Lecture 14: Virtual Memory

• Topics: virtual memory  (Section 5.4)

• Reminders: midterm begins at 9am, ends at 10:40am
Virtual Memory

• Processes deal with virtual memory – they have the illusion that a very large address space is available to them

• There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk

• Thanks to locality, disk access is likely to be uncommon

• The hardware ensures that one process cannot access the memory of a different process
Address Translation

- The virtual and physical memory are broken up into pages

8KB page size

Virtual address

virtual page number

13

page offset

Translated to physical page number

Physical address
Memory Hierarchy Properties

• A virtual memory page can be placed anywhere in physical memory (fully-associative)

• Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)

• A page table (indexed by virtual page number) is used for translating virtual to physical page number

• The memory-disk hierarchy can be either inclusive or exclusive and the write policy is writeback
TLB

- Since the number of pages is very high, the page table capacity is too large to fit on chip

- A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses

- A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!

- A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory wastage
TLB and Cache

- Is the cache indexed with virtual or physical address?
  - To index with a physical address, we will have to first look up the TLB, then the cache → longer access time
  - Multiple virtual addresses can map to the same physical address – can we ensure that these different virtual addresses will map to the same location in cache? Else, there will be two different copies of the same physical memory word

- Does the tag array store virtual or physical addresses?
  - Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present
Virtually Indexed Caches

- 24-bit virtual address, 4KB page size → 12 bits offset and 12 bits virtual page number
- To handle the example below, the cache must be designed to use only 12 index bits – for example, make the 64KB cache 16-way
- Page coloring can ensure that some bits of virtual and physical address match

![Diagram showing virtually indexed cache and data cache with page coloring.](image-url)
Cache and TLB Pipeline

Virtually Indexed; Physically Tagged Cache
Superpages

- If a program’s working set size is 16 MB and page size is 8KB, there are 2K frequently accessed pages – a 128-entry TLB will not suffice.

- By increasing page size to 128KB, TLB misses will be eliminated – disadvantage: memory wastage, increase in page fault penalty.

- Can we change page size at run-time?

- Note that a single page has to be contiguous in physical memory.
Superpages Implementation

• At run-time, build superpages if you find that contiguous virtual pages are being accessed at the same time

• For example, virtual pages 64-79 may be frequently accessed – coalesce these pages into a single superpage of size 128KB that has a single entry in the TLB

• The physical superpage has to be in contiguous physical memory – the 16 physical pages have to be moved so they are contiguous
Ski Rental Problem

- Promoting a series of contiguous virtual pages into a superpage reduces TLB misses, but has a cost: copying physical memory into contiguous locations.

- Page usage statistics can determine if pages are good candidates for superpage promotion, but if cost of a TLB miss is $x$ and cost of copying pages is $Nx$, when do you decide to form a superpage?

- If ski rentals cost $20 and new skis cost $200, when do I decide to buy new skis?
  
  ➢ If I rent 10 times and then buy skis, I’m guaranteed to not spend more than twice the optimal amount.
Protection

• The hardware and operating system must co-operate to ensure that different processes do not modify each other’s memory

• The hardware provides special registers that can be read in user mode, but only modified by instructions in supervisor mode

• A simple solution: the physical memory is divided between processes in contiguous chunks by the OS and the bounds are stored in special registers – the hardware checks every program access to ensure it is within bounds
Protection with Virtual Memory

• Virtual memory allows protection without the requirement that pages be pre-allocated in contiguous chunks

• Physical pages are allocated based on program needs and physical pages belonging to different processes may be adjacent – efficient use of memory

• Each page has certain read/write properties for user/kernel that is checked on every access
  ➢ a program’s executable can not be modified
  ➢ part of kernel data cannot be modified/read by user
  ➢ page tables can be modified by kernel and read by user
Alpha Paged Virtual Memory

- Each process has the following virtual memory space:

```
seg0 → kseg ← seg1
Reserved for User text, data Reserved for kernel Reserved for page tables
```

- The Alpha uses a separate instruction and data TLB

- The TLB entries can be used to map pages of different sizes
Example Look-Up

Virtual page abc → Physical page xyz

If each PTE is 8 bytes, location of PTE for abc is at virtual address abc/8 = lmn

Virtual addr lmn → physical addr pqr
Alpha Address Mapping

Virtual address

<table>
<thead>
<tr>
<th>Unused bits</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 bits</td>
<td>10 bits</td>
<td>10 bits</td>
<td>10 bits</td>
<td>13 bits</td>
</tr>
</tbody>
</table>

Page table base register

L1 page table \(+\) L2 page table \(+\) L3 page table

PTE

32-bit physical page number \(\rightarrow\) Page offset

45-bit Physical address
Alpha Address Mapping

- Each PTE is 8 bytes – if page size is 8KB, a page can contain 1024 PTEs – 10 bits to index into each level

- If page size doubles, we need 47 bits of virtual address

- Since a PTE only stores 32 bits of physical page number, the physical memory can be addressed by at most 32 + offset

- First two levels are in physical memory; third is in virtual

- Why the three-level structure? Even a flat structure would need PTEs for the PTEs that would have to be stored in physical memory – more levels of indirection make it easier to dynamically allocate pages
Title

• Bullet