Mechanics of Address Translation

- How are addresses translated?
  - In software (now) but with hardware acceleration (a little later)
- Each process is allocated a page table (PT)
  - Maps VPs to PPs or to disk (swap) addresses
  - VP entries empty if page never referenced
  - Translation is table lookup

```c
struct {
    union { int ppn, disk_block; }
    int is_valid, is_dirty;
} PTE;
struct PTE pt[NUM_VIRTUAL_PAGES];
int translate(int vpn) {
    if (pt[vpn].is_valid)
        return pt[vpn].ppn;
}
```

Page Table Size

- How big is a page table on the following machine?
  - 4B page table entries (PTEs)
  - 32-bit machine
  - 4KB pages
- Solution
  - [material in class]
- How big would the page table be with 64KB pages?
- How big would it be for a 64-bit machine?
- Page tables can get enormous
  - There are ways of making them smaller

Multi-Level Page Table

- One way: multi-level page tables
  - Tree of page tables
  - Lowest-level tables hold PTEs
  - Upper-level tables hold pointers to lower-level tables
  - Different parts of VPN used to index different levels
- Example: two-level page table for machine on last slide
  - Compute number of pages needed for lowest-level (PTEs)
    - 4KB pages / 4B PTEs → 1K PTEs fit on a single page
    - 1M PTEs / (1K PTES/page) → 1K pages to hold PTEs
  - Compute number of pages needed for upper-level (pointers)
    - 1K lowest-level pages → 1K pointers
    - 1K pointers * 32-bit VA → 4KB → 1 upper level page
Multi-Level Page Table

- Have we saved any space?
  - Isn't total size of 2nd level PTE pages same as single-level table (i.e., 4MB)?
  - Yes, but...

- [material in class]

Address Translation Mechanics

- The six questions
  - What? address translation
  - Why? compatibility, multi-programming, protection
  - How? page table
  - Who performs it?
  - When?
  - Where does page table reside?

- Option I: process (program) translates its own addresses
  - Page table resides in process visible virtual address space
    - Bad idea: implies that program (and programmer)...
      - ...must know about physical addresses
        - Isn't that what virtual memory is designed to avoid?
      - ...can forge physical addresses and mess with other programs
    - Translation on L2 miss or always? How would program know?

Who? Where? When? Take II

- Option II: operating system (OS) translates for process
  - Page table resides in OS virtual address space
    + User-level processes cannot view/modify their own tables
    + User-level processes need not know about physical addresses
  - Translation on L2 miss
    - Otherwise, OS SYSCALL before any fetch, load, or store

- L2 miss: interrupt transfers control to OS handler
  - Translate VA by accessing process' page table
  - Accesses memory using PA
  - Returns to user process when L2 fill completes
    - Still slow: added interrupt handler and PT lookup to memory access
    - What if PT lookup itself requires memory access? Head spinning...

Translation Buffer

- Functionality problem? Add indirection!
- Performance problem? Add cache!

- Address translation too slow?
  - Cache translations in translation buffer (TB)
    - Small cache: 16–64 entries, often FA
    - Exploits temporal locality in PT accesses
    - OS handler only on TB miss
TB Misses

- **TB miss**: requested PTE not in TB, but in PT
  - Two ways of handling
  - Either way is relatively short, process just stalls

- [material in class]

Nested TB Misses

- **Nested TB miss**: when OS handler itself has a TB miss
  - TB miss on handler instructions
  - TB miss on page table VAs
  - Not a problem for hardware FSM: no instructions, PAs in page table

- Handling is tricky but possible
  - First, save current TB miss info before accessing page table
  - So that nested TB miss info doesn’t overwrite it
  - Second, **lock nested miss entries into TB**
    - Prevent TB conflicts that result in infinite loop
    - Another reason to have a highly-associative TB

Page Faults

- [material in class]

Virtual Caches

- Memory hierarchy so far: **virtual caches**
  - Indexed and tagged by VAs
  - Translate to PAs only to access memory
    - Fast: avoids translation latency in common case

- What to do on process switches?
  - Flush caches? Slow
  - Add process IDs to cache tags

- Does inter-process communication work?
  - **Aliasing**: multiple VAs map to same PA
    - How are multiple cache copies kept in sync?
    - Also a problem for I/O (later in course)
    - Disallow caching of shared memory? Slow
Physical Caches

- Alternatively: **physical caches**
  - Indexed and tagged by PAs
  - Translate to PA to at the outset
  + No need to flush caches on process switches
  + Processes do not share PAs
  + Cached inter-process communication works
    - Single copy indexed by PA
    - Slow: adds 1 cycle to $t_{hit}$

Virtual Physical Caches

Compromise: **virtual-physical caches**

- Indexed by VAs
- Tagged by PAs
- Cache access and address translation in parallel
  + No context-switching/aliasing problems
  + Fast: no additional $t_{hit}$ cycles
- A TB that acts in parallel with a cache is a **TLB**
  - Translation Lookaside Buffer
- Common organization in processors today

Cache/TLB Access

- Two ways to look at VA
  - Cache: TAG+IDX+OFS
  - TLB: VPN+POFS
- Can have parallel cache & TLB ...
  - If address translation doesn’t change IDX
  - VPN/IDX don’t overlap

Cached Inter-process Communication:

- Single copy indexed by PA
- Slow: adds 1 cycle to $t_{hit}$

Cache Size And Page Size

- Relationship between page size and L1 I$\&(D\$) size
  - [material in class]
TLB Organization

- **Like caches**: TLBs also have ABCs
  - What does it mean for a TLB to have a block size of two?
  - Two consecutive VPs share a single tag

- **Rule of thumb**: TLB should "cover" L2 contents
  - In other words: #PTEs * page size ≥ L2 size
  - Why? Think about this ...

Back to Virtual Memory

- Virtual memory almost ubiquitous today
  - Certainly in general-purpose (in a computer) processors
  - But even some embedded (in non-computer) processors support it

- Several forms of virtual memory
  - **Paging** (aka flat memory): equal sized translation blocks
  - Most systems do this
  - **Segmentation**: variable sized (overlapping?) translation blocks
  - IA32 used this
  - Makes life very difficult
  - **Paged segments**: don't ask

- How does virtual memory work when system starts up?

Summary

- DRAM
  - Two-level addressing
  - Refresh, access time, cycle time

- Building a memory system
  - DRAM/bus bandwidth matching

- Memory organization

- Virtual memory
  - Page tables and address translation
  - Page faults and handling
  - Virtual, physical, and virtual-physical caches and TLBs

Next part of course: I/O