Memory Management

CSE 410 - Computer Systems

December 3, 2001

Readings and References

- Reading
  - › Chapter 9, *Operating System Concepts*, Silberschatz, Galvin, and Gagne

- Other References

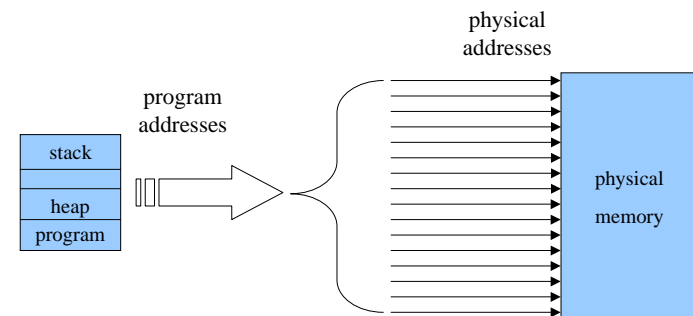
Program Memory Addresses

- Program addresses are fixed at the time the source file is compiled and linked
- Small, simple systems can use program addresses as the physical address in memory
- Modern systems usually much more complex
  - › program address space very large
  - › other programs running at the same time
  - › operating system is in memory too

Direct Physical Addressing

physical addresses

program addresses

stack

heap

program

physical memory
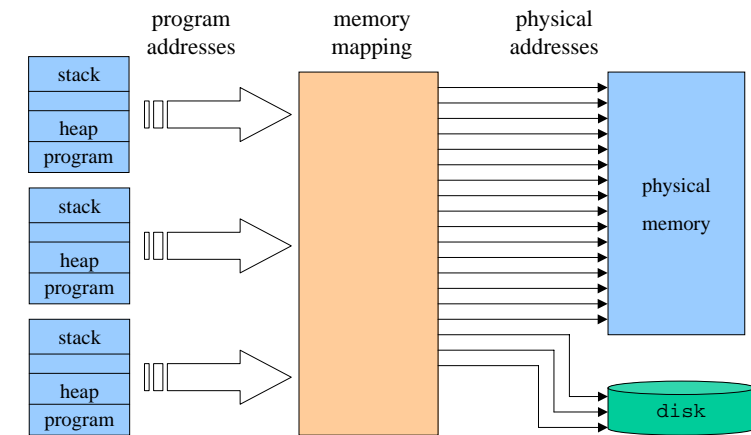
Physical Addresses

- Address generated by the program is the same as the address of the actual memory location
- Simple approach, but lots of problems
  - › Only one process can easily be in memory at a time
  - › There is no way to protect the memory that the process isn't supposed to change (ie, the OS or other processes)
  - › A process can only use as much memory as is physically in the computer
  - › A process occupies all the memory in its address space, even if most of that space is never used
    - 2 GB for the program and 2 GB for the system kernel

Memory Mapping

Virtual Addresses

- The program addresses are now considered to be "virtual addresses"
- The memory management unit (MMU) translates the program addresses to the real physical addresses of locations in memory
- This is another of the many interface layers that let us work with *abstractions*, instead of all details at all levels

Physical Memory Layout

- Contiguous Allocation
  - › Each process gets a single range of addresses
  - › Single-partition allocation
    - one process resident at a time
  - › Multiple-partition allocation
    - multiple processes resident at a time
- Noncontiguous allocation
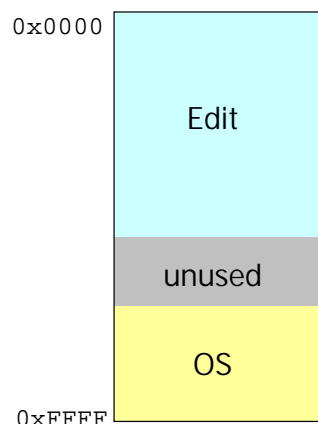 - › Paging, segmentation, or a combination

Uniprogramming without Protection

- Application always runs at the same place in physical memory
- Process can access all memory even OS
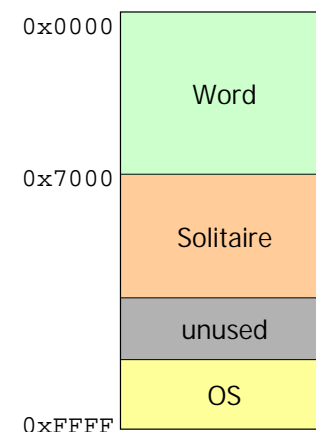 - › program bug crashes the machine
- MS-DOS

0x0000

Edit

unused

OS

0xFFFF

Multiprogramming without Protection

- When a program loaded the **linker-loader** translates a programs memory accesses (loads, stores, jumps) to where it will actually be running in memory
- Still no protection
- Once was very common
- Windows 3.1

