Operating System
CS 370
Spring 2019

Project 3
Date System Call

Due Date: 11:59 pm,18th February 2019
Must be submitted before deadline

Total Points: 50

Teaching Assistant:
Office Hours:
Graduate Assistant: Pradip Maharjan
Office Hours: MoWe 12 PM - 2 PM
Teaching Assistant: Luis Maya-Aranda
Office Hours: MoWe: 1 PM - 4 PM
TuTh: 10 AM - 12 PM

Course Web Page: http://osserver.cs.unlv.edu/moodle/

Objective: Become familiar with the **xv6** Teaching Operating System, shell organization, and system calls

Introduction:

The **xv6** OS is an educational OS designed by MIT for use as a teaching tool to give students hands-on experience with topics such as shell organization, virtual memory, CPU scheduling, and file systems. Although **xv6** lacks the full functionality of a modern OS, it is based on Sixth Edition Unix (also called V6) and is very similar in design and structure.

Resources:

The following resources provide more in-depth information regarding **xv6**. They include the **xv6** reference book, source code (pdf), and a tutorial on running and debugging.

1. **xv6** Reference Book: https://pdos.csail.mit.edu/6.828/2016/xv6/book-rev9.pdf
2. **xv6** Source Code PDF: https://pdos.csail.mit.edu/6.828/2016/xv6/xv6-rev9.pdf
3. Running and Debugging Tutorial: http://zoo.cs.yale.edu/classes/cs422/2010/lec/l2-hw

Project:

- Implement basic **date** system call
    - See instructions

Submission

When complete, submit:

- Zip the entire **xv6** folder and submit (not qemu).
- A copy of the ***zipped xv6 folder*** via the class web page (assignment submission link) by 11:59 pm 18th February, 2019.

We will check for code similarities using MOSS. Please do not download code from the internet or distribute solutions anywhere including Github, Bitbucket and any other public code publishing platforms.

Submissions received after the due date/time will not be accepted.

Displaying Pretty Error Messages

Before creating the date system call, we will take a short detour into the trap function. The trap function is located in the file **trap.c**. You may recall from lecture, that the trap function is the main "dispatch" for services executed in the kernel. There are three reasons for entering the kernel (and therefore three reasons for entering the trap function): handling interrupts, exceptions, and system calls. An exception is created whenever an illegal instruction is executed on the CPU. Examples include: dividing by zero, segmentation fault, array out of bounds errors, floating point instruction errors, etc.

Your task is to modify the trap function in **trap.c** so that when an error occurs, the trap function will print a short description of the error. Currently, in **trap.c** when an error occurs, it is handled by the **default** case in the switch statement. There are two cases, one is for an exception generated in the kernel, and the other is for an exception generated in user code. In both cases, the variable **tf->trapno** holds the error code and it is printed using the function **cprintf**.

You can see all of the trap error numbers (**trapno**) in the file **traps.h**. There are comments in that file which tell you what error each number corresponds to. You only need to worry about printing the descriptions of trap numbers 0 to 19, inclusive.

Why cprintf?

You may already be familiar from Project 1 with the **printf** function which is used to print to the console. So, then why is the function **cprintf** used inside trap.c? It is because **printf** is a user program, not a system call! You cannot call a user program from inside the trap function! The function **cprintf** is defined in **console.c** and it is available for use in kernel mode. In short, you can use **cprintf** to help you debug your system calls. To make things slightly more tedious, **cprintf** takes slightly different arguments than **printf** since **cprintf** only prints to the console (and cannot be used to output to a file, such as in the case of **printf**). The function prototypes are:

```
void cprintf(char* fmt, …)
void printf(int fd, char* fmt, …)
```

The use of these functions should need no explanation.

Testing

You will probably have many chances to test the error messages, but just to be safe we can edit the sample_program we made in Project 1 to test that they work. First, replace the **exit** call in sample_program.c with "return 0". This should generate the error "General Protection Fault". See the output below.

```
$ sample_program
This is a sample xv6 user program.
pid 3 sample_program: trap 13 err 0 on cpu 0 eip 0x2 addr 0x3f31--kill proc
Trap Error: General Protection Fault
```

Similarly, you can modify sample_program to create an array of size 10 using malloc. Then attempt to access the 11th element (that does not exist) to generate a page fault error.

```
pid 4 sample_program: trap 14 err 6 on cpu 0 eip 0x50 addr 0xb000--kill proc
Trap Error: Page Fault
$
```

Date System Call

A system call is how a program requests service from an operating system's kernel. System calls may include hardware related services, creation and execution of new processes and process scheduling. It is also possible for a system call to be made only from user space processes.

In this assignment you will implement a simplified version of the UNIX date command. This command gives current system date.

For example, if we run "**date 0**", then the current system date is returned in 24 hour format in GMT time zone. If we run "**date 1**", then the current system date is returned in 12 hour format in GMT time zone.

