CIS 429/529
Advanced Memory Operations

Prof. Michel A. Kinsky
Memory Management

• The Fifties:
  ‣ Absolute Addresses
  ‣ Dynamic address translation

• The Sixties:
  ‣ Paged memory systems and TLBs
  ‣ Atlas’ Demand paging

• Modern Virtual Memory Systems
Names for Memory Locations

- **Machine language address**
  - as specified in machine code

- **Virtual address**
  - ISA specifies translation of machine code address into virtual address of program variable

- **Physical address**
  - operating system specifies mapping of virtual address into name for a physical memory location
Absolute Addresses

- EDSAC, early 50’s
  - effective address = physical memory address

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)

- Addresses in a program depended upon where the program was to be loaded in memory

- But it was more convenient for programmers to write location-independent subroutines
  - How could location independence be achieved?
Dynamic Address Translation

• Motivation:
  ‣ In the early machines, I/O operations were slow and each word transferred involved the CPU
  ‣ Higher throughput if CPU and I/O of 2 or more programs were overlapped. Why?
    • multiprogramming

• Location independent programs:
  ‣ Programming and storage management ease
  ‣ need for a base register
Dynamic Address Translation

• Protection:
  ‣ Independent programs should not affect each other inadvertently
  • need for a bound register
Simple Base and Bound Translation

Base and bounds registers only visible/accessible when processor running in \textit{kernel} (a.k.a \textit{supervisor}) mode
Separate Areas for Program and Data

- What is an advantage of this separation?
  - Used today on Cray vector supercomputers
Memory Fragmentation

- As users come and go, the storage is “fragmented”. Therefore, at some stage programs have to be moved around to compact the storage.

<table>
<thead>
<tr>
<th>OS Space</th>
<th>Users 4 &amp; 5 arrive</th>
<th>OS Space</th>
<th>Users 2 &amp; 5 leave</th>
<th>OS Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>16 K</td>
<td>user 1</td>
<td>16 K</td>
<td>user 1</td>
</tr>
<tr>
<td>user 2</td>
<td>24 K</td>
<td>user 2</td>
<td>24 K</td>
<td>user 2</td>
</tr>
<tr>
<td>free</td>
<td>24 K</td>
<td>user 4</td>
<td>16 K</td>
<td>user 4</td>
</tr>
<tr>
<td>free</td>
<td></td>
<td>free</td>
<td>8 K</td>
<td>free</td>
</tr>
<tr>
<td>user 3</td>
<td>32 K</td>
<td>user 3</td>
<td>32 K</td>
<td>user 3</td>
</tr>
<tr>
<td>free</td>
<td>24 K</td>
<td>user 5</td>
<td>24 K</td>
<td>free</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Paged Memory Systems

- Processor generated address can be interpreted as a pair `<page number, offset>`

- A page table contains the physical address of the base of each page
Each user has a page table

Page table contains an entry for each user page
Where Should Page Tables Reside?

- Space required by the page tables is proportional to the address space, number of users, ...
  - Space requirement is large too expensive to keep in registers
- Special registers just for the current user:
  - What disadvantages does this have?
    - may not be feasible for large page tables
- Main memory:
  - needs one reference to retrieve the page base address and another to access the data word
    - doubles number of memory references!
Page Tables in Physical Memory

User 1

Page Table, User 1

User 2

Page Table, User 2
Demand Paging in Atlas (1962)

- “A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor.” Tom Kilburn

- Primary memory as a cache for secondary memory

Diagram:

- **Primary Memory**: 32 Pages, 512 words/page
- **Secondary (Drum)**: 32x6 pages
- **User sees**: $32 \times 6 \times 512$ words of storage
Modern Virtual Memory Systems

- Protection & Privacy
  - several users, each with their private address space and one or more shared address spaces
    - page table $\equiv$ name space
- Demand Paging
  - ability to run a program larger than the primary memory
- What is another big benefit?
  - The price is address translation on each memory reference
Address Translation and Protection

- Every instruction and data access needs address translation and protection checks.
- A good VM design needs to be fast (~ one cycle) and space efficient.

