

# Project 0: Getting Real

## Preliminaries

Fill in your name and email address.

Jiaqi Si [sigongzi@stu.pku.edu.cn](mailto:sigongzi@stu.pku.edu.cn)

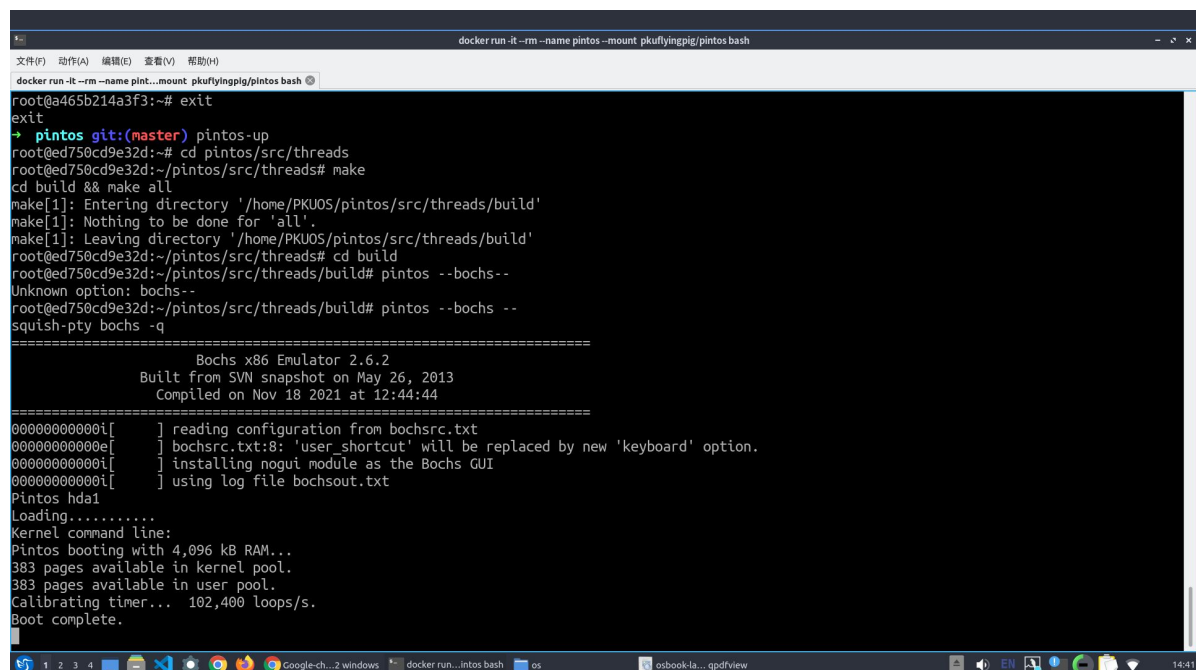
If you have any preliminary comments on your submission, notes for the TAs, please give them here.

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.

