Last lecture: Memory virtualization

②

Transparency: I think I own memory

Base & bound:

addr translation → adder
+ check bounds

✓ Perf
✓ Hardware complexity

× Flexibility
× utilization

3

Paging

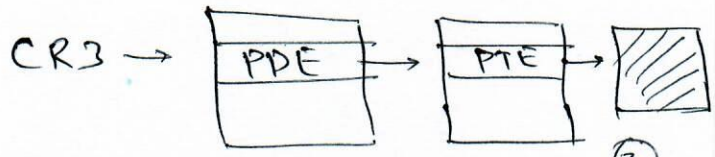
②

- Bits: valid, present, dirty, r/o
- Page table
  - Large ⇒ Hierarchical
- TLB – spatial locality for TLB hits
- Demand paging

Perf ✓
HW Complexity ×

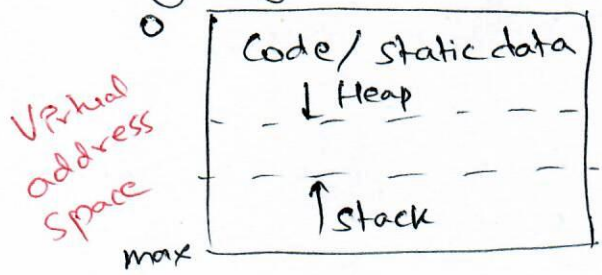
Flexibility ✓
Resource util ✓

HPT

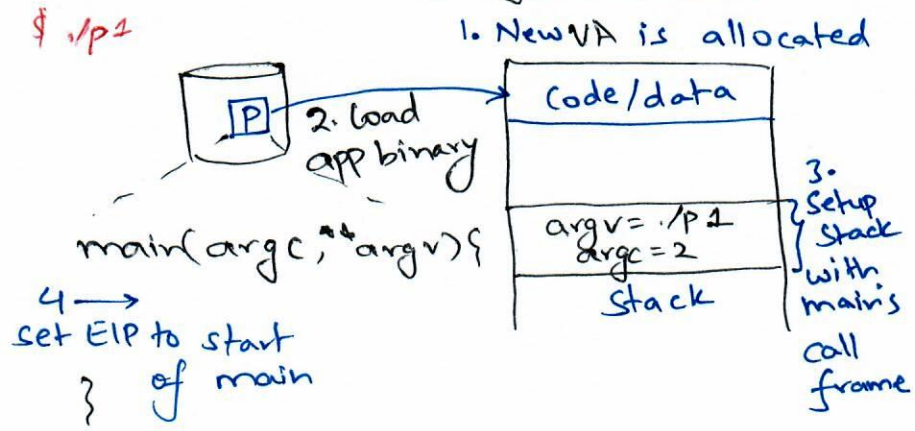


②

Memory layout

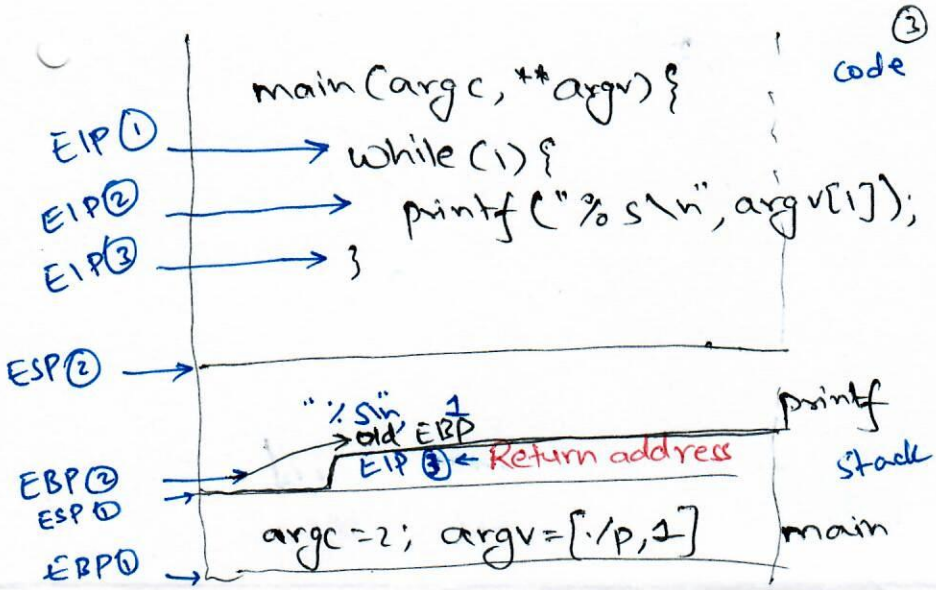


How does a program run? ③



---

③



---

② CPU register (x86)

