CSCI-GA.2250-001

Operating Systems

Lecture 5: Memory Management I

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Programmer's dream

Memory

- Private
- Infinitely large
- Infinitely fast
- Non-volatile
- Inexpensive
Programmer’s Wish List

Memory

- Private
- Infinitely large
- Infinitely fast
- Non-volatile
- Inexpensive

Programs are getting bigger faster than memories.
Memory Hierarchy

- Cache (SRAM)
- Main Memory (DRAM)
- Disk Storage (Magnetic media)
Memory Hierarchy

1. Disk Storage (Magnetic media) - Managed by OS
2. Main Memory (DRAM) - Managed by OS
3. Cache (SRAM) - Usually managed by hardware
Memory Hierarchy

- Cache (SRAM): Usually managed by hardware
- Main Memory (DRAM): Managed by OS
- Disk Storage (Magnetic media): Managed by OS
Question: Who Cares About the Memory Hierarchy?

“Moore’s Law”

Processor-Memory Performance Gap:
(grows 50% / year)

CPU-DRAM Gap

µProc 60%/yr.

DRAM 7%/yr.
Memory Abstraction

• The hardware and OS memory manager makes you see the memory as a single contiguous entity

• How do they do that?
  – Abstraction

Is abstraction necessary?
No Memory Abstraction

Even with no abstraction, we can have several setups!
No Memory Abstraction

Only one process at a time can be running (threads??)
No Memory Abstraction

• What if we want to run multiple programs?
  – OS saves entire memory on disk
  – OS brings next program
  – OS runs next program

• We can use swapping to run multiple programs concurrently
  – Memory divided into blocks
  – Each block assigned protection bits
  – Program status word contains the same bits
  – Hardware needs to support this
  – Example: IBM 360

Swapping
No Memory Abstraction
No Memory Abstraction

Using absolute address is wrong here
No Memory Abstraction

We can use static relocation at program load time

Using absolute address is wrong here
No Memory Abstraction

We can use static relocation at program load time

**Bad Idea!**
- Slow
- Require extra info from program

Bottom line: Memory abstraction is needed!
Memory Abstraction

• To allow several programs to co-exist in memory we need
  – Protection
  – Relocation
  – Sharing
  – Logical organization
  – Physical organization

• A new abstraction for memory: **Address Space**

• Address space = set of addresses that a process can use to address memory
Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
Relocation

• Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program

• Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization

• Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  – may need to relocate the process to a different area of memory
Sharing

• It is advantageous to allow each process access to the same copy of the program rather than have their own separate copy

• Memory management must allow controlled access to shared areas of memory without compromising protection
Logical Organization

• We see memory as linear one-dimensional address space.

• A program = code + data + ... = modules

• Those modules must be organized in that logical address space
Physical Organization

• Memory is really a hierarchy
  – Several levels of caches
  – Main memory
  – Disk

• Managing the different modules of different programs in such a way as:
  – To give illusion of the logical organization
  – To make the best use of the above hierarchy
Address Space: 
Base and Limit

- Map each process address space onto a different part of physical memory
- Two registers: Base and Limit
  - **Base**: start address of a program in physical memory
  - **Limit**: length of the program
- For every memory access
  - Base is added to the address
  - Result compared to Limit
- Only OS can modify Base and Limit
Address Space:  
Base and Limit

Main drawback:
Need to add and compare for each memory address

What if memory space is not enough for all programs?
Address Space: Base and Limit

Main drawback:
Need to add and compare for each memory address

What if memory space is not enough for all programs?

The we may need to swap some programs out of the memory.
Swapping
Swapping

• Programs move in and out of memory
• Holes are created
• Holes can be combined -> memory compaction

• What if a process needs more memory?
  – If a hole is adjacent to the process, it is allocated to it
  – Process has to be moved to a bigger hole
  – Process suspended till enough memory is there
Managing Free Memory

Bitmap

Linked List
Managing Free Memory

Bitmap

Linked List

Slow: To find k-consecutive 0s for a new process
Managing Free Memory: Linked List

- Linked list of allocated and free memory segments
- More convenient be double-linked list
Managing Free Memory: Linked List

• How to allocate?
  – First fit
  – Best fit
  – Next fit
  – Worst fit
  – ...

Memory Management Techniques

- Memory management brings processes into main memory for execution by the processor
  - involves virtual memory
  - based on segmentation and paging
<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Partitioning</td>
<td>Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.</td>
<td>Simple to implement; little operating system overhead.</td>
<td>Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.</td>
</tr>
<tr>
<td>Dynamic Partitioning</td>
<td>Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.</td>
<td>No internal fragmentation; more efficient use of main memory.</td>
<td>Inefficient use of processor due to the need for compaction to counter external fragmentation.</td>
</tr>
<tr>
<td>Simple Paging</td>
<td>Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.</td>
<td>No external fragmentation.</td>
<td>A small amount of internal fragmentation.</td>
</tr>
<tr>
<td>Simple Segmentation</td>
<td>Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.</td>
<td>No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.</td>
<td>External fragmentation.</td>
</tr>
<tr>
<td>Virtual Memory Paging</td>
<td>As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.</td>
<td>No external fragmentation; higher degree of multiprogramming; large virtual address space.</td>
<td>Overhead of complex memory management.</td>
</tr>
<tr>
<td>Virtual Memory Segmentation</td>
<td>As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.</td>
<td>No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.</td>
<td>Overhead of complex memory management.</td>
</tr>
</tbody>
</table>
Conclusions

• Process is CPU abstraction
• Address space is memory abstraction
  • OS memory manager and the hardware helps providing this abstraction
• Two main tasks needed from OS regarding memory management:
  – managing free space
  – making best use of the memory hierarchy