## Booting Pintos

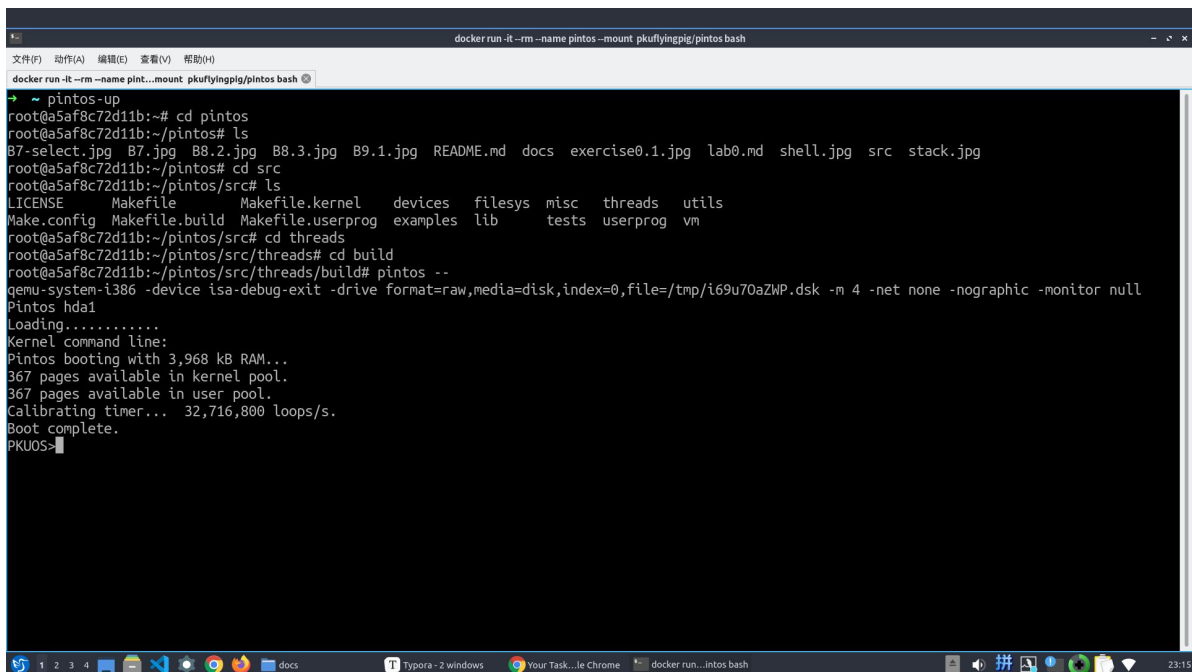
A1: Put the screenshot of Pintos running example here.

running by bochs



```
docker run -it --name pintos --mount pkuylngp/pintos bash
root@a465b214a3f3:~# exit
exit
→ pintos git:(master) pintos-up
root@ed750cd9e32d:~# cd pintos/src/threads
root@ed750cd9e32d:~/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@ed750cd9e32d:~/pintos/src/threads# cd build
root@ed750cd9e32d:~/pintos/src/threads/build# pintos --bochs-
Unknown option: bochs-
root@ed750cd9e32d:~/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.
```

running by qemu



```
docker run -it --rm --name pintos --mount pkuflyingpig/pintos bash
~ pintos-up
root@a5af8c72d11b:~# cd pintos
root@a5af8c72d11b:~/pintos# ls
B7-select.jpg B7.jpg B8.2.jpg B8.3.jpg B9.1.jpg README.md docs exercise0.1.jpg lab0.md shell.jpg src stack.jpg
root@a5af8c72d11b:~/pintos# cd src
root@a5af8c72d11b:~/pintos/src# ls
LICENSE Makefile Makefile.kernel devices filesys misc threads utils
Make.config Makefile.build Makefile.userprog examples lib tests userprog vm
root@a5af8c72d11b:~/pintos/src# cd threads
root@a5af8c72d11b:~/pintos/src/threads# cd build
root@a5af8c72d11b:~/pintos/src/threads/build# pintos --
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/i69u70aZWP.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... 32,716,800 loops/s.
Boot complete.
PKUOS>
```

# Debugging

## QUESTIONS: BIOS

B1: What is the first instruction that gets executed?

ljmp \$0xf000, \$0xe05b

B2: At which physical address is this instruction located?

physical address: 0xfffff0

## QUESTIONS: BOOTLOADER

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

use BIOS interrupt call (under the real mode, by instruction **int**)

int **0x14** ah 0x00 INITIALIZE PORT

al = 0xe3 = 0b11100011

dx=0x0 (for COM1 and 0x1 for COM2... and so on)

this table is copy from [INT0x14](#)

Bits	Function
5..7	Select baud rate
	000- 110 baud
	001- 150
	010- 300
	011- 600
	100- 1200
	101- 2400
	110- 4800
	111- 9600
3..4	Select parity
	00- No parity
	01- Odd parity
	10- No parity
	11- Even parity
2	Stop bits
	0- One stop bit
	1- Two stop bits
0..1	Character Size
	10- 7bits
	11- 8bits

Serial Port Initialization (from [BIOS wiki](#) )

