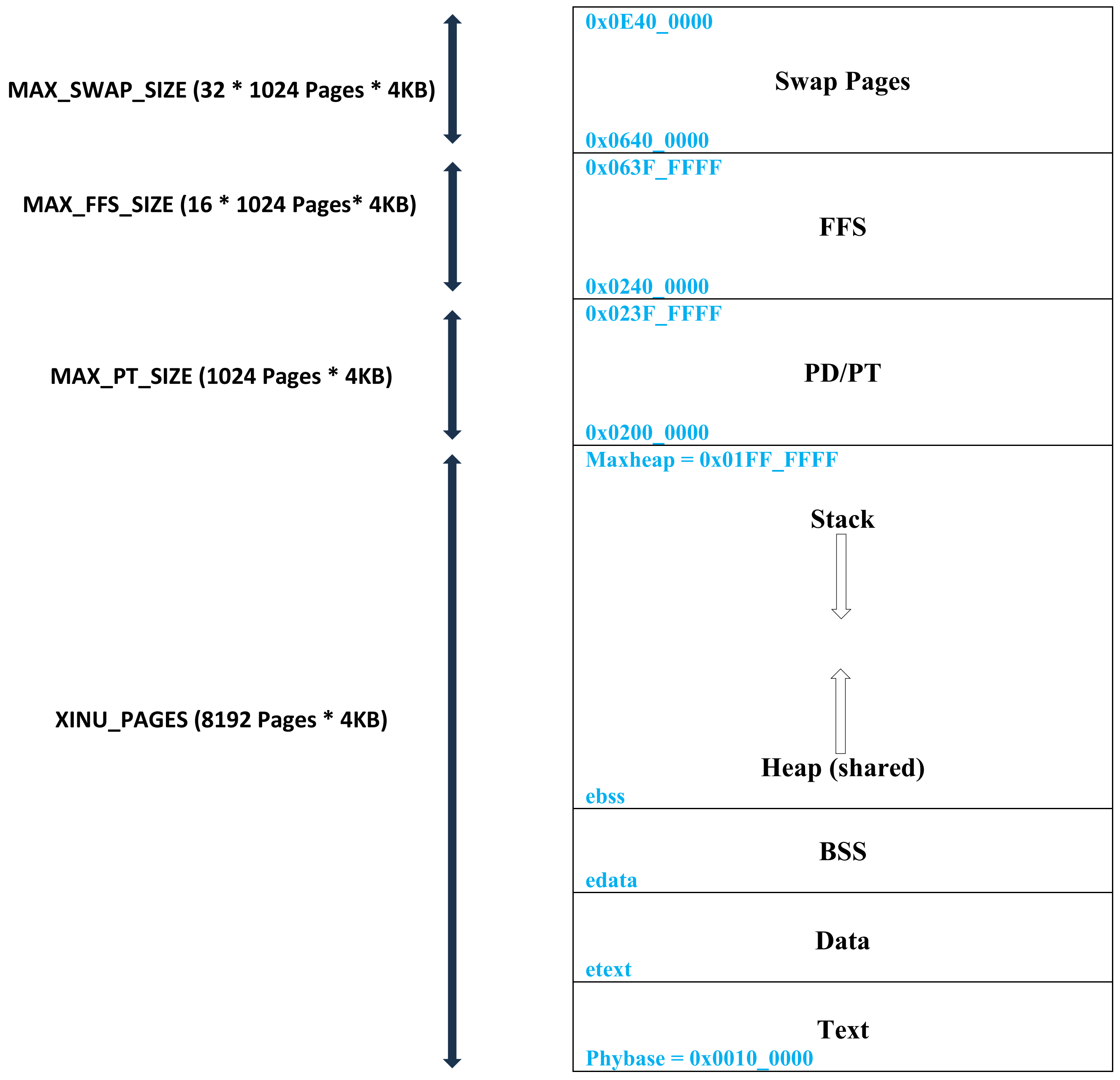
Project #4

Raghul Srinivasan and Vishvas Sudarshan

1. Physical memory layout



2. <u>**Initialization of page directories and page tables**</u>
   - First, we need to map the whole Xinu space so that all processes can have access to it. XINU_PAGES is defined as 8192 pages. This means we need 8 PDEs to create these mappings. Additionally, to access the area containing the PD/PTs we need to create a mapping. We initialize one more PDE for this, that can access 1024 pages.
     So, in total we map 9216 pages starting from address 0

   - All processes (system and user) will be mapped to XINU_PAGES.
   - System process will additionally have mappings to the PD/PT area.
   - When we initialize PDE/PTE for mapping XINU_Pages following bits will be set
     - Protection bit = 1
     - Present bit = 1
     - Valid bit = 1 (Valid bit not used for PDE) (x86 dosn't have a dedicated valid bit. We can use one of the three available bits as Valid bit)
     - pd_base/pt_base = Physical frame number of respective PTE/Xinu page.

