CS 423
Operating System Design: Historical Memory Management

Tianyin Xu (MIC)

* Thanks for Prof. Adam Bates for the slides.
Storage Hierarchy

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

*How to create the illusion of an abundant fast resource?*
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.
Overlay Manager

Overlay 2

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

Used when process memory requirement exceeded the physical memory space
Used when process memory requirement exceeded the physical memory space.
Overlay Manager
Overlay 1
Secondary Storage
Overlay 1
Overlay 2
Overlay 3

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

![Fixed Partition Allocation Diagram]

Program 1: 4k
Program 2: 16k
Program 3: 64k
Free Space: 128k

History: Fixed Partition Allocation
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
History: Variable Partition Allocation

Memory wasted by **External Fragmentation**
Best Fit?
Use the hole whose size is equal to the need, or if none is.

First Fit?
Use the first available hole whose size is sufficient to meet

Next Fit?
Minor variation of first fit approach from the last hole used

Worst Fit?
Use the largest available hole.
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…

Memory

Virtual Memory Stored on Disk

Page Table
VM Frame

3 1
2
3
4
Request Page 1…

Virtual Memory Stored on Disk

Memory

1 2 3 4

Page Table

VM Frame

3 1
1 2
3
4

Paging
Paging

Request Page 6…

Memory

Page Table
VM Frame

Virtual Memory Stored on Disk
Paging

Request Page 2…

Memory

Virtual Memory Stored on Disk

Page Table

VM Frame

1 2 3 4

1 2

6 3

2 4
Paging

Request Page 8. Swap Page 1 to Disk First…
Paging

Request Page 8. ... now load Page 8 into Memory.
Note: Virtual Memory also supports shared pages.
Page Mapping Hardware

Virtual Address (P,D)

Virtual Memory

Contents(P,D)

Physical Memory

Contents(F,D)

Page Table

Physical Address (F,D)

4

0

1

0

1

1

0

1

0

1

P → F

F D

P D

CS 423: Operating Systems Design
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

- 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?)
  - $(2\times2) \times 2^{20} = 4$ MB of space required to represent our page table in physical memory.
- What is the consequence of this?
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