5..7 111- Select 9600 baud rate

3..4 00- Select No parity (parity bit for check)

2 0 One stop bit

0..1 11- Select Character size (8 bits)

call **read\_sector** which sends **0x13** interrupt to BIOS

- int 0x13 ah 0x42 EXTENDED READ
- dl 0x80 hard disk 1 (we may add this number to check next disk if this disk doesn't have partition)
- DS:SI SI select data segment and **si** is the top of the stack(mov %sp, %si)
- the specific information about Disk Address Packet [INT 0x13 wiki](#)

offset	size	content	description
00h	1 Byte	0x10	size of DAP (always set to 0x10)
01h	1 Byte	0	unused, should be zero
02h..03h	2 Bytes	0x1	Number of sectors to read
04h..05h	2 Bytes	0x0 0x0	Buffer offset
06h..07h	2 Bytes	%es = 0x20000	Buffer segment
08h..0Fh	4 Bytes	0x00 0x0 0x0	LBA sector number (we read sector 0)

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

if **read\_sector** fails, the **CF** flag set on error

When the disk read fails, jc condition (CF = 1) is met, so the instruction change pc to **no\_such\_drive**

If we read a disk to 0x20000 successfully, we check the **MBR signature** of the hard disk (0xaa55 for the two bytes in the first sector)

If the signature check fails, we try to find the **next\_drive**

If it is the partitioned hard disk we need, we check whether the partition is a Pintos kernel (0x20 in 0x201c2) and a bootable partition (0x80 in 0x201be)

If the check fails, we will try **next\_partition**

Or we find the next hard disk

After above checking, we jump to **load\_kernel**

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

The execute stream comes to the assembly code with label **no\_such\_drive** or **no\_boot\_partition**

we print error information **Not found**

and send **0x18** interrupt call to BIOS (Called when there is no bootable disk available to the system [INT0x18](#))

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

(The space is so scarce that the author had to use the code text segment. I was shocked.)

we reading the whole kernel (must less than 512KiB) from 0x20000 to 0x9ffff

in 0x7cd3 physical address a long jump to the \*start in kernel

## QUESTIONS: KERNEL

B7: At the entry of pintos\_init(), what is the value of expression

`init_page_dir[ptov(pd_no(ptov(0)))]` in hexadecimal format?

```
(gdb) p init_page_dir[pd_no(ptov(0))]
=> 0xc000efef: int3
=> 0xc000efef: int3
$1 = 0
```

hexadecimal format: 0x0

pdov: physical address to virtual address (after pdov) -> (void \* 0xc0000000 + 0)

pd\_no: get page directory index (after pd\_no) -> (uint) (0b1100000000 -> 0x300)

init\_page\_dir: we have not initialized it already...

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

B8.1 what does the call stack look like?

```

104 address.
105 If PAL_USER is set, the page is obtained from the user pool,
106 otherwise from the kernel pool. If PAL_ZERO is set in FLAGS,
107 then the page is filled with zeros. If no pages are
108 available, returns a null pointer, unless PAL_ASSERT is set in
109 FLAGS, in which case the kernel panics. */
110 void *
111 pallocc_get_page (enum pallocc_flags flags)
112 {
113     return pallocc_get_multiple (flags, 1);
114 }
115
116 /** Frees the PAGE_CNT pages starting at PAGES. */
117 void
118 pallocc_free_multiple (void *pages, size_t page_cnt)
119 {
120 }
/home/PKUOS/pintos/src/threads/pallocc.c
0x00000000 in ?? ()
(gdb) b pallocc_get_page
Breakpoint 1 at 0xc0023290: file ../../threads/pallocc.c, line 112.
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0xc0023290 <pallocc_get_page>: push %ebp

Breakpoint 1, pallocc_get_page (flags=(PAL_ASSERT | PAL_ZERO)) at ../../threads/pallocc.c:112
(gdb) bt
#0 pallocc_get_page (flags=(PAL_ASSERT | PAL_ZERO)) at ../../threads/pallocc.c:112
#1 0xc0020526 in paging_init () at ../../threads/init.c:214
#2 0xc002031b in pintos_init () at ../../threads/init.c:104
#3 0xc002013d in start () at ../../threads/start.S:180
(gdb)

