Goal: Understanding an OS by building a small OS.

Why OS?

- OS is a library (explain)
- OS share the resources among the applications

How an OS interface looks like.
Disk interface is a sequence of sectors. The disk device lets users read/write data at the sector granularity.
OS interface is the file system.
fopen (filename, “w”)

PC ARCHITECURE (x86)

Draw a picture of resources with the system bus, cache, registers, etc.

- CPU, DISK, NIC, RAM, KEYBOARD, MOUSE, MONITOR

MEMORY LAYOUT of PHYSICAL ADDRESS:

---------- 0xFFFFFFFF (4 GB)
32-bit memory
mapped devices

Unused
---------- depends on the amount of RAM

Extended Memory

---------- 0x10000 (1 MB)
BIOS ROM
---------- 0x0F000 (960 KB)
16-bit devices
Expansion ROMs
---------- 0x0C000 (768 KB)
VGA DISPLAY
---------- 0x0A000 (640 KB)

LOW MEMORY

---------- 0x00000

X86 has eight 32-bit registers:

EAX, ECX, EDX, EBX, ESI, EDI, ESP, EBP

A typical application requires more memory.

- CPU sends out address on address lines
- Data comes back on data lines

Explain how an application looks like.

```
gcc -m32 -fno-pic example.c
objdump -dx a.out |less
```

CPU interface:

Let us say initially, instruction is set to foo.

```
while (1) {
  len = length (instruction);
  execute (instruction);
  instruction += len;
}
```

- Instructions are also in memory
    - EIP points to the next instruction in memory
    - EIP is incremented after every instruction
    - EIP can be modified by call, ret, jmp, conditional jumps

Conditional Jumps

X86 also maintains a flag register

Some instructions set specific bits in the flag registers

- Whether last arithmetic instruction overflowed
- Was positive/negative
- Was [not] zero
- Carry/borrow on add/subtract
- J[N]O, J[N]E, J[N]Z, J[N]C

X86 Instruction set.

- AT&T (gcc) syntax: op src, dst (labs, xv6)
    - Uses b, w, l suffix on instructions to specify size of operands
- Operands are registers, constants, direct memory access, indirect memory access
- Examples:

AT&T syntax	"C"-ish equivalent

movl %eax, %edx	edx = eax;	register mode
movl $0x123, %edx	edx = 0x123;	immediate
movl 0x123, %edx	edx = *(int32*)0x123;	direct
movl (%ebx), %edx	edx = *(int32*)ebx;	indirect
movl 4(%ebx), %edx	edx = *(int32*)(ebx + 4);	displaced
movl (%ebx, %ecx, 4)	ebx = ((int32*)ebx)[ecx]	indirect
movl (%ebx, %ecx, 1)	ebx = ((char*)ebx)[ecx]	indirect

Homework on x86 instructions:

Where local variables are allocated:

- A stack is used for automatic memory management of local variables
- Returning the address of a local variable is not-permitted
- pass-by-value, pass-by-reference

Instructions relevant to stack,

PUSH and POP

Other arithmetic instructions,

ADD, SUB