HoH: COL331/COL633 Labs

What's the best way for learning OS? Create one!

Jan 1, 2017

Shell

Threads

Concurrency

UserProgram

VirtualMem

Shell Again

Also available in pdf, slides. beamer. latex.

Introduction

Introduction

Hello! I'm your lab-instructor for this course.
In this series, you will join forces with me, and together, we will build a *kernel from the scratch*. We both will be working on this kernel. I'll do some coding in a branch, and ask you to implement some functionality. You can get my code by merging the branch with yours, and implement the functionality I asked. Once you implement it and commit the changes in your repository, I'll again work on the kernel on some other branch..

Status so far - our kernel boots into C code

So far, I have managed to write: See osdev barebones

1. x86/boot.S : containing seven lines of 32-bit x86 assembly instructions to:
   - set the stack pointer,

     ```
     movl  $tmpstack_bottom, %esp
     ```

   - clear flags,

Setup

So here's what you should do:

Tools

Please ensure you have latest version of:

- qemu (package: qemu qemu-system)
- g++ (package: g++-multilib >=4.7)
- git (package: git-all)
- grub2 (package: grub2 grub-pc-bin)
- boost library (package: libboost-all-dev)
- xorriso (to create iso image. Otherwise you'll get a warning that)
- coreutils(for makefile)

In debian/ubuntu, do:

```
bash$ sudo apt-get install qemu qemu-system g++-multilib
```

Clone the repository

Since we both will work on this kernel, we need to have a version

Shell

MMIO

MergeRequest

I've added few more code in origin/vgatext branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/vgatext
```

Aim

In this part, we'll program a memory mapped device while enhancing our kernel by adding the functionality to display "Hello, world!".

Information

In VGA text mode, 16 bit (2 bytes) of information is stored for each screen character and is stored in row-major order. First byte(MSB) is the ASCII code of the screen character and the next byte(LSB) encodes background(4 bit: msb) and foreground color(4 bit: lsb). Color: 0x0 corresponds to black pallete, 0x7 corresponds to white pallete, 0x1 corresponds to blue pallete.

PMIO

MergeRequest

Now it's my turn. I've added few more code in origin/serial branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/serial
```

Aim

In this part, we'll program an I/O mapped device while enhancing our kernel by adding debugging routines which will print debug messages to serial port.

Information

Serial port aka pc16550d uart(universal asynchronous receiver transmitter). In pc16550d uart,
Registers:

- the "transmitter holding" register of size 8 bits(1 byte) is I/O mapped at zeroth offset, and

Abstract mmio/pmio

MergeRequest

I've added few more code in origin/keyboard branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/keyboard
```

Aim

In this part, we'll look at one way of abstracting out details of mmio::read8 vs io::read8 while enhance our kernel by adding a simple keyboard driver.

Information

In Keyboard(8042, name=lpc_kbd), there are two main registers

- status register: size="8 bits" The status register has several fields

```
name="perr",      size="1 bit", description="Parity
name="timeout",   size="1 bit", description="General
```

kShell

MergeRequest

I've added few more code in origin/shell branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/shell
```

Aim

In this part, we'll look at one design approach while implementing a toy shell supporting builtin functions only.

- You need to implement the shell by implementing the given interfaces(in labs/shell.h and labs/shell.cc).
- You are *not* allowed to modify the interface and it's usage in x86/main.cc.
- You are *not* allowed to use any global variables or static variables in your functions.
- To make sure we have a personalized UI for each student, exact user interface is open - So be creative!

Threads

Stackless Coroutine

MergeRequest

I've added few more code in origin/coroutine branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/coroutine
```

Aim

In this part, we'll learn about "asymmetric-stackless coroutines" while enhancing our kernel to make it responsive to key presses while long computation task is running.

- You shall implement the long computation task as a stackless coroutine using the given APIs and add a new menu item/builtin command for the same.
- On key press, the status bar shall be updated with 'the number of keys pressed so far' while this long computation task is running(not after it finishes).
- If we select older menu item, shell still take seconds to respond

Fiber

MergeRequest

I've added few more code in origin/fiber branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/fiber
```

Aim

In this part, we'll learn about "fibers" while enhancing our kernel to make it responsive to key presses while long computation task is running.

- You shall implement the long computation task as a fiber using the given APIs and add a new menu item/builtin command for the same.
- On key press, the status bar shall be updated with 'the number of keys pressed so far' while this long computation task is running(not after it finishes).
- Result of all three menu items should be same.

Non-preemptive scheduling

MergeRequest

I've added few more code in origin/fiber_scheduler branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/fiber_schedul
```

Aim

In this part, we'll learn about non-preemptive sheduling while enhancing our shell to support multiple pending long computation task.

- You will need to support at least two additional long computation tasks. Now that you have implemented fibers, your computational tasks could involve the use of stack. For example, you can implement a recursive implementation of the fibonacci series computation, which has exponential complexity.
- For these additional long computation tasks:
  - You shall support multiple pending long computation tasks

Preemption (threads)

MergeRequest

I've added few more code in origin/preemption branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/preemption
```

Aim

In this part, we'll learn about "preemption" while enhancing our kernel to make it responsive to key presses while long computation task is running.

