Memory Addressing

Memory Management:
Memory management in Unix is complex.
Unix uses virtual memory for various benefits.
Virtual memory acts as a middleman between apps and hardware.
It allows running big apps, sharing memory, and simplifies coding.
Virtual memory uses virtual addresses, translated to physical by the CPU.
RAM is divided between kernel and virtual memory usage.
Balancing memory types is crucial, and fragmentation is an issue.

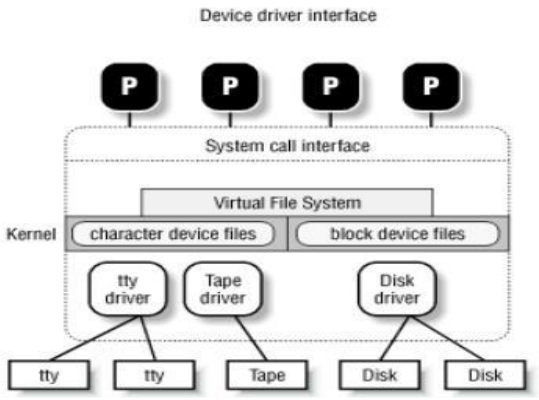
RAM (Random Access Memory):
RAM is computer memory used for active tasks.
It's divided into two parts: kernel and virtual memory.
Kernel part stores the core of the operating system.
The rest is used by applications for their data and processes.
RAM is faster than storage but volatile (loses data when powered off).
Balancing memory usage is crucial, and fragmentation can be a problem.

Kernel Memory Allocator (KMA):
KMA handles memory requests from different parts of the system.
It's crucial for speed because all kernel subsystems use it.
Good KMA features include speed, minimal wasted memory, reduced fragmentation, and cooperation with other memory management subsystems.
Different KMA techniques include resource map allocator, power-of-two freelists, McKusick-Karels allocator, buddy system, Mach's zone allocator, Dynix allocator, and Solaris's slab allocator.

Process Virtual Address Space:
In Unix, a process has a virtual address space that includes program code, data, stack, libraries, and dynamic memory. Recent Unix systems use demand paging, which means pages are loaded into physical memory only when needed. When a process accesses a missing page, the system allocates a page and fills it with the required data. This helps manage memory efficiently and allows processes to start even if not all pages are in physical memory.

Swapping and Caching:
Unix uses swapping to manage memory by moving pages between disk and RAM when needed.
Caching stores frequently used data in RAM to speed up access.

Device Drivers:
Device Drivers:
1. Kernel modules for I/O devices.
2. Each has a unique interface.
3. Benefits: encapsulation, vendor-friendly, uniform.
4. Load/unload without rebooting.
5. Simplifies device management.



Memory Addresses:
Memory address is how we access memory cell content.
Three kinds of addresses:

Logical Address:
Used in machine language instructions to specify operand or instruction address.
Comprises a segment and an offset, denoting the distance from segment start to the actual address.

Linear Address:
A single 32-bit unsigned integer.
Addresses up to 4GB, or 4,294,967,296 memory cells.

Physical Addresses:
Used to address memory cells in memory chips.
Correspond to electrical signals sent from the processor to the memory bus.
Represented as 32-bit unsigned integers.

Segmentation in Hardware:
Intel microprocessors perform address translation in two ways: real mode and protected mode.
Real mode is used for compatibility with older models and bootstrapping.
The focus is on protected mode, which provides advanced memory protection and security features.

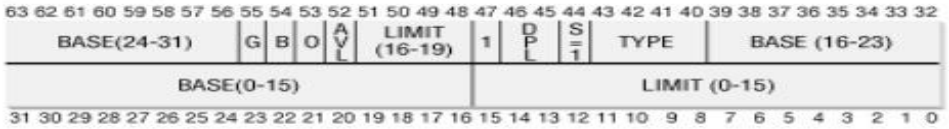
Segmentation Registers:
Logical address = Segment Selector (16-bit) + Offset (32-bit) within the segment.
Processors have segmentation registers: cs, ss, ds, es, fs, and gs.
These registers hold Segment Selectors.
Programs can reuse these registers for different purposes by saving content in memory and restoring it later.