```

start call `pintos_init`

`pintos_init` call `paging_init`

`paging_init` call `pallocc_get_page`

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

```

106 otherwise from the kernel pool. If PAL_ZERO is set in FLAGS,
107 then the page is filled with zeros. If no pages are
108 available, returns a null pointer, unless PAL_ASSERT is set in
109 FLAGS, in which case the kernel panics. */
110 void *
111 pallocc_get_page (enum pallocc_flags flags)
112 {
113     return pallocc_get_multiple (flags, 1);
114 }
115
116 /** Frees the PAGE_CNT pages starting at PAGES. */
117 void
118 pallocc_free_multiple (void *pages, size_t page_cnt)
119 {
120     struct pool *pool;
121     size_t page_idx;
122 }
/home/PKUOS/pintos/src/threads/pallocc.c
(gdb) ni
=> 0xc002311d <pallocc_get_page+9>: push $0x1
(gdb) ni
=> 0xc002311f <pallocc_get_page+11>: pushl 0x8(%ebp)
(gdb) ni
=> 0xc0023122 <pallocc_get_page+14>: call 0xc0023040 <pallocc_get_multiple>
(gdb) ni
=> 0xc0023127 <pallocc_get_page+19>: add $0x10,%esp
(gdb) p $eax
$3 = -1072689152
(gdb) ni
=> 0xc002312a <pallocc_get_page+22>: leave
(gdb) p/x $eax
$4 = 0xc002312d
(gdb)

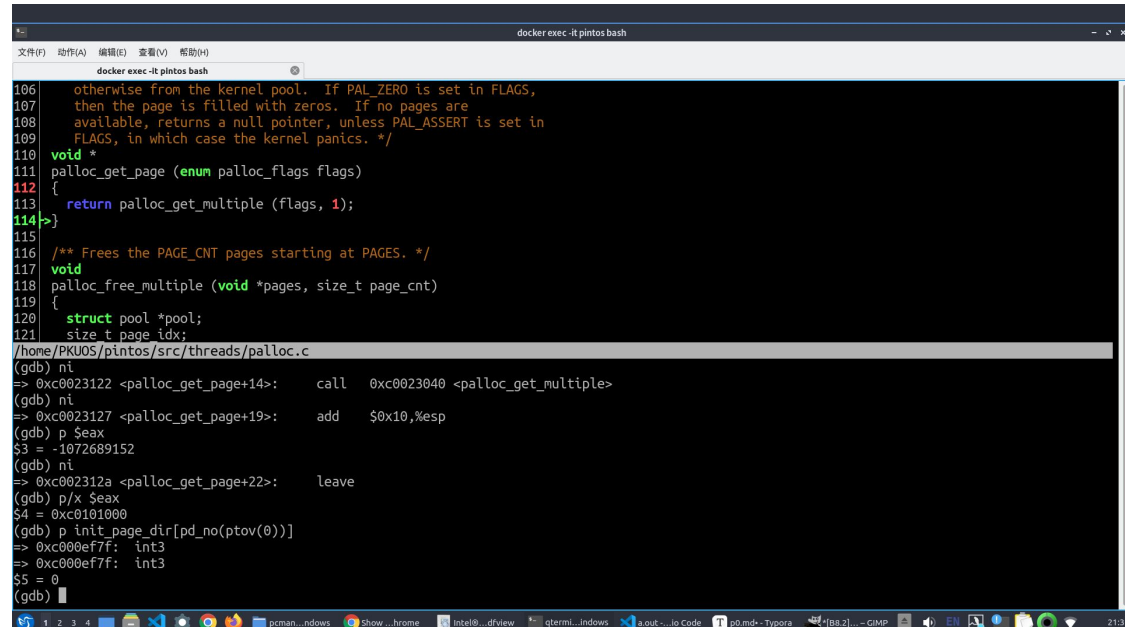
```

return **0xc0101000**

an virtual address direct mapping to physical address

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

you still have NOT initialized page table here yet!!!! why I need to take a screenshot again?

