CSCI 4717/5717
Computer Architecture

Topic: Memory Management

Reading: Stallings, Sections 8.3 and 8.4

Memory Management

- Uni-program – memory split into two parts
  - One for Operating System (monitor)
  - One for currently executing program
- Multi-program
  - Non-O/S part is sub-divided and shared among active processes
- Remember segment registers in the 8086 architecture
  - Hardware designed to meet needs of O/S
  - Base Address = segment address

Swapping

- Problem: I/O (Printing, Network, Keyboard, etc.) is so slow compared with CPU that even in multi-programming system, CPU can be idle most of the time
- Solutions:
  - Increase main memory
    - Expensive
    - Programmers will eventually use all of this memory for a single process
  - Swapping

What is Swapping?

- Long term queue of processes stored on disk
- Processes “swapped” in as space becomes available
- As a process completes it is moved out of main memory
- If none of the processes in memory are ready (i.e. all I/O blocked)
  - Swap out a blocked process to intermediate queue
  - Swap in a ready process or a new process
- But swapping is an I/O process!
  - It could make the situation worse
  - Disk I/O is typically fastest of all, so it still is an improvement

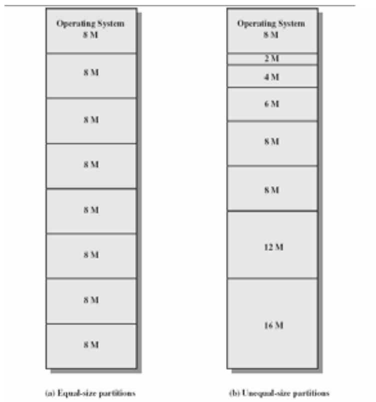
Partitioning

- Splitting memory into sections to allocate to processes (including Operating System)
- Two types
  - Fixed-sized partitions
  - Variable-sized partitions

Fixed-Sized Partitions (continued)

- Equal size or Unequal size partitions
- Process is fitted into smallest hole that will take it (best fit)
- Some wasted memory due to each block having a hole of unused memory at the end of its partition
- Leads to variable sized partitions

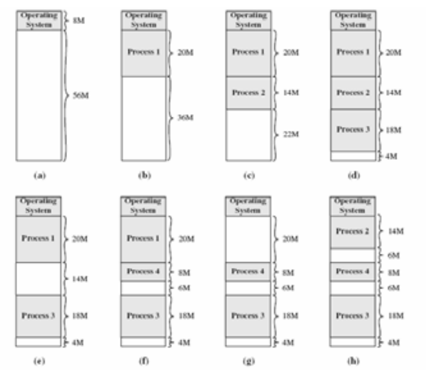
Fixed-sized partitions



Variable-Sized Partitions

- Allocate exactly the required memory to a process
- This leads to a hole at the end of memory, too small to use – Only one small hole - less waste
- When all processes are blocked, swap out a process and bring in another
- New process may be smaller than swapped out process
- Reloaded process not likely to return to same place in memory it started in
- Another hole
- Eventually have lots of holes (fragmentation)

Variable-Sized Partitions



Solutions to Holes in Variable-Sized Partitions

- Coalesce - Join adjacent holes into one large hole
- Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)

Relocation

- No guarantee that process will load into the same place in memory
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- **Logical address** – relative to beginning of program
- **Physical address** – actual location in memory (this time)
- **Base Address** – start of program or block of data
- Automatic conversion using base address

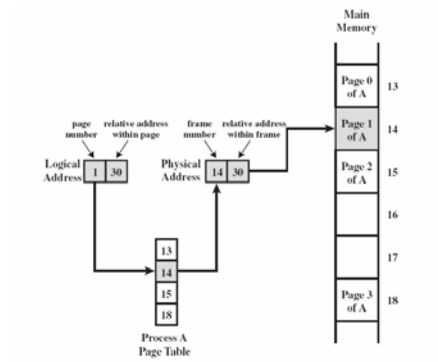
Paging (continued)

- Split memory into equal sized, small chunks -page frames
- Split programs (processes) into equal sized small chunks – pages
- Allocate the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames

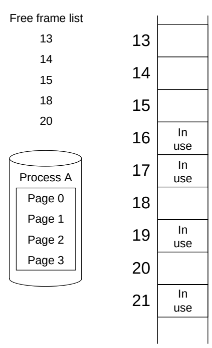
Paging (continued)

- Use page table to keep track of how the process is distributed through the pages in memory
- Now addressing becomes page number:relative address within page which is mapped to frame number:relative address within frame.

