PAGING & SEGMENTATION:

Non-Contiguous Memory

AUGUST 12, 2025

SINDHULI COMMUNITY TECHNICAL INSTITUTE
Submitted by: Susmita Karki

1. **Paging Basics**

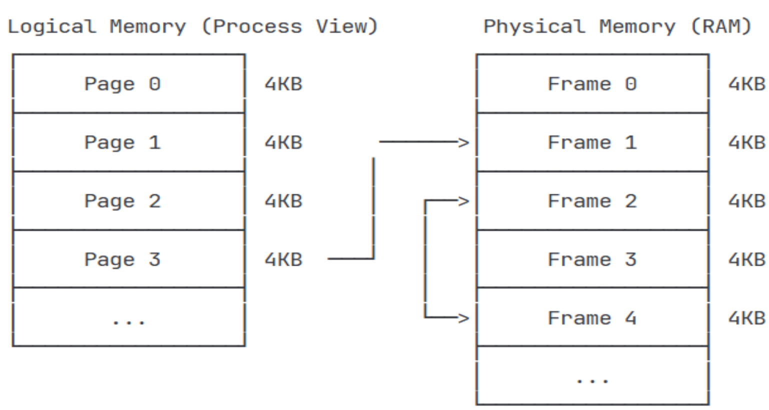
Definition of Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. The physical memory is divided into fixed-size blocks called **frames**, while the logical memory is divided into blocks of the same size called **pages**. This allows the operating system to load pages of a process into any available frames in physical memory.

Key Characteristics:

- Fixed-size memory blocks (typically 4KB, 8KB, or 16KB)
- Eliminates external fragmentation
- Enables non-contiguous memory allocation
- Supports virtual memory implementation

Paging Diagram




```
Page Table Translation:
Page 0 → Frame 1
Page 1 → Frame 4
Page 2 → Frame 2
Page 3 → Not in memory (Page Fault)
```

Page Table Structure

The page table is a data structure maintained by the operating system that maps logical page numbers to physical frame numbers.

Sample Page Table - Before Page Fault:

Page Number	Frame Number	Valid Bit	Reference Bit	Dirty Bit
0	1	1	1	0
1	4	1	0	1
2	2	1	1	0
3	-	0	0	0
4	7	1	1	1
5	-	0	0	0
6	3	1	0	0
7	-	0	0	0

Page Fault Scenario: When the CPU attempts to access Page 3 (which has Valid Bit = 0), a page fault occurs.

Sample Page Table - After Page Fault (Page 3 loaded into Frame 5):

Page Number	Frame Number	Valid Bit	Reference Bit	Dirty Bit
0	1	1	1	0
1	4	1	0	1
2	2	1	1	0
3	5	1	1	0

Page Number	Frame Number	Valid Bit	Reference Bit	Dirty Bit
4	7	1	1	1
5	-	0	0	0
6	3	1	0	0
7	-	0	0	0

Address Translation Process

Logical Address Structure:

Page Number (p bits)	Offset (d bits)

Translation Steps:

1. Extract page number (p) from logical address
2. Use page number as index into page table
3. Extract frame number (f) from page table entry
4. Combine frame number with offset to get physical address

2. Page Replacement Algorithms

When all frames in physical memory are occupied and a page fault occurs, the operating system must select a page to replace. Various algorithms determine which page to evict.

- **First-In-First-Out (FIFO) Algorithm**

FIFO replaces the page that has been in memory the longest, regardless of usage patterns.

Algorithm Steps:

1. Maintain a queue of pages in memory
2. On page fault, remove the page at the front of the queue

3. Add the new page to the rear of the queue

- **Least Recently Used (LRU) Algorithm**

LRU replaces the page that has not been used for the longest time, based on the principle of temporal locality.

Algorithm Steps:

1. Track the last access time for each page
2. On page fault, replace the page with the oldest last access time
3. Update access times on each page reference

- **Optimal Page Replacement Algorithm**

The optimal algorithm replaces the page that will not be used for the longest period in the future. While not implementable in practice, it serves as a benchmark.