Virtual Address

**Kernel/User Mode**

**Read/Write**

**Exception?**

**Protection Check**

**Address Translation**

- **Virtual Page No. (VPN)**
- **offset**

- **Physical Page No. (PPN)**
- **offset**

- **Physical Address**
Address Translation

Virtual Address

TLB Lookup

Page Table Walk

- miss
  - the page is not in memory
    - Page Fault (OS loads page)
  - the page is in memory
    - Update TLB

Protection Check

- hit
  - denied
    - Protection Fault
  - permitted
    - Physical Address (to cache)

Where?
Page Fault Handler

• When the referenced page is not in DRAM:
  ‣ The missing page is located (or created)
  ‣ It is brought in from disk, and page table is updated
    • Another job may be run on the CPU while the first job waits for
      the requested page to be read from disk
  ‣ If no free pages are left, a page is swapped out
    • approximate LRU replacement policy
Page Fault Handler

• Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS

  ‣ Untranslated addressing mode is essential to allow kernel to access page tables
Translation for Page Tables

- Can references to page tables cause TLB misses?

- User VA translation causes a TLB miss

- Page table walk: User PTE Base and appropriate bits from VA are used to obtain virtual address VP for page table entry

- Get a TLB miss when we try to translate VP
Translation for Page Tables

- When we get a TLB miss on VP translation, OS adds System PTE Base to bits from VP to find physical address of page table entry for VP
Swapping a Page of a Page Table

- A PTE in primary memory contains primary or secondary memory addresses
- A PTE in secondary memory contains only secondary memory addresses
- A page of a PT can be swapped out only if none its PTE’s point to pages in the primary memory
Address Translation in CPU Pipeline

- Software handlers need a restartable exception on page fault or protection violation.
- Handling a TLB miss needs a hardware or software mechanism to refill TLB.
Address Translation in CPU Pipeline

- Need mechanisms to cope with the additional latency of a TLB:
  - slow down the clock
  - pipeline the TLB and cache access
  - virtual address caches
  - parallel TLB/cache access
Physical or Virtual Address Caches?

- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- aliasing problems due to the sharing of pages (-)

Alternative: place the cache before the TLB

(StrongARM)
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

- General Solution: Disallow aliases to coexist in cache
Aliasing in Virtual-Address Caches

- **Software (i.e., OS) solution for direct-mapped cache**
  - VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!
Concurrent Access to TLB & Cache

- Index L is available without consulting the TLB
  - cache and TLB accesses can begin simultaneously
Concurrent Access to TLB & Cache

- Tag comparison is made after both accesses are completed
  - Cases: \( L + b = k \), \( L + b < k \), \( L + b > k \) what happens here?

![Diagram showing the relationships between virtual and physical addresses, VPN, TLB, and cache blocks.](image)
Virtual-Index Physical-Tag Caches

• After the PPN is known, W physical tags are compared

• Allows cache size to be greater than $2^{L+b}$ bytes
Virtual Memory Use Today

- Desktops/servers have full demand-paged virtual memory
  - Portability between machines with different memory sizes
  - Protection between multiple users or multiple tasks
  - Share small physical memory among active tasks
  - Simplifies implementation of some OS features
Virtual Memory Use Today

- Vector supercomputers have translation and protection but not demand-paging (Older Crays: base&bound, Japanese & Cray X1: pages)
  - Don’t waste expensive CPU time thrashing to disk (make jobs fit in memory)
  - Mostly run in batch mode (run set of jobs that fits in memory)
  - Difficult to implement restartable vector instructions
Virtual Memory Use Today

• Most embedded processors and DSPs provide physical addressing only
  ▸ Can’t afford area/speed/power budget for virtual memory support
  ▸ Often there is no secondary storage to swap to!
  ▸ Programs custom written for particular memory configuration in product
  ▸ Difficult to implement restartable instructions for exposed architectures
Next Class

• Process Synchronization