CR3 → Page table ← Hidden from programs

EIP → Instruction pointer ← Auto Incremented by CPU

ESP → Stack pointer. Top of stack

EBP → Bottom of Current call frame — Where to allocate call frames

managed by compiler

General: eax, ebx, ecx...

using actual hardware registers ②

CPU: [P1 P1 P1 P1] [P2 P2 P2 ...] [P3]

Early batch computers → Time

wait for program to end

↓ Resource utilization

CPU: P1 P1 wasted cycles P1 P1 P1

↓ I/O ↑

I/O is slow

Operator (manual): ①

- Kill run away process
- Prioritize process
  - manage list
- Only used during office hours
- Only way to intervene is to kill a process

P: ①

```
main(argc, **argv) {
    while(1)
        printf("%s\n", argv[1]);
}
```

modifies actual hardware registers

```
$ ./p A &; ./p B &; ./p C &...
```

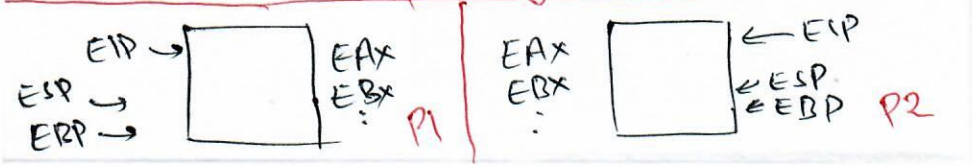
A
B
A
C
B

multiple processes are able to run even though there is single CPU

CPU virtualization!

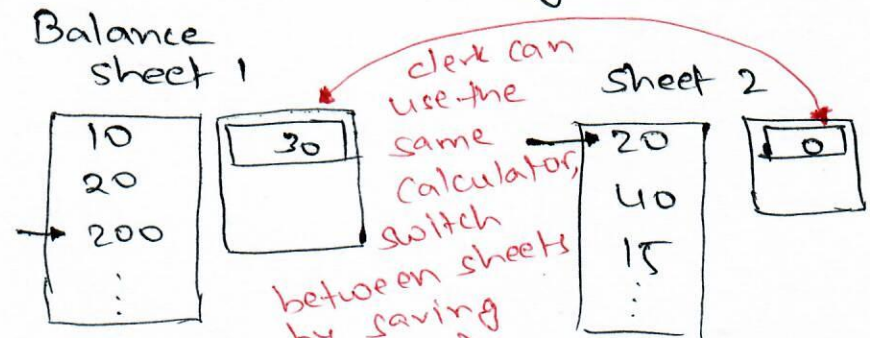
②

Every process THINKS that it owns CPU registers and directly changes them when it runs! Transparency



Bank clerk analogy

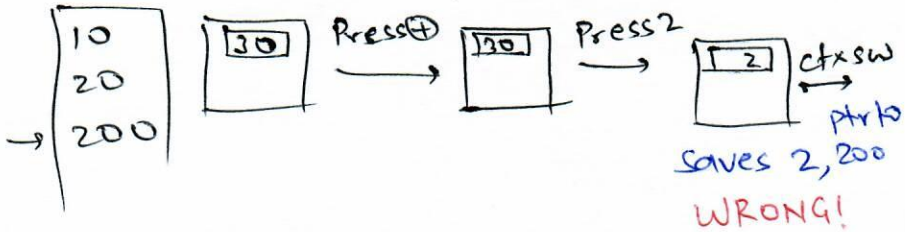
①



next value, calc screen value } sheet context: assume atomic instructions

Context sw can't happen in middle of instruction

②



Saves 2, 200

WRONG!

