

OS-CA1

Kasra Noorbakhsh - Kouros Sadjadi - Mehdi Jamalkhah

810100230 - 810100587 - 810100111

Contents

1	Introduction	3
1.1	XV6 Architecture(Q1)	3
1.2	Process Details(Q2)	3
1.3	Fork and Execute(Q4)	3
2	Compile	3
2.1	Make-file(Q8)	3
3	Boot	4
3.1	Boot file(Q11)	4
3.2	Objcopy(Q12)	8
3.3	X86 Registers(Q14)	8
3.4	Entry.s(Q18)	8
4	Xv6 kernel	9
4.1	Entry Address(Q19)	9
4.2	Segmentation(Q22)	9
5	User Programs	10
5.1	Proc(Q23)	10
5.2	System Prep(Q27)	10
6	Debugging	11
6.1	Breakpoint(Q1)	11
6.2	Breakpoint(Q2)	11
6.3	Bt command(Q3)	11
6.4	x and print(Q4)	11
6.5	Status(Q5)	11
6.6	Struct Input(Q6)	11
7	Assembly Debugging	11
7.1	Layout(Q7)	11
7.2	Transport(Q8)	15

8	Boot	15
9	Command Outputs	16
9.1	Commands	16
10	User Program	20

1 Introduction

1.1 XV6 Architecture(Q1)

Xv6 has a monolithic kernel where most of the operating system components are implemented as a large program called the kernel. The system calls are Unix based.

And in general is a simple OS for educational purposes.

1.2 Process Details(Q2)

An xv6 process consists of user-space memory (instructions, data, and stack) and per-process state private to the kernel.

Xv6 can time-share processes: it transparently switches the available CPU's among the set of processes waiting to execute.

1.3 Fork and Execute(Q4)

A process may create a new process using the fork system call. Fork creates a new process, called the child process, with exactly the same memory contents as the calling process, called the parent process. Fork returns in both the parent and the child. In the parent.

The exec system call replaces the calling process's memory with a new memory image loaded from a file stored in the file system. When exec succeeds, it does not return to the calling program; instead, the instructions loaded from the file start executing at the entry point declared in the ELF header.

Because if they are separate, the shell can fork a child, use open, close, dup in the child to change the standard input and output file descriptors, and then exec. No changes to the program being exec-ed (cat in our example) are required. If fork and exec were combined into a single system call, some other (probably more complex) scheme would be required for the shell to redirect standard input and output, or the program itself would have to understand how to redirect I/O.

2 Compile

2.1 Make-file(Q8)

UPROGS is a list of user programs in the XV6 operating system. Each entry in UPROGS represents a user program that is compiled and linked to create an executable file.

ULIB is a list of user library object files. These object files contain functions and routines commonly used by user programs.

3 Boot

3.1 Boot file(Q11)

This file has a direct binary machine language. the difference between this file and other files in xv6 is that it contains starting code that runs when machine starts. We used this boot file because it is simple and direct and it avoids complicated software processing.

bootblock: file format binary

Disassembly of section .data:

00000000 <.data>:

0:	fa	cli
1:	31 c0	xor %eax,%eax
3:	8e d8	mov %eax,%ds
5:	8e c0	mov %eax,%es
7:	8e d0	mov %eax,%ss
9:	e4 64	in \$0x64,%al
b:	a8 02	test \$0x2,%al
d:	75 fa	jne 0x9
f:	b0 d1	mov \$0xd1,%al
11:	e6 64	out %al,\$0x64
13:	e4 64	in \$0x64,%al
15:	a8 02	test \$0x2,%al
17:	75 fa	jne 0x13
19:	b0 df	mov \$0xdf,%al
1b:	e6 60	out %al,\$0x60
1d:	0f 01 16	lgdtl (%esi)
20:	78 7c	js 0x9e
22:	0f 20 c0	mov %cr0,%eax
25:	66 83 c8 01	or \$0x1,%ax
29:	0f 22 c0	mov %eax,%cr0
2c:	ea 31 7c 08 00 66 b8	ljmp \$0xb866,\$0x87c31
33:	10 00	adc %al,(%eax)
35:	8e d8	mov %eax,%ds
37:	8e c0	mov %eax,%es
39:	8e d0	mov %eax,%ss
3b:	66 b8 00 00	mov \$0x0,%ax
3f:	8e e0	mov %eax,%fs
41:	8e e8	mov %eax,%gs
43:	bc 00 7c 00 00	mov \$0x7c00,%esp
48:	e8 f0 00 00 00	call 0x13d
4d:	66 b8 00 8a	mov \$0x8a00,%ax
51:	66 89 c2	mov %ax,%dx

