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 dream

Programs are getting bigger faster than memories.

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

Memory
Memory Hierarchy

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

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

- **Cache (SRAM)**: Usually managed by hardware
- **Main Memory (DRAM)**: Managed by OS
- **Disk Storage (Magnetic media)**: Managed by OS

**Memory Manager**
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.
No Memory Abstraction

(a) User program
   Operating system in RAM
   0xFFF ...

(b) Operating system in ROM
    User program
    0

(c) Device drivers in ROM
    User program
    Operating system in RAM
    0
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 bit
  – Hardware needs to support this
  – Example: IBM 360
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

Using absolute address is wrong here
Memory Abstraction

- To allow several programs to co-exist in memory we need
  - Protection
  - Relocation
- A new abstraction for memory: Address Space
- Address space = set of addresses that a process can use to address memory
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?
Swapping

(a) Operating system

(b) Operating system

(c) Operating system

(d) Operating system

(e) Operating system

(f) Operating system

(g) Operating system
Swapping

- Programs move in and out of memory
- Holes are created
- Holes can be combined \rightarrow 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

(a) Bitmap

(b) Linked List

Hole Starts at 18 Length 2

Process
Managing Free Memory

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

- Linked list of allocated and free memory segments
- More convenient be double-linked list

![Diagram showing memory management before and after termination of X]
Managing Free Memory: Linked List

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

Virtual Memory

• Each program has its own address space
• This address space is divided into pages
• Pages are mapped into physical memory
Virtual Memory
Virtual Memory

• Main memory can act as a cache for the secondary storage (disk)

• Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection
Physical address (an address in main memory)
Virtual address

31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Virtual page number  Page offset

Translation

Page table inside MMU

Physical address  (an address in main memory)

Physical page number  Page offset
Structure of a Page Table Entry
Speeding Up Paging

- Challenges:
  - Mapping virtual to physical address must be fast
  - If address space is large, page table will be large
Speeding Up Paging

• Challenges:
  – Mapping virtual to physical address must be fast
  – If address space is large, page table will be large

  Translation Lookaside Buffer (TLB)

  • Multi-level page table
  • Inverted page table
**TLB**

- **Observation:** most programs tend to make a large number of references to a small number of pages -> only fraction of the page table is heavily used

- **TLB**
  - Hardware device inside the MMU
  - Maps virtual to physical address without going to the page table
TLB

- In case of TLB miss -> MMU accesses page table
- TLB misses occur more frequently than page faults
- Optimizations
  - Software TLB management
    - Simpler MMU
# TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virtual page</th>
<th>Modified</th>
<th>Protection</th>
<th>Page frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>1</td>
<td>RW</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>R X</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>1</td>
<td>RW</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
<td>1</td>
<td>RW</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
<td>R X</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0</td>
<td>R X</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>860</td>
<td>1</td>
<td>RW</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>861</td>
<td>1</td>
<td>RW</td>
<td>75</td>
</tr>
</tbody>
</table>
Multi-Level Page Table

- To reduce storage overhead in case of large memories
Inverted Page Table

• One entry per page frame
  + Save vast amount of storage
  - virtual-to-physical translation much harder
Conclusions

- Process is CPU abstraction
- Address space is memory abstraction
- Many optimizations
  - TLB
  - Multi-level page table
  - Inverted page table
- We covered from beginning of chap 3 till 3.3