Review 1/1

2 Approaches to Writes

- **Write Through**: Write twice, current level and next level down
- **Write Back**: Write only to current level, write dirty block to next level down only on miss

Reduce Miss Penalty?

- Add a (L2) cache

Manage memory-disk transfers?

- Explain as cache: block = page, miss = page fault, write back, fully associative, LRU...
- Included protection as bonus, now critical

Outline

- Sharing/Protecting Memory by Swapping
- Sharing/Protecting Memory by Base/Bounds
- Weakness of Base/Bounds Protection
- Administrivia, “Advice on courses”
- Virtual Memory/Paging
- Problems and solutions of Virtual Memory
- Conclusion

Review: C memory allocation

<table>
<thead>
<tr>
<th>Address</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^{32}-1)</td>
<td>Stack space for saved procedure information</td>
</tr>
<tr>
<td>$sp$</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>$gp$</td>
<td>Global pointer</td>
</tr>
</tbody>
</table>

Space for saved procedure information

- Explicitly created space, e.g., malloc(); C pointers
- Variables declared once per program
- Program
Memory Allocation in Reality

Prior schemes assumed 1 process is running; reality is many processes

Even a single person using:
- Browser
- Compiler
- Mail
- ...

(For now, ignore I/O split of address space)

“Sharing” Solution #1 (earliest)

- Copy other Users (processes) to Disk when not in use
- When time to run a new process, swap new user and old user
- Protection: only 1 user at a time, so OK; (but OS not protected)
- Problem: slow to swap processes between memory and disk

“Sharing” Solution #2: Base/Bounds

- Add Hardware to protect users from each other so that must not always swap to disk to share machine
- Add special registers:
  - Bound Register: address must fit inside this value
    - Also called “Protection Register”
  - Base Register: added to each memory address
    - Also called “Relocation Register”

Base/Bounds mapping Function

- Real memory address:
  - if Program Address $\geq$ Bound ADDRESS ERROR
  - if Program Address $<$ Bound $\Rightarrow$ Base + Program Address

- Note:
  - Allows safe sharing without always swapping to disk
  - $\$base, $\$bound$ are special registers, like $\$epc$, NOT like $\$0, ..., $\$31
1: What if need to grow size of Process B?
° If lucky, space above process available

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

2: What if need to grow size of Process B?
° If unlucky, must copy to create space

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

3: What if need to grow size of Process B?
° If unluckier, must swap to disk

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Danger of Base/Bounds Sharing Scheme?
° Lots of free memory, but in small fragments ⇒ too small to be useful

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Address

User C
$base$
User B
$base+$ $bound$
User A

0

OS

Enough space for User D, but discontinuous so cannot use
° Called “Memory Fragmentation” Problem

• Happens when processes quit too

° Base/bounds HW will not solve
Administrivia
- Project 5: Due 4/14: design and implement a cache (in software) and plug into instruction simulator.
- Next Readings: 5.1 (skip clocking), 5.2, 4.5 (pages 230-240), 4.6 (pages 250-256, 264), 4.7 (pages 265-273)
  - How many lectures to cover: 2? 3?
- 9th homework: Due 4/14 or 4/16? 7PM
  - Exercises 7.35, 4.24
  - Vote on leaving on Wed vs. delaying to Friday (only when there is a project due on Wed)
  - Wed: TA office hours; Fri, spread 61C load

Administrivia: General Course Philosophy
- Take variety of undergrad courses now to get introduction to areas
  - Can learn advanced material on own later once know vocabulary
- Who knows what you will work on over a 40 year career?

Administrivia: Courses for Telebears
- General Philosophy
  - Take courses from great teachers!
  - HKN ratings; > 6 very good, < 5 not good
  - hkn.eecs/toplevel/coursesurveys.html
- Top Faculty / Course (may teach soon)
  - CS 150 logic design Katz 6.2 F92
  - CS 152 computer Patterson 6.7 S95
  - CS 164 compilers Rowe 6.1 S98
  - CS 169 SW engin. Brewer 6.2 S98
  - CS 174 combinatorics Sinclair 6.1 F97
  - CS 186 data bases Wang 6.2 S98

If many good teachers: My recommendations
- CS169 Software Engineering
  - Everyone writes programs, even HW designers
  - Often programs are written in groups ⇒ learn skill now in school (before it counts)
- EE122 Introduction to Communication Networks
  - World is getting connected; communications must play major role
- CS162 Operating Systems
  - All special-purpose HW will run a layer of SW that uses processes and concurrent programming; CS162 is the closest thing
If many good teachers: Courses to consider

° E190 Technical Communication
  • Talent in writing and speaking critical for success
  • Now required for EECS majors

° CS 150 Lab Hardware Design
  • Hands on HW design

° CS 152 Design a Computer

° CS 188 Understand databases
  • Information more important now than computation?

