Chapter 11

Memory Management

Main memory is a resource that must be allocated and deallocated

Memory Management Techniques determine:

- Where and how a process resides in memory
- How addressing is performed

Binding:

Identifiers --> compiled relative addresses (relative to 0)

--> physical addresses
Memory Management Techniques

1) Single Contiguous
2) Overlays
3) Fixed (Static) Partitions
4) Relocation (Dynamic) Partitions
5) Paging
6) Demand Paging
7) Segmented
8) Segmented / Demand Paging

For each technique, observe:
- Algorithms
- Advantages / Disadvantages
- Special Requirements
I. Single Contiguous

\[
\text{While ( job is ready ) Do} \\
\quad \text{If ( JobSize} \leq \text{ MemorySize )} \\
\quad \quad \text{Then Begin} \\
\quad \quad \quad \text{Allocate Memory} \\
\quad \quad \quad \text{Load and Execute Job} \\
\quad \quad \quad \text{Deallocate Memory} \\
\quad \quad \quad \text{End} \\
\quad \text{Else Error}
\]

Advantages:

- Simplicity
- No special hardware

Disadvantages:

- CPU wasted
- Main memory not fully used
- Limited job size
II. Overlays

- Programs can be sectioned into modules
- Not all modules need to be in main memory at the same time

```
  A
 /   \
 B     E
|     |
C     D
```

- Programmer specifies which modules can overlay each other
- Linker inserts commands to invoke the loader when the modules are referenced
- The "parent" must stay in memory
- Used in DOS as an alternative to Expanded Memory.

Illustration of Overlays

Program Component: A B C D E

Memory: 40K 30K 10K 10K 40K

Without Overlays

```
Without Overlays
  0  40  70  130
A
B
C
D
E
```

With Overlays

```
With Overlays
  0  40  80
A
B
C
D
E
```
Overlays ...

**Advantages:**
- Reduced memory requirements

**Disadvantages:**
- Overlap map must be specified by programmer
- Programmer must know memory requirements
- Overlapped modules must be completely disjoint

---

Fixed (Static) Partitioning with Absolute Translation

- Earliest attempt at multiprogramming
- Partition memory into fixed sized areas:

```
0M → Partition #1 → 6M
    ↓ Partition #2 → 2M
    ↓ Partition #3 → 8M
16M →
```
Fixed (Static) Partitioning with Absolute Translation ...

- Each partition can hold ONE process
- Code generated using an ABSOLUTE address reflecting the starting address of the partition in which it is supposed to execute (relative to 0, 6M, or 8M in picture)
- Queue of processes waiting for each partition

---

Fig. 7.6 Fixed partition multiprogramming with absolute translation and loading.
Fragmentation is a situation in which the free cells in main memory are not contiguous.

**Internal fragmentation:**
A situation in which free memory cells are within the area allocated to a process

**External fragmentation:**
A situation in which free memory cells are not in the area allocated to any process
Fixed Partition Fragmentation

<table>
<thead>
<tr>
<th>Partition</th>
<th>Job Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>20K</td>
<td>20K</td>
</tr>
<tr>
<td>35K</td>
<td></td>
</tr>
<tr>
<td>25K</td>
<td>10K</td>
</tr>
<tr>
<td>30K</td>
<td>30K</td>
</tr>
</tbody>
</table>

External fragmentation: 35K partition
Internal fragmentation: 25-10 => 15K wasted inside 25K partition

Advantages:
- Simplicity
- Multiprogramming now possible
- Works with any hardware (8088, 68000, etc)
Fixed Partitioning with Absolute Translation: Pros/Cons ...

**Disadvantages:**

- Job Size \( \leq \) Max Partition Size \( \leq \) MM Size

- Storage wasted due to *internal fragmentation*:
  
  process size \( < \) partition size

- Storage wasted due to *external fragmentation*:
  
  A partition may be idle because none of the jobs assigned to it are being run

- Once compiled a job can *only* be executed in designated partition

---

Fixed (Static) Partitions with Relative Address Translation

- Allows process to run in *any* free partition

- *ALL* Code generated using addresses

  *relative to zero*
Fixed Partitions with Relative Address Translation...

Illustration:

Let:

- \( B \) denote base (absolute) address of a partition
- \( L \) denote partition length

QTP: Would Pointers work?

Multiprogramming Protection

Fixed partitions with relative addressing supports multiprogramming protection

\[ \Rightarrow \text{Ensure that one process does not access memory space dedicated to another process} \]

Method:

Each relative address is compared to the bounds register
**Multiprogramming Protection...**

```
B
B + L
```

- **Base Reg**
  - B

- **Bounds Reg**
  - B+L

- **“Virtual” Address**

- **OK**
- **T**
- **F**
- **Error:** Legal Address

---

**Fixed Partitioning with Relative Addressing: Pros/Cons**

- **Advantage compared to absolute addressing:**
  - Dynamic allocation of programs to partitions improves system performance

- **Still some disadvantages:**
  - Partition sizes are fixed at boot time
  - Can't run process larger than largest partition
  - Partition selection algorithm affects system performance
  - Still has internal and external fragmentation
Consider following scenario (100K memory):

1. Job 1 arrives; size= 22 K
2. Job 2 arrives; size= 24 K
3. Job 3 arrives; size= 30 K
4. Job 4 arrives; size= 10 K
5. Job 1 terminates
6. Job 3 terminates
7. Job 5 arrives; size= 12K

Where should job 5 be put?

