Mem Mgmt

- How to share main memory?
- Adv/Disadv of static relocation?
- dynamic
- H/W interactions

Motivation

- Simple uniprocessor env.
  early batch, PCs
  why bad?
  - 1 process @ a time, protection

=> Multiprogramming: goals
  sharing: > 1 process coexist in mem
  transparency: to process, looks like large private memory
  works even w/o others running
  protection: cannot corrupt OS, other processes
  cannot read from others: why?
  flexible: allow proc to do what it wants
  efficiency: should not waste too many cycles doing this
  should not waste memory either

Static relocation:

- Static relocation: can run anywhere in memory
  modify addresses statically at load time

- compile all as if loaded @ address 0
- loader (which gets process running) gets through and changes addresses @ run time

Positives: simple, no h/w required
Negatives: 1. fragmentation, 2. may not be able to increase space while running
3. protection: non-existent
Dynamic Relocation

Process \[ \Rightarrow \text{compiled as if @} \] \[ \Rightarrow \text{H/W translates addresses \rightarrow physical addresses} \]

\[ \text{CPU} \rightarrow \text{L/S} \rightarrow \text{Mem} \rightarrow \text{Phys Mem} \]

\[ \Rightarrow \text{all mem P can address: address space} \]

Address Space

H/W needed

Operating modes:

- Privileged: in kernel
  - when trap into OS (syscall)
  - can access all of mem
- User: when process runs
  - provides logical view of mem

How done? Base + Bounds registers

- Base: start location
- Bounds: max location

Implementation:

Translation: on every mem access (LD/ST, itetch)

compare logical to bounds reg

if \( LA > \text{Bounds} \) error

To get \( PA \):

\[ \text{PA} = \text{LA} + \text{Base} \]

Key to protection:

H/W limits access of process to memory
Context Switch

Add Base + Bounds to PCB

Steps during context switch:
- privileged mode
- save base + bounds of old
- restore new
- change to user mode + jump to new process

Protection

(user cannot change base + bounds
user cannot arbitrarily become "privileged")

Key: Modes provided by HW

Advantages:
- supports dynamic relocation
- protection
- simple: base + bounds
- fast

Disadvantages

> Allocation of process contiguuous in real mem

> Fragmentation: can't alloc new process

> Wasteful

> Swapping:
  - must allocate entire regime

> Slowing is hard

Running two copies of Netscape
  - could share code
Segmentation and Paging

+/- of segmentation
+/- of paging
combine?
speed up?

Segmentation

> Divide A$ into "logical" segments
  Each has base/bound
  Per segment: read/write bits, protection (before, all in one)

> How to designate?
  > Part of logical address
    
    \[
    \begin{array}{c}
    01 \\
    \# \\
    \# \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
Managing Processes

> Creation
  Find contiguous space per segment
  Fill in base/bounds

> Mem alloc when no contiguous space
  Compact memory (move segments, update bases)
  (Swap segment to disk?)

> Ctxt switch
  Segment Table \(\rightarrow\) PCB

> Exit
  Return to free list

Pros/Cons

+/ Diff. protection/segment
  (read-only for code)
  Sharing is easier for some segments
  Can relocate segment easily
  Enables sparse, large address space

-\/ Still have to find contiguous mem
  \(\Rightarrow\) External fragmentation: wasted memory
Paging

Divide mem into fixed-sized pages

typical page size: \(4K, 8K\)

Translation

\[
\begin{array}{c|c}
\text{Page number} & \text{offset} \\
\hline
19 & 13 \\
\end{array}
\]

\(8K\) page

Xlate: \underline{Page table}
\(\rightarrow\) just a data structure

\[
\text{PPN} = \text{lookup}(\text{PPN})
\]

> One "page table" per process

Base addr (PPN) + protection bits

> How many entries in table? \(2^{17}\) entries

BIG
Example

Advantages

+ Fast to allocate / free
  alloc / keep free list / grab first
  (no best fit needed -> all fit!)
  free / add page to free list
  (no sort needed)

+ Easy to swap to disk
  page size ~ block size
  just move needed pages to disk

Disadvantages

+ Every load -> 2 loads, every PC fetch -> 2 fetches
  can't keep entire page table in memory!
  (must keep in main memory)
  MMU: base addr of table

+ Huge! Lots of mem required
  Simple: entry for all pages, used or not
  Better: base/bounds

+ Fragmentation: internal (w/ buddy) (waste)
  Larger page => fewer entries => more internal frag

Why?
Combine Paging + Segmentation

Structure: segment is a logical unit of AS
- code, data, stack: vary in size, large
- each seg. is made up of many page)

2-level lookup
- Page-table per segment
- Base (real addr) + bound (size) per table
- Use to check for valid page

LA: 3 components

<table>
<thead>
<tr>
<th>Seg#</th>
<th>Page#</th>
<th>Page offset</th>
</tr>
</thead>
</table>

Plus:

+ Segments: Sparse addr space
  - no wasted page table space

+ Paging
  - External fragment problem gone
  - easy to grow

+ Both: flexible sharing
  - seg/page/both (code)
Disadvantages:

- Internal frag. inc
  (more segs => more pages)

- Overhead
  - tables in main memory
  - 1 or 2 extra loads / ref

- Large page tables
  - don't want to allocate PT contiguously

> TLB
> Inverted PT
> Paging Page Tables
Advantages of Paging

- Fast allocation, freeing
  - no search, list ops easy
- Easy to move back and forth from disk

Problems

- Page table is too big
  - must keep in main memory, not MMU
    - MMU might hold info about where processes page table is
- One entry per page
  - base/bounds may help, but not w/ stack, heap
- Internal fragmentation
  - how much waste?

Solution: Combine paging w/ segmentation

- Segments are logical units (code, data, stack)
- Segment consists of a bunch of pages

2-levels of mapping

- Page table per segment
- Base (real address) + bound (size) for each

![Diagram]

- Segment Table
- Seg #, Page Table
- Offset
- PP #, Offset
- PA
Advantages

Segmentation: sparse address spaces

Paging: no worry of external fragmentation
easy to grow w/o reshuffling

Both: can share page/segment/page tables between processes

Minus

(slight increase in waste due to internal fragmentation)
mem. overhead: 1 or 2 references per access
site overhead: large page tables still a problem

Multi-level page tables

Before: entire PT for segment must be contiguous in memory

Physical

outer page  Inner page  offset

rest of bits  10 bits  12 bit

inner-level:
1 page/pt
4 bytes/entry

register
base
bound

outer

2^10 2^2 = 2^12

 => how many bits for outer page?
Inverted Page Tables

Idea: why have so much of VA defined by page table? instead, have table that corresponds to size of physical memory

Translation: \( f(V_P^m) \rightarrow P_P^m \)

Could be any data structure

Could keep list

why bad?

efficient lookup: hash on VP#

bad: sharing is difficult: one VA per PA (typically)

Slow: TLB

need: speed up accesses w/ H/W support

use H/W cache of translations

\[
\begin{array}{c|c}
V_P^m & \text{offset} \\
\hline
\end{array}
\]

in TLB \( \rightarrow P_P^m + \text{offset} \rightarrow PA \)

not?

\[
\begin{pmatrix}
\text{do full translation} \\
\text{update TLB}
\end{pmatrix}
\]

who does this?

H/W or S/W

Size: usually small

Policies: fully associative

(32, 64 entries)