Operating Systems–2: CS3523
January 2022

Programming Assignment 6: Paging

Submission Date: 20th April 2022 (Wednesday), 9:00 pm

You can continue on this assignment using the same xv6 repo where you implemented the system call in Assignment-2.

While we discussed xv6 code in detail in our lectures, there is a book containing internal details of xv6, to understand some deeper insights into xv6. It is located at:
https://drive.google.com/file/d/1ujCX6t0T6o9MGR9vrUdsyWQmCkygByn3/view?usp=sharing

Part-1: Printing the page table entries

Your first task is to quickly revisit and recap your system call implementation from assignment-2. Now, modify the xv6 kernel to implement a new system call named **pgtPrint()** which will print the page table entries for the current process. Since the total number of page table entries can be very large, the system call should print the entries only if it is valid and the page is allowed access in user mode.

Hint: Use the PTE_P (present bit) to check for valid pages and PTE_U (user) bit to check for user mode access.
Hint: The pointer to the page directory can be obtained using myproc()->pgdir. You need to go through each pgdir entry to find the location of the page table and then read the mappings from the page table.
But page directory stores the **physical** address of page tables and therefore, OS needs to read the physical addresses directly to read the page table entries. This is one of the places where the OS might need to read a physical location directly (discussed in our google classroom interaction).

The system call should print something of the following format:

```
Entry number: 0, Virtual address: 0x00000000, Physical address: 0xdee0000
Entry number: 1, Virtual address: 0x00001000, Physical address: 0xde20000
```

and so on.

Implement a user program (use filename as mypgtPrint.c) to invoke the newly added system call. After a successful implementation, typing mypgtPrint on xv6 shell should print the page table. How to add a user program was covered in assignment-2.

Perform the following experiments on this user program and document your observations along with reasoning in report.pdf.

1. Declare a large size global array (int arrGlobal[10000]) and check if the number of valid entries changes or not.
2. Declare a large size local array (int arrLocal[10000]) within the main function and check if the number of valid entries change or remain the same.
3. Repeat the execution of the user program and check if the number of entries remains the same or not. Also, check if the virtual and physical addresses change or not across multiple executions.

Part-2: Implement demand paging

We discussed demand paging in our lectures where pages are not allocated on process creation but based on demand. The base implementation of xv6 does not implement demand paging and our task in this assignment is to implement demand paging. We would implement a simpler version of demand paging where the read-only code associated with the process is mapped during the process creation, but the memory required for heap and globals is not assigned pages during process creation but allocated on demand. Also, our demand paging would be simpler to assume that sufficient memory is available and we do not need to replace/evict any page during the demand paging process.

Hint-1: modify exec() in exec.c to prevent allocating memory space for dynamic variables but allocate for the read-only code/data. How to do it? Read about elf file format. A short summary is given below.

ELF (Executable and Linkable Format) is used to represent executable and object files. The first few bytes contain an ELF header. The ELF header has a fixed *magic* number at a defined location and is checked by the loader to confirm that it is an ELF file. The ELF file also has a pointer to various *program headers (ph)* which store information about different program segments. ELF header also contains the information about the total number of program headers. Each program header represents a program segment (e.g., code, data, etc.). Each program header contains the following information:

- type: the type of the segment (Loadable segments are of our interest).
- offset: the file offset at which the program segment is in the file (or on the disk).
- filesz: the size of the program segment in the file.
- paddr: the physical address at which the segment should be loaded.
- vaddr: the virtual address at which the segment should be loaded.
- memsz: the size of the program segment in memory (includes read-only and dynamic data).

memsz must be greater than filesz else the ELF file is not valid. If memsz is not equal to filesz, the remaining bytes (memsz - filesz) are initialized to zero by the loader.

You can print the program headers for a given executable file using the following unix command:

```
objdump -p <executable file name>
```

On Linux, type man elf to get full details about the ELF format.

Hint-2: Inside the trap.c file, the current trap function exits with an error. Modify it to check if the trap corresponds to page fault and if yes, then implement a handler to implement demand paging. The handler should check for the faulting address and if it is a valid address in the virtual memory range of the process. If yes, assign a page and map it to the page directory/page table. If it is not a valid page, then generate errors as was happening earlier.
Do make sure that you zero out the physical page before assigning to the process.

Implement a user program (mydemandPage.c) to exercise this condition and see that it works. Your user program should use a large sized global array write/read from this array to check for functioning of demand paging. Also, invoke pgtPrint system call intermittently to check that the page table is expanding as per demand paging. A template for the user program is given below.

```
#include "types.h"
#include "stat.h"
#include "user.h"

#define N 300           //global array size - change to see effect. Try 3000, 5000, 10000
int glob[N];
int main(){
        glob[0]=2;      //initialize with any integer value
        printf (1, "global addr from user space: %x\n", glob);
        for (int i=1;i<N;i++){
                glob[i]=glob[i-1];
                if (i%1000 ==0)
                        pgtPrint();
        }
        printf (1, "Printing final page table:\n");
        pgtPrint();
        printf(1, "Value: %d\n", glob[N-1]);

        exit();
}
```

On executing this program, the following kind of messages should be displayed (N=3000 in this example):

```
global addr from user space: B00
page fault occurred, doing demand paging for address: 0x1000
  pgdir entry num:0, Pgt entry num: 0, Virtual addr: 0, Physical addr: dee2000
  pgdir entry num:0, Pgt entry num: 1, Virtual addr: 1000, Physical addr: dfbc000
  pgdir entry num:0, Pgt entry num: 5, Virtual addr: 5000, Physical addr: dedf000
page fault occurred, doing demand paging for address: 0x2000
  pgdir entry num:0, Pgt entry num: 0, Virtual addr: 0, Physical addr: dee2000
  pgdir entry num:0, Pgt entry num: 1, Virtual addr: 1000, Physical addr: dfbc000
  pgdir entry num:0, Pgt entry num: 2, Virtual addr: 2000, Physical addr: df76000
  pgdir entry num:0, Pgt entry num: 5, Virtual addr: 5000, Physical addr: dedf000
page fault occurred, doing demand paging for address: 0x3000
Printing final page table:
  pgdir entry num:0, Pgt entry num: 0, Virtual addr: 0, Physical addr: dee2000
  pgdir entry num:0, Pgt entry num: 1, Virtual addr: 1000, Physical addr: dfbc000
  pgdir entry num:0, Pgt entry num: 2, Virtual addr: 2000, Physical addr: df76000
  pgdir entry num:0, Pgt entry num: 3, Virtual addr: 3000, Physical addr: dfbf000
  pgdir entry num:0, Pgt entry num: 5, Virtual addr: 5000, Physical addr: dedf000
Value: 2
```

Submission Instructions

Submission is to be done at the appropriate link. Just bundle the modified files as per below instructions.

1. Go inside the xv6 directory
2. make clean
3. tar -zcvf xv6.tar.gz * --exclude .git
4. A small report (report.pdf) with max. 2 pages explaining your implementation in brief, observations for various experiments, and your learning from this assignment.
5. Create a zip file containing xv6.tar.gz and report.pdf and upload it. The zip file should follow the name: Assgn6-<RollNo>.zip

Grading Policy

1. Part-1 working: 30%
2. Part-2 working: 45%
3. Report: 25%

Note: We would run plagiarism checks on the submissions and copy cases would be appropriately dealt with. If needed, we might conduct a viva to confirm the same.