Project 0: Getting Real

Preliminaries

> Fill in your name and email address.

Hao Wang tony.wanghao@stu.pku.edu.cn

> If you have any preliminary comments on your submission, notes for the TAs, please give them here.

I tested my console on a macos laptop and it seems the key bindings are a bit different for mac (the physical backspace actually corresponds to `0x7f` while <C-h> corresponds to `0x08`, the actual <BS>). In my submission I used `\b` as the <BS>.

> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.

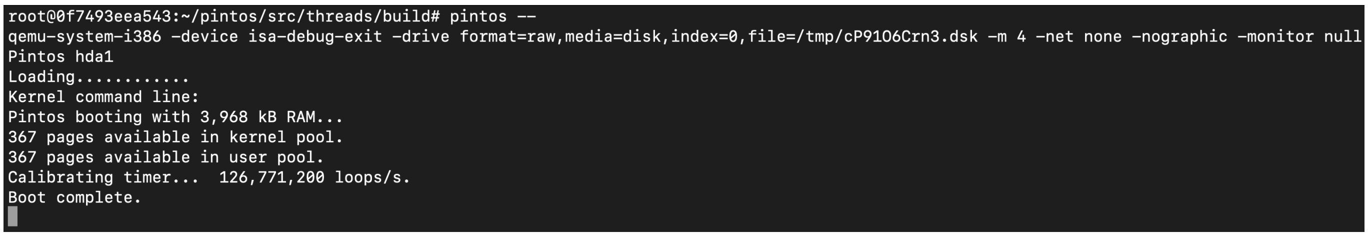
BIOS_interrupt_call wiki

ASCII wiki

Booting Pintos

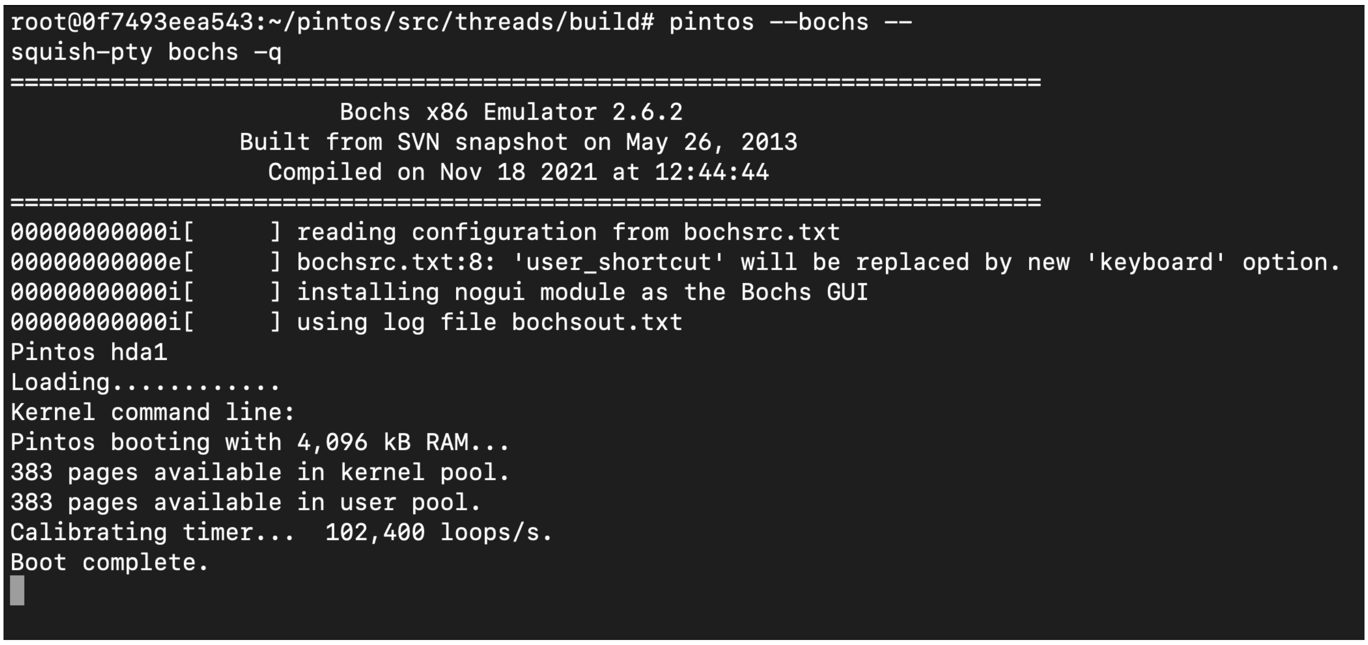
> A1: Put the screenshot of Pintos running example here.

