Operating Systems–2: CS3523
January 2023

Programming Assignment 5: Syscall Implementation

Submission Date: 7th April 2023 (Friday), **9:00 pm**

This assignment requires downloading the xv6 learning OS and implementing a system call in it. Instructions already shared for the same. The xv6 version uses Intel architecture.

Setup (if not done already): Please install the following libraries on your Ubuntu system:
sudo apt-get install qemu qemu-system g++-multilib git-all grub2 grub-pc-bin libsdl-console-dev

$ git clone https://github.com/mit-pdos/xv6-public xv6
$ cd xv6

Booting xv6:
$ make
$ make qemu-nox (or use make qemu for GUI)

It will open a shell in your terminal where you can run some basic commands just like a unix shell. Try echo or ls or cat commands and see that they work.
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/1eS5PwWmKk6o7mdnzGe7Uj5RrQhjcGaks/view?usp=share_link

Part-1: Understanding and tracing system call

Your first task is to modify the xv6 kernel to print out a line for each system call invocation. You should print the name of the current process, name of the system call, the system call number, and the return value; you don't need to print the system call arguments. **The format is process name: syscallName->number->return_value**

When you're done, you should see output like this when booting xv6:

...
init: write->16->1
init: fork->1->2
sh: exec->7->0
sh: open->15->3
sh: close->21- >0
$sh: write->16->1
 sh: write->16->1

That's init forking and exec-ing sh, sh making sure only two file descriptors are open, and sh writing the $ prompt.

Hint: modify the syscall() function in syscall.c.

Part-2: Implementing your own system call: date

Your second task is to add a new system call to xv6. The main point of the exercise is for you to see some of the different pieces of the system call machinery. Your new system call will get the current date/time and return it to the user program. You may want to use the helper function, cmostime() (defined in lapic.c), to read the real time clock. date.h contains the definition of the struct rtcdate struct, which you will provide as an argument to cmostime() as a pointer.

You should create a user-level program that calls your new date system call; here's some source you should put in mydate.c:

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

int main(int argc, char *argv[])
{
  struct rtcdate r;

  if (date(&r)) {
    printf(2, "date failed\n");
    exit();
  }

  // your code to print the time in any format you like...
  /* example output format should be like this:
          Year: 2016
           Month: 1 or January
           Date: 26
           Hour: 15
           Minute: 12
           Second: 11
  */
  exit();
}
```

In order to make your new date program available to run from the xv6 shell, add _mydate to the UPROGS definition in Makefile.

Your strategy for making a date system call should be to clone all of the pieces of code that are specific to some existing system call, for example the "uptime" system call. You should grep for uptime in all the source files, using grep -n uptime *.[chS].

When you're done, typing mydate to an xv6 shell prompt should print the current time/date.

Note: If you feel a need, you might AVOID such printing of system call and return value (as done in part-1) for ONE particular system call ONLY. It is upto you to figure out which one and why.

Part-3: Printing the page table entries

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 - how to achieve it?

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.

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.

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. Create a small report (report.pdf) of max. 2 pages explaining
    a. Your understanding of how system call works and what you learnt from this assignment. - 1 page

    b. Your observations along with reasoning for various experiments of different size arrays in part-3 system call.
5. Create a zip file containing xv6.tar.gz and report.pdf and upload it. The zip file should follow the name: Assgn5-<RollNo>.zip

Weightage/Marks Breakup

1. Part-1: 20%
2. Part-2: 25%
3. Part-3: 35%
4. Report: 20%

Required background concepts:

You can go through the following document to learn about concepts related to this assignment.
https://docs.google.com/document/d/1GuTgJbMAKQQwoKUoliVvL_eOCF29aFdmq6q5OoCws9o/edit?usp=share_link