54:	66 ef	out	%ax, (%dx)
56:	66 b8 e0 8a	mov	\$0x8ae0, %ax
5a:	66 ef	out	%ax, (%dx)
5c:	eb fe	jmp	0x5c
5e:	66 90	xchg	%ax, %ax
...			
68:	ff	(bad)	
69:	ff 00	incl	(%eax)
6b:	00 00	add	%al, (%eax)
6d:	9a cf 00 ff ff 00 00	lcall	\$0x0, \$0xffff00cf
74:	00 92 cf 00 17 00	add	%dl, 0x1700cf(%edx)
7a:	60	pusha	
7b:	7c 00	jl	0x7d
7d:	00 ba f7 01 00 00	add	%bh, 0x1f7(%edx)
83:	ec	in	(%dx), %al
84:	83 e0 c0	and	\$0xffffffffc0, %eax
87:	3c 40	cmp	\$0x40, %al
89:	75 f8	jne	0x83
8b:	c3	ret	
8c:	55	push	%ebp
8d:	89 e5	mov	%esp, %ebp
8f:	57	push	%edi
90:	53	push	%ebx
91:	8b 5d 0c	mov	0xc(%ebp), %ebx
94:	e8 e5 ff ff ff	call	0x7e
99:	b8 01 00 00 00	mov	\$0x1, %eax
9e:	ba f2 01 00 00	mov	\$0x1f2, %edx
a3:	ee	out	%al, (%dx)
a4:	ba f3 01 00 00	mov	\$0x1f3, %edx
a9:	89 d8	mov	%ebx, %eax
ab:	ee	out	%al, (%dx)
ac:	89 d8	mov	%ebx, %eax
ae:	c1 e8 08	shr	\$0x8, %eax
b1:	ba f4 01 00 00	mov	\$0x1f4, %edx
b6:	ee	out	%al, (%dx)
b7:	89 d8	mov	%ebx, %eax
b9:	c1 e8 10	shr	\$0x10, %eax
bc:	ba f5 01 00 00	mov	\$0x1f5, %edx
c1:	ee	out	%al, (%dx)
c2:	89 d8	mov	%ebx, %eax
c4:	c1 e8 18	shr	\$0x18, %eax
c7:	83 c8 e0	or	\$0xffffffffe0, %eax
ca:	ba f6 01 00 00	mov	\$0x1f6, %edx
cf:	ee	out	%al, (%dx)
d0:	b8 20 00 00 00	mov	\$0x20, %eax
d5:	ba f7 01 00 00	mov	\$0x1f7, %edx

da:	ee		out	%al, (%dx)
db:	e8	9e ff ff ff	call	0x7e
e0:	8b	7d 08	mov	0x8(%ebp), %edi
e3:	b9	80 00 00 00	mov	\$0x80, %ecx
e8:	ba	f0 01 00 00	mov	\$0x1f0, %edx
ed:	fc		cld	
ee:	f3	6d	rep insl	(%dx), %es:(%edi)
f0:	5b		pop	%ebx
f1:	5f		pop	%edi
f2:	5d		pop	%ebp
f3:	c3		ret	
f4:	55		push	%ebp
f5:	89	e5	mov	%esp, %ebp
f7:	57		push	%edi
f8:	56		push	%esi
f9:	53		push	%ebx
fa:	83	ec 0c	sub	\$0xc, %esp
fd:	8b	5d 08	mov	0x8(%ebp), %ebx
100:	8b	75 10	mov	0x10(%ebp), %esi
103:	89	df	mov	%ebx, %edi
105:	03	7d 0c	add	0xc(%ebp), %edi
108:	89	f0	mov	%esi, %eax
10a:	25	ff 01 00 00	and	\$0x1ff, %eax
10f:	29	c3	sub	%eax, %ebx
111:	c1	ee 09	shr	\$0x9, %esi
114:	83	c6 01	add	\$0x1, %esi
117:	39	df	cmp	%ebx, %edi
119:	76	1a	jbe	0x135
11b:	83	ec 08	sub	\$0x8, %esp
11e:	56		push	%esi
11f:	53		push	%ebx
120:	e8	67 ff ff ff	call	0x8c
125:	81	c3 00 02 00 00	add	\$0x200, %ebx
12b:	83	c6 01	add	\$0x1, %esi
12e:	83	c4 10	add	\$0x10, %esp
131:	39	df	cmp	%ebx, %edi
133:	77	e6	ja	0x11b
135:	8d	65 f4	lea	-0xc(%ebp), %esp
138:	5b		pop	%ebx
139:	5e		pop	%esi
13a:	5f		pop	%edi
13b:	5d		pop	%ebp
13c:	c3		ret	
13d:	55		push	%ebp
13e:	89	e5	mov	%esp, %ebp
140:	57		push	%edi

