Today =>
Intro to Operating Systems
(take 537 to learn more)
What is an operating system?

software that makes machine much easier to use

Examples:

- overcome physical limitations of machine

  - one (or few) CPUs (physical)

    - transforms into many virtual CPUs

    (Illusion)
OS roles:

-- memory: cpu

many programs, running all at once
Sharing same phys memory

illusion: virtual memory
each large, private address space (memory)

=> Virtualization

=> CPU 1 => many

=> Memory large, private mem per program
Other roles for OS

\[ \Rightarrow \text{Standard} \quad \text{protected} \quad \text{library} \]

files:
- open, read/write, close
- create, delete
- notion: file permissions

\[ \Rightarrow \text{OS as resource manager} \]
\[ (\text{CPU, mem}) \text{ etc.} \]

\[ \Rightarrow \text{which prog gets which resource when?} \]
Virtualization:

key mechanism: sharing ➔ CPU ➔ memory

⇒ time sharing, A, B

CPU A A A A B B B... B A A... A...

⇒ space sharing

if resource can be divided into pieces,
give some ⇒ A, some ⇒ B

(memory)
Virtualize CPU!

⇒ OS is a program

⇒ Runs first (system boots)

(Setup)

⇒ ?

Eventually, want to run program: A

⇒ OS load A's code/data into memory (creating A's address space)

⇒ What next? 1) Setup A's stack

2) Jump to main (entry point)

int main (int argc, char* argv[])
Problems w/ approach #1:

\[ \Rightarrow \text{malicious/buggy code} \]

\[ \text{while (1)} \]

\[ ; \]

\[ \text{os } \Rightarrow \text{A} \]

\[ \text{while (1)} \]

\[ ; \]

what if

\[ \Rightarrow \text{os wants to } \]

\[ \text{after "a while"}, \]

\[ \text{run B?} \]

Need: mechanism to regain

control of CPU (for os)

help:

\[ \Rightarrow \text{timer [interrupt]} \Rightarrow \text{hardware} \]
Timer interrupt:

Concept: interrupt

Normal execution: eip, ...

CPU:

while (1)
  → fetch (advance inst. ptr)
  → decode
  → execute

interrupt ⇒ forces break between instructions to run some other code (interrupt handler) is part of OS
Timer interrupt:

- Programmable timers
  - Program to interrupt CPU after some period of time (e.g., 10 milliseconds)

Protocol:

1. Inform CPU of location of timer interrupt handler.
2. OS program timer interrupt after X ms

OS: Load and jump to A

Interrupt:
- OS can run again (switch to B, keep running A, etc.)
Problem: (turn on/off timer int., change location of int. handler)

OS \rightarrow OK to do

user programs \rightarrow not OK to do

(need: h/w support)

\Rightarrow bit per CPU (in status register)

mode bit:

\begin{align*}
0 & : \text{privileged mode (kernel)} \\
1 & : \text{not-priv. mode (user)}
\end{align*}

OS: boot \Rightarrow (kernel mode)

: set up int. handlers (w/ special inst)

: set up timer

: run user prog A: not jump to code

need new inst: jumps to user code AND lowers privilege of CPU
Review:

boot (kernel)

setup (kernel)

(reduces priv. to user mode)

(int handler of OS)

OS

A

time

when OS → prog,
prog → OS

⇒ register state

(save/restore)

timer interrupt
CPU: virtualize

many CPUs from few physical CPUs

Mechanism: h/w + OS

timer interrupt

mode: kernel/user mode (privilege level)

transitions: (be careful)

e.g. when interrupt occurs
    ⇒ h/w save register state
    of CPU ⇒ OS memory

Running program:

Process

<table>
<thead>
<tr>
<th>code</th>
</tr>
</thead>
</table>

Address space

registers
Example: user program tries to change location of interrupt handler detected by CPU (user mode => nope)

=> raise exception
✓ (some inst. did something bad)

OS exception handler:

=> reaction: [kill process]

Need: OS in role as standard library
OS as std library:

\[ \text{Phys Mem} \rightleftharpoons \text{OS} \]

\[ \text{files:} \]

\[ \text{open( )} \]

\[ \rightarrow \text{check permissions} \]

\[ \rightarrow \text{if (fail)} \]

\[ \rightarrow \text{return fd} \]

\[ \text{need: protected function call} \]
System calls:
set of functions that OS exposes (in safe way) to user programs

How does OS allow safe exec. of system calls?

\[ \Rightarrow \text{restrict how transfer of control occurs} \]

\[ \Rightarrow \text{system call:} \]
Special instruction: \text{trap inst.}
Specify which sys call via number (e.g., put \#16 in register)

OS \rightarrow \text{system call handler} \rightarrow \text{h/w: saves reg state of A} \rightarrow \text{ret from trap} \rightarrow \text{(user)}

A \rightarrow \text{(user) trap}
Review

OS:

@ boot time: (kernel mode)

- set up handlers
  - timer interrupt handler
  - exception (tries to do handler something bad)
  - system call handler