Specific Registers:
cs: Code Segment Register, points to the segment with program instructions.
ss: Stack Segment Register, points to the segment with the current program stack.
ds: Data Segment Register, points to the segment with static and external data.

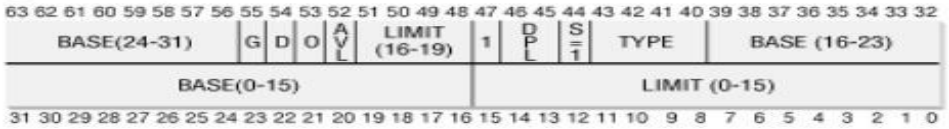
Segment Descriptors:
Each segment has an 8-byte Segment Descriptor describing its characteristics.
Segment Descriptors are stored in the Global Descriptor Table (GDT) or the Local Descriptor Table (LDT).
Typically, there's only one GDT, but each process can have its LDT.

The GDT's address in memory is stored in the gdtr processor register, while the currently used LDT's address is in the ldtr processor register.

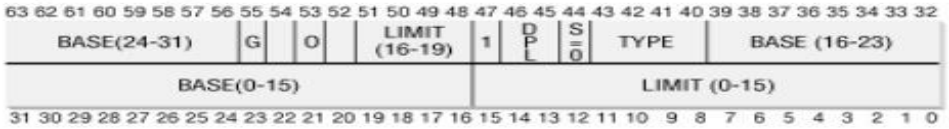
Data Segment Descriptor



Code Segment Descriptor



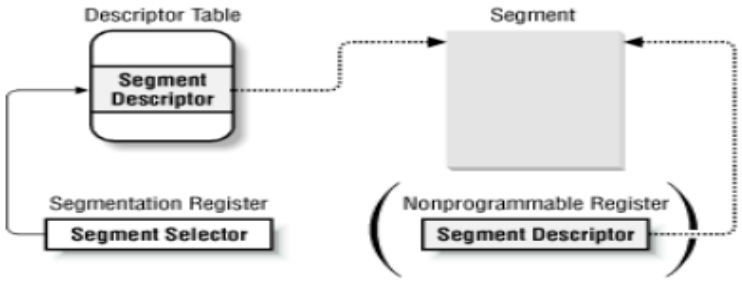
System Segment Descriptor



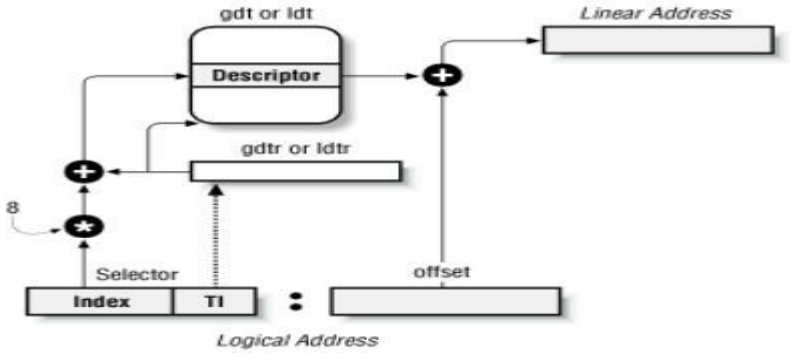
Segment Descriptor Fields:
Each Segment Descriptor has these fields:
32-bit Base: Linear address of the segment's first byte.
G Granularity Flag: Indicates segment size (bytes or multiples of 4096).
20-bit Limit: Denotes segment length in bytes (1B to 1MB or 4KB to 4GB).
S System Flag: System or normal segment (kernel data or code).
4-bit Type: Characterizes segment type and access rights.
D or B Flag: Depends on code or data, indicating 32-bit or 16-bit addresses.
Reserved bit: Always set to 0.
AVL (Available) Flag: Optionally used by the operating system but ignored in Linux.