141:	56	push	%esi
142:	53	push	%ebx
143:	83 ec 10	sub	\$0x10,%esp
146:	6a 00	push	\$0x0
148:	68 00 10 00 00	push	\$0x1000
14d:	68 00 00 01 00	push	\$0x10000
152:	e8 9d ff ff ff	call	0xf4
157:	83 c4 10	add	\$0x10,%esp
15a:	81 3d 00 00 01 00 7f	cmpl	\$0x464c457f,0x10000
161:	45 4c 46		
164:	75 21	jne	0x187
166:	a1 1c 00 01 00	mov	0x1001c,%eax
16b:	8d 98 00 00 01 00	lea	0x10000(%eax),%ebx
171:	0f b7 35 2c 00 01 00	movzwl	0x1002c,%esi
178:	c1 e6 05	shl	\$0x5,%esi
17b:	01 de	add	%ebx,%esi
17d:	39 f3	cmp	%esi,%ebx
17f:	72 15	jb	0x196
181:	ff 15 18 00 01 00	call	*0x10018
187:	8d 65 f4	lea	-0xc(%ebp),%esp
18a:	5b	pop	%ebx
18b:	5e	pop	%esi
18c:	5f	pop	%edi
18d:	5d	pop	%ebp
18e:	c3	ret	
18f:	83 c3 20	add	\$0x20,%ebx
192:	39 de	cmp	%ebx,%esi
194:	76 eb	jbe	0x181
196:	8b 7b 0c	mov	0xc(%ebx),%edi
199:	83 ec 04	sub	\$0x4,%esp
19c:	ff 73 04	push	0x4(%ebx)
19f:	ff 73 10	push	0x10(%ebx)
1a2:	57	push	%edi
1a3:	e8 4c ff ff ff	call	0xf4
1a8:	8b 4b 14	mov	0x14(%ebx),%ecx
1ab:	8b 43 10	mov	0x10(%ebx),%eax
1ae:	83 c4 10	add	\$0x10,%esp
1b1:	39 c1	cmp	%eax,%ecx
1b3:	76 da	jbe	0x18f
1b5:	01 c7	add	%eax,%edi
1b7:	29 c1	sub	%eax,%ecx
1b9:	b8 00 00 00 00	mov	\$0x0,%eax
1be:	fc	cld	
1bf:	f3 aa	rep stos	%al,%es:(%edi)
1c1:	eb cc	jmp	0x18f
1fb:	00 00	add	%al,(%eax)

```
1fd:          00 55 aa                add    %dl,-0x56(%ebp)
```

3.2 Objcopy(Q12)

Objcopy command is often used in the Makefile to convert binary files from one format to another.

Specifically, Objcopy is used to transform the output of the assembler (usually an ELF binary) into a flat binary file, which can be directly loaded and executed by the bootloader without any additional header information.

3.3 X86 Registers(Q14)

- **General purpose Register:** EAX(Extended Accumulator Register). used for and logic operations, storing function return values.
- **Segment Register:** CS(Code Segment). holds the starting address of the code segment in memory.
- **Status Register:** EFLAGS(Extended Flags Register). EFLAGS register holds the status and control flags that represent the current state of the processor.
- **Control Register:** CR0(Control registers). Control registers are used to control various operations of the processor. CR0 is a control register that is used to control the operating mode and other essential processor operations.

3.4 Entry.s(Q18)

The equivalent of entry.s in Linux kernel is arch/x86/entry/entry-64.S :

```
.section .text
.globl startup_64

startup_64:
    # Set up the stack pointer
    movq $init_stack, %rsp

    # Call the kernel initialization function
    call kernel_init

    # If kernel_init returns, enter an infinite loop
1:    jmp 1b

.section .data
```



```

align 8
init_stack:
    .skip 8192 # 8KB stack space for the kernel

# Kernel initialization function
.globl kernel_init
kernel_init:
    # Set up the data segment selector
    movq $0x10, %rax
    movq %rax, %ds
    movq %rax, %es
    movq %rax, %fs
    movq %rax, %gs

    # Clear the BSS section (zero out uninitialized data)
    movq $kernel_bss_start, %rdi
    movq $kernel_bss_end - $kernel_bss_start, %rcx
    xorq %rax, %rax
    rep stosq

    # Call the main function
    call main

    # Halt the CPU if main returns
    hlt

.section .bss
align 8
kernel_bss_start:
kernel_bss_end:

```

4 Xv6 kernel

4.1 Entry Address(Q19)

If it were virtual than it will need a page table to translate it to physical address but when the OS is just starting it can't produce page tables so it must be physical.