- start timer interrupt

To run user program:
- load into memory
- use special inst to
  - jump to main
  - reduce priv to user mode
    - (ret-from-trap)
    - special inst: A
Allows safely virtualize CPU

(Time Sharing)

OS
A
B

+--------+--------+
| Timer  | Timer  |
+--------+--------+

use CPU  sleep

A

disk

100 x \[
\frac{use \text{ (cpu)}}{use \text{ (cpu)} + sleep \text{ time} (I/O)}\] = CPU utilization (%)
OS: virtualize memory

illusion:
- private, large address space (memory) per process

technique:
- space sharing

lots of techniques:
- base/bounds (dyn. relocation)
- segmentation
- paging

phys mem.
Virtual Address Space (A)

physical page frames

VP₀(A) (fixed-size)

VP₁(A) (fixed-size)

4KB

VAS (B)

VP₀(B)

VP₁(B)

phys memory

VP₀(B)

VP₁(B)

VP₁(A)

3000
Virtual Memory:

Program is running:

Every address (instruction, data) is virtual address (not physical).

Upon every memory reference (inst, data):

H/W+OS: translation must occur

(virt addr \rightarrow) phys addr

\downarrow

phys memory
Virtual Memory:

prog: every address is [virtual]

translation: \( v \rightarrow p \)

how much info? (translation info)

\( \Rightarrow \) one phys frame per virtual page

Example:

32-bit virt. addr. space

4 KB pages

how many translations do we need for this single process?
virt.
Address space

4KB pages

$$\{8\,\text{KB}\}$$

$$\Rightarrow 2$$

$$2^{13} \text{ bytes}$$

$$\Rightarrow 2$$

$$2^{12} \text{ (pagesize)}$$

13 bits

$$\Rightarrow 32\text{-bit address}$$

$$1\,\text{KB} \approx 2^{10}$$

$$1\,\text{MB} \approx 2^{20}$$

$$\approx 1\,\text{million translations}$$
300 processes

each process has its own translation info

( $2^{32}$ addr space

\approx \frac{1}{4}$m$ trans. info (4 bytes)

\implies 4 MB

\implies \approx 1.2$ GB of translation info

\implies memory \implies data structure (page table)
Page Table: simple

array ⇒ linear(page table)

indexed by [virtual page #] ⇒ VPN

Example: 64 byte address space (unrealistically small)

Page size: 8 bytes

Virtual address:

```
0 1 2 3 4 5 6 7...
```

```
8 bytes
```

```
3 bits
```

```
3 bits
```

```
VPN
```

```
Offset
```

```
6-bit VA
```
Example: `movb 63, %eax`

Hardware: virt. addr.

\[ \text{VA} \quad 111 \\ 111 \quad \text{offset} \]

\[ \Rightarrow \text{extract VPN from VA} \]

\[ \Rightarrow 111 \Rightarrow 7 \]

Page Table: \# of entries = \# of virt pages

PT \rightarrow PT[0] \rightarrow PT[1] \rightarrow \ldots \rightarrow PT[7]

PFN: 1

Phys Addr: 00111111

\Rightarrow 15

Virt AS

0 of Process

Phys mem
Review:

- every inst fetch +
  - every explicit data load/store

=> Translation

  start w/ H/W: VA

  added extra => [fetch page table entry (PTE)]
  mem reference !

  => extract PFN from it

  => form Phys Addr (PA)

  => do mem op w/ PA
Translation:

- a lot of translation info

=> page table per process

Translation:

where is the page table for this process?

Need: register (CPU)

Page Table Base Register

=> Phys. Addr of beginning of page table of currently running process
 problem:

normally, process does not use all of its addr. space

(large, sparse addr space)

=) support into page table

page table

Unused

Valid bit

translation:

PFN
H/W:
\[
\text{PTBR → CPU}
\]

Virt addr: \(
\Rightarrow \) virt. page number

extracts VPN

\[
\text{uses PTBR + (VPN \times \text{sizeof(Page Table Entry)})}
\]

\[
\Rightarrow \text{PA of PTE (phys. addr)}
\]

\[
\text{(page table entry)}
\]

\[
\text{fetch PTE}
\]

\[
\text{if (PTE.valid = = 0)}
\]

\[
\text{raise exception} \Rightarrow \text{OS handler}
\]

\[
\text{kll process}
\]

\[
\text{else}
\]

\[
\text{extract PFN, form phys. addr (PA)}
\]

\[
\text{fetch (PA)}
\]
Page tables too big

$2^{32} \times 4$ KB pages

$\Rightarrow \approx 4$ MB of info/process

100 processes $\Rightarrow 400$ MB of page table info in mem

(translation)

$\Rightarrow$ Paging too slow

extra mem access per

mem access

$mov$ 

$\left[ \begin{array} \text{inst fetch} \\
\text{translation}
\end{array} \right]$

$\left[ \begin{array} \text{explicit load} \\
\text{translation}
\end{array} \right]$