PAGING & SEGMENTATION

Non-Contiguous Memory

AUGUST 12, 2025

SMMUNITY TECHNICAL INSTITUTE
Submitted by: Man Kumar Ghising

1. Introduction

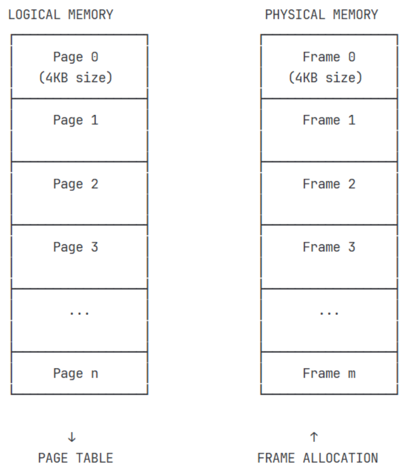
Modern operating systems face the challenge of efficiently managing memory resources among multiple processes. When simple contiguous allocation becomes inadequate due to external fragmentation and inflexibility, operating systems turn to more sophisticated memory management schemes: paging and segmentation. These techniques allow processes to be placed non-contiguously in physical memory while maintaining the illusion of a continuous logical address space.

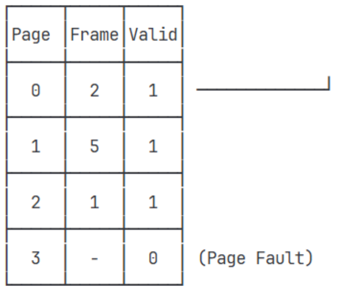
2. Paging Basics

2.1 Definition and Concept

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. In paging, the logical address space is divided into fixed-size blocks called **pages**, while the physical address space is divided into blocks of the same size called **frames** or **page frames**. The operating system maintains a page table for each process to track the mapping between pages and frames.

2.2 Paging Architecture Diagram





Page	Frame	Valid
0	2	1
1	5	1
2	1	1
3	-	0

(Page Fault)

2.3 Address Translation Process

The logical address in paging consists of two parts:

- **Page Number (p)**: Used as an index into the page table
- **Page Offset (d)**: Combined with the base address to define the physical address

Address Translation Formula: Physical Address = Frame Number × Frame Size + Page Offset

2.4 Sample Page Table Example

Consider a system with 4KB page size and the following page table:

Before Page Fault:

Page #	Frame #	Valid Bit	Protection	Last Access
0	2	1	R/W	100
1	5	1	R/W	95
2	1	1	R/W	120
3	-	0	-	-
4	7	1	R/W	80
5	-	0	-	-
6	3	1	R/W	110
7	9	1	R/W	105

After Page Fault (Page 3 loaded into Frame 4):

Page #	Frame #	Valid Bit	Protection	Last Access
0	2	1	R/W	100
1	5	1	R/W	95
2	1	1	R/W	120
3	4	1	R/W	125
4	7	1	R/W	80
5	-	0	-	-
6	3	1	R/W	110
7	9	1	R/W	105

3. Page Replacement Algorithms

When all frames are occupied and a page fault occurs, the OS must select a victim page to replace. Three primary algorithms are commonly used:

3.1 First-In-First-Out (FIFO)

FIFO replaces the oldest page in memory. It maintains a queue of pages in order of arrival.

Advantages:

- Simple to implement and understand
- Low overhead

Disadvantages:

- May suffer from Belady's anomaly
- Does not consider page usage patterns

3.2 Least Recently Used (LRU)

LRU replaces the page that has been unused for the longest time, based on the principle of temporal locality.

Advantages:

- Generally performs well in practice
- Considers usage patterns

Disadvantages:

- Higher implementation overhead
- Requires tracking access times

3.3 Optimal Algorithm

The optimal algorithm replaces the page that will not be used for the longest period in the future.