4.2 Segmentation(Q22)

The SEG-USER flag indicates that the segment is accessible from user-mode code. User-mode code should not have unrestricted access to system-critical parts of memory.

Therefore, the operating system sets the SEG-USER flag in the segment descriptor to restrict user-mode programs from accessing certain areas of memory, ensuring memory protection and security.

5 User Programs

5.1 Proc(Q23)

- **uint sz**: Size of process memory (bytes)
- **pde-t* pgdir**: Page table
- **char *kstack**: Bottom of kernel stack for this process
- **enum procstate state**: Process state
- **int pid**: Process ID
- **struct proc *parent**: Parent process
- **struct trapframe *tf**: Trap frame for current syscall
- **struct context *context**: swtch() here to run process
- **void *chan**: If non zero, sleeping on chan
- **int killed**: If non zero, have been killed
- **struct file *ofile[NOFILE]**: Open files
- **struct inode *cwd**: Current directory
- **char name[16]**: Process name (debugging)

In Linux the equivalent is task_struct.

5.2 System Prep(Q27)

The part of system preparation that is shared between all cores of a shared processor is the kernel code and data such as system calls cause

System calls provide a standardized interface for user programs to interact with the operating system. Sharing these calls ensures that all processes and cores behave consistently when interacting with the kernel.

The part that is exclusive to each core is the per-core data. This includes the processor control block (PCB) or task structure, which contains information specific to the currently executing process on that core.

6 Debugging

6.1 Breakpoint(Q1)

Using "info break" command we can see all our breakpoints.

6.2 Breakpoint(Q2)

Using "delete number" command whereas number is the number of the breakpoint added, we can delete the breakpoint. While creating breakpoints, they get a number from 1 to end.

6.3 Bt command(Q3)

Every thing happened after the breakpoint is pushed to stack. This command will print one line per frame for frames in the stack. By default, all stack frames are printed.

6.4 x and print(Q4)

x command examines memory. Display the contents of a memory location. But print command is used to evaluate and display the value of a variable or an expression. For printing the value of a special register we can use "x address" command where address is the address of the register.

6.5 Status(Q5)

Using "info registers" command, we can see the register names and corresponding values. Using "info locals" command, we can see the local variable names and corresponding value. Both edi and esi are general-purpose registers. edi(extended destination index), is commonly used for string operations. esi(extended source index), it holds the memory address from where the data is to be read during string operations.

6.6 Struct Input(Q6)

This struct consist of a buffer, r(read index), w(write index), e(end index). buffer is for holding the data, r shows the index where we read from, w shows where we write through, e shows the end of the line.

7 Assembly Debugging

7.1 Layout(Q7)

Layout asm shows the assembly of the c code that we breakpointed before and Layout src shows the source code where we put breakpoint on.

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS... x kasra@kasra-Linux: ~/Documents/OS... x
307     for(int i =0; i < COMMAND_BUF; i++){
308         if(c[i] == '\n' || c[i] == C('D'))
(gdb) info registers
eax             0x0             0
ecx             0x0             0
edx             0x0             0
ebx             0x80113e50      -2146353584
esp             0x8010b510      0x8010b510 <stack+3920>
ebp             0x8010b518      0x8010b518 <stack+3928>
esi             0x80114340      -2146352320
edi             0x80113da4      -2146353756
eip             0x80103d35      0x80103d35 <mycpu+21>
eflags          0x46           [ IOPL=0 ZF PF ]
cs              0x8             8
ss              0x10            16
ds              0x10            16
es              0x10            16
fs              0x0             0
gs              0x0             0
fs_base         0x0             0
gs_base         0x0             0
k_gs_base       0x0             0
cr0             0x80010011      [ PG WP ET PE ]

```

Figure 1: info registers output

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS... x kasra@kasra-Linux: ~/Documents/OS... x
eflags          0x46           [ IOPL=0 ZF PF ]
cs              0x8             8
ss              0x10            16
ds              0x10            16
es              0x10            16
fs              0x0             0
gs              0x0             0
fs_base         0x0             0
gs_base         0x0             0
k_gs_base       0x0             0
cr0             0x80010011      [ PG WP ET PE ]
cr2             0x0             0
cr3             0x3ff000        [ PDBR=0 PCID=0 ]
--Type <RET> for more, q to quit, c to continue without paging--c
cr4             0x10            [ PSE ]
cr8             0x0             0
efer            0x0             [ ]
xmm0            {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_in
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0
x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm1            {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_in
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0
x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}