QEMU:



```
root@0f7493eea543:~/pintos/src/threads/build# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/cP91O6Crn3.dsk -m 4 -net none -nographic -monitor null
Pintos hda1
Loading...........
Kernel command line:
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  126,771,200 loops/s.
Boot complete.
```

Bochs:



```
root@0f7493eea543:~/pintos/src/threads/build# pintos --bochs --
squish-pty bochs -q
========================================================================
                       Bochs x86 Emulator 2.6.2
                Built from SVN snapshot on May 26, 2013
                  Compiled on Nov 18 2021 at 12:44:44
========================================================================
00000000000i[     ] reading configuration from bochsrc.txt
00000000000e[     ] bochsrc.txt:8: 'user_shortcut' will be replaced by new 'keyboard' option.
00000000000i[     ] installing nogui module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
Pintos hda1
Loading...........
Kernel command line:
Pintos booting with 4,096 kB RAM...
383 pages available in kernel pool.
383 pages available in user pool.
Calibrating timer...  102,400 loops/s.
Boot complete.
```

Debugging

QUESTIONS: BIOS

> B1: What is the first instruction that gets executed?

ljmp $0xf000,$0xe05b

> B2: At which physical address is this instruction located?

0xffff0

QUESTIONS: BOOTLOADER

> B3: How does the bootloader read disk sectors? In particular, what BIOS interrupt is used?

It first set up the serial port at the very beginning using `int $0x14`. During the kernel-finding process, it calls `read_sector`, which then uses BIOS interrupt `int $0x13` coupled with `0x42` in `%ah` to perform an extended read.

> B4: How does the bootloader decides whether it successfully finds the Pintos kernel?

The bootloader first iterates through every sector, every drive and every partition. It first verifies the presence of an MBR signature (`cmpw $0xaa55, %es:510`) to check if the drive has been indeed partitioned. Then, after finding an used (`cmpl $0, %es:(%si)`), Pintos kernel (`cmpb $0x20, %es:4(%si)`), and bootable (`cmpb $0x80, %es:(%si)`) partition, it decides that it has successfully found the Pintos kernel.

> B5: What happens when the bootloader could not find the Pintos kernel?

It outputs "Not Found" and notify BIOS that boot failed by triggering an BIOS interrupt (`$0x18`).

> B6: At what point and how exactly does the bootloader transfer control to the Pintos kernel?

At address `7cd3`, after successfully finding and booting the Pintos kernel, the bootloader stores the start address of the Pintos kernel into a proxy (`%dx` or `start`), which is then used in an indirect long jump (`ljmp *(%esi)` or `ljmp *start`) to the `pintos_init()` function in Pintos kernel.

QUESTIONS: KERNEL

> B7: At the entry of pintos_init(), what is the value of expression `init_page_dir[pd_no(ptov(0))]` in hexadecimal format?

