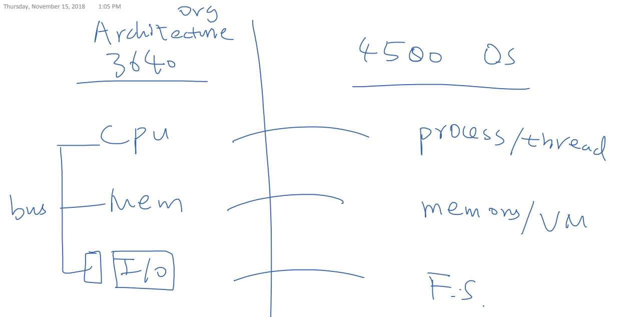
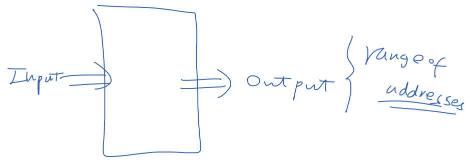
Thursday, November 15, 2018 1:05 PM



1) protection { dual mode (Kernel v.s. user) priviledsed instruction

memory address space

Translation map

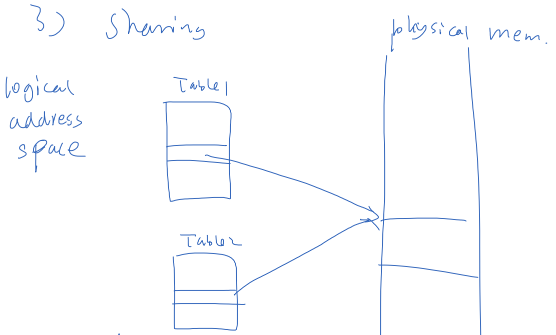


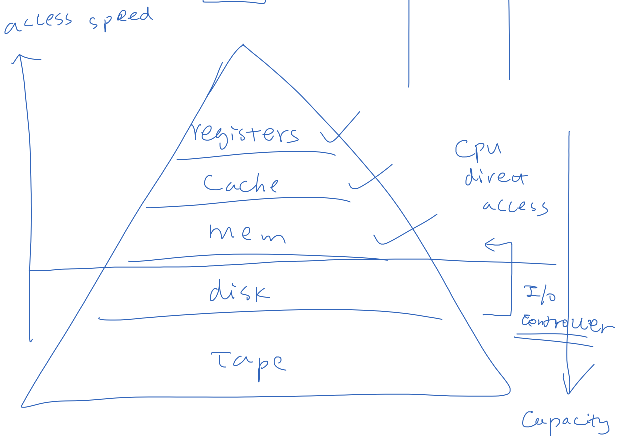
2) Infinite amount of mem (usability)

V.M.

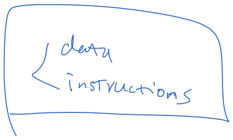
- no concerns on the availability
- exact location.

