Stacks and Heaps

- **Stack**: LIFO allocations/deallocations (*push* and *pop*)
  - Memory is allocated when a function, procedure or block is entered and is deallocated when it is exited

![Stack Diagram](image)

**Figure 11.7** Stack after (a) main calls sample; (b) sample calls calc.

Stacks and Heaps (continued)

float *floatptr1, *floatptr2;
int *intptr;
floatptr1 = (float *) malloc(5, sizeof(float));
floatptr2 = (float *) malloc(4, sizeof(float));
intptr = (int *) malloc(10, sizeof(int)),
free (floatptr2);

![Heap Diagram](image)

**Figure 11.8** (a) A heap; (b) A "hole" in the allocation when memory is deallocated.

- A heap permits random allocation/deallocation
  - Used for program-controlled dynamic data (PCD data)
The Memory Allocation Model

Memory Protection

- Memory protection uses *base* and *size* registers
  - *Memory protection violation* interrupt is raised if an address used in a program lies outside \([\text{base}, \text{base}+\text{size})\)
    - On processing interrupt, kernel aborts erring process
  - Kernel loads appropriate values into *base* and *size* registers while scheduling a process
    - Loading and saving are privileged instructions
  - When a *relocation* register is used, memory protection checks become simpler if every program has the linked origin of 0
    - Only need check whether address is smaller than *size*
Memory Fragmentation

Definition 11.3 Memory Fragmentation

The existence of unusable areas in the memory of a computer system.

Table 11.3 Forms of Memory Fragmentation

<table>
<thead>
<tr>
<th>Form of fragmentation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External fragmentation</td>
<td>Some area of memory is too small to be allocated.</td>
</tr>
<tr>
<td>Internal fragmentation</td>
<td>More memory is allocated than requested by a process, hence some of the allocated memory remains unused.</td>
</tr>
</tbody>
</table>

- Fragmentation leads to poor memory utilization

Contiguous Memory Allocation

- In contiguous memory allocation each process is allocated a single contiguous area in memory
  - Faces the problem of memory fragmentation
    - Memory compaction and reuse can be applied to overcome external fragmentation
      - Compaction requires a relocation register
      - Lack of this register is also a problem for swapping

![Diagram of Memory Compaction](image)
Noncontiguous Memory Allocation

- Portions of a process address space are distributed among different memory areas
  - Reduces external fragmentation

<table>
<thead>
<tr>
<th>Process component</th>
<th>Size</th>
<th>Memory start address</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1</td>
<td>50 KB</td>
<td>100K</td>
</tr>
<tr>
<td>P-2</td>
<td>30 KB</td>
<td>300K</td>
</tr>
<tr>
<td>P-3</td>
<td>60 KB</td>
<td>450K</td>
</tr>
</tbody>
</table>

Figure 11.17 Noncontiguous memory allocation to process P.

Logical Addresses, Physical Addresses, and Address Translation

- **Logical address**: address of an instruction or data byte as used in a process
  - Viewed as a pair \((\text{comp}_i, \text{byte})\)

- **Physical address**: address in memory where an instruction or data byte exists

Figure 11.18 A schematic of address translation in noncontiguous memory allocation.