---

Partition Selection Algorithms

- Implementation requires a free block table
- Sorting table in a particular manner results in a specific selection algorithm:
  1) First Fit -- Table sorted by location, searched top to bottom
  2) Best Fit -- Table Sorted by size (ascending)
     [don't break up big blocks]
  3) Worst Fit -- Table sort by size (descending)
     [break up big blocks]
  4) Next Fit
### Where does Job 5 Go?  
**First Fit**

<table>
<thead>
<tr>
<th></th>
<th><strong>Free List Table - First Fit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>FREE - 22 K</td>
</tr>
<tr>
<td>b</td>
<td>IN USE (J2) - 24 K</td>
</tr>
<tr>
<td>c</td>
<td>FREE - 30K</td>
</tr>
<tr>
<td>d</td>
<td>IN USE (J4) - 10 K</td>
</tr>
<tr>
<td>e</td>
<td>FREE - 14 K</td>
</tr>
</tbody>
</table>

**Start addr** | **Length**
--- | ---
 a | 22
 c | 30
 e | 14

7. Job 5 arrives; size=12K

---

### Where does Job 5 Go?  
**Best Fit**

<table>
<thead>
<tr>
<th></th>
<th><strong>Free List Table - Best Fit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>FREE - 22 K</td>
</tr>
<tr>
<td>b</td>
<td>IN USE (J2) - 24 K</td>
</tr>
<tr>
<td>c</td>
<td>FREE - 30K</td>
</tr>
<tr>
<td>d</td>
<td>IN USE (J4) - 10 K</td>
</tr>
<tr>
<td>e</td>
<td>FREE - 14 K</td>
</tr>
</tbody>
</table>

**Start addr** | **Length**
--- | ---
 e | 14
 a | 22
 c | 30

7. Job 5 arrives; size=12K
Where does Job 5 Go?
Worst Fit

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

7. Job 5 arrives; size=12K

Where does Job 5 Go? Next Fit

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

7. Job 5 arrives; size=12K
Dynamic partitions...

(a) Room for growth
Actually in use
B
Operating system

(b) Room for growth
Actually in use
A
Operating system

Memory Management with Bitmaps

- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list
Memory Management with Linked Lists

<table>
<thead>
<tr>
<th>Before X terminates</th>
<th>After X terminates</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) A X B</td>
<td>A X B</td>
</tr>
<tr>
<td>(b) A X</td>
<td>A</td>
</tr>
<tr>
<td>(c) X B</td>
<td></td>
</tr>
<tr>
<td>(d) X</td>
<td></td>
</tr>
</tbody>
</table>

Dynamic Partitions

Requires two OS operations:

- **Allocation:**
  
  Form a partition from a free partition of ample size

- **Deallocation:**
  
  Return partition to free table and *merge* where possible
Merge Example

Suppose b becomes free

<table>
<thead>
<tr>
<th></th>
<th>FREE - 22 K</th>
<th>IN USE - 24 K</th>
<th>FREE - 30K</th>
<th>IN USE - 10 K</th>
<th>FREE - 14 K</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Free List Table - *First Fit*

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

What does Free List Table look like?

CS3204 - Arthur

Merge Example

Suppose b becomes free

<table>
<thead>
<tr>
<th></th>
<th>FREE - 22 K</th>
<th>IN USE - 24 K</th>
<th>FREE - 30K</th>
<th>IN USE - 10 K</th>
<th>FREE - 14 K</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Free List Table - *Best Fit*

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>14</td>
</tr>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
</tr>
</tbody>
</table>

What does Free List Table look like?

CS3204 - Arthur
### Merge Example

Suppose b becomes free

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

What does Free List Table look like?

### Free List Table - *Worst Fit*

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

### Merge Example

Suppose b becomes free

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>22</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>14</td>
</tr>
</tbody>
</table>

What does Free List Table look like?

### Free List Table - *Next fit*
What if we cannot find a big enough hole for an arriving job?

Suppose a 35K job arrives?
Suppose a 90K job arrives?

What do you do?

<table>
<thead>
<tr>
<th>Free</th>
<th>22 K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24 K</td>
</tr>
<tr>
<td>Free</td>
<td>30 K</td>
</tr>
<tr>
<td></td>
<td>10 K</td>
</tr>
<tr>
<td>Free</td>
<td>14 K</td>
</tr>
</tbody>
</table>

Compaction

Shuffle jobs to create larger contiguous free memory

Job A 15 K
FREE
Job B 20 K
FREE
Job C 7K
FREE

A
B
C
FREE

15K
20K
7K
58K

Now 35 K job can run

QTP: How about pointers?

CS3204 - Arthur
Pros/Cons of Dynamic Partitions

😊 **Advantages:**
- Efficient memory usage

😊 **Disadvantages:**
- Partition Management
- Compaction *or* external fragmentation
- Internal fragmentation (if blocks composing partitions are always allocated in fixed sized units -- e.g. 2k)