0x0

> B8: When `palloc_get_page()` is called for the first time,

> > B8.1 what does the call stack look like?
> >
> > #0 palloc_get_page (flags=(PAL_ASSERT | PAL_ZERO)) at ../../threads/palloc.c:112
> >
> > #1 0xc00203aa in paging_init () at ../../threads/init.c:168

> > #2 0xc002031b in pintos_init () at ../../threads/init.c:100
> >
> > #3 0xc002013d in start () at ../../threads/start.S:180

> > B8.2 what is the return value in hexadecimal format?
> >
> > 0xc0101000

> > B8.3 what is the value of expression `init_page_dir[pd_no(ptov(0))]` in hexadecimal format?
> >
> > 0x0

> B9: When palloc_get_page() is called for the third time,

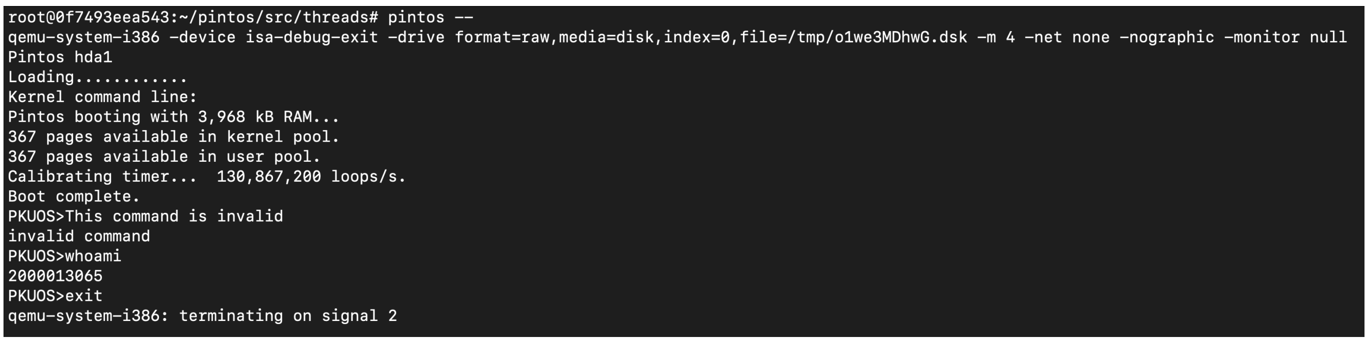
> > B9.1 what does the call stack look like?
> >
> > #0 palloc_get_page (flags=PAL_ZERO) at ../../threads/palloc.c:112
> >
> > #1 0xc0020a81 in thread_create (name=0xc002e895 "idle", priority=0, function=0xc0020eb0 , aux=0xc000efbc) at ../../threads/thread.c:178
> >
> > #2 0xc0020976 in thread_start () at ../../threads/thread.c:111
> >
> > #3 0xc0020334 in pintos_init () at ../../threads/init.c:119
> >
> > #4 0xc002013d in start () at ../../threads/start.S:180

> > B9.2 what is the return value in hexadecimal format?
> >
> > 0xc0103000

> > B9.3 what is the value of expression `init_page_dir[pd_no(ptov(0))]` in hexadecimal format?
> >
> > 0x102027

Kernel Monitor

> C1: Put the screenshot of your kernel monitor running example here. (It should show how your kernel shell respond to `whoami`, `exit`, and `other input`.)



```
root@0f7493eea543:~/pintos/src/threads# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/o1we3MDhwG.dsk -m 4 -net none -nographic -monitor null
Pintos hda1
Loading............
Kernel command line:
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer...  130,867,200 loops/s.
Boot complete.
PKUOS>This command is invalid
invalid command
PKUOS>whoami
2000013065
PKUOS>exit
qemu-system-i386: terminating on signal 2
```

> C2: Explain how you read and write to the console for the kernel monitor.

I use `input_getc` from `devices/input.c` for input. For output, I use both `printf` from `lib/stdio.c` and `putchar`, `puts` from `lib/kernel/console.c`.

I implemented backspace in my monitor. When it receives `\b` (`0x08`), it outputs consecutively `\b`, (space), `\b` to the monitor to cover up the previous character.