- You shall enhance the fiber implementation by adding preemption.
- You need to write a part of trap handler - ring0_preempt - which should switch stack to 'main_stack'
  - We would like to reuse shell_step_fiber_scheduler to do the scheduling.

Concurrency

SPSC Queue: Execute task on remote core

MergeRequest

I've added few more code in origin/multicore branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/multicore
```

Aim

In this part, we'll learn about multicore programming by implementing a SPSC queue and use it to send messages between two cores.

- I've modified the apps/labs.cc to execute the render() function in another core. The output of shell_render() - renderstate_t object - will be send to core #1 using the SPSQ queue. And core #1, will call the render() when it receives the renderstate_t object. Note: this means you won't see shell untill you implement SPSC queue correctly.
- You'll have to implement Leslie Lamport's portable lock-free

UserProgram

Ring3

MergeRequest

I've added few more code in origin/ring3 branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3
```

Aim

In this part, we'll learn about ELF headers, page table handling and user mode switching while enhancing our kernel to load arbitary user program and execute it.

- You need to implement elf loader (elf_load_function):
  - You shall only support loading of Position Independent Executable.
  - The entire program memory code address space shall be read only. You can safely ignore the flags in ELF and override with 'WRITE' = 0 flags in page table for code segments.
  - The entire program memory address space shall fit into a single

Ring3 Preemption

MergeRequest

I've added few more code in origin/ring3 branch. Please merge it with your master branch

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3
```

Aim

In this part, we'll learn about preempting user program while enhancing our kernel to make it responsive to key presses while long computation task is running in ring3/user mode.

- We'll have single kernel stack for the all user processes.
- Note: On timer interrupt, hardware will automatically switch to main_stack. and ring3_preempt macro will eventually be called.
- You need to write a part of trap handler - ring3_preempt - which should:
  - save all register state to current running process's state

Upcall/Signals

MergeRequest

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3
```

Aim

In this part, we'll learn about upcalls (passing information to the exception handler running in user mode) by letting the user process manage the exceptions(like INT3, page faults etc).

- Whenever exception occur, You need to:
  - we had already allocated emergency stack at the end of the page shared between kernel and user in 4.1
  - setup the emergency stack layout correctly (as explained below) at the end of this page
  - Set the esp to this allocated emergency stack
  - Set the eip to proc.startip+4.
  - all other register values including eflags shall remain unchanged
- user's exception handler is located at _start+4. ie.

Downcall/System call

MergeRequest

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3
```

Aim

In this part, we'll learn about downcalls/system calls by implementing following system calls: - You need to define the following function: system_call();

0. *nop*: no-operation/do-nothing
   - do-nothing
   - System call should *NOT* modify/write to the system call memory. (See Tip)

     `nop()`
1. *done*: done/exit.
   - mark the process as done(proc->state=PROC_DONE). process shouldn't be scheduled after this. So make sure, in your

VirtualMem

App: Virtual Memory

MergeRequest

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3
```

Aim

In this part, we'll learn about virtual memory by emulating an array of N_v virtual pages using N_p physical pages.

- **Note: There is a change : $N_v = 16$ and $N_p = 4$ instead of $N_v = 16$ and $N_p = 8$. Ie. you need to emulate 16 page array using 4 physical pages. not 8**
- Please read lecture videos on demand paging and page replacement policy.
- You need to emulate an array of size N_v pages, say varray. Starting address of varray shall be 2GB ie. 2u<<30.
- $N_v = 16$ and $N_p = 4$

Shell Again

App: Shell in user mode

Congrats on making it so far! It's been a pleasure working with you. Hope you enjoyed it as well!
Let's try to summarize the plot:

- You wrote a shell in kernel mode.
- Then I split the shell into two and rendered the UI on another core.
- Now I've moved your shell into user mode.
- And then I'll split the user shell into two and render the UI on another process.

In short: - You've implemented kernel coroutines, kernel threads and a kernel thread scheduler. And implmented a kernel application - shell. - You also have implemented user coroutines, user threads and a user thread scheduler. And implemented a user application - shell.

Shell is already done for you! Your shell which you implemented in part 1.4 is already moved to user mode as an application. So the role has been revereed - whatever you've done till parts 1.9 are now in user mode. And parts 1.10 - parts 1.13 are in kernel.

MergeRequest

```
user@host:~/hohlabs$ git pull
user@host:~/hohlabs$ git merge origin/ring3_shell
```

Aim

Get User shell working

Please don't make the source code public even after you finish this course - The code you been working on is part of Deepak Ravi's PhD. We hope to release the code under AGPL3 licence (Current LICENSE doesn't allow the code). Till then please don't publish.

End of lab

```
Please make sure you submit the feedback form
--
Regards,
DR H
Hoh labs
```