Agenda

- Intro: Memory management
- Memory hierarchy
- Address binding
- Memory allocation techniques
  - Contiguous Memory Allocation
    - Fixed and Dynamic Allocations
  - Paging Concepts
Introduction

- Our machines today have 10,000 times more memory than the IBM 7094 - leading edge machine of the 1960’s
- Cost of memory has dropped dramatically
- Bill Gates (former chair of Microsoft) once said “640K should be enough”
- There is a high-performance machine at Western (for bioinformatics research) with 1 terabyte of memory
Introduction

- Software and data sets expand to fill the memory available
  - The 1 terabyte of memory – the researchers already want more

- Operating systems must manage memory
Introduction

- Memory management requires
  - Allocate memory to processes when needed
  - Keep track of what parts of memory are in use
  - Deallocate memory when processes are done
Instruction-Execution Cycle

- You can think of memory as a large array of bytes
  - Each byte has its own address
- Fetch an instruction from memory
- Instruction is decoded
- After instruction execution
  - Results may be stored back in memory
- Each of these operations require memory addresses
Memory Hierarchy

- Registers
- On-chip Cache
- Main Memory
- Disk

- Registers built into the CPU are generally accessible within one cycle of the CPU clock
- Completing a memory access may take many cycles of the CPU clock
Memory Hierarchy

- A CPU waiting for data from main memory is not desired
- Remedy: Add fast memory between the CPU and main memory called a cache
Memory Management

- Memory is to be shared by multiple processes
- Processes should not be able to reference another process’s memory without permission
  - Each process is given a range of legal addresses that the process may access
Address Binding

- Program execution requires that a program be brought into memory from the disk.
- The process can reside in any part of the physical memory.
- Address space starts at 00000 but the first address of a user/process need not be 00000.
Address Binding

- User program goes through several steps before it is ready to be executed.
- The binding of instructions/data to memory addresses can be done at any step along the way.
Address Binding

- Addresses in the source program are generally symbolic, e.g., count
- Compiler will bind these symbolic addresses to relocatable addresses
The loader will in turn bind the relocatable addresses to absolute addresses.
Address Binding

- **Execution time:**
  - Binding to actual physical memory addresses must be delayed until runtime.
Address Binding

- Typically the programmer does not know where the program will be placed in memory when it is executed.

- While the program is executing, it may be placed in disk and returned to main memory at a different location (swapping).
Address Binding

- Memory references in the code (virtual or logical) must be translated to actual physical memory addresses.

- Run-time Mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU).

- Run-time mappings depend on how memory is allocated to processes.
Memory Allocation Techniques

- Contiguous Memory Allocation
- Paging Concepts
In this section we will discuss approaches to allocating memory to processes

- Contiguous
- Paging
Contiguous Memory Allocation
Contiguous Memory Allocation

- We will start out with the most basic method used that allows multiple processes to reside in memory.
- With contiguous memory allocation each process is contained in a single section of memory that is contiguous.
  - Fixed and Dynamic Partitioning
Fixed Partitioning

- Any program, no matter how small, occupies an entire partition.

- Equal-size partitions

<table>
<thead>
<tr>
<th>Operating System</th>
<th>8 M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 M</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
</tbody>
</table>
Fixed Partitioning

- Unequal-size partitions
  - Lessens the problem with equal-size partitions
Placement Algorithm with Partitions

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

- Unequal-size partitions
  - Each process can be assigned to the smallest partition within which it will fit
  - There is a queue for each partition or a queue for all partitions
  - Processes are assigned in such a way as to minimize wasted memory within a partition
One Process Queue per Partition
One Common Process Queue

- When its time to load a process into main memory the smallest available partition that will hold the process is selected.
Simple MMU for user program
Address Mapping

- The **base** register holds the physical memory address
- The **limit** register specifies the range
- These registers can be loaded only by the operating system
- Ensures the user program doesn’t access anything beyond the range
Simple MMU for Address Mapping

- Simple mapping from logical addresses to physical
- Relocation register: Value is added to every address generated by a user process
Questions

?? Single Queue:
  ?? What is the problem with taking the first job that fits into the partition?
  ?? What is the problem with taking the largest job that fits into the partition?

?? Multiple Queues:
  ?? What is the disadvantage?
Multiprogramming with Fixed Partitions

- Was used by OS/360 on large IBM mainframes for many years
- Incoming jobs were queued until a suitable partition was available
- Today no modern OS uses fixed partitions
Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Leads to external fragmentation
- **Compaction** is required to obtain a large block at the end of memory
  - Shift processes so they 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
  - Best-fit, First-fit, and Next-fit algorithms

- Best-fit algorithm
  - Choose the block that is closest in size to the request
  - This has the worst overall performance
  - The smallest block is found for a process
    - The smallest amount of fragmentation is left
    - Memory compaction must be done more often
Dynamic Partitioning Placement Algorithm

- **First-fit algorithm**
  - Starts scanning memory from the beginning and chooses the first available block that is large enough.

- **Next-fit**
  - Starts scanning memory from the location of the last placement and chooses the next available block that is large enough.