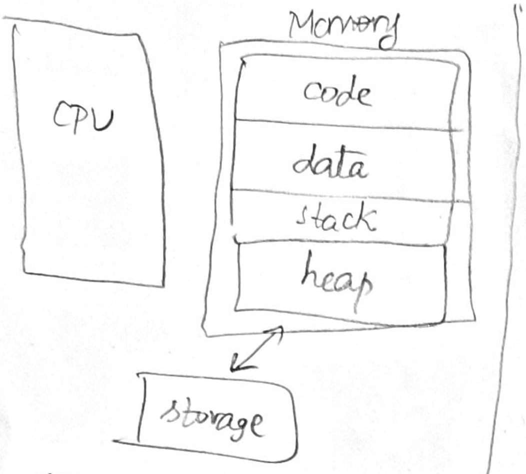


① Process abstraction

② Virtualization

- time sharing
- resource sharing

"limited direct execution"

Isolation Performance

③ Isolation why?

- process can be buggy
  - crash
  - infinite loop
- process can be malicious
  - take over other processes / users

④ System Resources (cpu, mem, disk, net)

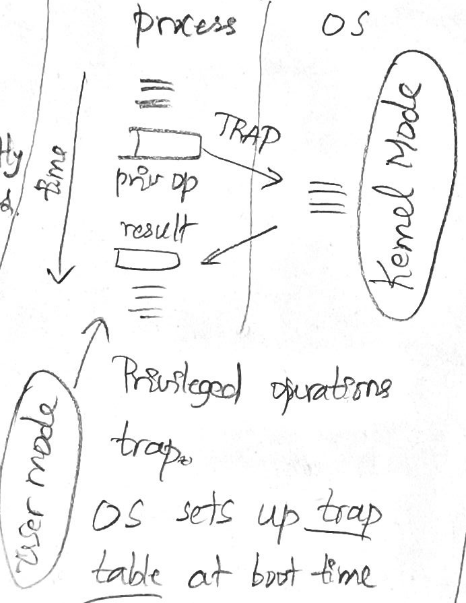
⊛ access only after vetting ⇒ but process directly executes.

⊛ Get hardware support

privilege levels

traps

⑤ trap process



Privileged operations trap.

OS sets up trap table at boot time

⑥ trap table — HW data structure

trap#	address of func/handler
0	→
1	→
:	

→ examples

Boot time ⊛ setup

① access the storage

② execute any privileged operation, in general

⑦ system calls

→ library to do req assembly

⑧ Basic limited direct exec.

OS (Kern mode)	process (user mode)
Boot time set up all trap handlers & trap table	(nothing. All this eqn before process starts)
Run time Initialize text, data, stack w/ args save regs return from trap →	

⑨ sharing between processes

① cooperative approach:

→ wait for sys call

→ yield()

② Non cooperative approach

Timer Interrupt

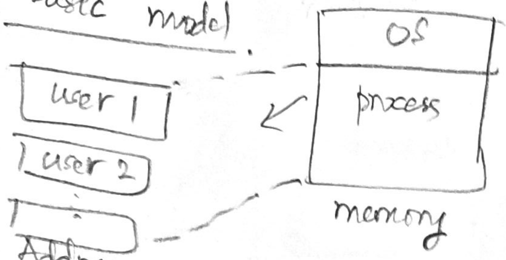
hardware support once again.

⑩ Timer Interrupt

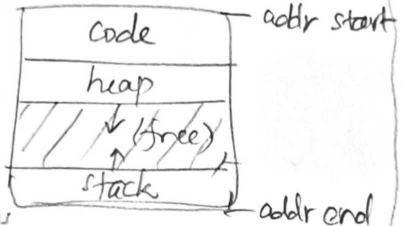
Boot time kernel	hardware
→ start timer interrupt & set up handler	CONTEXT SWITCH ⊛
	timer interrupt
(whenever chose to move to diff process)	⊛ save regs to kernel stack move to kern mode Jmp to handler

⑪ virtualizing memory.

Basic model



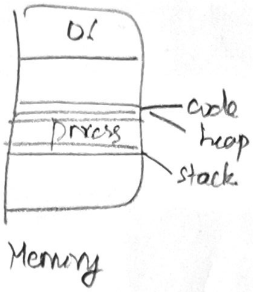
Address space



⊛ virtual addresses

⊛ actually located at different locations

⑫ Dynamic relocation



User abstraction

→ "fixed" addressing/ instructions regardless of location

⑬ Hardware translation

⊛ Base & bounds

load register, addr (// vaddr)

→ virtual to phy addr

if (base < base + vaddr < bound

load from base + vaddr

⊛ else fault // trap

Memory Management unit (MMU)

⑭ primitives for hardware address translation.