```

Figure 2: info registers output

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS... x kasra@kasra-Linux: ~/Documents/OS... x
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm1 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm2 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm3 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm4 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm5 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm6 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm7 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 12 times>, 0x80, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0

```

Figure 3: info registers output

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS... x kasra@kasra-Linux: ~/Documents/OS... x
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm5 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm6 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
xmm7 {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
t8 = {0x0 <repeats 12 times>, 0x80, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0
mxcsr 0x1f80 [ IM DM ZM OM UM PM ]
(gdb) info locals
apicid = <optimized out>
i = <optimized out>
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
0x80103d35 in mycpu () at proc.c:45
45     apicid = lapicid();
(gdb) 

```

Figure 4: info locals output

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS-... x kasra@kasra-Linux: ~/Documents/OS-... x
B+> 0x80100ad0 <char_handler> endbr32
0x80100ad4 <char_handler+4> push %ebp
0x80100ad5 <char_handler+5> mov %esp,%ebp
0x80100ad7 <char_handler+7> push %edi
0x80100ad8 <char_handler+8> push %esi
0x80100ad9 <char_handler+9> push %ebx
0x80100ada <char_handler+10> sub $0x1c,%esp
0x80100add <char_handler+13> mov 0x8(%ebp),%ebx
0x80100ae0 <char_handler+16> mov 0xc(%ebp),%eax
0x80100ae3 <char_handler+19> cmp $0x15,%ebx
0x80100ae6 <char_handler+22> je 0x80100cfa <char_handler+554>
0x80100aec <char_handler+28> jg 0x80100b40 <char_handler+112>
0x80100aee <char_handler+30> cmp $0xc,%ebx

remote Thread 1.1 In: char handler L238 PC: 0x80100ad0
#1 0x80100e3a in consoleintr (getc=0x8010b4a0 <stack+3808>) at console.c:325
(gdb) down
#0 char_handler (c=102, doprocdump=0x8010b4a0 <stack+3808>) at console.c:238
(gdb) down
Bottom (innermost) frame selected; you cannot go down.
(gdb) layout src
(gdb) layout asm
(gdb)

```

Figure 5: layout asm output

