Memory Management

- Program must be brought from disk into internal memory and placed within a process for it to be executed.
- There are many programs on the disk that can be brought into memory for execution. Obviously, not all of the programs will be brought into memory.

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time**: Memory location is known a priori; must recompile code if the location is changed.
- **Load time**: Memory location is not known at compile time.
- **Execution time**: The process can be moved during its execution from one memory segment to another.

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - *Logical address* – generated by the CPU; also referred to as *virtual address*.
  - *Physical address* – address seen by the internal memory.
- Memory-Management Unit (MMU)
  - Hardware device that maps virtual to physical address.
  - In MMU scheme, the value in the relocation register (or base register) is added to every address generated by a user process at the time it is sent to memory.
  - The user program deals with logical addresses; it never sees the real physical addresses.

Memory Relocation Using a Relocation Register

A diagram illustrating the process of memory relocation using a relocation register.
Swapping

- A process can be swapped temporarily out of memory to a Swap Area, and later is brought back into memory for continued execution.
- Swap Area – Fast and large enough disk which accommodates copies of all memory images for all users;
- Roll out, roll in – A lower-priority process is swapped out so a higher-priority process can be loaded and executed.
- Major part of swap time is the transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems, e.g., UNIX and Microsoft Windows.

Schematic View of Swapping

Contiguous Allocation

- Single-partition allocation:
  - Main memory is split into two partitions:
    - Resident operating system is usually held in low addresses in memory with interrupt vector.
    - User processes are held in high addresses in memory.
  - Relocation-register scheme is used to protect user processes from changing the operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - Hole – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about:
    a) allocated partitions       b) free partitions (hole)

```text
<table>
<thead>
<tr>
<th>OS</th>
<th>process 5</th>
<th>OS</th>
<th>process 5</th>
<th>OS</th>
<th>process 5</th>
<th>OS</th>
<th>process 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>process 8</td>
<td></td>
<td>process 9</td>
<td></td>
<td>process 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>process 2</td>
<td></td>
<td>process 2</td>
<td></td>
<td>process 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Dynamic Storage-Allocation Problem

How to satisfy a request of size $n$ from a list of free holes.

- **First-fit**: Allocates the first hole that is big enough.
- **Best-fit**: Allocates the smallest hole that is big enough; must search the entire list, unless is ordered by size. Produces the smallest leftover hole.
- **Worst-fit**: Allocates the largest hole; must also search the entire list. Produces the largest leftover hole.

Usually, First-fit and best-fit are better than worst-fit in terms of speed and storage utilization.

Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called frames. Size is power of 2. Nowadays, the size is usually 4Kbytes or 8Kbytes.
- Sunshine in Bar-Ilan has 8192 bytes page size.
- Divide logical memory into blocks of same size called pages.
- To run a program of size $n$ pages, need to find $n$ free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.

Fragmentation

- External Fragmentation – The total memory space exists to satisfy a request, but it is not contiguous.
- Internal Fragmentation – The allocated memory may be slightly larger than the requested memory.
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible only if relocation is dynamic, and is done at execution time.
  - I/O problem
    - Latch job in memory while it is involved in I/O.
    - Do I/O only into OS buffers.

Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** ($p$) – used as an index into a page table which contains base address of each page in physical memory.
  - **Page offset** ($d$) – combined with base address to define the physical memory address that is sent to the memory unit.
Address Translation Architecture

Paging Example

Yet other Paging Examples

Implementation of Page Table

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the using of a special fast-lookup hardware cache called associative registers or translation look-aside buffers (TLBs)
**Associative Register**

- Associative registers – parallel search

<table>
<thead>
<tr>
<th>Page #</th>
<th>Frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address translation:
Let \( P_i \) be a virtual page number
- If \( P_i \) is in associative register, get frame \# out.
- Otherwise get frame \# from page table in memory

**PAGING HARDWARE WITH TLB**

- Effective Access Time
  - Associative Lookup = \( \varepsilon \) microseconds
  - Assume memory reference (average) time is 1 microsecond. (If page is in the disk, it will take much more time. If values are in cache, it will take less time, so we take an average time)
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
  - Hit ratio = \( \alpha \)
  - Effective Access Time (EAT)
    
    \[
    EAT = (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\
    = 2 + \varepsilon - \alpha
    \]
A logical address on a 32-bit machine with 4K page size is divided into:
- a page number consisting of 20 bits.
- a page offset consisting of 12 bits.
Since the page table is paged, the page number is further divided into:
- a 10-bit page number.
- a 10-bit page offset.
Thus, a logical address is as follows:

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table.

Since each level is stored as a separate table in memory, converting a logical address to a physical one, may take more memory accesses.
Even though time needed for one memory access is multiplied, caching permits performance to remain reasonable if Cache hit rate will be high.

Uncommon in use.
The virtual page number is hashed into a page table. This page table contains a chain of elements which have been hashed to the same location.
Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.
Shared Pages

- Shared code
  - One copy of read-only (reentrant) code is shared among several processes (e.g. text editors, window systems).
  - Shared code must appear in the same location in the logical address space of all processes.

- Private code and data
  - Each process keeps a separate copy of the code and data.
  - The pages of the private code and data can appear anywhere in the logical address space.

Shared Pages Example

Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - main program,
  - procedure,
  - function,
  - local variables,
  - global variables,
  - stack,
  - arrays

- external fragmentation
Example of Segmentation

Segmentation Architecture

- Logical address consists of two components:
  `<segment-number, offset>`,
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - limit – specifies the length of the segment.
- Segment-table base register (STBR) points to the segment table’s location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program;
  segment number \( s \) is legal if \( s < \text{STLR} \).

Segmentation Hardware

Sharing of segments
The UNIX operating system solved problems of external fragmentation and lengthy search times by paging the segments.

Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a page table for this segment.

The segments are of text, data and stack.