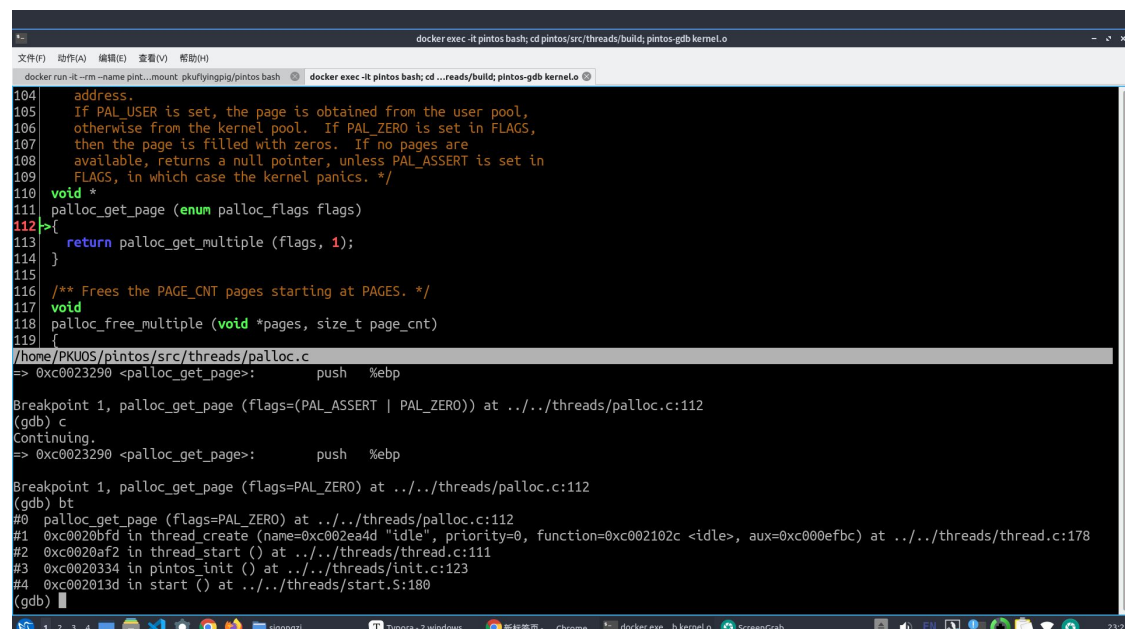


```
106 otherwise from the kernel pool. If PAL_ZERO is set in FLAGS,
107 then the page is filled with zeros. If no pages are
108 available, returns a null pointer, unless PAL_ASSERT is set in
109 FLAGS, in which case the kernel panics. */
110 void *
111 palloc_get_page (enum palloc_flags flags)
112 {
113     return palloc_get_multiple (flags, 1);
114 }
115
116 /** Frees the PAGE_CNT pages starting at PAGES. */
117 void
118 palloc_free_multiple (void *pages, size_t page_cnt)
119 {
120     struct pool *pool;
121     size_t page_idx;
122 }
/home/PKUOS/pintos/src/threads/palloc.c
(gdb) ni
=> 0xc0023122 <palloc_get_page+14>: call 0xc0023040 <palloc_get_multiple>
(gdb) ni
=> 0xc0023127 <palloc_get_page+19>: add $0x10,%esp
(gdb) p $eax
$3 = -1072689152
(gdb) ni
=> 0xc002312a <palloc_get_page+22>: leave
(gdb) p/x $eax
$4 = 0xc0101000
(gdb) p init_page_dir[pd_no(ptov(0))]
=> 0xc000ef7f: int3
=> 0xc000ef7f: int3
$5 = 0
(gdb) |
```