Segment Selector and Segment Descriptor:
Intel processors use Segment Selectors as memory segment pointers. Loading a Selector into a segmentation register fetches its Segment Descriptor into a CPU register. This speeds up address translation for that segment without constantly accessing the GDT or LDT in memory, making memory operations more efficient.

A 13-bit index in the Segment Selector points to a specific entry in either the GDT or LDT. The TI flag (Table Indicator) distinguishes between the GDT (TI=0) and LDT (TI=1). The RPL (Requestor Privilege Level), a 2-bit field, indicates the CPU's current privilege level when loading the Selector. The Segment Descriptor's position in the GDT or LDT is calculated by multiplying the top 13 bits of the Selector by 8.

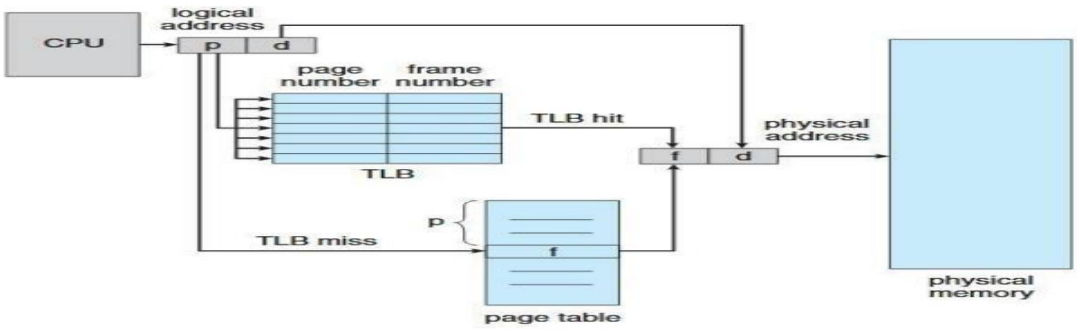


Segmentation Unit:

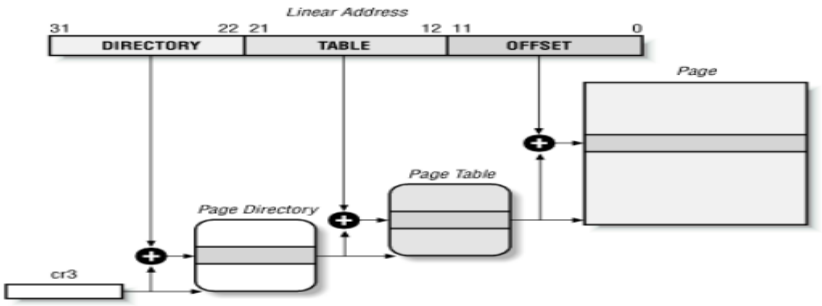


- Translates logical addresses to linear addresses.
- Determines Descriptor Table (GDT or LDT) based on the TI field in Segment Selector.
- Computes Descriptor address by multiplying index by 8 and adding to gdtr/ldtr register.
- Adds logical address offset to the Segment Descriptor's Base to get linear address.

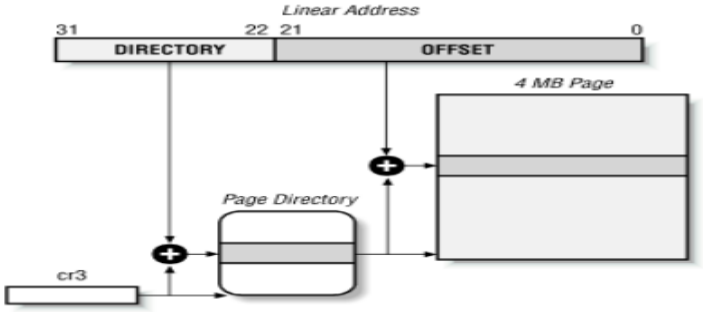
Paging in Hardware:
- Hardware translates linear addresses to physical ones.
- Checks access rights, generates page fault if invalid.
- Linear addresses grouped into fixed-size pages.
- Each page corresponds to a page frame in RAM.
- Page tables map linear to physical addresses.
- Enabled by setting the PG flag in the cr0 register.



