Lecture Topics

- Today: Memory Management  
  (Stallings, chapter 7.1-7.4)
- Next: Virtual Memory  
  (Stallings, chapter 8.1-8.4)

Announcements

- Exam scores sent via email
- Self-Study Exercise #6 (this week)
- Project #4 (due 2/23)
- Project #5 (due 3/2)
Cache Memory

- Cache: fast (and thus small and expensive)
- Main memory: slow (and thus larger and cheaper)
- Processor first checks cache for requested word
- If not found in cache, a block of memory containing the word is moved to the cache

Example

- RAM: 4 GB (32-bit addresses)
- Data cache
  - 4 bits per index (16 lines in cache)
  - 16 bytes per block
  - direct mapped
  - uses "write back" as the write policy
Example (continued)

- Address (32 bits) viewed as three fields:
  - Byte offset: 4 bits to identify byte within block
  - Line: 4 bits to identify cache line
  - Tag: 28 bits (remaining bits)

- Example: 0003A529

Example (continued)

- How many lines in the cache?
  \[ 2^4 = 16 \text{ lines} \]

- How many bytes in one block?
  \[ 2^4 = 16 \text{ bytes} \]

- How many control bits in one line?
  \[ V + M + \text{Tag} = 1 + 1 + 28 = 30 \text{ bits} \]

- How many total bits in one line?
  \[ \text{control + data} = 30 + 128 = 158 \text{ bits} \]
Example (continued)

- Overview of cache operation
  - Determine if requested address is already present in the cache
  - If not present, put it in the cache (miss)
  - Process read or write request

- Miss processing
  - If necessary, write back before replacing block
  - Copy block from RAM to cache

Example (continued)

- Full example on course website:
  ~cse410/Examples/example12.pdf
Main Memory (RAM)

- Process manages registers: uses Load and Store instructions to interact with RAM

- Hardware manages cache: transfers blocks between RAM and cache when necessary (not visible to process, except indirectly through timing)

- OS manages main memory: allocates space for processes

Main Memory (RAM)

- Processes must be resident in RAM in order to be executed:
  - Process control block
  - Space for machine language instructions
  - Space for data objects

- OS also must be resident in RAM (set of kernel functions which are called by user processes)
Need for Memory Management

- **Goal:** keep set of processes in the Ready state so that utilization of the CPU is high

- If only a few processes can be kept in RAM, then there will be times when all processes will be blocked (waiting for I/O) and the CPU will be idle

- Hence, RAM must be managed efficiently in order to keep multiple processes in the Ready state
Memory Management Themes

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Relocation

- Programmer does not know where program will be placed in RAM when it is executed
- While a program is executing, it may be swapped to disk and returned to RAM at a different location (relocated)
- Memory references in a program must be translated to physical memory addresses
Protection

- A process cannot reference memory locations in other processes without permission
- Not possible to check physical addresses at compile time – must be checked at run time
- To be effective, memory protection requirement must be satisfied by the processor (hardware) rather than the OS

Sharing

- Protection mechanism must allow shared access (in a controlled manner)

- Examples:
  - single copy of library functions in memory (shared by all processes)
  - data objects in memory shared by cooperating processes
Logical Organization

- Programs are usually written as modules
  - Modules can be written and compiled independently
  - Different types of protection can be given to different modules (read-only, read-write)
  - Modules can be shared among processes

Physical Organization

- Secondary memory (disk): long term storage for programs and data
- Primary memory (RAM): holds programs and data currently in use
- Moving information between these two levels of memory is a major responsibility of the OS (impractical and undesirable to make programmer responsible)
Simplistic Memory Management

- Assume a process must be loaded entirely into main memory in order to execute

- Simple strategies (without virtual memory):
  - fixed partitioning
  - dynamic partitioning
  - simple paging
  - simple segmentation

Fixed Partitioning

- Subdivide memory into partitions with fixed boundaries

- Partitions can all be the same size or can be several different sizes
Fixed Partitioning

- Any process whose size is less than or equal to a partition size can be loaded into the partition
- If all partitions are occupied, the operating system can swap a process out of a partition, then load a different process into that partition

Fixed Partitioning: Problems

- A program may be too large to fit in a partition. Programmer must then design the program with overlays (programmer identifies "sections" of program which don’t have to be resident in memory at the same time)
- Memory utilization is low due to internal fragmentation (memory allocated to a process, but not needed by that process)
Fixed Partitioning: Placement Algorithms

- Equal-size partitions: because all partitions are of equal size, it does not matter which partition is used

- Unequal-size partitions: processes are assigned in such a way as to minimize wasted memory within a partition

Dynamic Partitioning

- Partitions are of variable length and number

- Process is allocated exactly as much memory as required

- External fragmentation: gaps develop between processes

- Must use compaction to move processes in RAM so that the processes are contiguous and all free memory is in one block
Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process

- Three common algorithms:
  - Best-fit
  - First-fit
  - Next-fit
Partitioning: Relocation

- A process may occupy different partitions due to swapping, which means different absolute memory locations during execution.

- Compaction may also cause a program to occupy a different partition, which means different absolute memory locations during execution.

- Hardware must support relocation.
Terminology Related to Addresses

- Logical Address: reference to a memory location independent of the current assignment of data to memory; translation must be made to physical address
- Relative Address: address expressed as a location relative to some known point
- Physical Address: the absolute address or actual location in main memory

Hardware Translation of Addresses

- Process moved to Running state:
  - base register loaded with starting physical address
  - bounds register loaded with ending physical address
- Every relative address is added to the contents of the base register to obtain the physical address and then is compared with the contents of the bound register
Hardware Translation of Addresses

- Relocation: process can be swapped out, swapped back into a different partition

- Protection: each process can only access memory within its process image