MEMORY MANAGEMENT
**Background**

- Programs must be brought (from disk) into memory for them to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory access privileges required to ensure correct operation
Base and Limit Registers

Simplistically, a pair of base and limit registers define the logical 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 memory unit

- Memory-Management Unit (MMU) is the hardware device that maps virtual to physical address

- Logical and physical addresses are the same in compile-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme. The user program deals with *logical* addresses - it never sees the *real* physical addresses
Address binding of instructions and data to memory addresses can happen at two different stages

- **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes

- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Needs hardware support for address maps.
Dynamic Loading

- Routine is not loaded until it is called.
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required, implemented through program design.
Allocation Possibilities
Contiguous Allocation

- Main memory usually divided into two partitions:
  - Resident operating system, usually held in low memory.
  - User processes then held in high memory.
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data.
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register.
  - MMU maps logical address dynamically.
HW Address Protection with Base and Limit Registers

Logical + relocation

CPU

address

≥ yes

< no

> no

trap to operating system monitor—addressing error

base

base + limit

memory
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)
Dynamic Storage Allocation Problem

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

- **First-fit**: Allocate the *first* hole that is big enough
- **Next-fit**: Like first fit, but allocate the first hole from *last allocation* that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the largest leftover hole
Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

- **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
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,
- method,
- object,
- local variables, global variables,
- common block,
- stack,
- symbol table, arrays
User’s View of a Program

- subroutine
- stack
- symbol table
- Sqrt
- main program

logical address
Logical View of Segmentation

user space

physical memory space
Segmentation Architecture

• Logical address consists of a two tuple:
  
  \(<\text{segment-number}, \text{offset}>\)

• **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 Architecture - Cont.

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ illegal segment
    - read/write/execute privileges
  - Protection bits associated with segments; code sharing occurs at segment level.
  - Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
Example of Segmentation

[Diagram showing a logical address space with segments for subroutine, stack, symbol table, and main program. There is also a segment table with limits and bases for segments 0 to 4.]
Paging

- Logical address space of 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).

- Divide logical memory into blocks of same size called pages.

- Keep track of all free frames.

- 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.
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

$\begin{array}{|c|c|}
\hline
\text{page number} & \text{page offset} \\
\hline
p & d \\
\hline
m - n & n \\
\hline
\end{array}$

• For given logical address space of size $2^m$ and page size is $2^n$
Paging Hardware
Shared Pages

• Shared code
  – One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  – Each page table maps onto the same physical copy of the shared code.

• Private code and data
  – Each process keeps a separate copy of the code and data.
Paging Model of Logical and Physical Memory

<table>
<thead>
<tr>
<th>page 0</th>
<th>page 1</th>
<th>page 2</th>
<th>page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical memory</td>
<td>page table</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Frame number:
  - 0: page 0
  - 1: page 0
  - 2: page 2
  - 3: page 2
  - 4: page 1
  - 5: page 1
  - 6: page 3
  - 7: page 3

Physical memory
### Paging Example

#### 32-byte memory and 4-byte pages

<table>
<thead>
<tr>
<th>Logical memory</th>
<th>Physical memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>f</td>
</tr>
<tr>
<td>6</td>
<td>g</td>
</tr>
<tr>
<td>7</td>
<td>h</td>
</tr>
<tr>
<td>8</td>
<td>i</td>
</tr>
<tr>
<td>9</td>
<td>j</td>
</tr>
<tr>
<td>10</td>
<td>k</td>
</tr>
<tr>
<td>11</td>
<td>l</td>
</tr>
<tr>
<td>12</td>
<td>m</td>
</tr>
<tr>
<td>13</td>
<td>n</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>page table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

- 25-byte memory and 4-byte pages
Free Frames

(a) Before allocation

(b) After allocation
To check if a page is a valid memory address we have a **Valid-invalid** bit attached to each entry in the page table:

- “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
- “invalid” indicates that the page is not mapped at this time.
Thrashing may be caused by programs or workloads that present insufficient locality of reference.

If the working set of a program or a workload cannot be effectively held within physical memory, then constant data swapping, i.e., thrashing, may occur.
To resolve thrashing due to excessive paging, a user can do any of the following.

- Increase the amount of RAM in the computer (generally the best long-term solution).
- Decrease the number of programs being run on the computer.
- Replace programs that are memory-heavy with equivalents that use less memory.
Swapping

• A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

• **Backing store** – disk large enough to accommodate copies of all memory images for all users;

• **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed

• System maintains a **ready queue** of ready-to-run processes which have memory images on disk
Schematic View of Swapping

1. Swap out
2. Swap in
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 (PRLR)** 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 use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process
Paging Hardware With TLB

CPU → logical address → TLB

TLB:

- Page frame number
- TLB hit
- TLB miss

Page table:

- p
- f

Physical memory:

- Physical address
As the number of processes increases, the percentage of memory devoted to page tables also increases.

The following structures solved this problem:

– Hierarchical Paging

– Hashed Page Tables

– Inverted Page Tables
Hierarchical Page Tables

• Break up the logical address space into multiple page tables.

• A simple technique is a two-level page table (the page table is paged).
Two-Level Page-Table Scheme
A logical address (on 32-bit machine with 1K page size) is divided into:
- a page number consisting of 22 bits
- a page offset consisting of 10 bits

Since the page table is paged, the page number is divided into:
- a 12-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>
</tbody>
</table>

12 10 10

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.
Address-Translation Scheme

Logical address

\[
\begin{array}{c}
p_1 \\
p_2 \\
d \end{array}
\]

p_1 \rightarrow \text{outer page table}

p_2 \rightarrow \text{page of page table}

d
Hashed Page Tables

• Common in address spaces > 32 bits.

• The virtual page number is hashed into a page table. This page table contains a chain of elements hashing 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.
Inverted Page Table

• One entry for each real page of memory.
• Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
• Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
• Use hash table to limit the search to one - or at most a few - page-table entries.
Inverted Page Table Architecture

CPU → Logical address

page table

physical address

physical memory