3) Sharing





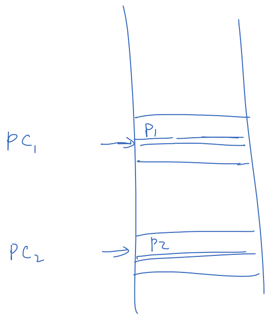
Cpu access

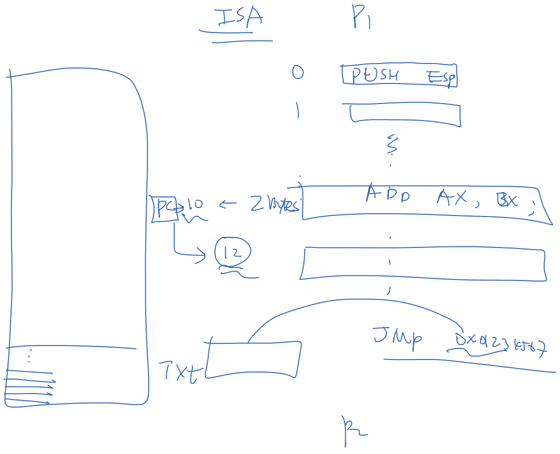


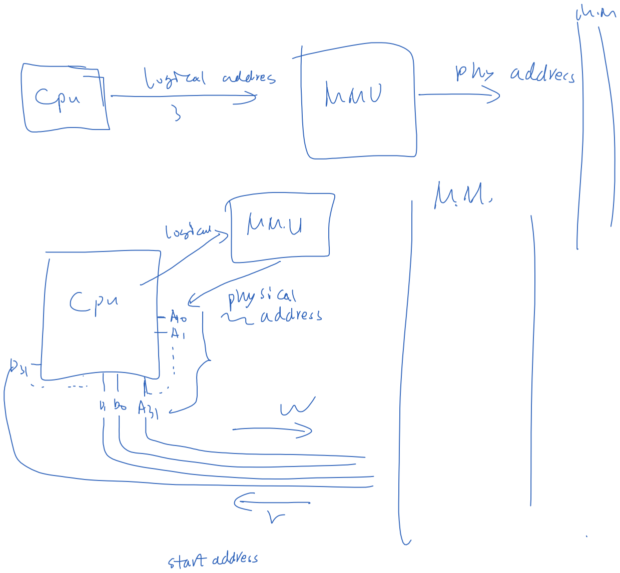
programs ⟸ 1) stored initially on "disk" in the form of "files"

2)

process Instructions & data loaded in mem address space

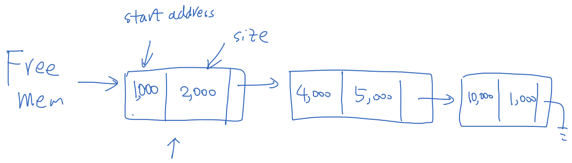






start address

size



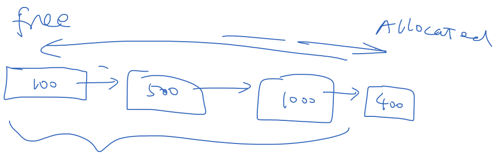
request of 800 bytes

First fit:

Free list: (1800, 1200) → (4,000, 5,000) →

Best: (1000, 2000) → (4,000, 5,000) → (10,800, $\boxed{200}$)   (10,000, 1000)

Worst: (1,000, 2,000) → (4,800, 4,200) → (10,000, 1000)



Total: 2000 bytes

request r: if $1000 < r < 2000$

$r = 1500, \left\} \begin{matrix} 1000 \\ 500 \end{matrix} \right.$

fragmentation ← Contiguous allocation various sizes

↑

Compaction

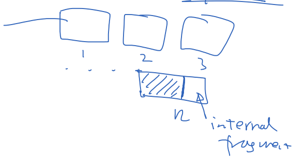
↑

1) remove the constraints

2) make sizes "equal"

$S \neq n.p$

$\left\lceil \frac{S}{p} \right\rceil = n$



$P = 4KB$

~~program~~

process 1	process 2
20KB	32KB
n=5	n=8

process 2 terminates

process 1 arrives,

P1 can use the mem that P2 just used

page size 4KB in logical address space

⇓

frame w/ equal size (4KB) in phy mem.

physical mem 1 GB

$$\frac{1GB}{4KB} = \frac{1K \cdot 1MB}{4KB} = \boxed{256K}$$

$$= \frac{1K \cdot 1K \cdot 4KB}{4KB}$$

$$= \frac{1K \cdot 256 \cdot \cancel{4} \cdot \cancel{1KB}}{\cancel{4KB}}$$

4KB pages size

physical mem 4MB

in total there are

1K frames.

a process size ~~of~~ of 512KB

then it has $\frac{512KB}{4KB} = 128$

pages.

Contiguous Allocation

OS maintains

1) where in the phy mem, & How large

is allocated to which process

?