Administrivia: Courses for Telebears

° Remember:
  • Teacher quality more important to learning experience than official course content
  • Take courses from great teachers!

Inspiration for Solution #3

° Provide Hardware to allow discontinuous memory to look continuous to new process

Solution #3 called Virtual Memory, or Paging

° Will specify new HW mapping function

Mapping Virtual Memory to Physical Memory

° Divide into equal sized chunks (about 4KB)
  ° Any chunk of Virtual Memory assigned to any chunk of Physical Memory (“page”)

64 MB

Virtual Memory

Stack

Heap

Static

Code

Physical Memory

User A

User B

User C

OS

$base

$bounf

$base+
Virtual Memory Mapping Function

- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
</table>

- Use table lookup ("Page Table") for mappings: Page number is index
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number = PageTable[Virtual Page Number]
  (P.P.N. also called "Page Frame")

Notes on Page Table

- Solves Fragmentation: all chunks same size, so all holes can be used
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some pages to disk
    - (Least Recently Used to pick pages to swap)
- OS must reserve "Swap Space" on disk for each process
- Each process has own Page Table
- Will add details, but Page Table is essence of Virtual Memory

Virtual Memory Problem #1

- Not enough physical memory!
  - Only, say, 64 MB of physical memory
  - N processes, each 4GB of virtual memory!
  - Could have 1K virtual pages/physical page!
- Spatial Locality to the rescue
  - Each page is 4 KB, lots of nearby references
  - No matter how big program is, at any time only accessing a few pages
  - "Working Set": recently used pages

Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
- OS maps to disk if Not Valid

<table>
<thead>
<tr>
<th>V</th>
<th>A.R.</th>
<th>P. P.N.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val: Access Rights</td>
<td>Physical Page Number</td>
<td></td>
</tr>
</tbody>
</table>

- If valid, also check if have permission to use page: Access Rights may be Read Only, Read/Write, Executable
Virtual Memory Problem #2

- Map every address ⇒ 1 extra memory accesses for every memory access
- Observation: since locality in pages of data, must be locality in virtual addresses of those pages
- Why not use a cache of virtual to physical address translations to make translation fast? (small is fast)
- For historical reasons, cache is called a Translation Lookaside Buffer, or TLB

Typical TLB Format

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access Rights</th>
</tr>
</thead>
</table>

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- Ref: Used to help calculate LRU on replacement
- Dirty: since use write back, need to know whether or not to write page to disk when replaced

What if not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
- Option 2: Hardware traps to OS, up to OS to decide what to do
- MIPS follows Option 2: Hardware knows nothing about page table format

Virtual Memory Problem #3

- Page Table too big!
  - 4GB Virtual Memory ÷ 4 KB page ⇒ ~ 1 million Page Table Entries
  - 4 MB just for Page Table
- Variety of solutions to tradeoff memory size of mapping function for slower when miss TLB
  - Make TLB large enough, highly associative so rarely miss on address translation
  - CS 162 will go over more options and in greater depth
Page Table Shrink #1:

- Page Table Entry has Virtual Page Number and Physical Page Number (as in TLB entry)
- Keep only 1 Page Table Entry per Physical Page vs. per Virtual Page
- Find match on miss via hash function
- Size Reduction
  - e.g., 64 MB \( \div \) 4 KB \( \Rightarrow \) 16 K entries \( \times \) 4 Bytes
  - 64 KB vs. 4096 KB for normal page table

Page Table Shrink #2:

- Single Page Table

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

- Multilevel Page Table

<table>
<thead>
<tr>
<th>Super Page No.</th>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 bits</td>
<td>10 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

- Only have second level page table for valid entries of super level page table

2-level Page Table

- Virtual Memory
  - Stack
  - Heap
  - Static
  - Code

Super Page Table

- 2nd Level Page Tables

Physical Memory

64 MB

Space Savings for Multi-Level Page Table

- If only 10% of entries of Super Page Table have valid entries, then size is roughly 1/10-th of single level page table
  - Exercise 7.35 explores exact size
Virtual Memory allows protected sharing of memory between processes with less swapping to disk, less fragmentation than always swap or base/bound.

- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well.
- TLB to reduce performance cost of VM.
- Need more compact representation to reduce memory size cost of simple 1-level page table (especially 32-\(\rightarrow\) 64-bit address)

Next: Introduction to processors design.

And in Conclusion..