Go through chapter 1 in xv6 reference book to have a basic understanding of system calls in xv6.

Setting up a System Call

A system call must be both available in the user space and kernel space. It must be available in user space so that it can be used other programs and it must be available in kernel space so has permission to access resources that prohibited from user mode.

User Program (date.c)

In date.c file, you will implement the date user program. To write this program, you need to get familiar to ***rtcdate*** structure in ***date.h*** file, and ***cmostime*** function in ***lapic.c*** file. To add date.c as a user program, add date to **makefile** like in previous assignment.

You cannot call cmostime function from a user program. So, you need to write a system call getDate() to access cmostime for you. You will call the getDate system call in your user program to *return the current date by reference.*

You are responsible to create a new file called **date.c**, and write a program that will:

- Accepts arguments from the command line interface. You are expected to do the necessary error checking with the corresponding error message. *See examples at the end of the handout.*
- Get the date by calling getDate system call
    - *int getDate(struct rtcdate *myDate)*
- Print the date or the error message.

getDate System call

Make changes in **user.h, usys.s, syscall.h, syscall.c** like in previous assignment. You need to write getDate system call in **sysproc.c**. When your system call is called from your user program, the system call number and arguments are setup for you by the operating system. If the system call number is valid, it will begin running your system call function in sysproc.c. The arguments you passed to the system call in your user program are stored in the stack for security. This is why all functions in sysproc.c take no arguments even when the system call takes arguments when it is called from your user program. You need to retrieve them in your system call in sysproc.c by using argptr, argint, etc. Please refer to sys_sleep in sysproc.c for a hint. But you need to use argptr in this system call. Refer chapter 3, page 44-45 in xv6

reference book.
To implement getDate:

- create a variable to put the argument
- use argptr to get address of myDate struct passed as 0th argument (*Hint: look for argptr in syscall.c*)
- if argptr fails to get the pointer, return -1
- else, use cmostime to fill the struct and return 0 (*Hint: look for cmostime in lapic.c*)

Test cases

Case 1

$date

```
$ date
The date is: 21:23:12 on 12/28/2018
```

Returns current date in GMT time zone in 24 hr format

Case 2

$date 0

```
$ date 0
The date is: 21:23:14 on 12/28/2018
```

Returns current date in GMT time zone in 24 hr format

Case 3

$date 1

```
$ date 1
The date is: 9:23:15 PM on 12/28/2018
```

Returns current date in GMT time zone in 12 hr format

Case 4

$date -t -8

```
$ date -t -8
The date is: 13:23:19 on 12/28/2018
```

Returns current date in GMT-8 time zone in 24 hr format

Case 5

$date -t +2

```
$ date -t +2
The date is: 23:23:24 on 12/28/2018
```

Returns current date in GMT+2 time zone in 24 hr format

Case 6

$date 0 -t -8

```
$ date 0 -t -8
The date is: 13:23:31 on 12/28/2018
```

Returns current date in GMT-8 time zone in 24 hr format

Case 7

```
$date 1 -t +5
$ date 1 -t +5
The date is: 2:23:36 AM on 12/28/2018
```

Returns current date in GMT+5 time zone in 12 hr format

Case 8

```
$date 1 -t -8
$ date 1 -t -8
The date is: 1:23:43 PM on 12/28/2018
```

Returns current date in GMT-8 time zone in 12 hr format

Case 9

```
$date 1 -t +12
$ date 1 -t +12
The date is: 6:25:44 AM on 2/9/2019
$ date 1
The date is: 6:25:49 PM on 2/8/2019
```

Shows the behavior when the time zone offset shifts into the next day.

Notes:

- Don't forget to handle the edge cases. There is the possibility of the hours going negative when subtracting the time zone in which you will need to change the day (and similarly when the hours becomes greater than 24, see Case 9). What if the day is 1? What if the month is 1? What if the day is the last in the month and needs to increase? What if it is 12/31/18 and the time zone offset increases the hours to 30?
- Incorrect parameters should throw error and exit from the program
- Acceptable time zones are from GMT-12 to GMT+14 (whole numbers only)
- The following are some examples on error checking.

```
$ date -t -t -t -t
Usage: date [Optional: 12/24hr flag] [Optional: -t TIMEZONE]
1 for 12 hour, 0 for 24 hour
Use -t TIMEZONE for a timezone. Default is GMT
$ date -h
Error: Flag bit unrecognized
$ date 001256
Error: Flag bit unrecognized
$ date 0 -t 123
Error: incorrect timezone format
Example: -8 or +2
$ date 0 -t 16
Error: incorrect timezone format
Example: -8 or +2
$ date 1 -t -15
Error, timezone does not exist.
$ date 0 -t +123
Error, timezone does not exist.
$
```

Note: Please download latest test.sh file. You can test your program using script.c from 1st Assignment. E.g. "script test.sh"