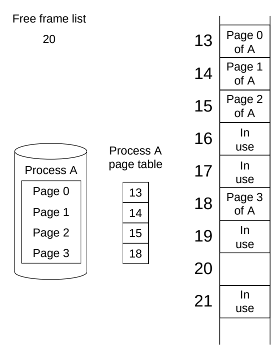
Paging (continued)



Paging Example – Before



Paging Example – After



Virtual Memory

- Remember the Principle of Locality which states that "active" code tends to cluster together, and if a memory item is used once, it will most likely be used again.
- Demand paging
  - Do not require all pages of a process in memory
  - Bring in pages as required

Page Fault in Virtual Memory

- Required page is not in memory
- Operating System must swap in required page
- May need to swap out a page to make space
- Select page to throw out based on recent history

Virtual Memory Bonus

- We do not need all of a process in memory for it to run
- We can swap in pages as required
- So - we can now run processes that are bigger than total memory available!
- Main memory is called real memory
- User/programmer sees much bigger memory - virtual memory

Thrashing

- Too many processes in too little memory
- Operating System spends all its time swapping
- Little or no real work is done
- Disk light is on all the time
- Solutions
  - Better page replacement algorithms
  - Reduce number of processes running
  - Get more memory

Page Table Structure

- VAX architecture – each process may be allocated up to 2^{31} = 2 GBytes of virtual memory broken in to 2^9=512 byte pages.
- Therefore, each process may have a page table with $2^{(31-9)}=2^{22}$=4 Meg entries.
- This uses a bunch of memory!

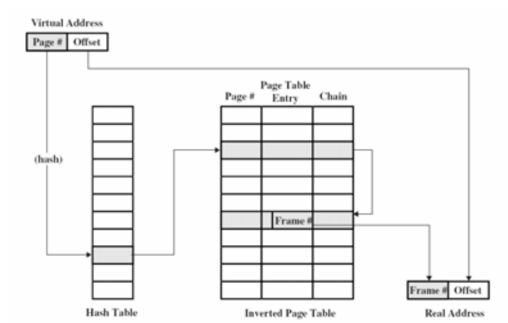
Pages of Page Table

- Some processors solve this with a page directory that points to page tables, each table of which is limited to a page and treated as such
- Another approach is the ***inverted page table structure***

Inverted Page Table

- Page tables based on logical (program's) address space can be huge
- Alternatively, restrict page table entries to real memory, not virtual memory
- Problem:
  - Simple page table says each line of table maps to logical page
  - Inverted Page Table need to have mapping algorithm because there isn't a one-to-one mapping of logical to virtual pages

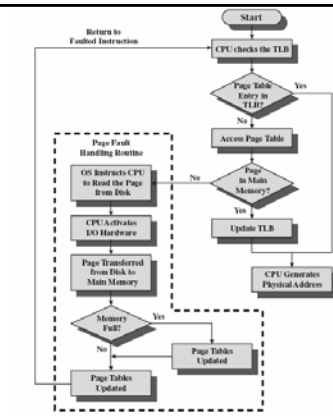
Page of Page Table (continued)



Translation Lookaside Buffer

- Every virtual memory reference causes two physical memory access
- Fetch page table entry
- Fetch data
- Use ***special cache*** for page table – TLB

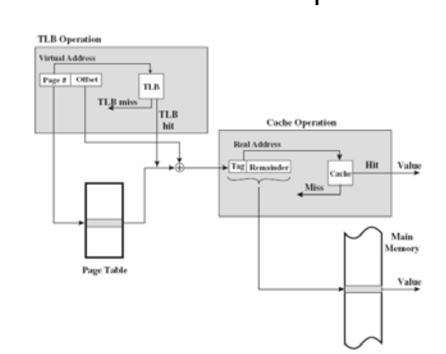
Translation Lookaside Buffer (continued)



Translation Lookaside Buffer (continued)

- Complexity! Virtual address translated to a physical address
- Reference to page table – might be in TLB, main memory, or disk
- Referenced word may be in cache, main memory, or disk
- If referenced word is on disk, it must be copied to main memory
- If in main memory or on disk, block must be loaded to cache and cache table must be updated

TLB and Cache Operation



Segmentation

- Paging is not (usually) visible to the programmer
- Segmentation is visible to the programmer
- Usually different segments allocated to program and data
- May be a number of program and data segments

Advantages of Segmentation

- Simplifies handling of growing data structures – O/S will expand or contract the segment as needed
- Allows programs to be altered and recompiled independently, without re-linking and re-loading
- Lends itself to sharing among processes
- Lends itself to protection since O/S can specify certain privileges on a segment-by-segment basis
- Some systems combine segmentation with paging

Recursion

- Many complex algorithmic functions can be broken into a repetitive application of a simple algorithm.
- The typical recursion function begins with an initial value of n which is decremented with each recursive call until the last call reaches a terminal value of n.
- A recursive function contains a call to itself.
- "Definition of recursion: See recursion"

Recursion – Factorial

- Non-Recursive Function:

```
int factorial(int n)
{
   int return_val = 1;
   for (int i = 1; i <= n; i++)
        return_val = return_val * i;
   return return_val;
}
```