Advantages:

- Guaranteed minimum page faults
- Theoretical benchmark

Disadvantages:

- Impossible to implement in practice
- Requires future knowledge

3.4 Algorithm Simulation

Reference String: 1, 3, 2, 4, 1, 5, 2, 3, 4, 5 **Number of Frames:** 3

FIFO Simulation:

Step	Reference	Frame 1	Frame 2	Frame 3	Hit/Miss	Queue Order
1	1	1	-	-	Miss	[1]

Step	Reference	Frame 1	Frame 2	Frame 3	Hit/Miss	Queue Order
2	3	1	3	-	Miss	[1,3]
3	2	1	3	2	Miss	[1,3,2]
4	4	4	3	2	Miss	[4,3,2]
5	1	4	1	2	Miss	[4,1,2]
6	5	4	1	5	Miss	[4,1,5]
7	2	2	1	5	Miss	[2,1,5]
8	3	2	3	5	Miss	[2,3,5]
9	4	2	3	4	Miss	[2,3,4]
10	5	5	3	4	Miss	[5,3,4]

FIFO Results: 10 misses, 0 hits, Page fault rate = 100%

LRU Simulation:

Step	Reference	Frame 1	Frame 2	Frame 3	Hit/Miss	LRU Order
1	1	1	-	-	Miss	[1]
2	3	1	3	-	Miss	[1,3]
3	2	1	3	2	Miss	[1,3,2]
4	4	4	3	2	Miss	[4,3,2]
5	1	4	3	1	Miss	[4,3,1]
6	5	5	3	1	Miss	[5,3,1]
7	2	5	2	1	Miss	[5,2,1]

Step	Reference	Frame 1	Frame 2	Frame 3	Hit/Miss	LRU Order
8	3	5	2	3	Miss	[5,2,3]
9	4	4	2	3	Miss	[4,2,3]
10	5	4	5	3	Miss	[4,5,3]

LRU Results: 10 misses, 0 hits, Page fault rate = 100%

Optimal Algorithm Simulation:

Step	Reference	Frame 1	Frame 2	Frame 3	Hit/Miss	Next Use
1	1	1	-	-	Miss	[5,-,-]
2	3	1	3	-	Miss	[5,8,-]
3	2	1	3	2	Miss	[5,8,7]
4	4	1	4	2	Miss	[5,9,7]
5	1	1	4	2	Hit	[-,9,7]
6	5	5	4	2	Miss	[10,9,7]
7	2	5	4	2	Hit	[10,9,-]
8	3	5	3	2	Miss	[10,-,-]
9	4	5	3	4	Miss	[10,-,-]
10	5	5	3	4	Hit	[-,-,-]

Optimal Results: 7 misses, 3 hits, Page fault rate = 70%

---

4. Segmentation

4.1 Definition and Concept

Segmentation is a memory management scheme that reflects the user's view of memory. Unlike paging, which divides memory into fixed-size pages, segmentation divides the logical address space into variable-size segments based on logical units such as functions, data structures, or program modules.

4.2 Segment Table Structure

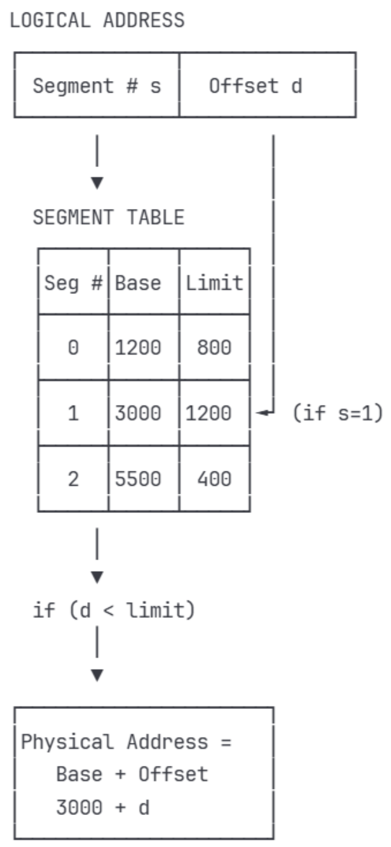
Each process has a segment table with entries containing:

- **Base Address**: Starting physical address of the segment
- **Limit**: Length of the segment
- **Protection**: Access permissions (read, write, execute)

Sample Segment Table:

Segment #	Base Address	Limit (bytes)	Protection	Description
0	1200	800	R-X	Main Program
1	3000	1200	RW-	Data Section
2	5500	400	RW-	Stack
3	2000	600	R--	Constants
4	7200	300	RWX	Shared Library

