CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 28

Main Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Loading vs Linking

- **Loading**
  - **Load executable into memory prior to execution**
- **Linking**
  - **Takes some smaller executables and joins them together as a single larger executable.**

Linking: Static vs Dynamic

- **Static linking** – system libraries and program code combined by the loader into the binary image
  - Every program includes library: wastes memory
- **Dynamic linking** –linking postponed until execution time
  - Operating system checks if routine is in processes' memory address

Colorado State University

Dynamic Linking

- **Dynamic linking** –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for
  - shared libraries
  - Separate patching

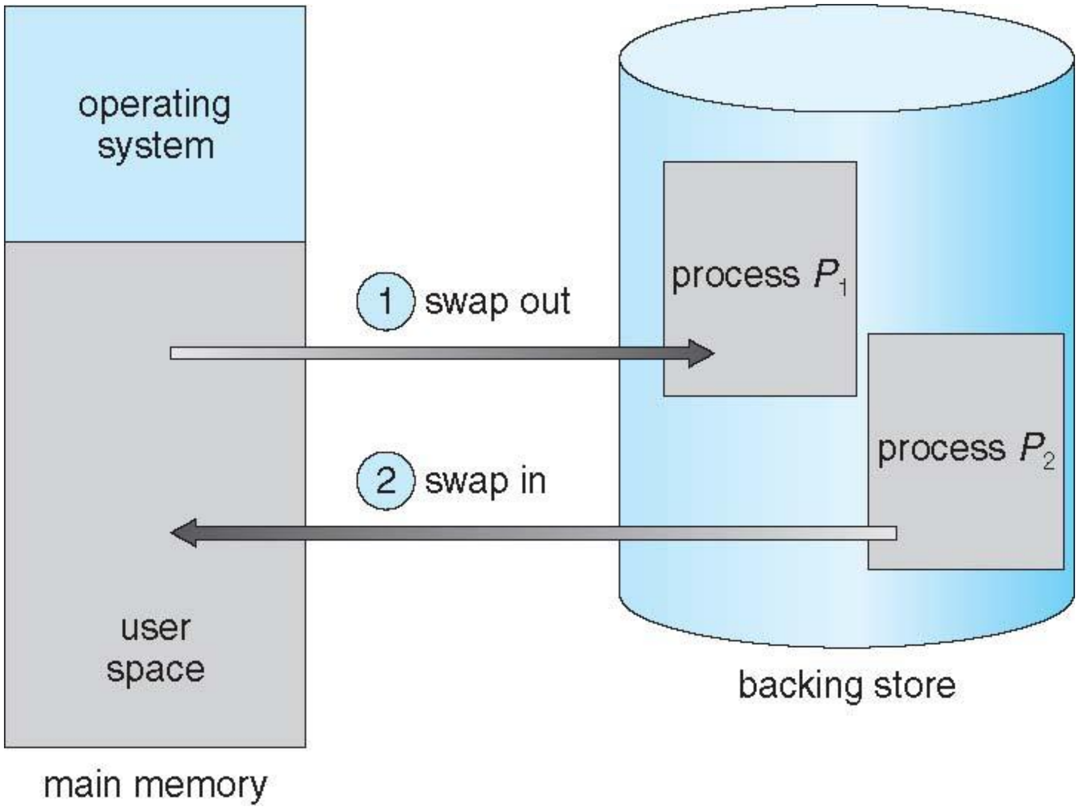
Dynamic loading of routines

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- OS can help by providing libraries to implement dynamic loading

Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Colorado State University

Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 100MB/50MB/s = 2 seconds
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4 seconds + some latency
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used

Context Switch Time and Swapping (Cont.)

- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Contiguous Allocation

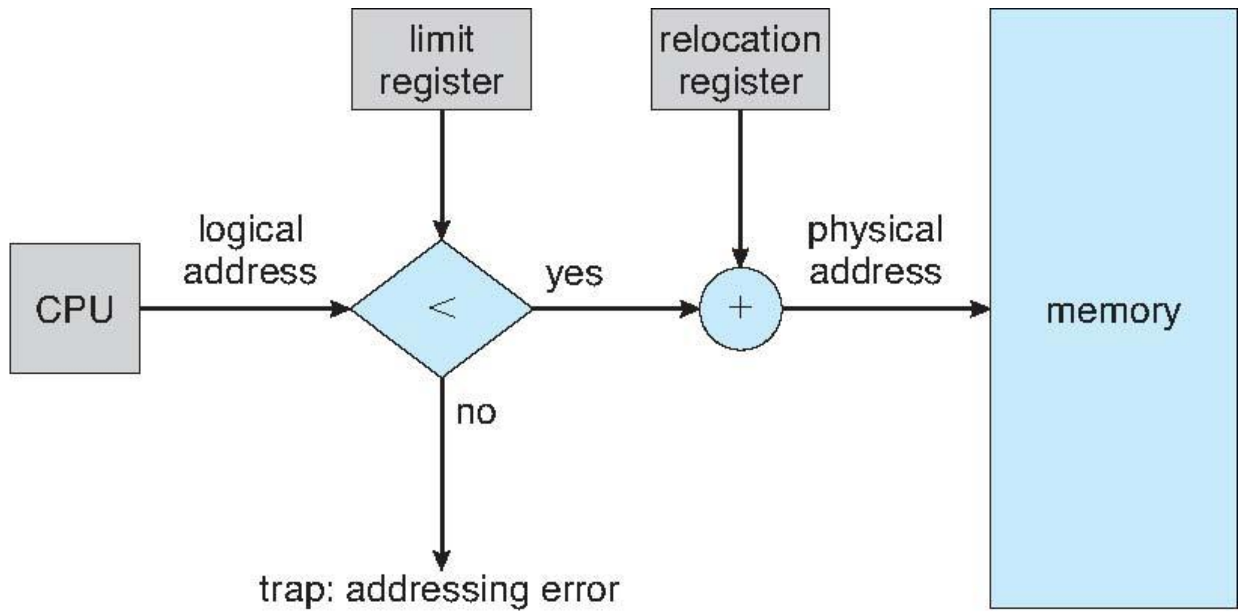
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

- Registers used to protect user processes from each other, and from changing operating-system code and data
  - Relocation (Base) register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

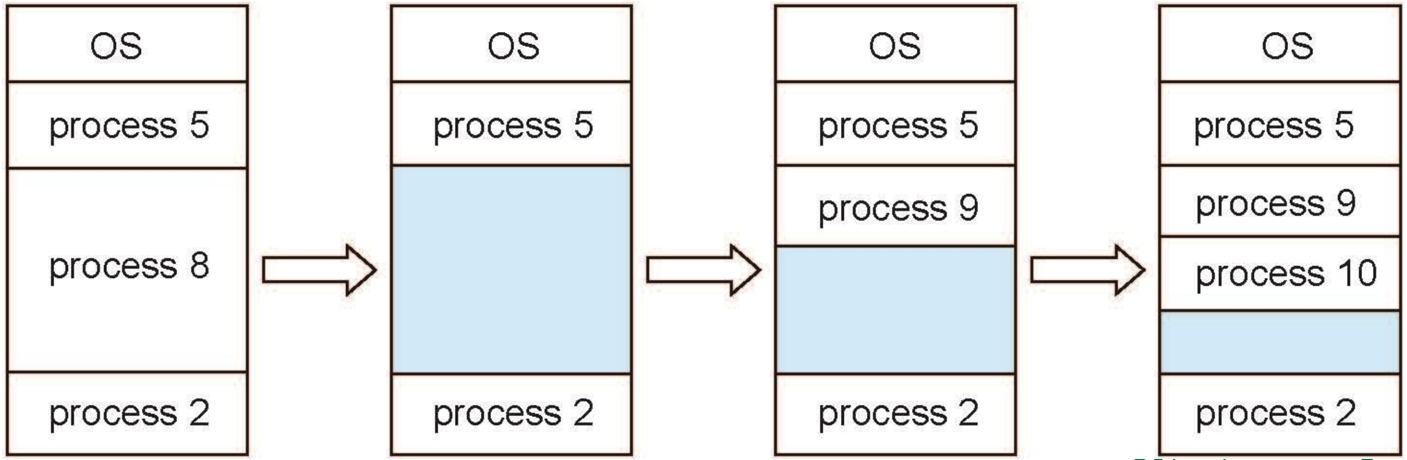
Hardware Support for Relocation and Limit Registers