```

kasra@kasra-Linux: ~/Documents/OS-Lab1
kasra@kasra-Linux: ~/Documents/OS-... x kasra@kasra-Linux: ~/Documents/OS-... x
console.c
234 }
235
236 void
237 char_handler(int c, int *doprocdump)
B+> 238 {
239     switch(c){
240         case C('P'): // Process listing.
241             // procdump() locks cons.lock indirectly; invoke later
242             *doprocdump = 1;
243             break;
244         case C('U'): // Kill line.
245             while(input.e != input.w &&
246                 input.buf[(input.e-1) % INPUT_BUF] != '\n'){
remote Thread 1.1 In: char handler L238 PC: 0x80100ad0
(gdb) down
#1 0x80100e3a in consoleintr (getc=0x8010b4a0 <stack+3808>) at console.c:325
(gdb) down
#0 char_handler (c=102, doprocdump=0x8010b4a0 <stack+3808>) at console.c:238
(gdb) down
Bottom (innermost) frame selected; you cannot go down.
(gdb) layout src
(gdb)

```

Figure 6: layout src output

7.2 Transport(Q8)

For traversing between the things that are in stack we can use up and down command.

8 Boot

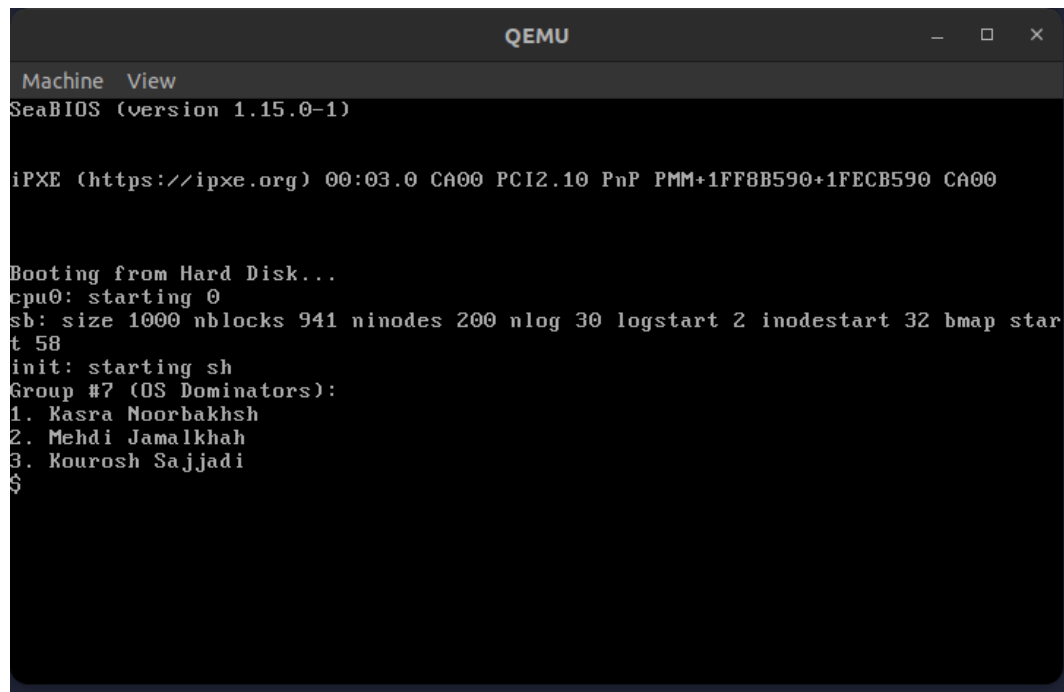
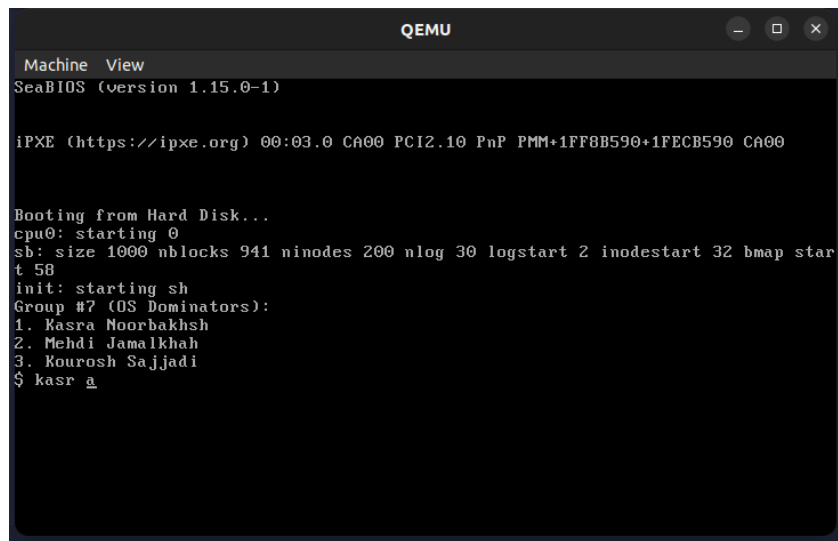


Figure 7: Names

9 Command Outputs

9.1 Commands



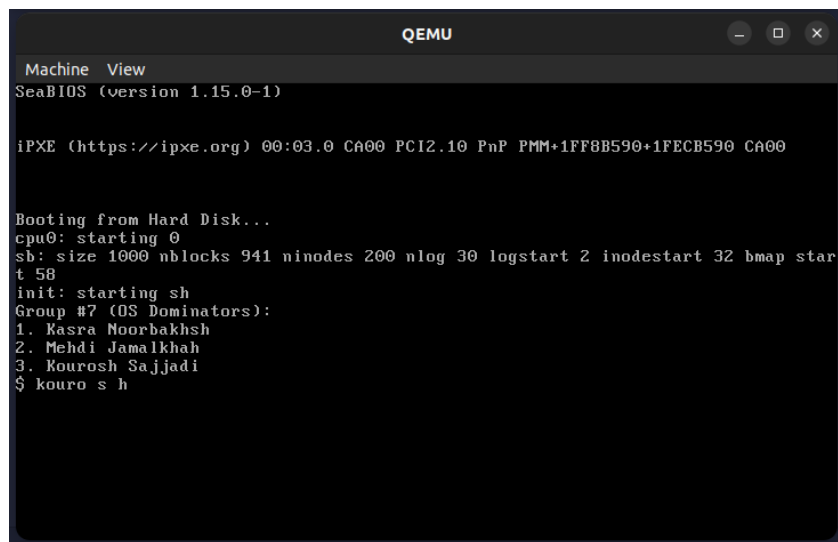
The screenshot shows a QEMU window with a terminal interface. The terminal displays the SeaBIOS boot process, including the iPXE boot loader, booting from a hard disk, and the initialization of the operating system. The prompt is \$ kasr _a.