it is 0x0

B9: When palloc\_get\_page() is called for the third time,

B9.1 what does the call stack look like?



```
104 address.
105 If PAL_USER is set, the page is obtained from the user pool,
106 otherwise from the kernel pool. If PAL_ZERO is set in FLAGS,
107 then the page is filled with zeros. If no pages are
108 available, returns a null pointer, unless PAL_ASSERT is set in
109 FLAGS, in which case the kernel panics. */
110 void *
111 palloc_get_page (enum palloc_flags flags)
112 {
113     return palloc_get_multiple (flags, 1);
114 }
115
116 /** Frees the PAGE_CNT pages starting at PAGES. */
117 void
118 palloc_free_multiple (void *pages, size_t page_cnt)
119 {
120 }
/home/PKUOS/pintos/src/threads/palloc.c
=> 0xc0023290 <palloc_get_page>: push %ebp
Breakpoint 1, palloc_get_page (flags=(PAL_ASSERT | PAL_ZERO)) at ../../threads/palloc.c:112
(gdb) c
Continuing.
=> 0xc0023290 <palloc_get_page>: push %ebp
Breakpoint 1, palloc_get_page (flags=PAL_ZERO) at ../../threads/palloc.c:112
(gdb) bt
#0 palloc_get_page (flags=PAL_ZERO) at ../../threads/palloc.c:112
#1 0xc0020bdf in thread_create (name=0xc002ea4d "idle", priority=0, function=0xc002102c <idle>, aux=0xc000efbc) at ../../threads/thread.c:178
#2 0xc0020af2 in thread_start () at ../../threads/thread.c:111
#3 0xc0020334 in pintos_init () at ../../threads/init.c:123
#4 0xc002013d in start () at ../../threads/start.S:180
(gdb) |
```

start call pintos\_init

pintos\_init call thread\_start

thread\_start call thread\_create

thread\_create call palloc\_get\_page

it is called in function **thread\_create**

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

it returns 0xc0103000

which means we allocate the third page

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

value: 0x102027

a PDE entry

0x102 physical index of this page frame

0x027 = 0b 0000 0010 0111

ACCESS BIT(5)

USER BIT(2)

WRITABLE BIT(1)

PRESENT BIT(0)

# 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@e98226be74af:~/pintos/src/threads/build# pintos --  
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/ygtyjWDiSe.disk -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... 52,377,600 loops/s.  
Boot complete.  
PKUOS>whoami  
2000012957  
PKUOS>sijiaqi  
sijiaqi invalid  
PKUOS>tooooooooooooooooooooooooooooooooooooooooooooooooooooo  
COMMAND TOO LONG  
PKUOS>LLLLLLLLLLLLLLLLLLLLLLLLLLLLOOOO00000000000000000000000000ONG  
COMMAND TOO LONG  
PKUOS>exit  
  
Timer: 9568 ticks  
Thread: 9546 idle ticks, 22 kernel ticks, 0 user ticks  
Console: 521 characters output  
Keyboard: 0 keys pressed  
Powering off...  
root@e98226be74af:~/pintos/src/threads/build#
```

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

firstly, the command need to be cached, so it can not be infinite long.

Buffer size is 64, so the maximal command line is 63. (to left a position for end character '\0')

The size is defined as a marco definition and can be changed easily.

## handle input

The two special inputs we need to handle are that **backspace**(ascii 0x8) and **carriage return**(ascii 0xD)

### backspace

The VGA helps us do the cursor move. But we can not delete the character which has been already printed.

A tricky method is that using space(ascii 0x20) to cover it.

When we receive a backspace, we transmit backspace(0x8) + space(0x20) + backspace(0x8), 3 characters totally, to VGA

The user is to believe that we can really delete his input!

### carriage return

0xD is the end of a command line. We use this value to check that input for this time is end.

### printable character

The ascii code for printable character is between 0x20 and 0x7e.

We just use **putchar** to append it to STDOUT

### too long input

We record the length of input. If the length reaches MAXCOMMANDLINE - 1, we terminate the getting process.

Tell the naughty user the shell is unwilling to handle command too long and clear the input buffer by **input\_init**

## handle command line

### whoami

**print** my student id

### exit

call **shutdown\_power\_off** to exit qemu



## other input

we **print** the command we cached in input handling.

and **print** " invalid" after it.