **only** use **these files**. Your program is not allowed to use other files from the /proc directory.

As mentioned before, we can see the **virtual memory layout** (virtual memory map) of a process pid by typing "cat /proc/pid/maps". We will get a listing of used virtual memory areas (regions) of the process. An example content of /proc/pid/maps file is shown below (numbers are in hexadecimal).

```
563962f0e000-563962f0f000 r-xp 00000000 08:01 45880146 /.../app
56396310e000-56396310f000 r--p 00000000 08:01 45880146 /.../app
56396310f000-563963110000 rw-p 00001000 08:01 45880146 /.../app
563963110000-563963111000 rw-p 00000000 00:00 0
563963d4e000-563963d6f000 rw-p 00000000 00:00 0        [heap]
7f94152ae000-7f9415495000 r-xp 00000000 08:01 67109118 /lib/.../libc-2.27.so
7f9415495000-7f9415695000 ---p 001e7000 08:01 67109118 /lib/.../libc-2.27.so
7f9415695000-7f9415699000 r--p 001e7000 08:01 67109118 /lib/.../libc-2.27.so
7f9415699000-7f941569b000 rw-p 001eb000 08:01 67109118 /lib/.../libc-2.27.so
7f941569b000-7f941569f000 rw-p 00000000 00:00 0
7f941569f000-7f94156c6000 r-xp 00000000 08:01 67108949 /lib/.../ld-2.27.so
7f94158ad000-7f94158af000 rw-p 00000000 00:00 0
7f94158c6000-7f94158c7000 r--p 00027000 08:01 67108949 /lib/.../ld-2.27.so
7f94158c7000-7f94158c8000 rw-p 00028000 08:01 67108949 /lib/.../ld-2.27.so
7f94158c8000-7f94158c9000 rw-p 00000000 00:00 0
7ffe45bb7000-7ffe45bd8000 rw-p 00000000 00:00 0          [stack]
7ffe45be5000-7ffe45be8000 r--p 00000000 00:00 0          [vvar]
7ffe45be8000-7ffe45bea000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0  [vsyscall]
```

The output above lists information about 19 **virtual memory areas** of a process (i.e., used regions of the process virtual memory), whose name is app. For each virtual memory area, on the left side, we see the range of virtual addresses [startvaddr-endvaddr) used by that virtual memory area. For example, the heap section of the process has **virtual addresses** in the range [0x563963d4e000-0x563963d6f000).

The possible **options** and **invocations** of the program are listed below. The virtual addresses (VA) and frame numbers (PFN) specified as part of an option can be in **decimal** (a number not starting with 0x) or in **hexadecimal** (a number starting with 0x). A pid value should be entered in decimal. The program will print out a physical address, a page number, or a frame number in **hexadecimal** form. The output format for each option is up to you, except the explicitly stated requirements below.

1. pvm -freefc PFN1 PFN2: When invoked with this option, your program will find and print out (in decimal) the number of empty (i.e., free) frames (mapped count = 0, or nopage flag is set) between physical frame numbers PFN1 and PFN2, i.e., in range [PFN1, PFN2). An example invocation is: pvm -freefc 0x00 0x1a. That means we want to count the free frames in range [0, 26) (in decimal). [this option is cancelled].

2. pvm -frameinfo PFN: When invoked with this option, your program will print detailed information (various **flag** values and **mapping count**) for the specified frame. PFN is the frame number. The flags whose value will be printed out are those that can found in the /proc/kpageflags file (see [1]). The format of the output is up to you.

3. pvm -memused PID: When invoked with this option, your program will find out the **total** amount of **virtual memory** and **physical memory** used by the process PID (in KB). You will calculate the total virtual memory usage from the virtual memory areas of the process (read from /proc/PID/maps). For physical memory usage, you will calculate two values (in KB). For the first value, you will only consider the process pages that are in memory and used only by that process (i.e., **exclusively** mapped). A page that is exclusively mapped will have a **mapping count** as 1 in the respective entry of the /proc/kpagecount file. For the second value, you will consider all the process pages that are in memory (pages with mapping count >= 1; in other words, pages for which **Present** bit is 1). Hence, when invoked with this option, your program will output three values (all in KB). All values will be in decimal.

   You can use the **pmap** -X pid command to check your results about memory usage. Type man pmap to learn more information about pmap program. You need to check the Size and Rss columns in the output of pmap for a process. Size gives the total virtual memory used in KB. Rss (resident set size) gives total physical memory used in KB (including shared pages).

4. pvm -mapva PID VA: When invoked with this option, your program will find and print out the **physical address** corresponding to the virtual address VA for the process PID. The physical address will be printed in hexadecimal in the form 0x.... The printed value will be 16 digits long.

The first 4 digits will be `0x0000`.

5. `pvm -pte PID VA`: When invoked with this option, your program will find and print out detailed **information** for the **page** corresponding to the virtual address `VA` of the process `PID`. The information will be obtained from `/proc/PID/pagemap file`. Numbers (physical frame number or swap offset) must be printed in hexadecimal form. For a given virtual address, if the corresponding page is not used or is not in memory, you need to print a proper message.