0x0000

Word

0x7000

Solitaire

unused

OS

0xFFFF

Multiprogramming with Protection

- Restrict what a program can do by restricting what it can touch
- User process is restricted to its own memory space
  - › can't crash OS
  - › can't crash other process

- How?
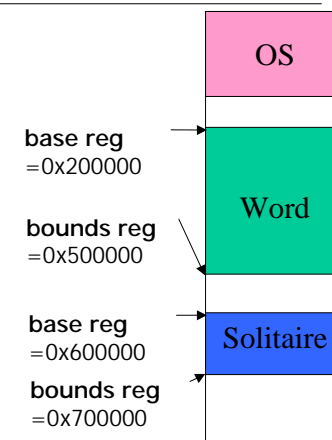 - › All problems can be solved with another level of indirection

Simple Translation: Base/Bounds

- Each process has a **base register**
  - › added to every memory reference
- Each process has a **bounds register**
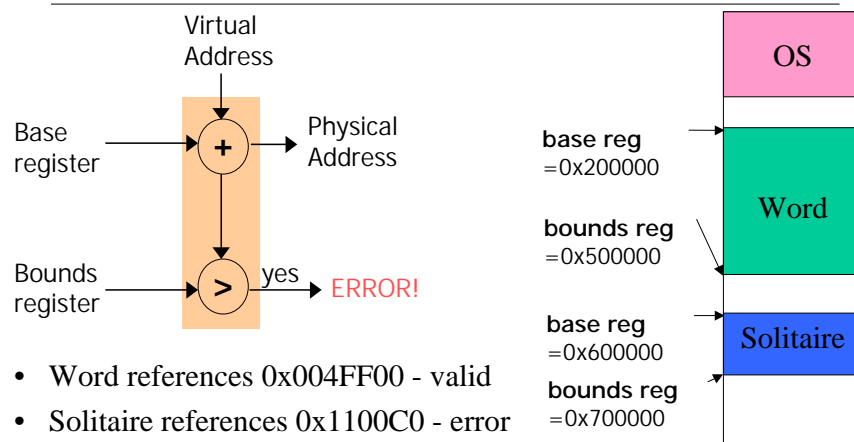 - › no memory reference allowed beyond here

OS

base reg =0x200000

Word

bounds reg =0x500000

base reg =0x600000

Solitaire

bounds reg =0x700000

Base/Bounds

Virtual Address

Base register → +

Physical Address

Bounds register → > yes → ERROR!

base reg
=0x200000

bounds reg
=0x500000

base reg
=0x600000

bounds reg
=0x700000

OS

Word

Solitaire

- Word references 0x004FF00 - valid
- Solitaire references 0x1100C0 - error

Fragmentation

OS	OS	OS	OS	OS	OS	OS	OS		OS
	P1	P1	P1	P1	P1		P5	compaction	P5
									P4
	P2	P2		P4	P4	P4			P3
		P3	P3	P3	P3	P3			P6
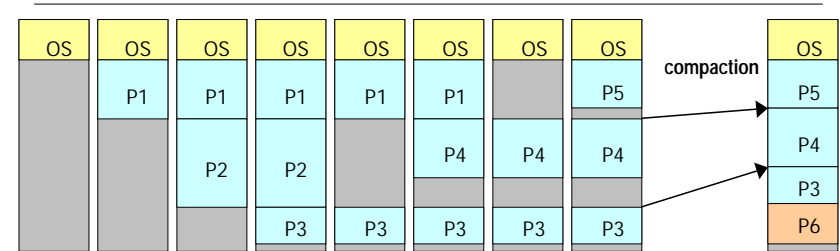
- Over time unused memory is spread out in small pieces
  › external fragmentation
- Rearrange memory to make room for the next program
  › compaction = lots of copying (expensive)
  › change base/bounds registers for moved programs

Base/bounds Evaluation

- Advantages of base/bounds
  › process can't crash OS or other processes
  › can move programs around and change base register
  › can change program memory allocation by changing bounds register
- Problems with base/bounds
  › external fragmentation
  › can't easily share memory between processes
  › programs are limited to amount of physical memory
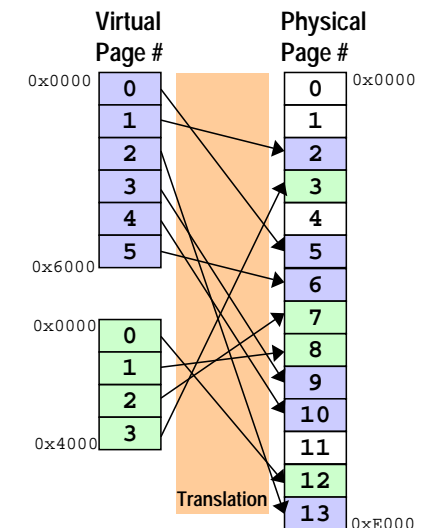 › doesn't improve support for sparse address spaces

