

① Process abstraction

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

⊛ access only after

letting ⇒ but process directly executed.

⊛ Get hardware support

privilege levels

traps

② Virtualization

- time sharing
- resource sharing

"limited direct"

↓ execution ↑

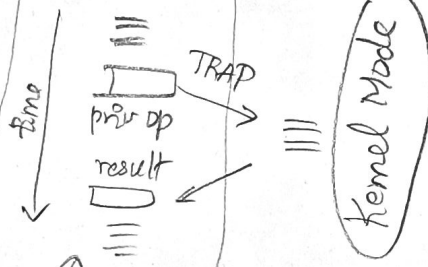
Isolation

Performance

③ Isolation why?

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

⑤ trap process



Privileged operations traps

OS sets up trap table at boot time

⑥ trap table - hw data structure

trap#	address of fun/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 (kernel mode) process (user mode))

Boot time

set up all trap handlers & trap table

Run time

initialize text, data, stack w/ args
save regs
return from trap

(nothing. All this even before process starts)

⑨ sharing between processes

processes

① cooperative approach:

- wait for sys call
- yield()

② Non cooperative approach

Timer Interrupt

hardware support

⑩ Timer Interrupt

boot time kernel
→ start timer
interrupt & set up handler

hardware

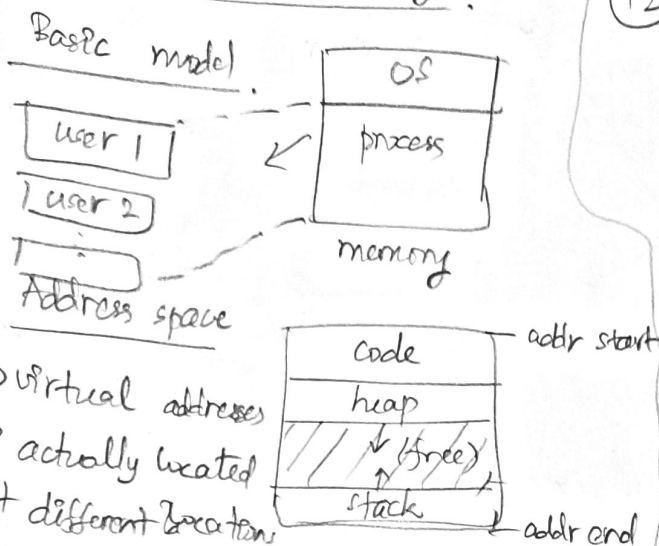


(whenever choose to move to diff process)

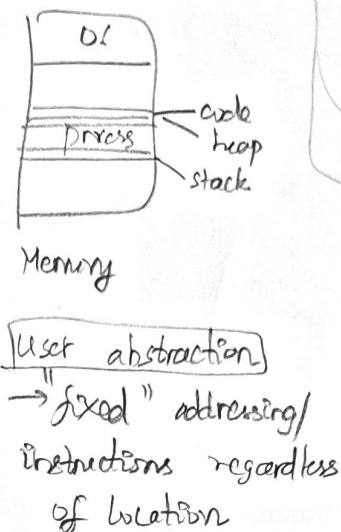
timer interrupt

⊛ save regs to kernel stack
move to kernel mode
Jump to handler

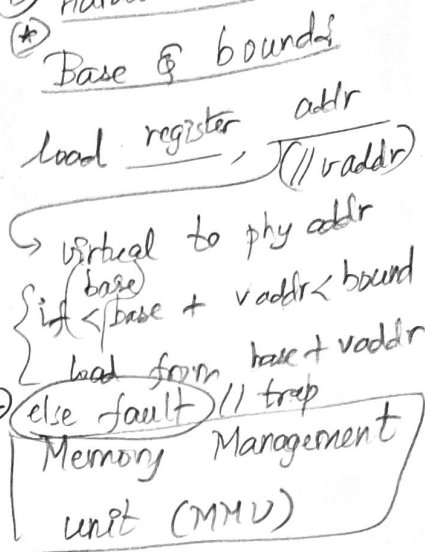
11 Virtualizing memory.



12 Dynamic relocation



13 Hardware translation

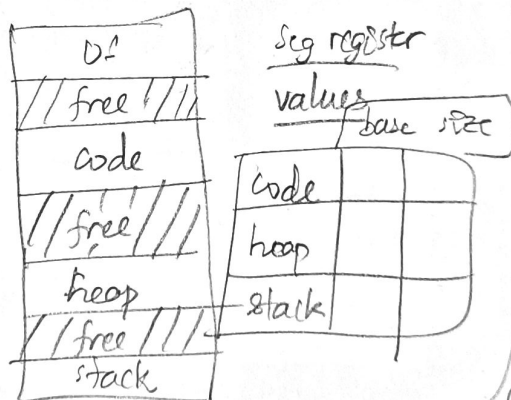


14 primitives for hardware address translation.

- 1 privilege modes
 - 2 base / bound registers
 - 3 check vaddr within bound
 - 4 privileged insn to update base / bounds \Rightarrow "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
 \Rightarrow generalize to many base and bounds.