```
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 50
init: starting sh
Group #7 (OS Dominators):
1. Kasra Noorbakhsh
2. Mehdi Jamalkhah
3. Kourosh Sajjadi
$ kasr _a
```

Figure 8: control + B



The screenshot shows a QEMU window with a terminal interface. The terminal displays the SeaBIOS boot process, including the iPXE boot loader, booting from a hard disk, and the initialization of the operating system. The prompt is \$ kouro s h.

```
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 50
init: starting sh
Group #7 (OS Dominators):
1. Kasra Noorbakhsh
2. Mehdi Jamalkhah
3. Kourosh Sajjadi
$ kouro s h
```

Figure 9: control + F

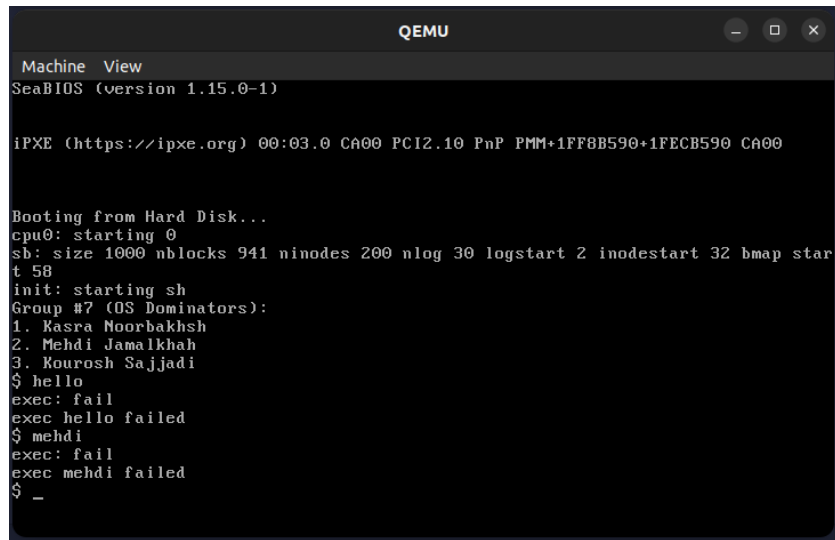


Figure 10: control + L

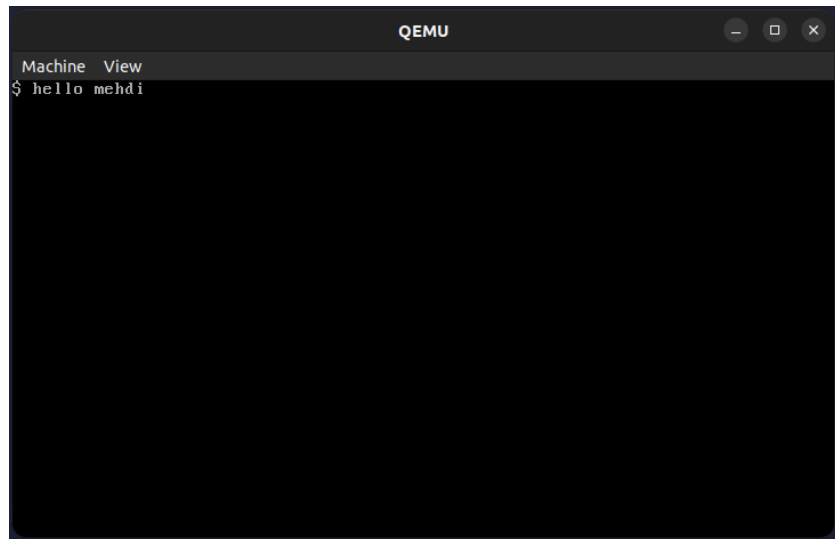


Figure 11: control + L

```
QEMU

Machine View
Nelson Elhage, Saar Ettinger, Alice Ferrazzi, Nathaniel Filardo, Peter
Froehlich, Yakir Goaron, Shivam Handa, Bryan Henry, Jim Huang, Alexander
Kapshuk, Anders Kaseorg, kehao95, Wolfgang Keller, Eddie Kohler, Austin
Liew, Imbar Marinescu, Yandong Mao, Matan Shabtay, Hitoshi Mitake, Carmi
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
Eldar Sehayek, Yongming Shen, Cam Tenny, tyfkda, Rafael Ubal, Warren
Toomey, Stephen Tu, Pablo Ventura, Xi Wang, Keiichi Watanabe, Nicolas
Wolovick, wxdao, Grant Wu, Jindong Zhang, Icenowy Zheng, and Zou Chang Wei.