Regular Paging:
- Intel processors from i80386 use 4KB pages.
- Linear address (32 bits) is divided into three fields:
  - Directory: Top 10 bits.
  - Table: Middle 10 bits.
  - Offset: Bottom 12 bits.
- Translation occurs in two steps with Page Directory and Page Table.
- Physical address of Page Directory is in cr3 processor register.
- Directory field points to the Page Table entry.
- Table field determines the Page Table entry containing the page frame's physical address.
- Offset field specifies the position within the page frame, as each page is 4096 bytes.



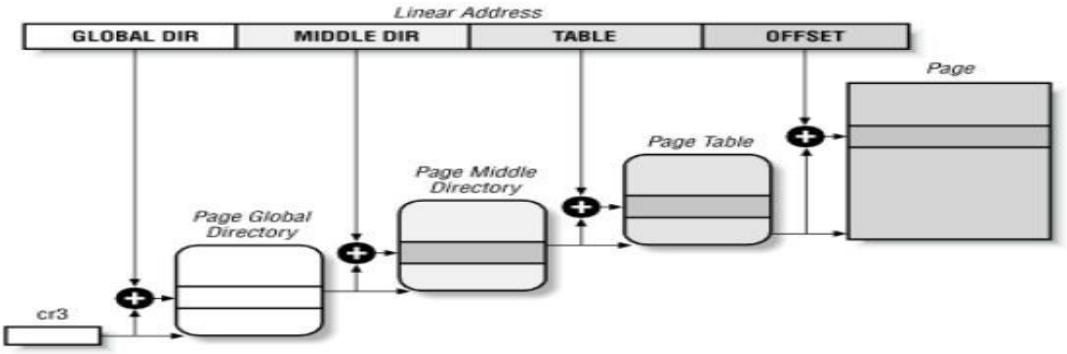
Extended Paging:
Extended Paging (Intel Processors):
- Introduced in Pentium and later CPUs.
- Enables 4KB or 4MB page sizes.
- Set PageSize flag in Page Directory entry.
- Splits linear address into Directory (top 10 bits) and Offset (remaining 22 bits).
- Coexists with regular paging, enabled by PSE flag in cr4 register.
- Translates large contiguous address ranges, conserving memory by eliminating intermediates.



Hardware Protection Scheme:
In hardware protection, the paging unit and segmentation unit employ distinct schemes. Two privilege levels are defined for pages and Page Tables, governed by the User/Supervisor flag. When the flag is 0, the page is only accessible when the Current Privilege Level (CPL) is less than 3. When the flag is 1, the page is always accessible. Pages also have two access rights (Read and Write). If the Read/Write flag in a Page Directory or Page Table entry is 0, the corresponding Page Table or page can only be read; otherwise, it can be both read and written to.

Three-Level Paging:
Introduced for 64-bit architectures.
Typically, 16 KB page size is chosen, which results in a 14-bit Offset field.
The remaining 50 bits of the linear address are distributed between the Table and Directory fields.
In a three-level paging scheme, 30 bits of the address are split into three 10-bit fields.
Page Tables have 210 (1024) entries at each level.
Allows addressing large memory spaces efficiently.

Linux and Paging:
Linux relies heavily on paging for process management.
Assigns a different physical address space to each process, ensuring protection against addressing errors.
Distinguishes between pages (groups of data) and page frames (physical addresses in main memory).
Enables features like virtual memory, allowing pages to be stored in different page frames, swapped to disk, and reloaded in different page frames.
Paging is a fundamental aspect of Linux memory management.



The Linear Address field
PAGE_SHIFT: Specifies page size, usually 4096 bytes (4 KB).
PMD_SHIFT: Defines the size of an area mapped by a Page Middle Directory entry, often 4 MB.
PGDIR_SHIFT: Specifies the size of an area mapped by a Page Global Directory entry, also typically 4 MB.
And these macros tell us:
PTRS_PER_PTE: The number of entries in a Page Table, usually 1024.
PTRS_PER_PMD: Entries in a Page Middle Directory, typically 1.
PTRS_PER_PGD: Entries in a Page Global Directory, usually 1024.