① privilege modes

② base / bound registers

③ check vaddr within bound

④ privileged insn to update base / bounds ⇒ "sensitive"

⑤ privileged insn to register exception handlers

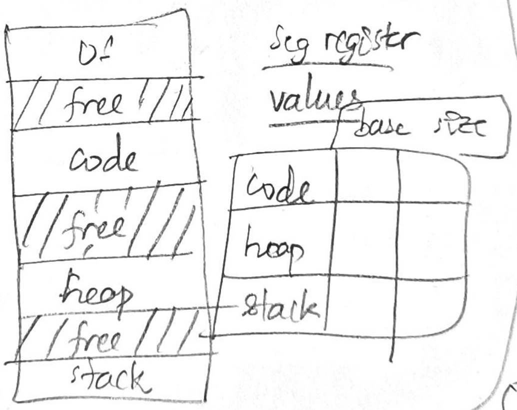
⑥ raise exceptions on invalid address

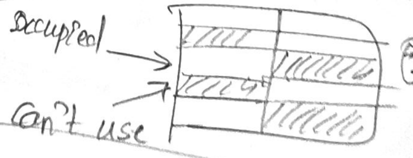
OS must start managing memory

⑮ segmentation

empty mostly addr spaces

⇒ generalize to many base and bounds.





⊛ can't use dedicated registers any more

⇒ page table in memory.

Per process data structure

⑯ hardware for Segmentation

① tell hardware which segment?

top k bits of addr

② Memory protection bits.

read? write? execute?

problems

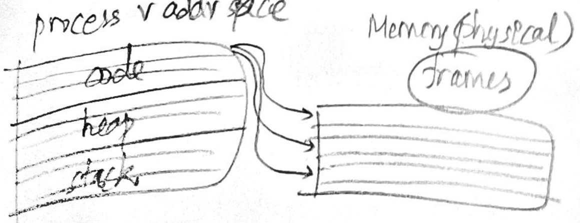
① Context switch

→ save & restore segdescriptors

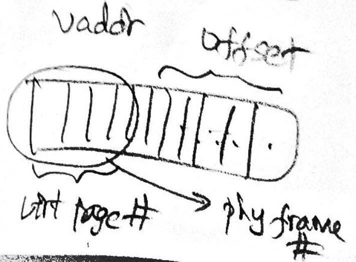
② non compact memory layouts!

⑰ Paging

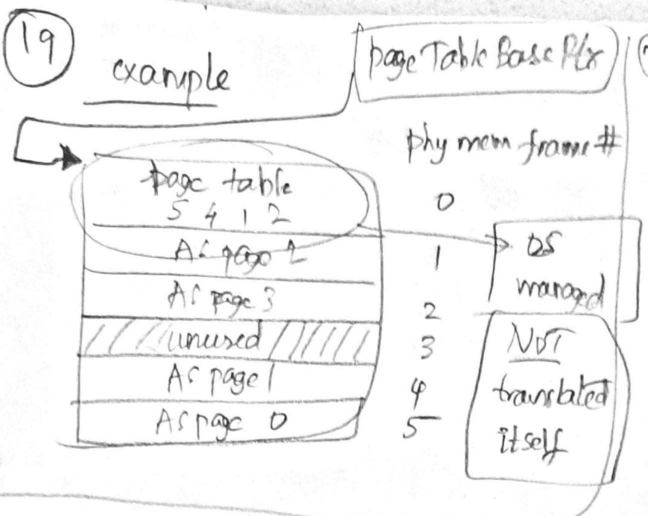
use fixed size chunks. Many of them.



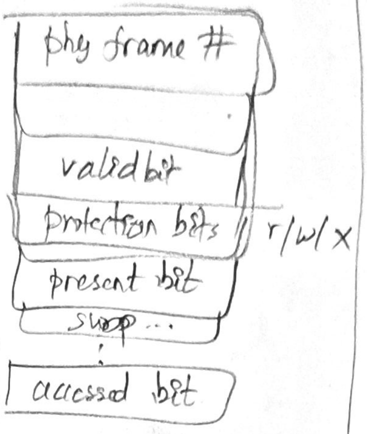
⑱ Translation



(19) example



(20) Page table entry



(21) translation (hardware)

VP# = (vaddr & vmask) >> SHIFT

PTE addr // linear pagetable = Page Table Base Ptr + (vp# x sizeof (PTE))

translate to get phy frame#

offset = vaddr & offset mask.