16 hardware for segmentation

- 1 tell hardware which segment?

top k bits of addr

2 Memory protection bits.

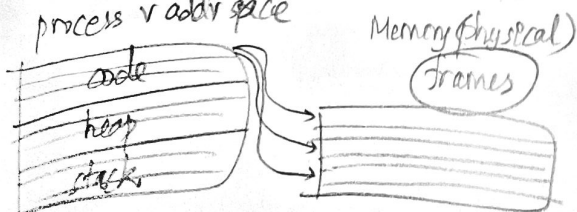
read? write? execute?

problems

- 1 context switch \rightarrow save & restore segdesc off - ors
- 2 non compact memory layouts!

17 Paging

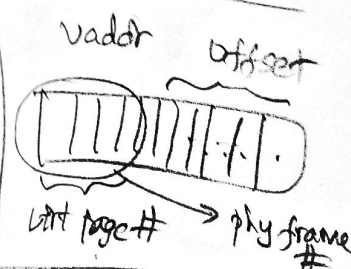
use fixed size chunks. Many of them.



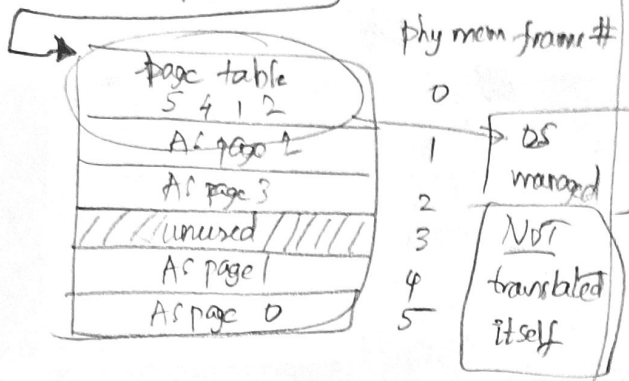
- * can't use dedicated registers any more
- \Rightarrow page table in memory.

Per process data structure

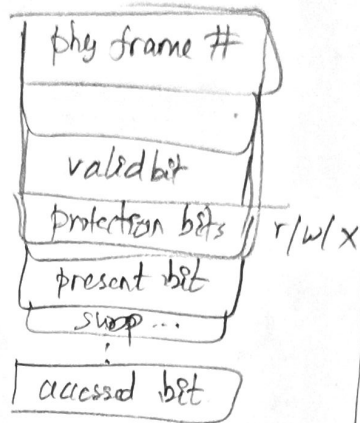
18 Translation



19 example



20 Page table entry



21 translation (hardware)
 $VP\# = (vaddr \& vmask) \gg \text{SHIFT}$

PTE addr // linear pageable
 $= \text{Page Table Base Ptr} + (vp\# \times \text{sizeof (PTE)})$

translate to get phy frame #
 $\text{offset} = vaddr \& \text{offset mask}$

$\text{Phy addr} = (\text{pf\#} \ll \text{SHIFT}) | \text{offset}$

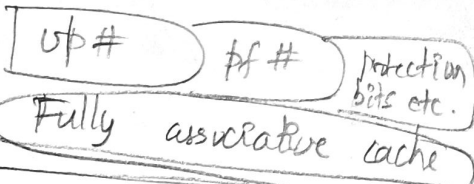
22 translation

check valid, protection, PTE valid.
 Else trap!
 → Minor fault (retrieve shared)
 → Major fault (retrieve from disk)
 → Invalid fault (vaddr not valid)

24 Address translation cache (TLB)

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

26 TLB entry



27 TLB + context switches

Per process data structure of pagetables
 option 1 TLB flush
 option 2 tagged TLB
 - use address space ID

25 Handling TLB miss

hardware managed TLB (Intel x86)
 software handled (use traps) (RISC)

30 Principles applied to entire operating systems!

- Xen
- amazon ec2

29 TLBs make paging feasible

physically indexed
 versus virtually indexed caches

28 smaller page tables?

→ multi level page tables
 page directory + table
 → superpages
 → hybrid segment & paging
 → swap page tables to disk

31 system virtualization

- ① OS assumes it is the most privileged entity.
- ② OS virtualization, why?
 - ⇒ time sharing uses
 - ⇒ unit of software/app deployment — any software & config can be run
- ③ actually properties of both OS & hardware arch matter (together provide isolation)

33 theorem (basic result)

$\text{efficiency virtualized if } S2 \cup S3 \subseteq S1 \}$ FULL virtualization
 → "trap and emulate" virtualization (for privileged instructions)

- sufficient condition,
- unmodified guest OS
- workaround tech

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

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

② {

- cache emulation code
- assist through hardware

② para-virtualization → port before running

layering Xen copy on write ← ③ union filesystems

32 virtual machine monitor

(VMM) 1974

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

3 requirements (popek & goldberg)
of interest

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

34 Containers

⊕ kernel namespaces (access isolation)

⊕ Control groups (resource isolation)
 process network
 mem
 cpu
 mem/blocks

35 xen

① Cpu → ring 1 of
(else trap ring 0)

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

② I/O virtualization

Ring buffers

asynchronous 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.