Project 0: Getting Real

Preliminaries

> Fill in your name and email address.

安昱达 2000017737@stu.pku.edu.cn

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

> 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.

R. Brown, Ralf Brown's Interrupt List, 2000.

Booting Pintos

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

QEMU running.

```
root@6f31139674ae:~# cd pintos/src/threads
root@6f31139674ae:~/pintos/src/threads# make
cd build && make all
make[1]: Entering directory '/home/PKUOS/pintos/src/threads/build'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/PKUOS/pintos/src/threads/build'
root@6f31139674ae:~/pintos/src/threads# cd build
root@6f31139674ae:~/pintos/src/threads/build# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/3N37TkbbWW.dsk -m 4 -net none -no
graphic -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...  104,755,200 loops/s.
Boot complete.
```

Bochs running.

```
root@6f31139674ae:~/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?

The 1st instruction being executed is "ljmp $0xf000,$0xe05b".

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

At 0xffff0.

QUESTIONS: BOOTLOADER

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

Firstly it sets up segement registers and stack, then initializes ports, using **BIOS interrupt 13 with AH=0x0 *[INITIALIZE PORT]***. Then, configuring disk number in **DL** and sector number in **EBX**, it uses **interrupt 14 with AH=0x42 *[EXTENDED READ]*** to read the sector.

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

Firstly, checking the return of INTR14, if BIOS failed to read disk then the finding failed, too.

Secondly, checking if there is a MBR signature in the sector, if not the disk is not bootable, going to the next disk.

Then, it starts to check partitions on the disk. Checking if a partition is unused, kernel partition and bootable partition. If one of these failed, bootloader will go to next partition, until it find a bootable kerbel partition. This process is in a finite loop.

If a partition is found, bootloader starts to load the kernel.

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

It will use INPR18, notifying BIOS a failed boot, and print an infomation.

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

After found a kernel, bootloader will use BIOS to load it to memory, then through reading the start address in ELF header, it calculates out a 16:16 address, saves it on memory, and indirectly jump to it (since 8086 has no instruction for directly jumping by 16:16 address).

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?

0xc0023114 in palloc_get_page(flags=(PAL_ASSERT | PAL_ZERO))

0xc00203aa in paging_init()

0xc002031b in pintos_init()

0xc002013d in start()

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

(void *) 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?

0xc0023114 in palloc_get_page(flags=PAL_ZERO)

0xc0020a81 in thread_create(name=0xc002e895 "idle", priority=0, function=0xc0020eb0 , aux=0xc000efbc)

0xc0020976 in thread_start()

0xc0020334 in pintos_start()

0xc002013d in start()

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

(void *) 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`.)

```
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...  104,755,200 loops/s.
Boot complete.
No command line passed to kernel.
Start kernel monitor.
PKUOS> whoami
stu ID : 2000017737
PKUOS>
PKUOS> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Error : Out of buffer(63).
PKUOS> exit
Exit kernel monitor.
Kernel will shutdown.

root@89b3f25a24a3:~/pintos/src/threads#
```

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

To write a shell, I need use functions for IO. So I read code in `init.c`, `input.c` and `console.c`. I noticed `console.c` provides `puts()`, `putbuf()`, `putchar()` and `input.c` provides `input_getc()` which receive keyboard input. I used them. I also find `strcmp()` in `lib/string.h` so I used it for parsing cmdline.

I used infinite loop for running shell. I set up a buffer for cmdline. I read from keyboard char by char. I implemented backspace by using buffer and "\b \b".