Phy addr = (pf# << SHIFT) | offset

(22) translation

check valid, protection, PTE valid.

Else trap! →
- Minor fault (retrieve shared)
- Major fault (retrieve from disk)
- Invalid fault (vaddr not valid)

(23) Paging is too inefficient / slow

- at least two mem access
- Large memory just for page tables

(24) Address translation cache (TLB)

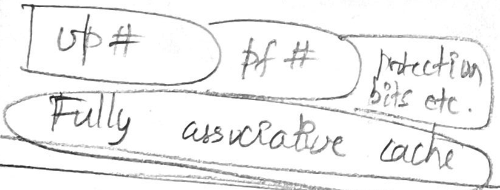
- Efficient memory, small → cache PTEs
- first check TLB
  - → hit ⇒ use (protection etc).
  - → miss ⇒ walk page table

(25) Handling TLB miss

→ hardware managed TLB (Intel x 86)

→ software handled (RISC) use traps

(26) TLB entry



Fully associative cache

(27) TLB + context switches

(*) Per process data structure of pagetables

option (1) TLB flush

option (2) tagged TLB

- use addr space ID

(28) smaller page tables?

→ multi level page tables page directory + table

→ superpages

→ hybrid segment & paging

→ swap page tables to disk

(29) TLBs make paging feasible

~~at~~, physically indexed versus virtually indexed caches

(30) Principles applied to entire operating systems!

- Xen
- amazon ec2

+ impact

(31) system virtualization

① OS assumes it is the most privileged entity.

② OS virtualization. why?

⇒ time sharing OSes

⇒ unit of software/app deployment — any software & config can be run

③ actually properties of both OS & hardware arch matter (together provide isolation)

(32) virtual machine monitor (VMM) 1974

→ virtualize all system resources (processor, mem, disk/net/I-O, ...)

3 requirements of interest (popek & goldberg)

① equivalence

– same as direct execution on underlying phy machine

② Resource control/safety

→ VMM must have complete control of resources

③ efficiency

→ mostly execute without VMM intervention (not a "requirement" for functionality. But need it in practice.)

(33) ISA classification

① privileged insn's

→ must trap in user mode

② control sensitive

→ can change processor mode or "whether something will trap in 1st place" (eg. change mem bounds)

③ Behaviour sensitive

→ effect of instruction depends on mode or bounds registers

(33) theorem (basic result)

(unmodified guest OS) efficiency virtualized if $S2 \cup S3 \subseteq S1$ } FULL virtualizn

→ "trap and emulate" virtualization (for privileged instructions)

→ sufficient condition,

x86 does not satisfy it (classically) → now Intel VT-x and AMD-V

→ unmodified guest OS

→ workaround tech

① Dynamic recompilation — replace critical insn's at run time.

- cache emulation code
- assist through hardware

② paravirtualization → port before running

Xen

(34) Containers

① kernel namespaces (access isolation) < process, network, (mem)

② Control groups — (process trees) (resource isolation) < cpu, mem/blkio

③ union filesystems ← ~~3.2~~ copy on write ← ③ layering

(35) xen

① cpu → ring 1 of
(else trap ring 0)

- synchronous hypercalls
- system calls
  easy.
  same handlers.
- page faults
  extended stack frame
  (cr3 unreadable)

② I/O virtualization

Ring buffers

async producer – consumer

descriptor – data separation / zero copy.

③ Memory

types

guest OSes manage pagetable

writes validated by xen.

avoid effect of TLBflush on xen

④ Management
- separate policy from mech
- dom 0.

(33A) (E, M, P, R)

E → executable storage (reg+mem)

M → mode

P → program counter (vaddr)

R → relocation register (bounds)

(E, M, P, R) ⟷ Memory-mapped IO

Simple dynamic relocation (base/bounds)

[memory trap: attempt to OOB access]

Trap: Storage left unchanged except currently active (m, p, r) storage

Privileged: (e, s, p, r) and (e, u, p, r)

(e, s, p, r) → does not trap

(e, u, p, r) → traps

(33B)

Control sensitive attempt to change mem resources (through reloc bounds) or affects processor mode without a memory trap (but regular trap OK)

eg, return to user mode.

(33C) Behaviour sensitive

effect of execution depends on values of relocation bounds R, or mode M

e.g., load directly phys addr; move data from previous instruction space.

(33D) VMM control program

- dispatcher (for traps)
- allocator (what resources to provide)
- interpreter for instructions that trap (simulate privileged instructions)

(*) VMM runs as supervisor, all others run in user mode.

(*) Hardware assumptions: trap, relocation, MMU, privilege levels