

Memory Management & Security

Prof. Khushal Bhoyar

Department of CSE (Cyber Security)
PIET, Parul University

3.1 Memory Allocation Techniques

Memory allocation strategies manage how processes obtain and use memory resources.

Fixed Partitioning:

Divides physical memory into predetermined-size partitions.

Simple to implement.

Suffers from **internal fragmentation** (unused space within an allocated partition).

Variable Partitioning:

Allocates memory blocks of exactly the required size to processes.

Eliminates internal fragmentation.

Creates **external fragmentation** over time (small free memory blocks scattered between allocated blocks).

-cont

Paging:

Divides both physical memory and process address spaces into fixed-size blocks.

Physical memory blocks are called **frames**.

Process address space blocks are called **pages**.

Uses **page tables** for address translation (mapping logical pages to physical frames).

Enables efficient memory utilization.

Causes slight internal fragmentation (last page of a process may not be completely full).

Segmentation:

Divides memory into logical units corresponding to program structure (e.g., code, data, stack segments).

More natural for programmers.

Complex to manage due to variable-sized segments, leading to external fragmentation.

-cont

Modern Systems:

Typically combine **paged segmentation**.

Uses segmentation for logical organization.

Uses paging for physical allocation.

3.2 Virtual Memory Implementation

Virtual memory creates the illusion of a memory space larger than physical RAM by storing inactive portions on secondary storage (like a disk).

Demand Paging:

Loads pages into physical memory only when they are accessed (on demand). Reduces initial process load time and memory requirements.

Page Replacement Algorithms:

Determine which page to evict from physical memory when a **page fault** occurs and no free frames are available.

First-In-First-Out (FIFO):

Removes the oldest page (first page brought in).

Simple to implement.

Suffers from **Belady's anomaly** (increasing page frames can sometimes increase page faults).

-cont

Optimal Algorithm:

Replaces the page that will not be used for the longest period in the future.
Impossible to implement practically (requires future knowledge).
Serves as a benchmark for comparing other algorithms.

Least Recently Used (LRU):

Replaces the page that has not been used for the longest time.
Performance is close to optimal.
Requires substantial hardware support to track usage.

Clock Algorithms:

Approximate LRU performance with lower overhead.
Use a **reference bit** (or use bit) to track page usage.

-cont

Thrashing:

Occurs when a system spends excessive time and resources on paging (swapping pages in and out) instead of executing useful work.

Addressed through:

Working Set Model: Ensuring a process has its required set of pages in memory.

Page Fault Frequency (PFF) Technique: Adjusting the number of frames allocated based on the rate of page faults.

3.3 Memory Protection Mechanisms

Memory protection prevents processes from accessing unauthorized memory regions through hardware-enforced access rights.

Base and Limit Registers:

Early protection mechanism.

Define each process's address space boundaries (a starting base address and a limit/range).

Page Table Entry Permission Bits:

Control read, write, and execute access at the granularity of individual pages.

Buffer Overflow Attacks:

Exploit software vulnerabilities to write data beyond the boundaries of an allocated buffer in memory.

Can overwrite return addresses to redirect program execution to malicious code.

-cont

Stack Canaries:

A defense against stack buffer overflows.

Place known values ("canaries") between buffers and control data (like return addresses) on the stack.

Check the integrity of the canary value before a function returns.

Non-executable Stack Protection:

Marks stack memory as non-executable (NX bit).

Prevents execution of malicious code (shellcode) injected into the stack.

Address Space Layout Randomization (ASLR):

Randomizes the memory locations of key areas (stack, heap, libraries) for each process.

Makes target addresses unpredictable for attackers.

These protections operate at multiple levels: hardware (Memory Management Units), operating system, and compiler.

3.4 Secure Memory Design Principles

Secure memory architectures implement isolation guarantees through hardware-enforced boundaries between security domains.

Virtualization-based Security:

Uses hardware virtualization features (like hypervisors) to create strongly isolated containers for sensitive operations.

Trusted Execution Environments (TEEs):

Provide hardware-protected, isolated areas for secure code execution and data processing.

Examples: **Intel SGX** (Software Guard Extensions) and **ARM TrustZone**.

Memory Encryption:

Protects data confidentiality in memory, even from physical attacks.

Approaches range from **full memory encryption** to **selective encryption** of only sensitive regions.

-cont

Memory Tagging:

Associates metadata tags (a small number of bits) with each memory allocation or granule.

Used to detect spatial (out-of-bounds access) and temporal (use-after-free) safety violations.

Control-Flow Integrity (CFI) Techniques:

Validate indirect control transfers (jumps, calls, returns) against a pre-determined, legitimate control-flow graph or policy.

Aims to prevent code reuse attacks (like ROP - Return-Oriented Programming).

These advanced mechanisms address sophisticated memory corruption attacks while aiming to maintain reasonable performance overhead.

1. <https://www.druva.com/glossary/what-is-a-disaster-recovery-plan-definition-and-related-faqs>
2. <https://www.konverge.co.in/virtualization-in-cloud-computing-need-types-and-importance/>
3. <https://www.crowdstrike.com/cybersecurity-101/cloud-security/cloud-application-security/>



<https://paruluniversity.ac.in/>

