CS 423
Operating System Design: Historical Memory Management

Professor Adam Bates
Goals for Today

• Learning Objective:
  • Explore historical strategies for memory management

• Announcements, etc:
  • MP1 is out! Due TODAY
  • Midterm Exam — Wednesday March 6th (in-class)
  • C4 Readings! Due FRIDAY

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 increases as you move up the hierarchy.
- Size increases as you move down the hierarchy.
Problem Statement

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

Overlay Manager
Overlay Area
Main Program

Secondary Storage
Overlay 1
Overlay 2
Overlay 3

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

History: Mem Overlays

Overlay Manager

Main Program

Overlay 1

Overlay 2

Overlay 3

Secondary Storage

Overlay 3
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 (possibly unequal)
• Problem?
• Approach: Multiprogramming with fixed memory partitions
• Divides memory into \( n \) fixed partitions (possible unequal)
• Problem?
• Approach: Multiprogramming with fixed memory partitions
• Divides memory into $n$ fixed partitions (possible unequal)
• Problem?
  • Internal Fragmentation
History: Fixed Partition Allocation

- Separate input queue for each partition
  - Sorting incoming jobs into separate queues
  - Inefficient utilization of memory
    - 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.

- One single input queue for all partitions.
  - Allocate a partition where the job fits in.
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.

- Enter “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
History: Relocation Register

- CPU Instruction Address
- Logical Address (MA)
- Base Register (BA)
- Physical Address (MA + BA)
- Memory

The diagram illustrates the process of converting a logical address to a physical address using a base register. The logical address (MA) is added to the base address (BA) to obtain the physical address (MA + BA).
History: Variable Partition Allocation

1. Monitor, Job 1, Job 2, Job 3, Job 4, Free

Memory wasted by External Fragmentation
History: Storage Placement Strategy

- **Best Fit?**

- **First Fit?**

- **Next Fit?**

- **Worst 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.

- **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.

- **Worst Fit?**

  - Use the largest available hole.

  - Problem: Gets rid of large holes making it difficult to run large programs.
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

Request Page 3...

Virtual Memory Stored on Disk

Memory

Page Table

VM Frame

1 2 3 4

3 1
2
3
4
Paging

Request Page 1…

Virtual Memory Stored on Disk

Memory

Page Table
VM Frame

1 2 3 4

3 1
1 2
1 3
1 4
Request Page 6…

Memory

Virtual Memory Stored on Disk

Page Table
VM Frame

Request Page 6…
Paging

Request Page 2...

Memory

Virtual Memory Stored on Disk

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

1 2 3 4 5 6 7 8
Request Page 8. ... now load Page 8 into Memory.
Note: Virtual Memory also supports shared pages.
Page Mapping Hardware

Virtual Address (P,D)

Page Table

Virtual Memory

Physical Memory

Contents(P,D)

Contents(F,D)

Physical Address (F,D)

Page Table

P  D

0 0
1 1
0 1
1 1
0 0
1 1

F  D

4

CS 423: Operating Systems Design
Page Mapping Hardware

Virtual Address (004006)

Page Table

0
1
0
1
1
0
1

Physical Address (F,D)

004 005 006

Virtual Memory

Contents(4006)

Physical Memory

Contents(5006)

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

- Occur when we 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
Reasoning about Page Tables

- On a 32 bit system we have $2^{32}$ B virtual address space
  - i.e., a 32 bit register can store $2^{32}$ values
- # of pages are $2^n$ (e.g., 512 B, 1 KB, 2 KB, 4 KB...)
- Given a page size, how many pages are needed?
  - e.g., If 4 KB pages ($2^{12}$ B), then $2^{32}/2^{12} = ...$
    - $2^{20}$ pages required to represent the address space
- **But!** each page entry takes more than 1 Byte of space to represent.
  - suppose page size is 4 bytes (Why?)
  - $(2*2) * 2^{20} = 4$ MB of space required to represent our page table in physical memory.
- What is the consequence of this?