CSC 4730 Fall 2023 Project 2

A RISC-V xv6 project to add a system call.

Objectives

The educational purpose for this project is to understand the flow of control that occurs when a user program makes a system call. You will accomplish this by first reading and understanding the source code of xv6 then you will demonstrate your mastery by adding a new custom system call to the operating system.

Deadlines

You and your assigned partner have 7 days to complete this assignment plus one grace day. If you have not turned anything in by 11:59 PM on the eighth day, you will win a grade of zero. Therefore, to receive partial credit, hand in something before the project expires!

Overview

Let's start by recalling the lessons of "Limited Direct Execution".

- We want user programs to run as fast as possible. We accomplish this by allowing them to run directly on the CPU.
- When the user needs to make a system call, there is a trap that causes control to pass from user space into kernel space. The kernel performs whatever service is requested and returns some value back to user space.

Pointers to where to study xv6 source code

- Start with `trampoline.S`. This file is riscv assembly language. You don't need to understand the assembly language although those of you who has taken CSC3510 can probably decode the instructions. Instead, see what you can get out of the comments.
- Next up is `trap.c` to look at `usertrap()`. Notice it is used and can handle traps caused by several kinds of events. Follow the code until it gets to `syscall()`.
- Next up is `syscall.c`. Review this entire file. Notice the `externs`. Notice the funny syntax defining `syscalls`. Look this up or ask me about it.

Figure out where things must be added

- You must add a user level program to xv6 that will test your new call. What files must change?

- You must be able to call your new system call from user level. What files must change?
- You must implement your system call inside the kernel. What files need to change?

Implementation

After you have made a plan and studied xv6, execute your implementation.

The system call to be added

Your system call will inspect the process table, counting how many of its slots are both used and not ZOMBIEs. The value found is to be returned.

Sample output

```
hart 2 starting
hart 1 starting
init: starting sh
$ count
The number of live processes: 3
$ sh
$ count
The number of live processes: 4
$ sh
$ count
The number of live processes: 5
$ $ $ count
The number of live processes: 3
$
```

`count` is the name of the user level program you must write.

This is your first user level program:

- Includes must be `kernel/types.h` and `user.h`.
- The user level program is trivial.

The line containing three prompt characters is where ^D was enter causing a shell to exit.

What to Hand In

Execute a `make clean`. Then zip the entire xv6 folder. Test the zip file to ensure you have done this correctly. Hand in the zip file.

Partners

- You must use only the partner I assign you.
- Only 1 person should hand in code. The code should clearly state who the partners are in a new text file `partner.txt` in the main directory.
- The non-code-submitting partner must submit a text file "partner.txt" that states who the partners are.
- Failure to list partners correctly as described above removes 5 points from your grade.

Software Kills

You goofed when adding a system call to the Linux kernel that runs a new pace maker. One time out of 100 million calls to your system call causes the kernel to crash. Patients die. No one knows why. Somewhere in the back of your mind you suspect it might be your system call. It haunts you for the rest of your life.

Grading

The class will receive a rubric when grading is complete that describes errors and penalties.

Both partners get the same grade without exception.