2) list of free partitions

⇑

<start, size>

Paging

OS will use

1) page table

2) bitmap to represent the free mem

1 bit w/ 0, or 1

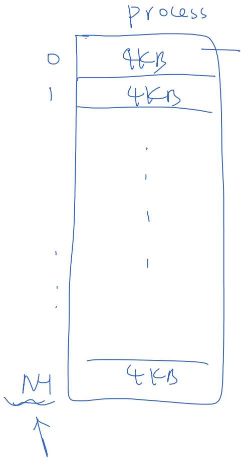
4GB phy mem w/ page size of 4KB

So we need 256K bits

⇙

32K Bytes

17 page



page # 0 ~ N-1

page ~~size~~ size = 4KB

process size = # of pages * page size

if N = 64 then

process size : 4KB * 64

= 256 KB

convert decimal # into binary format

$\log_2 N$

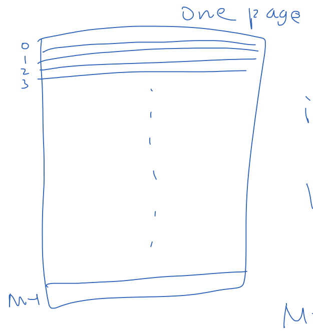
N=64

2^6

6 bits to represent

two page #

0 000000
1 000001
⋮
64-1 = 63 111111



byte addressing

it has M Bytes

$\log_2^M$ bits are needed for offset

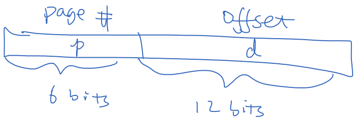
M = 4KB

we need $\log_2^{4K}$

= 12

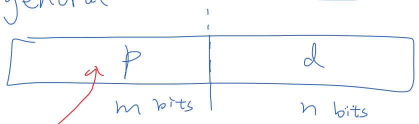
0 ... 0000 0000 0000
1 ... 0000 0000 0001
2 ... 0000 0000 0010
⋮
4095 $2^{12}-1$ ... 1111 1111 1111

Logical address



2^{p+d} + d

In general



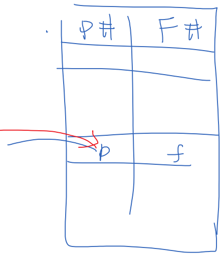
logical address

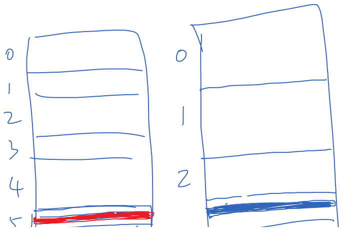
1) page size = 2^n

32bits address

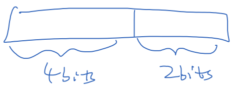
- if n = 12 ⇒ page size = 4KB first 20bits
  n = 10 ⇒ page size = 1 KB first 22bits

MMU Queries Page table

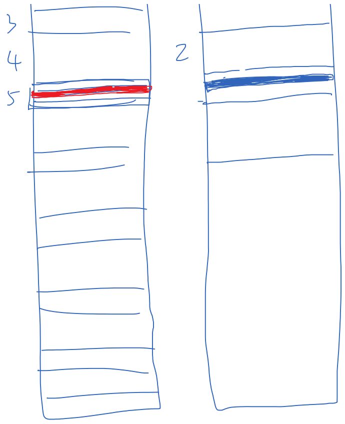




1) page size = 4Bytes



pages = 16



4 bits 2 bits
# pages = 16

6 bits address

0101 | 01

p = 0101 d = 01
(5) (1)

2) page size = 8 bytes

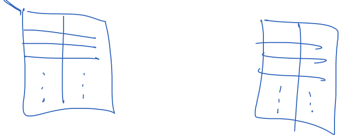
010 | 101

p = 010 d = 101
(2) (5)

OS's view on mem
1) used mem ←
2) free mem

bit map

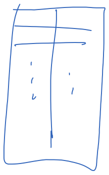
page table 1 PT2 ...

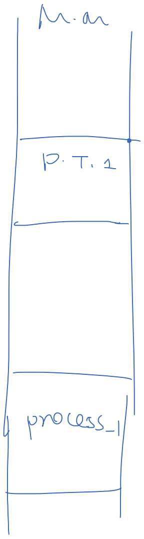


☆ page table is stored in mem.

access to the page table
— Access to mem
— " " register. ✓

Cpu → MMU
↓
page Table (Mem)

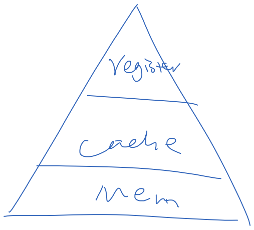




1) access P.T. to get frame#
using logical address (p)

2) access M.M according to the physical address

2.8



x (1 cpu cycle)

y

z (several)

X < Y < Z

if we could move data into Cache then

~~at~~ reference time

is $Y+Z < 2\cdot Z$

if $\underset{\sim}{Y}$ is 10% Z

the improvement is

$\cancel{\frac{2-1.1}{2}\% = 45\%}$

$$\frac{2\cdot Z - 1.1 Z}{2\cdot Z} = \underset{\sim}{45\%}$$

Hierarchical P.T..

process size: } determine the size of page table

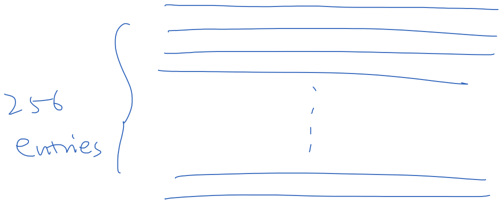
page size:

page size = 4KB

process size = 1 MB

the # of pages this process has

$$= \frac{1\text{ MB}}{4\text{KB}} = 256$$



if ~~we us~~ the system uses 4 Bytes for each page Table entry,

then page Table would need

$256 \times 4\text{Bytes} = \boxed{1\text{KB}}$

to store.

