Operating Systems

Memory management

Memory Management

List of Topics
1. Memory Management
2. Memory In Systems Design
3. Binding Times
4. Introduction to Memory Management
5. Raw Memory Model
6. Single User Contiguous Memory
7. Relocation - Why and How
8. Overlay Management

Topics (continued)
9. Protection
10. Fixed Partitions
11. Nonuniform Sized Fixed Partitions
12. Uniformly Sized Fixed Partitions
13. Simple Paging
14. Benefits of Simple Paging
15. Page Tables
16. Translation Lookaside Buffers
17. Hierarchical Address Caching
Topics (continued)
18. Dynamic Partitions
19. Fragmentation
20. Internal Fragmentation
21. External Fragmentation
22. Coalescing Holes
23. Compaction
24. Dynamic Partition Placement
25. Simple Segmentation
26. Memory Layout of A C Program

Memory In Systems design

Figure 1: memory Connects CPU and Peripherals
Introduction to Memory Management

Memory management issues:
1. Relocation
2. Protection
3. Sharing
4. Logical Organization
5. Physical Organization

Consider a series of solutions starting with the most primitive first

Raw Memory Model

The raw memory provides no services and gives the programmer complete control.

Figure 3: Raw Machine Model

Single User Contiguous Memory

Primitive operating systems (such as MS-DOS and CP/M) provide some interfaces to the hardware but not much else in the way of services.
Relocation - Why and How

Relocation refers to the ability to store a program at an arbitrary base memory address.

Actual memory locations have physical or absolute addresses, user program's access these locations using logical addresses.
Overlay Management

Overlays have gone out of fashion with cheaper memory, users (and compilers) determine which code to swap in and out.

![Overlay Management Example][2]

**Protection**

It is undesirable to permit user programs (accidentally or intentionally) to accesses memory outside of their partition.
Protection (continued)

Figure 7: Protection in Resident Monitor Model

Fixed Partitions

*Fixed partitioning* refers to memory being split into contiguous non-overlapping regions of precomputed sizes.

Fixed sized partitions make the selection of a partition for a job easy.

Nonuniform Sized Fixed Partitions

Fixed Partitions may have differing sizes.
Uniformly Sized Fixed Partitions

Memory is frequently partitioned into uniformly sized regions.

Figure 8: Nonuniform fixed Partitions [3]

Figure 9: Uniform Fixed Partition Allocation [3]
Simple Paging

Paging provides relocation, and splits memory into fixed length partitions called frames.

Figure 10: Simple Paging [1]
<table>
<thead>
<tr>
<th>Page frame number</th>
<th>Page frame size</th>
<th>Range of real storage addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p</td>
<td>0 → p-1</td>
</tr>
<tr>
<td>1</td>
<td>p</td>
<td>p → 2p-1</td>
</tr>
<tr>
<td>2</td>
<td>p</td>
<td>2p → 3p-1</td>
</tr>
<tr>
<td>3</td>
<td>p</td>
<td>3p → 4p-1</td>
</tr>
<tr>
<td>4</td>
<td>p</td>
<td>4p → 5p-1</td>
</tr>
<tr>
<td>5</td>
<td>p</td>
<td>5p → 6p-1</td>
</tr>
<tr>
<td>6</td>
<td>p</td>
<td>6p → 7p-1</td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>7p → 8p-1</td>
</tr>
</tbody>
</table>

Benefits of Simple Paging

Simple paging allows discontiguous storage for memory objects exceeding the page frame size.
Page Tables

One simple mechanism is to allocate some real memory space for a table, and hash page addresses using the high order address bits as pointers into the page table. There are 2 real memory accesses per virtual memory access.

Translation Lookaside Buffers

Translation lookaside buffers (TLB) eliminate one physical memory reference using special associative memory, which addressed by its contents in \(O(1)\) parallel search time.
Hierarchical Address Caching

Rather than placing all addresses in the TLB recently/frequently used addresses are stored in associative memory, with misses being serviced by the page table.

Hierarchical Address Caching (continued)
Dynamic Partitions

Dynamically partitioned memory allows placement of relocatable code in variable size contiguous memory regions.

Dynamic Partitions (continued)

<table>
<thead>
<tr>
<th>User needs</th>
<th>9K</th>
<th>18K</th>
<th>11K</th>
<th>32K</th>
<th>11K</th>
<th>32K</th>
<th>10K</th>
<th>20K</th>
<th>15K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>A</td>
<td>15K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>A</td>
<td>15K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>A</td>
<td>15K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fragmentation

Fragmentation makes available memory useless by breaking it into discontiguous pieces too small to use.
Fragmentation (continued)

There are two categories of memory fragmentation:
1. Internal Fragmentation — A fixed partition contains more memory than required by the user, and some is wasted.
2. External Fragmentation — Results from the holes left by dynamic partitions.

Internal Fragmentation

Internal fragmentation occurs when fixed size partitions are too large.

Figure 16: Internal Fragmentation [3]
External Fragmentation

External fragmentation happens when dynamic partitions are released. The fragments are frequently called holes.

Coalescing Holes

Adjacent holes in dynamic partitions should be coalesced into a single larger hole.
Compaction

If the amount of memory available in the holes is large enough to service a request, the holes may made contiguous by compacting storage.
Dynamic Partition Placement

(a) First-Fit Strategy
Place job in first storage hole on free storage place list in which it will fit

(b) Best-Fit Strategy
Place job in the smallest possible hole in which it will fit.
(Kept in ascending order by hole size.)

Free Storage List

<table>
<thead>
<tr>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>14K</td>
</tr>
<tr>
<td>e</td>
<td>5K</td>
</tr>
<tr>
<td>a</td>
<td>16K</td>
</tr>
<tr>
<td>g</td>
<td>30K</td>
</tr>
</tbody>
</table>

Request for 13K

In use

Operating systems

16K hole

14K hole

5K hole

30K hole

(c) Worst-Fit Strategy

Place job in the largest possible hole in which it will fit.

(Kept in descending order by hole size.)

Free Storage List

<table>
<thead>
<tr>
<th>Start</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>30K</td>
</tr>
<tr>
<td>a</td>
<td>16K</td>
</tr>
<tr>
<td>c</td>
<td>14K</td>
</tr>
<tr>
<td>e</td>
<td>5K</td>
</tr>
</tbody>
</table>

Request for 13K

In use

Operating systems

16K hole

14K hole

5K hole

30K hole
Simple Segmentation

Segmentation provides relocation, and supports contiguous variable length partitions.

Segmentation often provides protection (counterexample Intel 8086).

Memory Layout of A C Programs

Traditional Unix/C memory images of programs use segments.
Programmers often want to allocate data objects which persist beyond the function call creating them (e.g. constructors in OOP).

In C and C++, the `malloc` operator maintains a linked list of data objects, in the user program's Data segment (on the Heap).

**malloc**

Programmers often want to allocate data objects which persist beyond the function call creating them (e.g. constructors in OOP).

In C and C++, the `malloc` operator maintains a linked list of data objects, in the user program's Data segment (on the Heap).
**sbrk**

A user program can exhaust its default heap space allocation.

The Unix sbrk system call increases data segment allocation at run time.

---

**Memory Hierarchy**

Users want to:
1. Increase their address space, using slow cheaper memory to extend their more expensive faster memory.

2. Increase the speed at which they can the extended memory by using small amounts of expensive fast memory.