Sheet context assumes 1 instruction → adding num

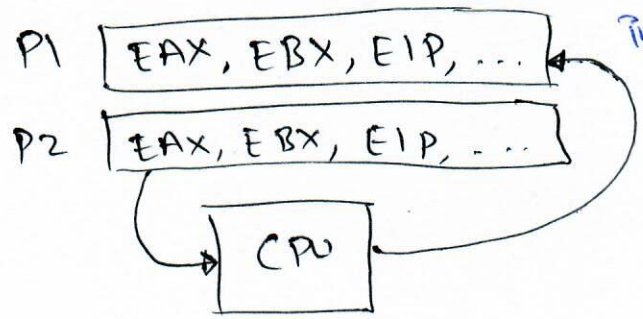
Each instruction → ATOMIC

Context switch

(1)

Process Control block

Can't happen in middle of instruction



Comparison w/ operator

(3)

Flexibility↑

→ Run multiple processes simultaneously

→ Insert high-priority job without rebooting

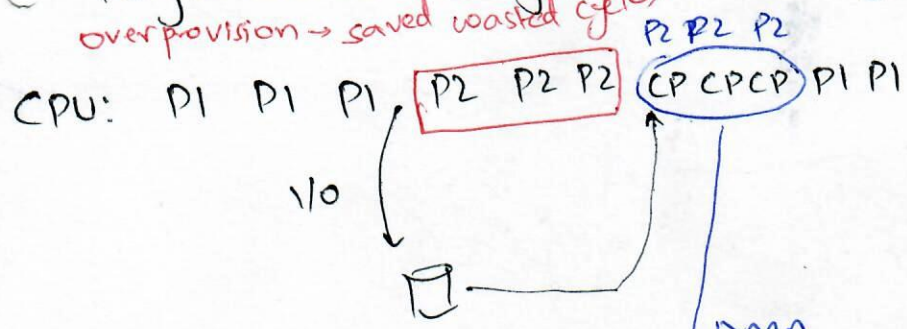
→ Dynamically Kill / spawn processes

→ No need for operator → OS!

Performance benefit

(1)

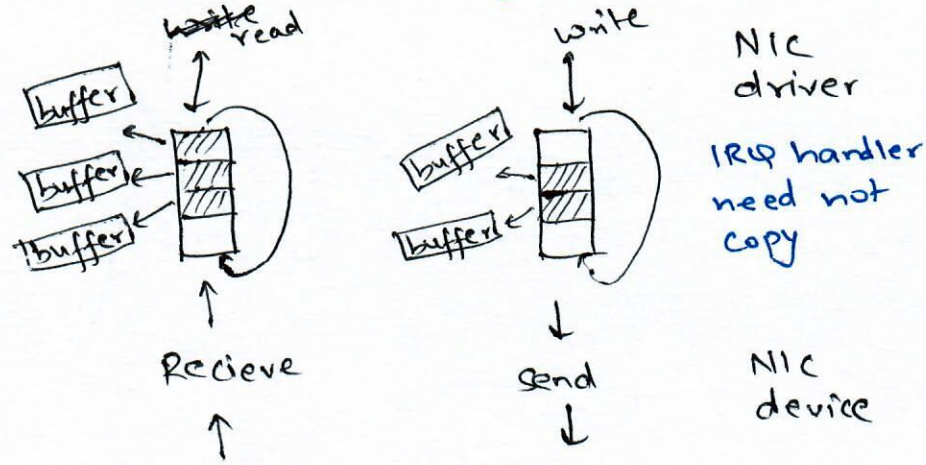
overprovision → saved wasted cycles



Further optimization

Improved resource utilization

Aside: DMA ring buffers ②



Problem: Context switch is light ②

⇒ few registers

Impact on performance is bad.