Algorithm Steps:

1. Look ahead in the reference string
2. Replace the page that is used farthest in the future
3. If no page is referenced again, replace any page

Simulation Example

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

FIFO Algorithm Simulation:

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

Step	Reference	Frames	Hit/Miss	Queue Order
4	4	[4, 3, 2]	Miss	3, 2, 4
5	1	[4, 1, 2]	Miss	2, 4, 1
6	5	[4, 1, 5]	Miss	4, 1, 5
7	2	[2, 1, 5]	Miss	1, 5, 2

FIFO Results: 7 page faults, 0 hits

LRU Algorithm Simulation:

Step	Reference	Frames	Hit/Miss	Last Used Order
1	1	[1, -, -]	Miss	1(1)
2	3	[1, 3, -]	Miss	1(1), 3(2)
3	2	[1, 3, 2]	Miss	1(1), 3(2), 2(3)
4	4	[4, 3, 2]	Miss	3(2), 2(3), 4(4)
5	1	[4, 1, 2]	Miss	2(3), 4(4), 1(5)
6	5	[5, 1, 2]	Miss	1(5), 2(3), 5(6)
7	2	[5, 1, 2]	Hit	1(5), 5(6), 2(7)

LRU Results: 6 page faults, 1 hit

Optimal Algorithm Simulation:

Step	Reference	Frames	Hit/Miss	Future References
1	1	[1, -, -]	Miss	1@5
2	3	[1, 3, -]	Miss	1@5, 3@∞

Step	Reference	Frames	Hit/Miss	Future References
3	2	[1, 3, 2]	Miss	1@5, 3@∞, 2@7
4	4	[1, 4, 2]	Miss	1@5, 4@∞, 2@7
5	1	[1, 4, 2]	Hit	1@∞, 4@∞, 2@7
6	5	[1, 5, 2]	Miss	1@∞, 5@∞, 2@7
7	2	[1, 5, 2]	Hit	1@∞, 5@∞, 2@∞

Optimal Results: 5 page faults, 2 hits

Performance Comparison

Algorithm	Page Faults	Hit Rate	Implementation Complexity
FIFO	7	0%	Low
LRU	6	14.3%	Medium
Optimal	5	28.6%	Impossible (theoretical)

3.Segmentation

Definition of Segmentation

Segmentation is a memory management technique that divides a process into variable-sized segments based on logical divisions such as functions, procedures, objects, or data structures. Unlike paging, segments vary in size and represent meaningful units of the program.

Key Characteristics:

- Variable-size memory segments
- Logical program divisions (code, data, stack)
- Facilitates sharing and protection

- May cause external fragmentation

Segment Table Structure

The segment table maps segment numbers to their base addresses and limits in physical memory.

```
Segment Table Structure:
```

Segment Number	Base Address	Limit
0	1000	400
1	4000	800
2	2500	300
3	7000	1200

Segmentation Diagram

```
Logical Memory (Segmented)           Physical Memory
┌──────────────────────────┐         ┌──────────────────────────┐
│    Segment 0 (Code)      │  400B   │          ...             │
│       400 bytes          │  ─────> │  Segment 0: 1000-1399    │
├──────────────────────────┤         ├──────────────────────────┤
│   Segment 1 (Data)       │  800B   │          ...             │
│       800 bytes          │  ──┐    │  Segment 2: 2500-2799    │
├──────────────────────────┤    │    ├──────────────────────────┤
│   Segment 2 (Stack)      │  300B ──┘ │          ...           │
│       300 bytes          │  ─────> │  Segment 1: 4000-4799    │
├──────────────────────────┤         ├──────────────────────────┤
│   Segment 3 (Heap)       │  1200B  │          ...             │
│      1200 bytes          │  ─────> │  Segment 3: 7000-8199    │
└──────────────────────────┘         └──────────────────────────┘
```

Address Translation in Segmentation

Logical Address Format:

Translation Process:

1. **Extract Segment Number and Offset** from logical address

2. **Check Segment Table** using segment number as index
3. **Validate Offset** against segment limit
4. **Calculate Physical Address** = Base Address + Offset

Example Translation:

- Logical Address: (Segment 1, Offset 250)
- From Segment Table: Segment 1 → Base = 4000, Limit = 800
- Validation: 250 < 800 ✓ (Valid)
- Physical Address: 4000 + 250 = 4250

Error Handling:

