Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation
Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space

![Diagram showing base and limit registers]

Address Binding

- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Further, addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic
  - Compiled code addresses **bind** to relocatable addresses
    - i.e. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. 74014
  - Each binding maps one address space to another
Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
  - **Logical address space** is the set of all logical addresses generated by a program
  - **Physical address space** is the set of all physical addresses generated by a program
Memory-Management Unit (MMU)

• Hardware device that at run time maps virtual to physical address
• Many methods possible, covered in the rest of this chapter

• To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  • Base register now called **relocation register**
  • MS-DOS on Intel 80x86 used 4 relocation registers

• The user program deals with *logical* addresses; it never sees the *real* physical addresses
• Execution-time binding occurs when reference is made to location in memory
• Logical address bound to physical addresses
Contiguous Allocation

- 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

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - 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
  - Can then allow actions such as kernel code being transient and kernel changing size

Hardware Support for Relocation and Limit Registers

- Diagram showing the process of checking logical address against limit register and relocation register to determine if the address is valid or if an addressing error has occurred.
Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - Slot – block of available memory; slots of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a slot 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 (slot)

Dynamic Storage-Allocation Problem

How to satisfy a request of size $n$ from a list of free slots?

- **First-fit**: Allocate the *first* slot that is big enough

- **Best-fit**: Allocate the *smallest* slot that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover slot

- **Worst-fit**: Allocate the *largest* slot; must also search entire list
  - Produces the largest leftover slot

First-fit and best-fit better than worst-fit in terms of speed and storage utilization
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
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- 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
- Does the swapped out process need to swap back in to same physical addresses?
  - Depends on address binding method
    - Plus consider pending I/O to/from process memory space
  - Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping
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
  - Plus disk latency of 8 ms
  - Swap out time of 2008 ms
  - Plus swap in of same sized process
- Total context switch swapping component time of 4016 ms (> 4 seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via request memory and release memory

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

- First fit analysis reveals that given $N$ blocks allocated, $0.5N$ 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

- Now consider that backing store has same fragmentation problems

- But, fragmentation becomes a big problem in long-running systems