int spaceInString (char *ct) {
    static char s[n];
    int i = 0;
    while (*ct) {
        s[i++] = *ct++;
        if (ct[0] == '
') return i;
    }
    return i;
}
Relocation and Protection

Multiple Programs

Single Program

Relocation
No Protection
Relocation and Protection

All programs see a 0-based address space.

Add all addresses to a base register (BR) in hardware to relocate.

Handles everything.

System code handles switching.

All or nothing protection.

Add length register (LR).

Hardware performs Base and Length bound check.

Signal exception on access violation.

Add privilege mode (S or U).

U-mode can only access \([B, B+L-1]\), can't modify BR or LR.

S-mode can access all memory, and BR, LR.

\[ \text{Base Register Addressing} \]

- Relocation
- No Protection

- Relocation
- Protection
Reasons:
-- Run two of the same program (share code)
-- Share data between programs (e.g., databases)

Cannot use 1 BR, LR per process because relocated process is not contiguous

Cannot use all or nothing protection

Use multiple BR, LR registers
Use multiple privileges for each mapping
R, W, X
Segments:
--Contiguous variable-sized set of memory locations
--Uniform privileges
--Unit of relocation, protection, and sharing
Segmentation Example

Ex: 32-bit logical address space
12-bit segment number ⇒ 4096 segments
20-bit offset ⇒ max segment size = 1 Mbyte

Process 0:

Seg 0
- - - 0x00100000
Seg 1
  DLL
- - - 0x00200000
Seg 2
- - - 0x00300000
Seg 3
- - - 0x00400000
... ...

Process 1:

Seg 0
- - - 0x00100000
Seg 1
- - - 0x00200000
Seg 2
- - - 0x00300000
Seg 3
  DLL
- - - 0x00400000
... ...

Physical Memory

Seg Table 0 Base 0x00010000
Seg Descriptor

Seg Table 1 Base 0x00020000
Seg Descriptor

Prot Base X0 0xOBAD1000
Length 0x1000

Prot Base X0 0xOBAD1000
Length 0x1000

0x0010AEEF

0xOBADBEEF

0xOBADBEEF

DLL Base 0xOBAD1000

DLL Code

0x10000
Resource Management

Given a name, where should the name reside in memory? or, "What parts of a name space should I keep in memory?"

Problems:

1. Physical memory is finite (i.e., small)
2. Multiple name spaces may not fit in memory
3. Each name space may not fit in memory

Solution: Virtual Memory

Keep most of name spaces in secondary storage (disk) and move "important" portions into physical memory automatically.

Solve 2 problems:

- Mapping (relocation)
- Management

(very similar to hardware caches)
As segments come and go, the storage is "fragmented"; therefore, at some point segments must be moved around to compact storage ⇒ "burping the memory"
Page Addressing

Virtual Address Space 0
(LNS 0)

Virtual Page Numbers

Virtual Address Space 1
(LNS 1)

Page Table 0

Page Table 1

Physical Memory

Physical Frame Numbers

Secondary Storage (Disk)
Paged Addressing

Translation Lookaside Buffer (TLB) Small, fully associative cache of PTEs (≈ 64 entries)

Physical Address

Dirty PFN or DPN

PFN Offset

VPN

Virtual Page # Offset

Virtual Address
## Caching vs. Paging

<table>
<thead>
<tr>
<th></th>
<th>Caching</th>
<th>Paging</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Size</strong></td>
<td><strong>Cache Block</strong> ~ 16 - 128 bytes</td>
<td><strong>Page</strong> ~ 4K - 64K bytes</td>
</tr>
<tr>
<td><strong>Miss Rate</strong></td>
<td><strong>Cache Miss Rate</strong> 1 - 20%</td>
<td><strong>Page Miss Rate</strong> 0.00001 - 0.001%</td>
</tr>
<tr>
<td><strong>Hit Time</strong></td>
<td><strong>Cache Hit Time</strong> 1 cycle</td>
<td><strong>Page Hit Time</strong> ~ 100 cycles</td>
</tr>
<tr>
<td><strong>Miss Penalty</strong></td>
<td><strong>Cache Miss Penalty</strong> ~ 100 cycles</td>
<td><strong>Page Fault</strong> ~ 10⁶ cycles</td>
</tr>
<tr>
<td><strong>Organization</strong></td>
<td>Direct-Mapped, Set-Associative, Fully Associative</td>
<td>Fully Associative</td>
</tr>
<tr>
<td><strong>Write Hit Policy</strong></td>
<td>Write-through or Write back</td>
<td>Write back</td>
</tr>
<tr>
<td><strong>Write Miss Policy</strong></td>
<td>Write allocate or Write no-allocate</td>
<td>Write allocate</td>
</tr>
</tbody>
</table>
Page Frame Management

- Allocate some number of page frames to each process.
- Maintain free page list.
  Evict pages when free pages falls below "low watermark".
- Evict pages from a process using LRU replacement policy.
- If dirty bit is clear, don't copy page back to disk.

How many page frames should each process get?
Monitor page fault frequency for each process.

- Try to keep working set of each process in memory
  - Page fault frequency above some upper limit
    \[\Rightarrow\text{Increase page frame allocation.}\]
  - Page fault frequency below some lower limit
    \[\Rightarrow\text{Decrease page frame allocation.}\]
- "Swap out" entire process if there are insufficient page frames for working set.