- Recursive Function:

```
int rfactorial(int n)
{
   if ((n == 1) || (n == 0)) return (1);
   else return (n*rfactorial(n - 1));
}
```


Recursion – Fibonacci Numbers

"f(i) = f(i–1) + f(i–2)"

- Non-Recursive Function:

```
int fibonacci(int n)
{
    int fibval_i = 1;
    int fibval_i_minus_1 = 0;
    int fibval_i_minus_2 = 0;
    if ((n == 0)||(n == 1)) return n;
    else
    {
          for (int i = 2; i <= n; i++)
          {
                fibval_i_minus_2 = fibval_i_minus_1;
                fibval_i_minus_1 = fibval_i;
                fibval_i = fibval_i_minus_1 +
                                  fibval_i_minus_2;
          }
    }
    return fibval_i;
}
```


Recursion – Fibonacci Numbers (continued)

- Recursive Function:

```
int rfibonacci(int n)
{
   if ((n == 0)||(n == 1)) return n;
   else return rfibonacci(n - 1) +
                    rfibonacci(n - 2);
}
```


Comparing Recursive and Non-Recursive Functions

- Non-recursive function has more variables. Where does recursive function store values.
- Non-recursive function has more code → recursive requires less code and therefore less memory.

In-Class Exercise

- In groups, discuss how recursion might affect an operating system
- Compare & contrast iterative vs. recursion algorithms in terms of growth/memory usage

Pentium II

- Hardware for segmentation and paging
- Unsegmented unpaged
  - virtual address = physical address
  - Low complexity
  - High performance
- Unsegmented paged
  - Memory viewed as paged linear address space
  - Protection and management via paging
  - Berkeley UNIX

Pentium II (continued)

- Segmented unpaged
  - Collection of local address spaces
  - Protection to single byte level
  - Translation table needed is on chip when segment is in memory
- Segmented paged
  - Segmentation used to define logical memory partitions subject to access control
  - Paging manages allocation of memory within partitions
  - Unix System V

Pentium II Segmentation

- Each virtual address is 16-bit segment and 32-bit offset
- 2 bits of segment are protection mechanism
- 14 bits specify segment
- Unsegmented virtual memory 2^{32} = 4Gbytes
- Segmented 2^{46}=64 terabytes
  - Can be larger – depends on which process is active
  - Half (8K segments of 4Gbytes) is global
  - Half is local and distinct for each process

Pentium II Protection

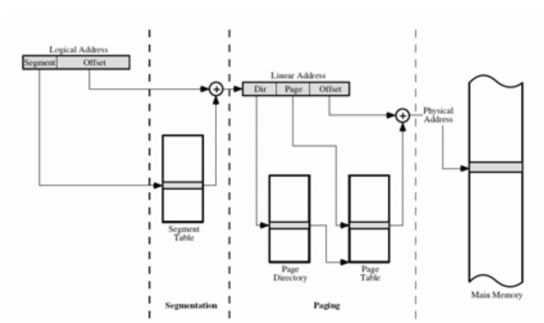
Protection bits give 4 levels of privilege

- 0 most protected, 3 least
- Use of levels software dependent
- Usually level 3 is for applications, level 1 for O/S and level 0 for kernel (level 2 not used)
- Level 2 may be used for apps that have internal security, e.g., database
- Some instructions only work in level 0

Pentium II Paging

- Segmentation may be disabled in which case linear address space is used
- Two level page table lookup
- First, page directory
  - 1024 entries max
  - Splits 4G linear memory into 1024 page groups of 4Mbyte
  - Each page table has 1024 entries corresponding to 4Kbyte pages
  - Can use one page directory for all processes, one per process or mixture
    Page directory for current process always in memory
- Use TLB holding 32 page table entries
- Two page sizes available 4k or 4M

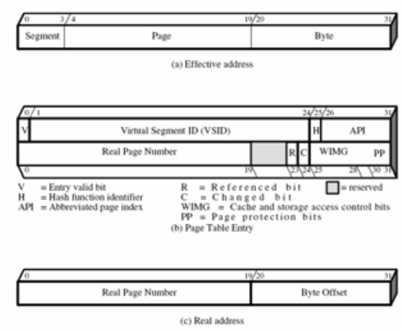
Pentium Segment/Paging Operation



PowerPC Memory Management Hardware

- 32 bit – paging with simple segmentation
  - 64 bit paging with more powerful segmentation
- Or, both do block address translation
  - Map 4 large blocks of instructions & 4 of memory to bypass paging
  - e.g. OS tables or graphics frame buffers
- 32 bit effective address
  - 12 bit byte selector → 4kbyte pages
  - 16 bit page id → 64k pages per segment
  - 4 bits indicate one of 16 segment registers → Segment registers under OS control

PowerPC 32-bit Memory Management Formats



PowerPC 32-bit Address Translation