CS 423
Operating System Design: Virtual Memory Mgmt

Professor Adam Bates
Spring 2017
Goals for Today

- **Learning Objective:**
  - Navigate the history of memory systems in OS
- **Announcements, etc:**
  - 4 Credit Project Preferences due today!
  - MP2 description is now available on compass. Due 3/8

Reminder: Please put away devices at the start of class.
Storage Hierarchy

- CPU Registers: 32-64 bits
- Cache: 4-128 words
- Memory: 512-16k words
- Secondary Storage

Performance vs. Size:

- Performance
  - CPU Registers
  - Cache
- Size
  - 32-64 bits
  - 4-128 words
  - 512-16k words
We have limited amounts of fast resources, and large amounts of slower resources…

*How to create the illusion of an abundant fast resource?*
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space.
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
Overlay Manager

Main Program

Overlay Area

Overlay 1

Overlay 2

Overlay 3

Secondary Storage

Used when process memory requirement exceeded the physical memory space
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
History: Mem Overlays

Used when process memory requirement exceeded the physical memory space
• Approach: Multiprogramming with fixed memory partitions
• Divides memory into $n$ fixed partitions (possible unequal)
• Problem?

History: Fixed Partitions

Free Space

0k
4k
16k
64k
128k
History: Fixed Partitions

- Approach: Multiprogramming with fixed memory partitions
- Divides memory into $n$ fixed partitions (possible unequal)
- Problems?
• Approach: Multiprogramming with fixed memory partitions
• Divides memory into $n$ fixed partitions (possible unequal)
• Problems?
  • Internal Fragmentation! Also,
  • Level of Multiprogramming
Placement Algorithms for Fixed Partitions

- Trivial for equal size partitions. For unequal...

- Multiple Queues:
  - Assign (i.e., enqueue) each incoming job to the smallest partition within which it fits
  - Decreases fragmentation
  - when the queue for a large partition is empty but the queue for a small partition is full. Small jobs have to wait to get into memory even though plenty of memory is free.

- Single Queue:
  - Assign each process to the smallest available partition within which it fits
  - Increases amount of multiprogramming on the expense of fragmentation
History: Relocation

- Correct starting address when a program should start in the memory
- Different jobs will run at different addresses
  - When a program is linked, the linker must know at what address the program will begin in memory.
- Logical addresses
  - Logical address space, range (0 to max)
  - Physical addresses, Physical address space range (R+0 to R+max) for base value R.
  - User program never sees the real physical addresses
- Relocation register
  - Mapping requires hardware with the base register
Relocation => "Variable Partition Allocation"
History: Variable Partition Allocation

1. Monitor | Job 1 | Job 2 | Job 3 | Job 4 | Free
2. Monitor | Job 1 | Job 3 | Job 4 | Free
3. Monitor | Job 1 | Job 5 | Job 3 | Job 4 | Free
4. Monitor | Job 5 | Job 3 | Job 4 | Job 6
5. Monitor | Job 7 | Job 5 | Job 3 | Job 8 | Job 6

Memory wasted by External Fragmentation
History: Storage Placement Strategy

- **Best Fit**
  - Use the hole whose size is equal to the need, or if none is equal, the hole that is larger but closest in size.
  - Problem: Creates small holes that can't be used.

- **Worst Fit?**
  - Use the largest available hole.
  - Problem: Gets rid of large holes making it difficult to run large programs.

- **First Fit**
  - Use the first available hole whose size is sufficient to meet the need.
  - Problem: Creates average size holes.

- **Next Fit.**
  - Minor variation of first fit: search from the last hole used.
  - Problem: slightly worse performance than first fit.
Virtual Memory

- Provide user with virtual memory that is as big as user needs
- Store virtual memory on disk
- Cache parts of virtual memory being used in real memory
- Load and store cached virtual memory without user program intervention
Paging

Memory
1 2 3 4

Page Table
VM Frame
1
2
3
4

Virtual Memory Stored on Disk
1 2 3 4 5 6 7 8
Request Page 3…

Virtual Memory Stored on Disk

Memory

Page Table
VM Frame

1 2 3 4

3 1
2
3
4
Request Page 1...

Virtual Memory Stored on Disk

Memory

Page Table

VM
Frame

1
2
3
4
5
6
7
8

1
2
3
4

CS 423: Operating Systems Design
Paging

Request Page 6...

Memory

Page Table
VM Frame

Virtual Memory Stored on Disk

1 2 3 4

3 1
1 2
6 3
4

1 2 3 4 5 6 7 8
Paging

Request Page 2

Virtual Memory Stored on Disk

Memory

Page Table VM Frame

1 2 3 4

3 1

1 2

6 3

2 4
Paging

Request Page 8. Swap Page 1 to Disk First…

Virtual Memory Stored on Disk

Memory

Page Table
VM Frame

1 2 3 4

3 1
2 2
6 3
2 4

CS 423: Operating Systems Design
Request Page 8. ... now load Page 8 into Memory.
Page Mapping Hardware

**Page Table**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

**Virtual Address (P,D)**

- P
- D

**Physical Address (F,D)**

- F
- D

**Virtual Memory**

- P
- D

- Contents(P,D)

- Physical Memory

- F
- D

- Contents(F,D)
Page Mapping Hardware

Page Table

0
1
0
1
1
0
1

Virtual Address (004006)

004 006

Physical Address (F,D)

005 006

Virtual Memory

Contents(4006)

Physical Memory

Contents(5006)

Page size 1000
Number of Possible Virtual Pages 1000
Number of Page Frames 8
Page Faults

- Access a virtual page that is not mapped into any physical page
  - A fault is triggered by hardware
- Page fault handler (in OS’s VM subsystem)
  - Find if there is any free physical page available
    - If no, evict some resident page to disk (swapping space)
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table
Paging Issues

- Page size is $2^n$
  - usually 512 bytes, 1 KB, 2 KB, 4 KB, or 8 KB
  - E.g. 32 bit VM address may have $2^{20}$ (1 MB) pages with 4k ($2^{12}$) bytes per page

- Page table:
  - $2^{20}$ page entries take $2^{22}$ bytes (4 MB)
  - Must map into real memory
  - Page Table base register must be changed for context switch

- No external fragmentation; internal fragmentation on last page only
Optimization:

Virtual address

VPage # offset

VPage# PPage# ...
VPage# PPage# ...
...
VPage# PPage# ...

TLB

Miss

Real page table

Hit

PPage # offset

Physical address
If a virtual address is presented to MMU, the hardware checks TLB by comparing all entries simultaneously (in parallel).

- If match is valid, the page is taken from TLB without going through page table.
- If match is not valid
  - MMU detects miss and does a page table lookup.
  - It then evicts one page out of TLB and replaces it with the new entry, so that next time that page is found in TLB.
Issues:

- What TLB entry to be replaced?
  - Random
  - Least Recently Used (LRU)

- What happens on a context switch?
  - Invalidate the entire TLB contents

- What happens when changing a page table entry?
  - Change the entry in memory
  - Invalidate the TLB entry
Translation Lookaside Buffers

Effective Access Time:

- TLB lookup time = $\sigma$ time unit
- Memory cycle = $m$ $\mu$s
- TLB Hit ratio = $\eta$
- Effective access time
  - $Eat = (m + \sigma)\eta + (2m + \sigma)(1 - \eta)$
  - $Eat = 2m + \sigma - m\eta$

*Note: Doesn’t consider page faults. How would we extend?*