P1 <> P2 : different address spaces

D-cache, I-cache, TLB misses

↑ flushed in some architectures

Improve perf. ①

- Only change registers
- Don't change address space

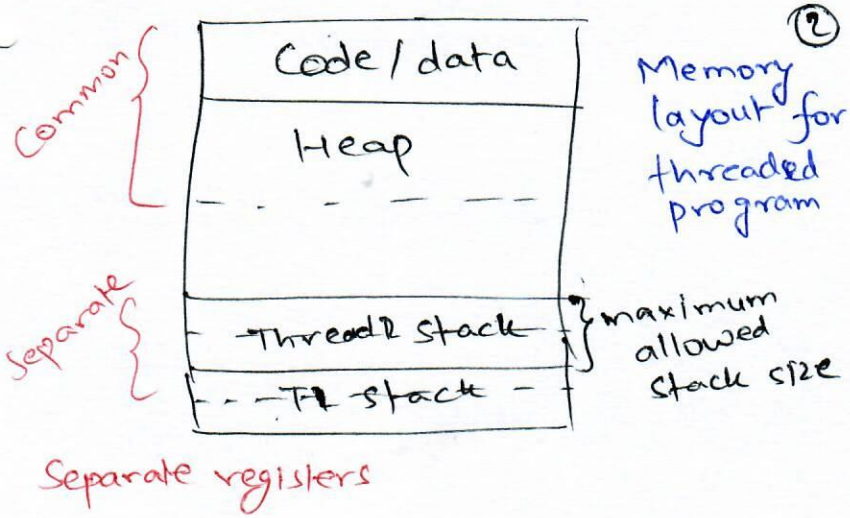
  ⇒ Threads

- T1 T1 T1 T2 T2 T2 T1 T1 T1

  fast

- very effective for I/O heavy processes

②



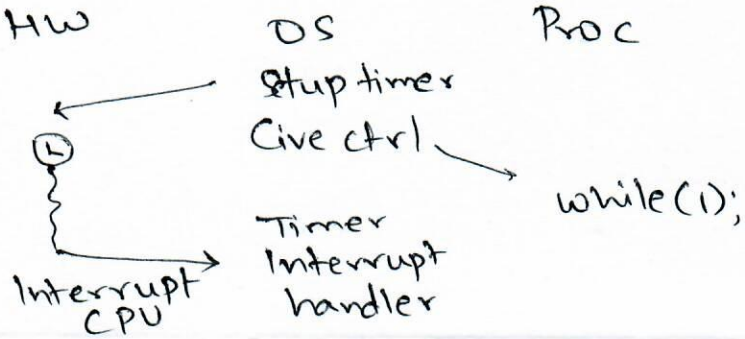
Memory layout for threaded program

Prob

Give control, keep ability to take back control

$\boxed{\text{while (1); Single CPU}}$ ①

Sol^n

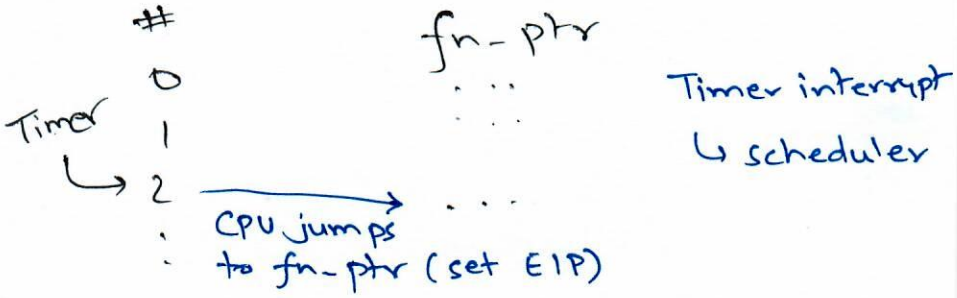


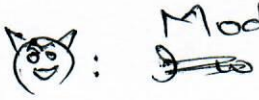
---

which handler? ②

- Interrupt descriptor table

  Set up at boot



 : Modify IDT ②

#	fn-ptr
0	...
1	...
↳ 2	 → does not call sched

Privileged Instruction ⇒ Traps to OS
⇒ Kill proc

---

x86 rings ①

ring 3	User
	Other privileged instⁿ:
	- Disable interrupts
	- Change CR3
ring 0	Kernel

Interesting programs Need to use privileged instructions ②

- Sending data on network
- Reading disk contents (100x heavier than function calls)

⇒ System calls

execute INT N Instruction
↳ IDT