Paging

- Divide a process's virtual address space into fixed-size chunks (called **pages**)
- Divide physical memory into pages of the same size
- **Any virtual page can be located at any physical page**
- Translation box converts from virtual pages to physical pages

Virtual Page #

0x0000 0
        1
        2
        3
        4
0x6000 5

0x0000 0
        1
        2
0x4000 3

Translation

Physical Page #

0 0x0000
1
2
3
4
5
6
7
8
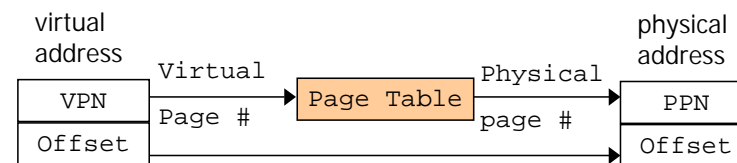9
10
11
12
13 0xE000

Paging and Fragmentation

- No **external fragmentation** because all pages are the same size
  - › don't have to rearrange pages
- Sometimes there is **internal fragmentation** because a process doesn't use a whole page
  - › some space wasted at the end of a page
  - › better than external fragmentation

Page Tables

- A page table maps virtual page numbers to physical page numbers
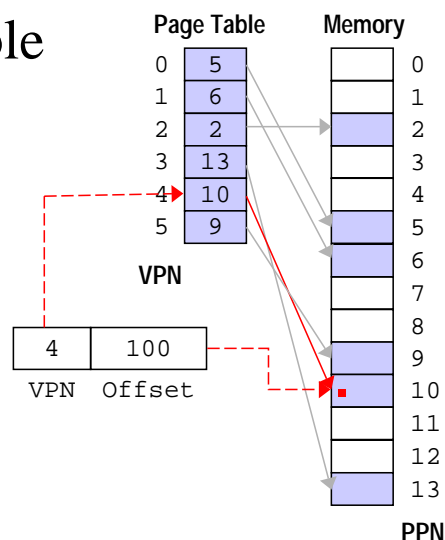- Lots of different types of page tables
  - › arrays, lists, hashes

Flat Page Table

- A flat page table uses the VPN to index into an array
- What's the problem? (Hint: how many entries are in the table?)

Flat Page Table Evaluation

- Very simple to implement
- Don't work well for sparse address spaces
  - › code starts at 0x00400000, stack starts at 0x7FFFFFFF
- With 4K pages, this requires 1M entries per page table
  - › must be kept in main memory (can't be put on disk)
- 64-bit addresses are a nightmare (4 TB)
- Addressing page tables in kernel virtual memory reduces the amount of physical memory used
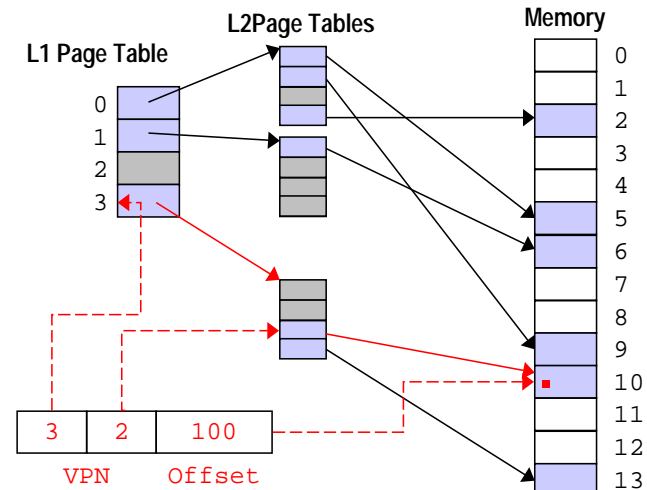
Multi-level Page Tables

- Use multiple levels of page tables
  - each page table entry points to another page table
  - the last page table contains the physical page numbers (PPN)
- The VPN is divided into
  - Index into level 1 page
  - Index into level 2 page
  - …

CSE 410 - Memory Management 21

Multi-level Page Tables

L1 Page Table   **L2Page Tables**   **Memory**

VPN		Offset
3	2	100

Multi-Level Evaluation

- Only allocate as many page tables as we need-- works with the sparse address spaces
- Only the top page table must be in pinned in physical memory
- Each page table usually fills exactly 1 page so it can be easily moved to/from disk
- Requires multiple physical memory references for each virtual memory reference

3-Dec-01 CSE 410 - Memory Management 23

Inverted Page Tables

- Inverted page tables **hash** the VPN to get the PPN
- Requires O(1) lookup
- Storage is proportional to number of physical pages being used **not** the size of the address space

Inverted Page Table   **Memory**

Hash Table

VPN	Offset