- If Offset ≥ Limit → Segmentation Fault
- If Segment Number ≥ Segment Table Size → Invalid Segment Error

Segmentation with Paging

Modern systems often combine segmentation with paging to leverage benefits of both techniques:

```
Hybrid Approach: Segmentation + Paging
┌─────────────┐    ┌─────────────┐    ┌─────────────┐
│  Logical    │ -> │  Segment    │ -> │  Physical   │
│  Address    │    │  Address    │    │  Address    │
│ (seg,page,  │    │ (seg,page,  │    │ (frame,     │
│  offset)    │    │  offset)    │    │  offset)    │
└─────────────┘    └─────────────┘    └─────────────┘

      |                  |                  |
      v                  v                  v
 Segment Table       Page Table       Physical Memory
```

4.Comparison & Nepal Telecom Example

4.1 Paging vs. Segmentation Comparison

Aspect	Paging	Segmentation
Size	Fixed-size pages (e.g., 4KB)	Variable-size segments
Fragmentation	Internal fragmentation only	External fragmentation possible
Address Space	Single linear address space	Multiple logical address spaces
Sharing	Page-level sharing	Segment-level sharing (more natural)
Protection	Page-level protection	Segment-level protection (more granular)
Implementation	Simpler hardware support	More complex address translation
Memory Utilization	Better for uniform access patterns	Better for logical program organization

4.2 Nepal Telecom Scenario

Nepal Telecom operates one of the largest telecommunications networks in Nepal, managing thousands of concurrent user sessions and various system services. Let's examine how different memory management approaches would affect their server infrastructure.

Scenario 1: Segmentation for User Sessions

Context: Nepal Telecom's authentication server handles user login sessions for mobile and internet services across Nepal's diverse geographic regions.

Segmentation Implementation:

```
User Session Segments:
┌──────────────────────────┐
│ Segment 0: Session Data│ - User credentials, location data
│       Size: 2KB          │ - Connection metadata
├──────────────────────────┤
│ Segment 1: Call Logs     │ - Recent call history
│       Size: 4KB          │ - Billing information
├──────────────────────────┤
│ Segment 2: Data Usage    │ - Internet usage statistics
│       Size: 1.5KB        │ - Bandwidth allocations
├──────────────────────────┤
│ Segment 3: Location      │ - Tower connections
│       Size: 512B         │ - Geographic data
└──────────────────────────┘
```

Advantages for Nepal Telecom:

- **Natural Protection:** Each segment can have different access permissions (read-only location data, read-write session data)
- **Efficient Sharing:** Common location data for users in the same area can be shared across sessions
- **Logical Organization:** Segments align with business logic (authentication, billing, location services)

Segment Table for Multiple Users:

User ID	Segment	Base Address	Limit	Permissions
NPL001	0	10000	2048	RW
NPL001	1	15000	4096	RW
NPL001	2	22000	1536	RW
NPL001	3	30000	512	R

User ID Segment Base Address Limit Permissions

NPL002	0	12000	2048	RW
NPL002	3	30000	512	R (Shared)

Scenario 2: Paging for OS Kernel and User Code

Context: Nepal Telecom's core network management system runs critical infrastructure code alongside user applications.

Paging Implementation:

```
Memory Layout (4KB pages):
┌──────────────────────────┐
│ Page 0-15: Kernel Code   │ 64KB - Network protocol stack
├──────────────────────────┤
│ Page 16-31: Kernel Data  │ 64KB - Routing tables, device drivers
├──────────────────────────┤
│ Page 32-47: User Code    │ 64KB - Billing application
├──────────────────────────┤
│ Page 48-63: User Data    │ 64KB - Customer databases
├──────────────────────────┤
│ Page 64-79: Buffer       │ 64KB - Network packet buffers
└──────────────────────────┘
```

Page Table Example (Network Management Server):

Page	Frame	Valid	Protection	Description
0	15	1	R-X	Network stack code
1	23	1	R-X	Protocol handlers
16	45	1	RW-	Routing tables
17	67	1	RW-	Device status
32	89	1	R-X	Billing engine
33	-	0	---	Not loaded

Page	Frame	Valid	Protection	Description
48	12	1	RW-	Customer DB cache
64	78	1	RW-	Packet buffers

5. Flowchart & Code Implementation

5.1 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
```