MMU maps logical address *dynamically*
Physical address = relocation reg + valid logical address

Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Colorado State University

Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

Simulation studies:

- First-fit and best-fit better than worst-fit in terms of speed and storage utilization
- Best fit is **slower** than first fit . Surprisingly, it also results in more **wasted memory** than first fit
  - Tends to fill up memory with tiny, useless holes

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Simulation analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Paging

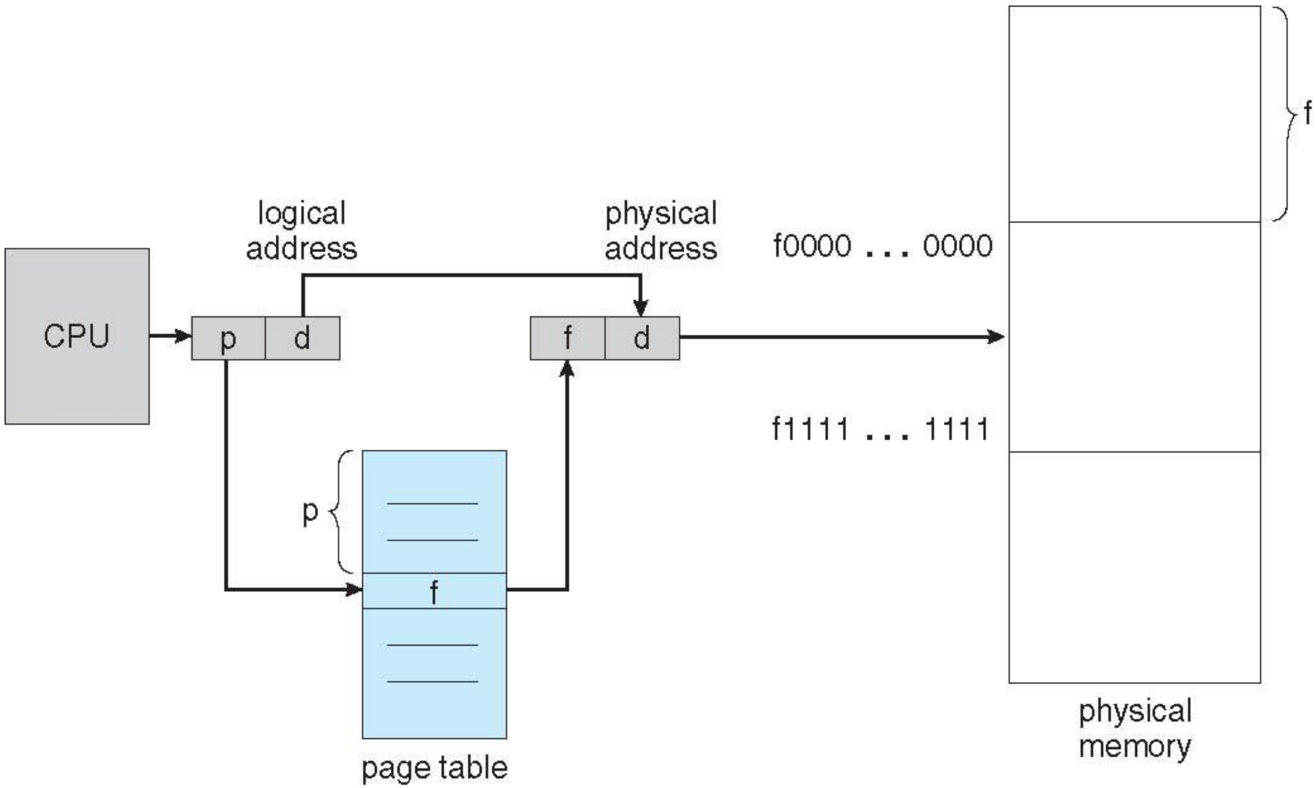
Paging

- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Still have Internal fragmentation
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks

Paging

- physical memory - **frames**
- logical memory - **pages**
- Keep track of all free frames
- Set up a **page table** to translate logical to physical addresses

Paging Hardware



Page number p to frame number f translation

Colorado State University