P_1

if process is 64MB

$$\frac{64\text{MB}}{4\text{KB}} = 16\text{K pages},$$

the size of page table is $16\text{K} * 4 = \underline{64\text{KB}}$

the size of page table is $16K * 4 = \boxed{64KB}$

p_2

p_3 : 128KB

p_6 : 57KB

logical wise # of pages is $2^{\boxed{22}}$ ← ?

page size = $2^{\boxed{10}}$ ←

$+$

= 4M pages $\overset{//}{32}$

then page Table would need

4M x 4Bytes / P.T.E = 16MB

1KB per page

$$\frac{16MB \text{ (page Table)}}{1KB \text{ (per page size)}}$$

= 16K pages

we need (2^{14} = 16K)

14 bits to represent the p#

Exercise: page size 4KB

32bit system

if we use two level Hierarchical str

to manage the page table,

What is length of P1 , P2 and d

1) if page size is 4KB then

# of bits for offset is B

A) 10

[✓ B) 12]

C) 14

$\boxed{2^{12} = 4K}$

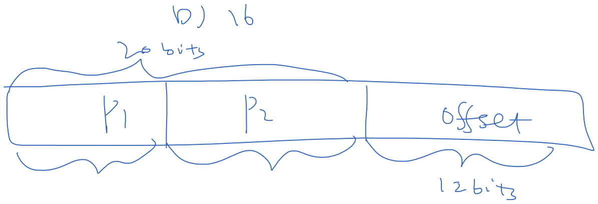
1 KB

offset
bits

ucture

b) 16

2)



2^{20} = 1 M pages

So the page table size is

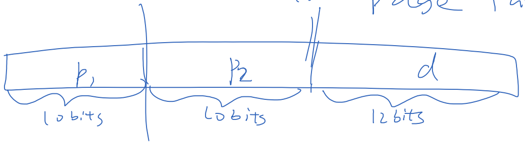
1 M × 4 Bytes/P.T.E. = 4MB

to store the page table in pages

$\frac{4MB}{4KB}$ = 1 K pages.

We need 10 bits (2^{10} = 1

to represent each page of the page table



$z + z + z$

Search in the outer page table

Inner P.T.

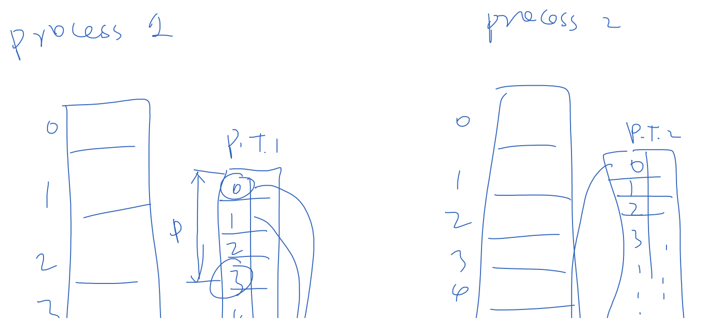
m.m for data/Ins

$y + z$

Cache access time.

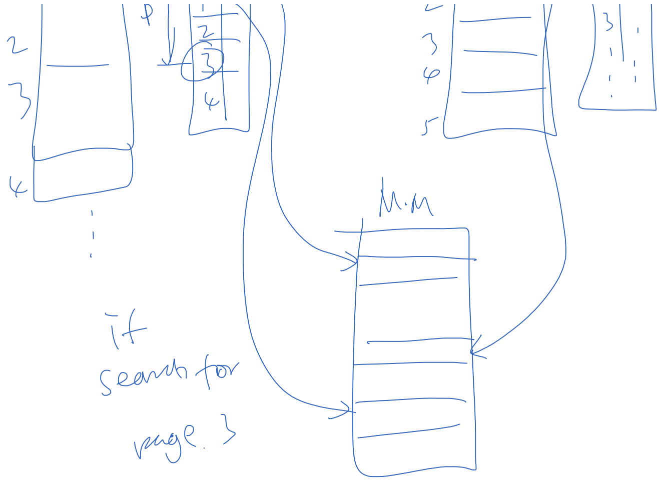
mem access time.

speed up $2 \cdot z \to y$



k)

tructions.



if
search for
page 3