Memory Management

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Today’s Topics

- Why is memory management difficult?

- Old memory management techniques:
  - Fixed partitions
  - Variable partitions
  - Overlays
  - Swapping
Memory Management (1)

- **Goals**
  - To provide a convenient abstraction for programming.
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead.
  - To provide isolation between processes.

- **Why is it so difficult?**
- An OS with one user process
  - Programs use physical addresses directly.
  - OS loads job, runs it, unloads it.
Multiprogramming

- Need multiple processes in memory at once.
  - To overlap I/O and CPU of multiple jobs
  - Each process requires variable-sized and contiguous space.
- Requirements
  - Protection: restrict which addresses processes can use.
  - Fast translation: memory lookups must be fast, in spite of protection scheme.
  - Fast context switching: updating memory hardware (for protection and translation) should be quick.
Fixed Partitions (1)

Partition 0

Base register
0x2000

Virtual address
0x0362

0x2362

Partition 1
Partition 2
Partition 3
Partition 4

Operating System

0x1000
0x2000
0x3000
0x4000
0x5000
0x1000
Fixed Partitions (2)

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - the number of partitions = degree of multiprogramming
  - Hardware requirements: base register
    - Physical address = virtual address + base register
    - Base register loaded by OS when it switches to a process

- Advantages
  - Easy to implement, fast context switch

- Problems
  - Internal fragmentation: memory in a partition not used by a process is not available to other processes
  - Partition size: one size does not fit all
    - Fragmentation vs. fitting large programs
## Fixed Partitions (3)

### Improvement

- Partition size need not be equal.
- Allocation strategies
  - Maintain a separate queue for each partition size
  - Maintain a single queue and allocate to the closest job whose size fits in an empty partition (first fit)
  - Search the whole input queue and pick the largest job that fits in an empty partition (best fit)

- IBM OS/MFT
  (Multiprogramming with a Fixed number of Tasks)

<table>
<thead>
<tr>
<th>Partition</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x1000</td>
</tr>
<tr>
<td>0</td>
<td>0×2000</td>
</tr>
<tr>
<td>1</td>
<td>0×4000</td>
</tr>
<tr>
<td>2</td>
<td>0×8000</td>
</tr>
</tbody>
</table>
Variable Partitions (1)

Virtual address offset

Limit register
P1’s Limit

Base register
P1’s Base

<?

Yes

No

<?

No

Yes

 Partition 0

Operating System

Partition 1

Partition 2

Partition 3

Virtual address

Partition 0

Partition 1

Partition 2

Partition 3

Virtual address offset

Limit register
P1’s Limit

Base register
P1’s Base

<?

Yes

No

<?

No

Yes

Partition 0

Operating System
Variable Partitions (2)

- **Physical memory is broken up into variable-sized partitions**
  - IBM OS/MVT
  - Hardware requirements: base register and limit register
    - Physical address = virtual address + base register
    - Base register loaded by OS when it switches to a process
  - The role of limit register: protection
    - If (physical address > base + limit), then raise a protection fault.

- **Allocation strategies**
  - First fit: Allocate the first hole that is big enough
  - Best fit: Allocate the smallest hole that is big enough
  - Worst fit: Allocate the largest hole
Variable Partitions (3)

- **Advantages**
  - No internal fragmentation
    - Simply allocate partition size to be just big enough for process
    - But, if we break the physical memory into fixed-sized blocks and allocate memory in unit of block sizes (in order to reduce bookkeeping), we have internal fragmentation.

- **Problems**
  - External fragmentation
    - As we load and unload jobs, holes are left scattered throughout physical memory
  - Solutions to external fragmentation:
    - Compaction
    - Paging and segmentation
Overlays (1)

- Overlays for a two-pass assembler
Overlays (2)

- **Overlays**
  - Keep in memory only those instructions and data that are needed at any given time.
  - Normally implemented by user

- **Advantages**
  - Needed when a process is larger than the amount of memory allocated to it.
  - No special support needed from operating system.

- **Problems**
  - Programming design of overlay structure is complex.
Swapping (1)

1. Swap out
2. Swap in
Swapping (2)

Swapping

- A process can be swapped temporarily out of memory to a backing store and then brought back into memory later for continued execution.
- Backing store
  - Fast disk large enough to accommodate copies of all memory images for all users
  - Must provide direct access to these memory images
- Major part of swap time is transfer time.
  - Directly proportional to the amount of memory swapped.
- Swapping a process with a pending I/O
  - Do not swap a process with pending I/O
  - Execute I/O operations only into OS buffers