PROJECT 4A: PAGING - KERNEL

Minimum Requirements: None

This should a short project but get familiar with paging so you are ready for project 4b.

TEST DISTRIBUTION

Public tests – 4 tests | 17 points

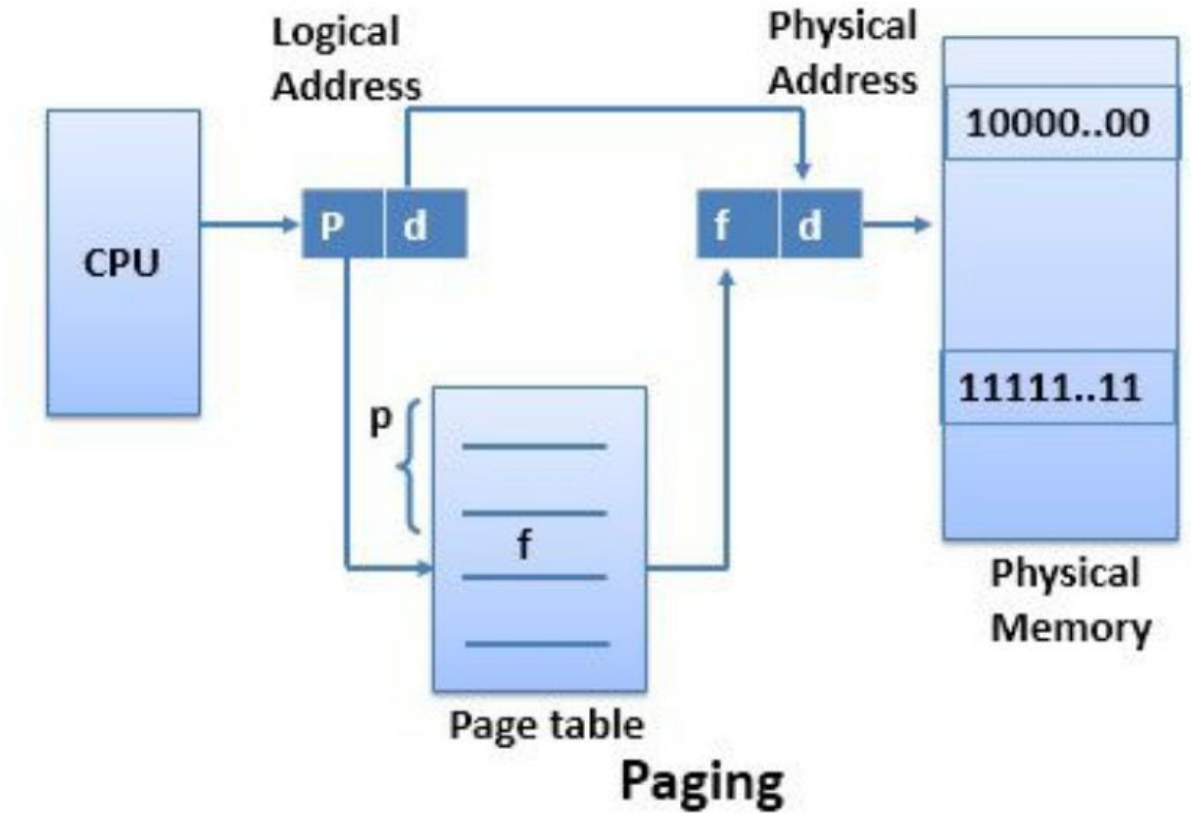
Release tests – 0 tests | 0 points

Secret tests – 1 tests | 5 points

WHAT IS PAGING?

- Paging allows physical address of a process to be non-continuous.
- The main idea behind the paging is to divide each process's memory space in the form of pages. The main memory will also be divided in the form of frames of the same size.
- The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.
- In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- Each Process has a page table which is the data structure to store the mapping between virtual addresses and physical addresses.