   - When we initialize PDE/PTE for mapping virtual heap following bits will be set/reset
     - Protection bit = 1
     - Present bit = 0 (Because of lazy allocation)
     - Valid bit = 1 (Valid bit not used for PDE) (x86 dosn't have a dedicated valid bit. We can use one of the three available bits as Valid bit)
     - Physical address of respective XINU_page is stored in Base address field of PT.
     - Base address of PT is stored in Base address field of PD.
     - Base address of PD is stored in PCB of the respective process.
   - For the user processes max PDE/PTE count will be
     - Number of PTE->
       Number of XINU Pages + MAX_FFS_SIZE = 8192+ (16*1024) = **24 *1024**
     - Number of PDE->
       Number of PTE / (Page Size/ Single PTE Size) = (24*1024) / (4096/4B) = **24**

3. <u>**System Initialization**</u>
   - Paging is enabled at the end of sysinit() in initialize.c by setting the 31st bit of CR0. Function enable_paging() in control_reg.c does this task.
   - Before paging is enabled, we need address translation for xinu pages (Text, Data, BSS, Heap, Stack) as the physical memory will be accessed in granularity of pages.
   - Even for system processes, we have to initialize Page Directory and Page Tables with PTEs/PDEs for xinu pages.

4. **<u>Process Creation</u>**
   - Add additional Process Control Block elements -> PD_Base_Address and User_Flag
   - PD_Base_Address -> To save the Page Directory base address of that process. This value is used to load to CR3 register when it is running.
   - User_Flag -> To differentiate System process and User Process. It is set if process is created using vcreate().
   - While creating a process, add appropriate PDEs/PTEs for xinu pages (Text, Data, BSS, Heap, Stack). All these entry's valid bit should be set to 1.

5. **<u>Process Termination</u>**
   - When a process is to be killed, its PDE/PTE entries should be invalidated, the physical pages the process was using, should be freed. And the pages that were being used for storing the PTEs and the PDEs should also be freed.
   - We need to set the PDBR register (CR3) to the system PDBR.
   - We should also free any swap pages that was being used by this process.

6. **<u>Context Switch</u>**
   - During context switching, we have to change the content of CR3 register.
   - The old process's Page Directory base address which would have been loaded in CR3 register should be replaced with new process's Page Directory base register.
   - This can be done using the PCB field PD_Base_Address.

7. **<u>Heap allocation, deallocation and access</u>**
   a. Heap Allocation:
      - During heap allocation, we loop through all the addresses in the heap starting at the lowest address and check if we can find the required number of free pages sequentially.
      - Once we find 'n' sequential pages, we create page directory entries and page table entries for these addresses.
      - Since we are using lazy allocation, in the page table entries, we set Valid bit -> 1, Present bit -> 0.
      - We do not allocate a physical page at this time.

   b. Heap Deallocation:
      - We go through all the page table entries belonging to the addresses that are being freed.
      - First, we check if it is possible to free these addresses, that is, we check if we are trying to free an address that was never allocated. If such a condition occurs, we return SYSERR

    - Once we have verified that we are only freeing the correct addresses, for each page table entry, we set the valid bit to 0
    - If the present bit was set to 1, we free the corresponding physical page.

c. Access:
    - Physical memory is allocated and mapped to the virtual heap address space by the page fault handler when the page is being accessed for the first time.

8. Page Fault Handler Design

- Hardware will raise a page fault when PRESENT bit is 0. PRESENT bit will be 0 under two scenarios.
    - The page is being accesses for the first time (Not allocated yet)
    - The page does not exist in physical memory, it was evicted to swap space.

- The page fault hander first reads the value from CR2. This is the virtual address that caused the page fault.
- Then we check if these exists a valid page directory and page table entry for this address, if not, raise a segmentation fault and kill the process.
- Next, we allocate a new FFS page by going through the FFS free list.
- The instruction that caused the page fault will be executed again, and to indicate that the page has now been allocated, we set the preset bit to 1

9. Page Replacement and Swapping Design

- When the page fault hander is called, it can happen that the FFS free list is empty. Meaning there are no more physical pages available. In the scenario, we need to evict one page from the FFS region and move it to the swap area.
- We use the approximate LRU policy to identify which physical page to evict and move that page to swap area.
- Then we assign the address of the page that was just evicted to the process that called the handler.
- We also need to store a mapping between FFS pages and swap pages, this is required in the case where a page from FFS was evicted, then it was swapped back into FFS and then needs to be evicted again.
    - Here, we should not request a new swap page, but map the FFS page back to the same swap page it was previously using.