CSE 490/590 Computer Architecture

Address Translation and Protection

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

- Prefetching
  - Speculate future I & d accesses and fetch them into caches
- Hardware techniques
  - Stream buffer
  - Prefetch-on-miss
  - One Block Lookahead
  - Strided
- Software techniques
  - Prefetch instruction
  - Loop interchange
  - Loop fusion
  - Cache tiling

Memory Management

- From early absolute addressing schemes, to modern virtual memory systems with support for virtual machine monitors
- Can separate into orthogonal functions:
  - Translation (mapping of virtual address to physical address)
  - Protection (permission to access word in memory)
  - Virtual memory (transparent extension of memory space using slower disk storage)
- But most modern systems provide support for all the above functions with a single page-based system

Absolute Addresses

EDSAC, early 50's

- 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
- Problems?

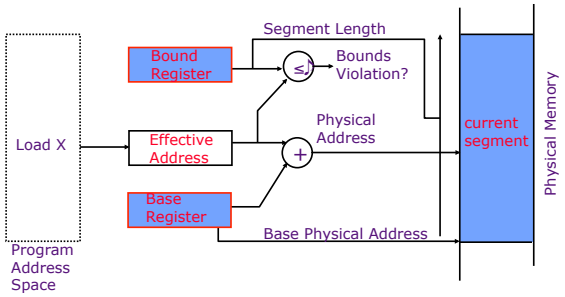
Dynamic Address Translation

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.
How?⇒ multiprogramming

Location-independent programs
Programming and storage management ease
⇒ need for a *base register*

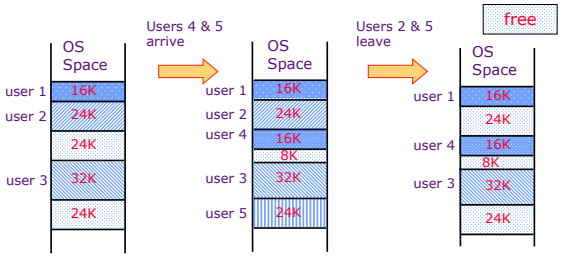
Protection
Independent programs should not affect each other inadvertently
⇒ need for a *bound register*

Simple Base and Bound Translation



Base and bounds registers are visible/accessible only when processor is running in the *supervisor mode*

Memory Fragmentation

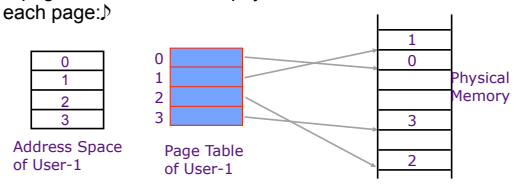


As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

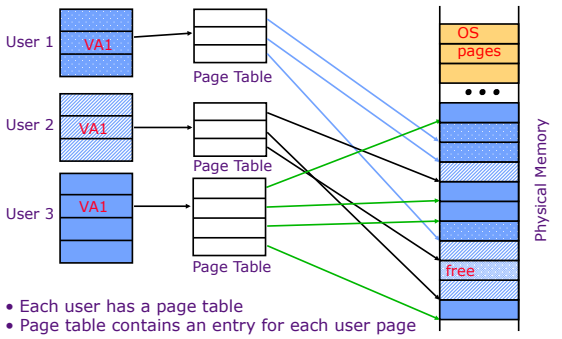
- Processor-generated address can be split into:

- A page table contains the physical address of the base of each page:♪



Page tables make it possible to store the pages of a program non-contiguously.

Private Address Space per User



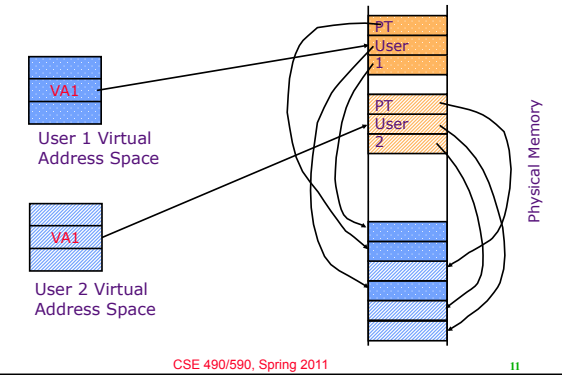
- Each user has a page table
- Page table contains an entry for each user page

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...
  - ⇒ Space requirement is large
  - ⇒ Too expensive to keep in registers
- Idea: Keep PTs in the main memory
  - needs one reference to retrieve the page base address and another to access the data word

    ⇒ *doubles the number of memory references!*

Page Tables in Physical Memory



CSE 490/590 Administrivia

- Midterm on Friday, 3/4
- Project 1 deadline: Friday, 3/11
- Project 2 list will be up soon
- Guest lectures possibly this month
- Quiz will be distributed Monday

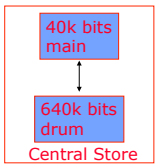
A Problem in the Early Sixties

- There were many applications whose data could not fit in the main memory, e.g., payroll
  - *Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory*
- Programmers moved the data back and forth from the secondary store by *overlaying* it repeatedly on the primary store

tricky programming!

Manual Overlays

- Assume an instruction can address all the storage on the drum
- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required
  - *Difficult, error-prone!*
- *Method 2:* automatic initiation of I/O transfers by software address translation
  - *Brooker's interpretive coding, 1960*
  - *Inefficient!*



Ferranti Mercury
1956

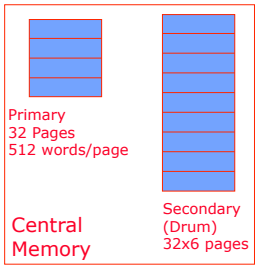
Not just an ancient black art, e.g., IBM Cell microprocessor used in Playstation-3 has explicitly managed local store!

Demand Paging in Atlas (1962)

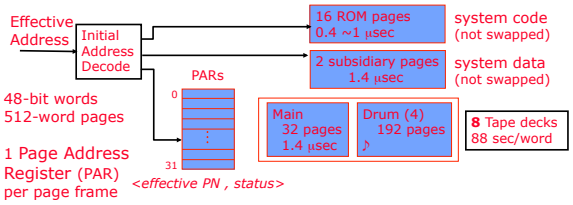
"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."
Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage



Hardware Organization of Atlas

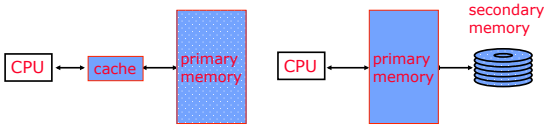


Compare the effective page address against all 32 PARs
match ⇒ normal access
no match ⇒ *page fault*
save the state of the partially executed instruction

Atlas Demand Paging Scheme

- On a page fault:
  - Input transfer into a free page is initiated
  - The Page Address Register (PAR) is updated
  - If no free page is left, a *page is selected to be replaced* (based on usage)
  - The replaced page is written on the drum
    - » to minimize drum latency effect, the first empty page on the drum was selected
  - The *page table is updated* to point to the new location of the page on the drum

Caching vs. Demand Paging



Caching	*Demand paging*
cache entry	page frame
cache block (~32 bytes)	page (~4K bytes)
cache miss rate (1% to 20%)	page miss rate (<0.001%)
cache hit (~1 cycle)	page hit (~100 cycles)
cache miss (~100 cycles)	page miss (~5M cycles)
a miss is handled in *hardware*	a miss is handled mostly in *software*

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252