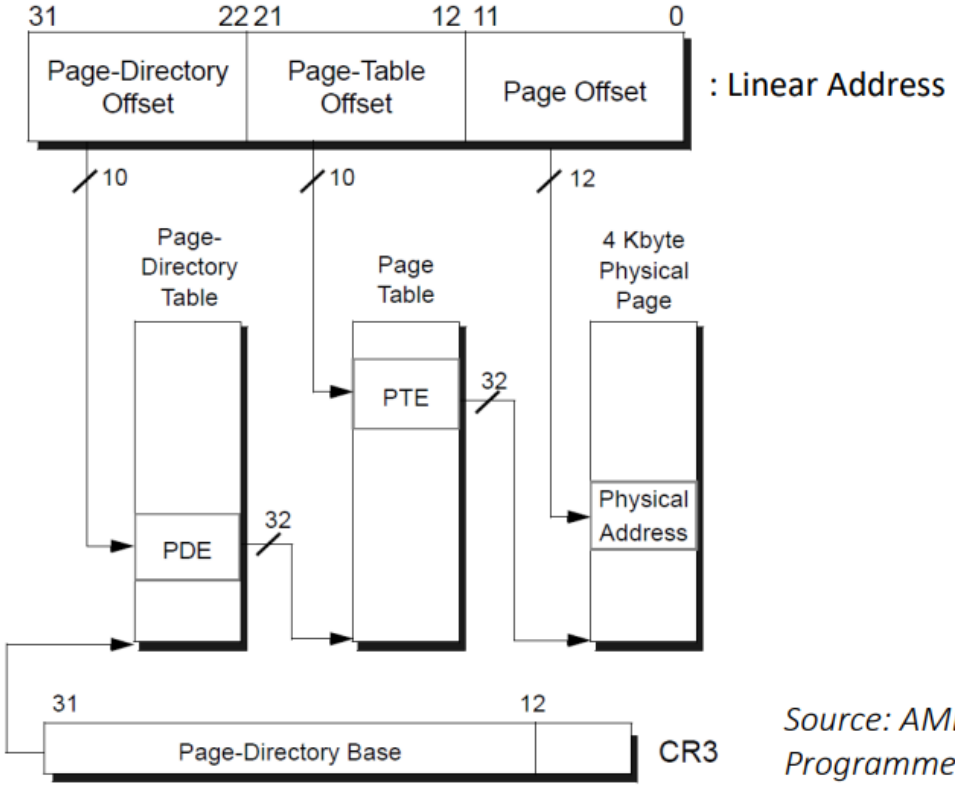
ONE LEVEL PAGING



Paging

TWO LEVEL PAGING

32-bit Translation Overview



Source: AMD64 Architecture Programmer's Manual (Volume 2)

ABOUT PREVIOUS PROJECTS

- You do not need Fork, Exec, Pipe or Signals for this project!
- Start with a fresh version of GeekOS!

FUNCTIONS TO IMPLEMENT/MODIFY

- Init_VM() (paging.c) – see page 9
- Identity_Map_Page() → optional (paging.c)
  - Clean way to map the kernel address space
- Init_Secondary_VM() (paging.c)
  - Used to initialize paging for secondary CPU, called in smp.c
  - Enable paging in this function.
- Kernel_Page_Dir() (paging.c)
  - Return the global page directory (static) that you have declared.
- Main() (main.c)
  - Include function calls to Init_VM() and checkPaging().
- You may have to make minor changes to other files depending on your implementation.

INIT_VM() (PAGING.C)

- Allocate a page for page directory (pde_t) and one page for each page table (pte_t) using Alloc_Page().
- Map entire kernel address space, use bootInfo->memSizeKB for kernel address space size, giving VM_READ, VM_WRITE and VM_USER access.
- Map APIC page and APIC-IO page (do not give VM_USER for this).
- Enable paging.
- Add interrupt handler for 14 (you can ignore 46 unless you see an unexpected interrupt 46).
- Use PAGE_DIRECTORY_INDEX and PAGE_TABLE_INDEX to check the indices of page directory and page table respectively.

INIT_VM() - MAPPING KERNEL ADDRESS SPACE

- Initialize fields of page directory
  - (important fields are present, pageTableBaseAddr and flags).
- Initialize fields of page table entry.
- PAGE_ALIGNED_ADDR is something you can look at for calculating the page base address.

WHAT DO WE MEAN BY "MAPPING" OF APIC/APIC-IO PAGES?

- It's a hardware and GeekOS accesses that hardware by mapping the hardware address to some address in the main memory.
- The kernel linear mapping page that contains the address APIC and APIC-IO is called the APIC page.
- Note: Their addresses are contained in only one page table.

SETUP AFTER 4A

- Page directory with 1024 pde_t structs pointing to page tables that cover size of the memory (bootInfo->memSizeKB).
- Each entry in the page table describes a 4KB memory region.
- Bytes 0 - 4095 should be unmapped (for checking null pointer dereference)
- Each pte_t (for now) maps a region in the virtual memory to the physical memory, and that mapping is identical (e.g. 0X100000 will map to 0X100000 after the two-level page table address translation which is done by hardware once you set up all the tables).
- Most memory regions are RWX for both the kernel and user.
- APIC regions are RW for the kernel and not user.