6. `pvm -maprange PID VA1 VA2`: When invoked with this option, your program will find and print out (page number, frame number) **mappings** for the virtual address **range** [`VA1`, `VA2`). For each page in the range, your program will print a line of information that contains the page number and the corresponding frame number (if any). If a page in the range is not a used page (i.e., is not in one of the virtual memory areas of the process), you will print `unused` instead of a frame number. If a page is a used page (i.e., in one of the virtual memory areas of the process), but is not in memory, you will print `not-in-memory` instead of a frame number.

7. `pvm -mapall PID`: When invoked with this option, your program will find and print out (page number, frame number) mappings for all the virtual memory areas (i.e., used pages) of the process `PID`. For each used page of the process (i.e., the page is in one of the virtual memory areas of the process), your program will print a line of information that contains the page number and the corresponding frame number (if any). If the page is not in memory, then you will print `not-in-memory` instead of a frame number.

8. `pvm -mapallin PID`: This is same with the previous option (option 7) except that your program will not print information about used pages that are not in memory. It will print (page number, frame number) mappings only for the pages in main memory.

9. `pvm -alltablesize PID`: When invoked with this option, your program will calculate the total memory required to store **page table information** for the process `PID`. That means it will find the total size (in KB) of the 1st level, 2nd level, 3rd level, and 4th level page tables needed for the process `PID`. There will be only one 1st level page table for a process. For the other levels, how many tables will be needed depends on the virtual memory areas (their start addresses and their sizes) of the process `PID`. In calculating the page table size for a process `PID`, your program should consider all the legal pages of the process `PID`, no matter whether the page is in memory or not. We consider a page as legal (used) if it is in one of the virtual memory areas of the process. For this option, you only need the virtual memory mapping information from `/proc/PID /maps` file and the 4-level paging details of x86-64

architecture.

The x86-64 architecture is using **4-level paging.** You can learn more about the paging scheme used in x86-64 architecture from related documentation in Internet and also from the course slides. Of a 64-bit virtual address, only least significant **48 bits** are used. The most significant 16 bits are not used. Therefore, a virtual address is effectively 48 bits long. The address split scheme is as follows: `[9, 9, 9, 9, 12]`. That means page offset is 12 bits. Hence page size is 4 KB. The top level table and each other level page table has 2^9 = 512 entries. An entry is 64 bits long.

You need to run your program by using the `sudo` command. For example:

```
sudo ./pvm -mapall PID
```

Submission

You will submit as in the previous projects.

References:

- [1]. Linux Kernel Pagemap Interface to User Space. *You **definitely** need to **read** this.* URL: https://www.kernel.org/doc/Documentation/vm/pagemap.txt
- [2]. Intel Architectures Software Developer's Manual, Systems Programming Guide. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
- [3] Pagemap Interface of Linux Explained. URL: https://blog.jeffli.me/blog/2014/11/08/pagemap-interface-of-linux-explained/
- [4]. Complete Virtual Memory Map with 4-level Page Tables. URL: https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

Tips and Clarifications

- Start early; work incrementally.
- As an example, for the use of your program, you can check the frames that are used by many processes and see if there are some shared frames/pages. You can also see if frames are shared by multiple processes running the same application.
- Other process and memory related Linux tools and `/proc` files are: `top`, `ps aux`, `cat /proc/meminfo`, `cat /proc/vmstat`, `cat /proc/zoneinfo`.
- In our tests, we will never use a virtual address outside of the range [`0x0000000000000000, 0x00007fffffffffff`].
- You can represent a virtual address, a physical address, a page number, or a frame number with `unsigned long` type, which is 8 bytes (64 bits) long.
- You program will be tested in a 64-bit machine with x86-64 architecture.

- A `/proc/pid/pagemap` file for a process `pid` is indexed by virtual page number. For a 48 bit-virtual address, this page number is the most significant 36 bits (4 x 9 = 36). The remaining 12 bits of a virtual address indicate a page offset. In other words, given a virtual address, you can obtain the corresponding virtual page number by dividing the virtual address by 4096 (page size). Hence, to find out the respective frame number, you don't need to walk over the 4-level paging information of the process. You just need to access the respective entry in the `/proc/pid/pagemap` file, that corresponds to the virtual page number (36 bits value). File offset of the entry is: virtual_page_number * 8. An entry in `pagemap` file is 8 bytes long (64 bits).
- You can choose a VA to convert to a physical address by looking to the maps file. Type: cat `/prod/pid/maps`.
- The last line of the maps output (`cat /proc/pid/maps`) corresponds to a virtual memory area that starts with virtual address `0xffffff600...` (`vsyscall`). It is one page long. Don't worry if you can not parse the related entry in the pagemap file properly (for example, for the `-pte` option, or, `-mapva` option). That one page will be counted to the virtual memory usage of a process (`-memused`). It will also be considered in the calculation of the total page table size (`-alltablesize`). It will not be counted to the physical memory usage of a process. (`-memused`).
- If PFN corresponds to a frame that has no page in it, then output of `–frameinfo` option is undefined; you can assume this.