Chapter 8: Main Memory
Memory

- We started with how to organize and coordinate a cohort of programs and threads executing concurrently.

- Dealt with correctness issues arising from concurrency.

- Dealt with scheduling issues that impact fairness, utilization.

- Now we get into the non-computational resources of the machine.
  - Memory
  - Disks

- Start with memory.
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
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
  - Hardware manages cache, not OS or user.
- Protection of memory required to ensure correct operation
Start simple

Simplest task we need to tackle.
- How to keep processes isolated.

First attempt:
- Define a start and length of the memory footprint of a process.

Machine can at least make sure the processes don’t wander outside the range this defines.
A pair of **base** and **limit** registers define the logical address space.
Hardware address protection

```
CPU → address

Base

>= no

< yes

base + limit

< no

Addressing error: trap to OS

memory```

Silberschatz, Galvin and Gagne ©2009
Process: “Where am I?”

- How does the process consider memory?
  - Where is zero?
  - How to know where data/code will be at runtime.

- Problem of defining how addresses are bound to code/data.
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)
Multistep Processing of a User Program

1. Source program
2. Compiler or assembler
3. Object module
4. Linkage editor
5. Load module
6. Loader
7. In-memory binary memory image
   - Dynamic linking
   - Dynamically loaded system library
   - Execution time (run time)
   - Load time
   - Compile time
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
Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address

- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory

- The user program deals with *logical* addresses; it never sees the *real* physical addresses
PPC750 Die

Data & Instr MMUs
Cell Broadband Engine Die
Dynamic relocation using a relocation register

- Logical address: 346
- Relocation register: 14000
- Physical address: 14346
- CPU
- MMU
- Memory
Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required implemented through program design

- Different from 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 needed to check if routine is in processes’ memory address
- Dynamic linking is particularly useful for libraries
- System also known as *shared libraries*
### C:\windows\system32 on XP

...  
- a---  4/14/2008  5:00 AM  41472 hhsetup.dll  
- a---  4/14/2008  5:00 AM  20992 hid.dll  
- a---  4/14/2008  5:41 AM  21504 hidserv.dll  
- a---  4/14/2008  5:00 AM  72704 hlink.dll  
- a---  4/14/2008  5:00 AM  344064 hnetcfg.dll  
- a---  4/14/2008  5:00 AM  14848 hnetmon.dll  
- a---  4/14/2008  5:00 AM  330752 hnetwiz.dll  
- a---  4/14/2008  5:00 AM  144896 hotplug.dll  
- a---  4/14/2008  5:00 AM  44544 hicons.dll  
- a---  4/14/2008  5:00 AM  24576 httpapi.dll  
- a---  4/14/2008  5:00 AM  41984 htui.dll  
...  

*Dynamically linked libraries*
Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

- **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
Swapping

- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows).
- System maintains a ready queue of ready-to-run processes which have memory images on disk.
Schematic View of Swapping

1. **Swap out**
   - Process $P_1$ is swapped out from main memory.

2. **Swap in**
   - Process $P_2$ is swapped in from the backing store.
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

- 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*
Hardware Support for Relocation and Limit Registers

CPU ➔ limit register > yes ➔ physical address ➔ memory

CPU ➔ < logical address ➔ no ➔ trap: addressing error

CPU ➔ relocation register ➔ physical address ➔ memory
Contiguous Allocation (Cont)

- Multiple-partition allocation
  - 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
  - Operating system maintains information about:
    a) allocated partitions  
    b) free partitions (hole)
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

First-fit and best-fit better than worst-fit in terms of speed and storage utilization
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
Fragmentation

- 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

- Approach to memory management that eliminates external fragmentation.
  - Does not eliminate internal fragmentation.
  - Some processes will consume more memory than they need.
    - Holes have well defined bounds due to fixed page size though.

- This approach is used all over the place.
  - So popular, hardware support exists in modern CPUs to support it.
Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes).
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames.
- To run a program of size $n$ pages, need to find $n$ free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.