The code in the files that constitute xv6 is
Copyright 2006-2018 Frans Kaashoek, Robert Morris, and Russ Cox.

ERROR REPORTS

We don't process error reports (see note on top of this file).

BUILDING AND RUNNING XV6

To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
will need to install a cross-compiler gcc suite capable of producing
x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC
simulator and run "make qemu". $ _
```

Figure 12: Up/Down 1

```
QEMU

Machine View
simulator and run "make qemu". $ ls
.          1 1 512
..         1 1 512
README    2 2 2286
strdiff.c 2 3 894
cat        2 4 15460
echo       2 5 14344
forktest   2 6 8788
grep       2 7 18304
init       2 8 15076
kill       2 9 14428
ln         2 10 14324
ls         2 11 16892
mkdir     2 12 14452
rm         2 13 14432
sh         2 14 28488
stressfs   2 15 15360
usertests  2 16 62860
wc         2 17 15888
zombie     2 18 14008
strdiff    2 19 14980
console    3 20 0
strdiff_result 2 21 8
$ _
```

Figure 13: Up/Down 2

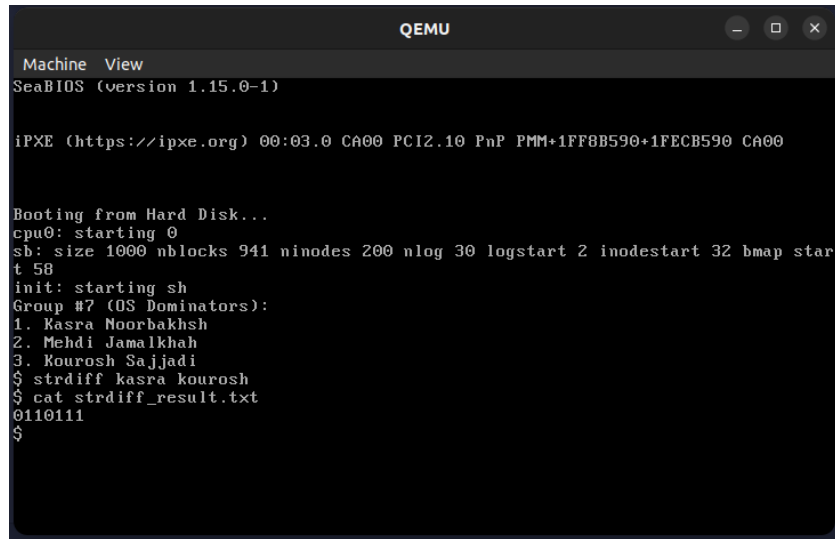
```
QEMU
Machine View
simulator and run "make qemu". $ ls
.          1 1 512
..         1 1 512
README    2 2 2286
strdiff.c 2 3 894
cat       2 4 15460
echo      2 5 14344
forktest  2 6 8788
grep      2 7 18304
init      2 8 15076
kill      2 9 14428
ln        2 10 14324
ls        2 11 16892
mkdir     2 12 14452
rm        2 13 14432
sh        2 14 28488
stressfs  2 15 15360
usertests 2 16 62860
wc        2 17 15888
zombie    2 18 14008
strdiff   2 19 14980
console   3 20 0
strdiff_result 2 21 8
$ ls
```

Figure 14: Up/Down 3

```
QEMU
Machine View
simulator and run "make qemu". $ ls
.          1 1 512
..         1 1 512
README    2 2 2286
strdiff.c 2 3 894
cat       2 4 15460
echo      2 5 14344
forktest  2 6 8788
grep      2 7 18304
init      2 8 15076
kill      2 9 14428
ln        2 10 14324
ls        2 11 16892
mkdir     2 12 14452
rm        2 13 14432
sh        2 14 28488
stressfs  2 15 15360
usertests 2 16 62860
wc        2 17 15888
zombie    2 18 14008
strdiff   2 19 14980
console   3 20 0
strdiff_result 2 21 8
$ cat README
```

Figure 15: Up/Down 4

10 User Program



```
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #7 (OS Dominators):
1. Kasra Noorbakhsh
2. Mehdi Jamalkhah
3. Kourosh Sajjadi
$ strdiff kasra kourosh
$ cat strdiff_result.txt
0110111
$
```

Figure 16: Strdiff