Operating Systems
Assignment 3 – *Easy*

April 14, 2025

Introduction

In this assignment, we add the following capabilities to xv6:

1. Memory Printer: This prints the number of pages allocated to a user process.
2. Page Swapping: Pages are periodically moved to the disk using an adaptive page replacement policy.

1 Memory Printer

In the first part, we create a handler that prints the number of pages (in the user space) residing on the RAM for each of the SLEEPING, RUNNING and RUNNABLE process with $pid >= 1$. This handler is invoked by pressing `Ctrl + I`. The format of the output should be as follows:

```
Ctrl+I is detected by xv6
PID NUM_PAGES
1   x
2   y
3   z
...
...
```

Hint: You can check how many pages the `exec` system call allocated while loading the program into memory. Later, when the program starts executing, it might expand its memory layout (stack, heap, etc.). This should also be taken into account when counting the number of pages in RAM. For the `init` process, the number of user-space pages will be 3 (assuming that no swapping is invoked and no change is made to the original `init.c` file).

Note: You must ensure that the execution of the currently running process resumes after the handler is serviced.

2 Swapping Implementation in `xv6`

For better understanding of xv6-memory system you can refer to xv6 Memory Layout Notes.

2.1 Modified Disk Layout

The current disk layout of xv6 is

`[ boot block | sb block | log | inode blocks | free bit map | data blocks ]`

To transfer the memory pages on the disk, you will have to reserve a set of disk blocks. Introduce a *swap blocks* partition between the `sb block` and the `log` of the disk layout (refer to `mkfs.c`).

Further, split the swap blocks into an array of swap slots, each representing eight consecutive disk blocks to store a page (since a page size is 4096 bytes and a disk block is 512 bytes). Each slot is of type `struct` with two attributes: `page_perm` (`int`) and `is_free` (`int`). `page_perm` stores the permission of a swapped memory page, and `is_free` denotes the availability of this swap slot. Note that you must initialize the array of swap slots at boot time. For this assignment, **800** swap slots should suffice, and you can accordingly change the disk size (FSSIZE in `param.h`).

The process of saving a memory page to disk involves storing the contents of the page into eight consecutive disk blocks. Unlike file system blocks, disk writes to swap blocks bypass the log layer because they store volatile memory pages (refer to `fs.c`). This ensures that the process of saving the contents of the memory pages to disk is not unnecessarily slowed down by the log layer.

2.2 Memory Organization

To allocate a memory page, xv6 invokes the `kalloc()` function. Currently, the system crashes if we allocate more than the available memory space. For this assignment, you need to limit the available physical memory to 4MB. This is done using `#define PHYSTOP 0x400000` in `memlayout.h`. To address this issue, you will add page swap functionality to xv6 in a new file, e.g., `pageswap.c`.

2.3 Page Replacement

Prior to discussing the swapping-in and swapping-out procedures, let us first define a policy to find a process and its page for swapping (referred to as victim process and victim page). To find a victim process, you first choose a process with the highest number of user-pages residing in the RAM. For this, you can augment the current `struct proc` with another attribute `rss`, which denotes the number of user space memory pages residing in the memory. In case two processes have the same `rss` value, choose the one with the lower `pid`. Whenever a process exits, make sure to clean the corresponding swap slots.

To find a victim page, iterate through the pages of the victim process and choose a page with the `PTE_P` flag set and the `PTE_A` flag unset. The `PTE_P` flag indicates whether the page is present in memory, and the `PTE_A` flag indicates whether the page is accessed by the process, which is set by the xv6 paging hardware (you can refer to Chapter 2 of the xv6 book).

2.4 Swapping-in Procedure

If a process attempts to access a swapped-out page, then x86 will generate the `T_PGFLT` trap signal (refer to `trap.c`). You will have to invoke a page fault handler on receiving this signal. The page fault handler first reads the virtual address of the page from the `cr2` register. Then, find the starting disk block ID of the swapped page from the page table entry. To load the page into memory, first allocate a memory page using `kalloc()`, then copy data from disk to memory, restore the page permissions, and update the page table entry of the swapped-in page.

2.5 Adaptive Page Replacement Strategy

Let $\alpha \in [0, 100]$, $\beta \in [0, 100]$, $Th = 100$ (threshold) and $Npg = 2$ (number of pages to be swapped out), $LIMIT = 100$.

You should actively check the current available free pages in memory (for this you need to understand how the kernel allocates memory to the user process from the free space list). If the number of free pages drops below the threshold (Th), then:

1. Print `"Current Threshold =` Th`, Swapping` Npg `pages"` in the exact format.
2. Swap-out Npg pages to the disk.
3. Update: $Th = Th(1 - \beta/100)$ (take floor value)
4. Update: $Npg = min(LIMIT, Npg(1 + \alpha/100))$ (take floor value)

α (`ALPHA`) & β (`BETA`) should be set using Makefile. For the initial configuration, you can set $\alpha = 25$ & $\beta = 10$.

This adaptive page replacement strategy might not be the most optimal one. Students are welcome to post more efficient strategies on Piazza (only justification, no implementation).

3 Testing

Set `#define PHYSTOP 0x400000` in `memlayout.h`.
We are providing you with `memtest.c` file. On running it without any swapping mechanism it will print `'Memtest Failed'`, due to lack of memory.

Upon correct implementation, you should see output something like this:

```
$ memtest
Current Threshold = A, Swapping B pages
Current Threshold = C, Swapping D pages
Current Threshold = E, Swapping F pages
...
...
...
...
...
Current Threshold = P, Swapping Q pages
Current Threshold = R, Swapping S pages
Memtest Passed
```

4 Submission Instructions

- We will run MOSS on the submissions. Any cheating will result in a zero in the assignment, a penalty as per the course policy and possibly much stricter penalties (including a fail grade).
- In your report clearly explain the working of swapping mechanism. Also give an in-depth explanation of how values of α and β determine the overall efficiency of the system while swapping the pages out.

How to submit:

1. Copy your report to the xv6 root directory.
2. Then, in the root directory run the following commands:

   ```
   make clean
   tar czvf \
     assignment3_easy_<entryNumber1>_<entryNumber2>.tar.gz *
   ```

   This will create a tarball with the name, $assignment3\_easy\_ < entryNumber1 > \_ < entryNumber2 > .tar.gz$ in the same directory that contains all the xv6 files. Entry number format: 2020CSZ2445 (*All English letters will be in capitals in the entry number.*). **Only one** member of the group is required to **submit** this tarball on Moodle.
3. If you are attempting the assignment as an individual, you do not need to mention the entryNumber_2 field.