PROJECT 3A: PAGING – KERNEL

Minimum Requirements: None

This should be a short project but get familiar with paging so you are ready for project 3b.

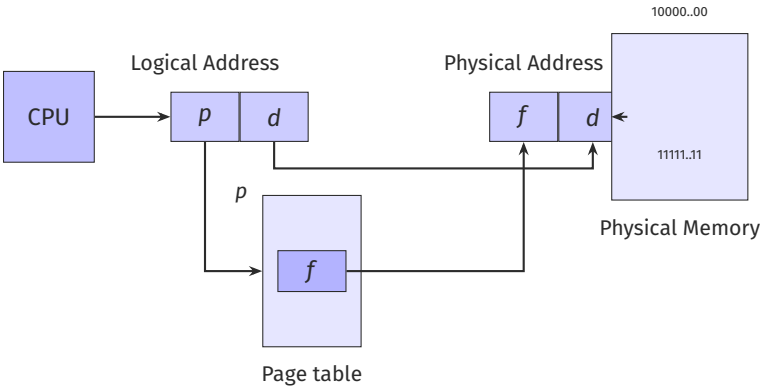
TEST DISTRIBUTION

Category	Tests	Points
Public tests	4	17
Release tests	0	0
Secret tests	1	5

WHAT IS PAGING?

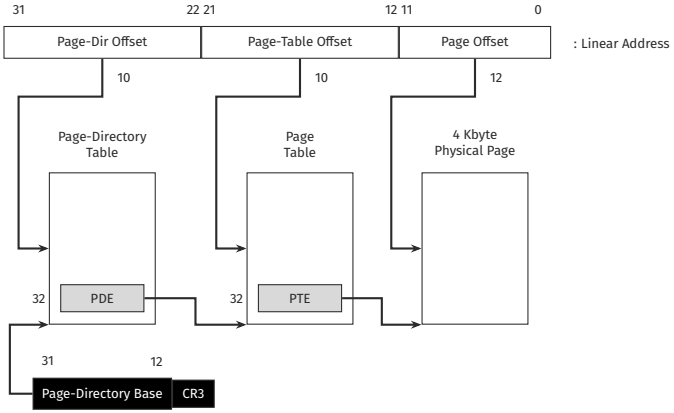
- Paging allows physical address of a process to be non-continuous.
- The main idea behind paging is to divide each process's memory space into *pages*. Main memory is also divided into *frames* of the same size.
- The mapping from virtual to physical address is done by the **Memory Management Unit (MMU)**,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

ONE LEVEL 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 next slide
- `Identity_Map_Page()` $\rightarrow$ 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 the 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 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 device 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 addresses APIC and APIC-IO is called the *APIC page*.
- Note: Their addresses are contained in only one page table.

SETUP AFTER 3A

- 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 4 KB memory region.
- Bytes 0–4095 should be **unmapped** (for checking null pointer dereference).
- Each `pte_t` (for now) maps a region in virtual memory to physical memory *identically* (e.g. `0x100000` $\rightarrow$ `0x100000` after two-level address translation done by hardware).
- Most memory regions are RWX for both the kernel and user.
- APIC regions are RW for the kernel and **not** user.