4.3 Address Translation Diagram



Seg #	Base	Limit
0	1200	800
1	3000	1200
2	5500	400

4.4 Address Translation Process

1. Extract segment number (s) and offset (d) from logical address
2. Check if segment number is valid (s < segment table length)
3. Retrieve base and limit from segment table entry s
4. Check if offset is within bounds (d < limit)
5. If valid, calculate physical address = base + offset
6. If invalid, generate segmentation fault

Example Translation:

- Logical Address: (1, 400) - Segment 1, Offset 400
- Segment Table Entry 1: Base = 3000, Limit = 1200
- Validation: 400 < 1200 ✓

- Physical Address: 3000 + 400 = 3400

---

5. Comparison and Nepal Telecom Example

5.1 Paging vs. Segmentation Comparison

Aspect	**Paging**	**Segmentation**
Fragmentation	Internal fragmentation only	External fragmentation possible
Size	Fixed-size pages	Variable-size segments
User Visibility	Transparent to programmer	Visible to programmer
Sharing	Difficult (page-level)	Easy (segment-level)
Protection	Page-level protection	Segment-level protection
Memory Utilization	May waste space in pages	Better utilization
Implementation	Simpler hardware	More complex validation

5.2 Nepal Telecom Server Farm Scenario

Context: Nepal Telecom operates a large server farm in Kathmandu serving millions of customers across Nepal. The infrastructure handles various services including voice calls, data services, and internet connectivity.

Segmentation Approach for User Sessions:

```
User Session Segments:
┌─────────────────────────────────┐
│ Segment 0: Session Control Data │ Base: 0x10000, Limit: 2KB
├─────────────────────────────────┤
│ Segment 1: Call History         │ Base: 0x20000, Limit: 8KB
├─────────────────────────────────┤
│ Segment 2: Billing Information  │ Base: 0x35000, Limit: 4KB
├─────────────────────────────────┤
│ Segment 3: Service Configuration│ Base: 0x42000, Limit: 3KB
└─────────────────────────────────┘
```

Advantages in this context:

- **Logical Organization**: Each segment represents a meaningful unit (billing, call history)
- **Protection**: Billing data can be read-only for regular processes
- **Sharing**: Service configuration can be shared among similar user types
- **Variable Size**: Accommodates different data requirements per segment

Paging Approach for OS Kernel and User Code:

```
Memory Layout:
┌─────────────┐ ┌─────────────┐ ┌─────────────┐
│   Page 0    │ │   Frame 3   │ │ Kernel Code │
│ (Kernel)    │→│   (4KB)     │ │             │
├─────────────┤ ├─────────────┤ ├─────────────┤
│   Page 1    │ │   Frame 1   │ │ User Code   │
│ (User Code) │→│   (4KB)     │ │             │
├─────────────┤ ├─────────────┤ ├─────────────┤
│   Page 2    │ │   Frame 5   │ │ Data Pages  │
│ (Data)      │→│   (4KB)     │ │             │
└─────────────┘ └─────────────┘ └─────────────┘
```

6. Flowchart and Code Implementation

6.1 Page Fault Handling Flowchart

```
Start: CPU Memory Access
          |
          v
┌──────────────────────┐
│    Page Fault        │
│    (Invalid Page)    │
└──────────────────────┘
          |
          v
┌──────────────────────┐
│     TRAP TO OS       │
│  (Hardware Signal)   │
└──────────────────────┘
          |
          v
┌──────────────────────┐
│    OS PAGE FAULT     │
│      ROUTINE         │
└──────────────────────┘
          |
          v
┌──────────────────────┐
│   FRAME ALLOCATION   │
│  Find free frame or  │
│  replace existing    │
└──────────────────────┘
          |
          v
 ┌─────────────────────┐
 │   LOAD PAGE FROM    │
 │   SECONDARY STORAGE │
 └─────────────────────┘
          |
          v
 ┌─────────────────────┐
 │  PAGE TABLE UPDATE  │
 │ Set valid bit = 1   │
 │ Update frame number │
 └─────────────────────┘
          |
          v
 ┌─────────────────────┐
 │   RESUME PROCESS    │
 │  Return to CPU      │
 └─────────────────────┘
          |
          v
     Continue Execution
```