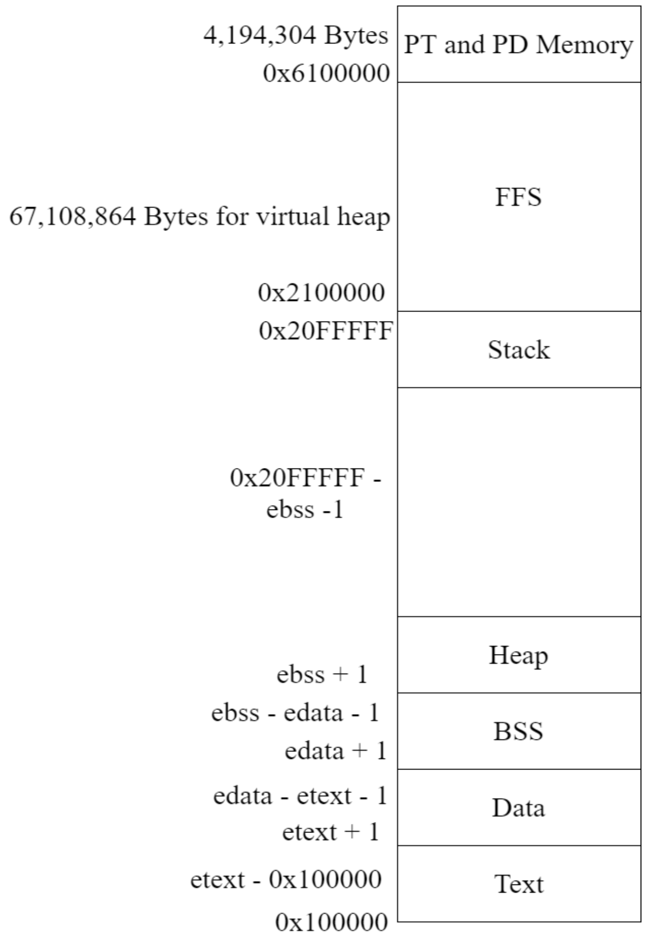
ECE 592 : Project #4 Design Document

Lakshmi Aasritha Pisupati, lpisupa; Kashyap Ravichandran, kravich2

Physical memory layout

Draw the physical memory layout, indicating the areas of memory devoted to code (text area), data, page directories and page tables, BSS, etc. For each memory area, indicate its start address and its size. Note that different teams may use different layouts.

The physical memory in xinu starts from 0x10000 and the xinu memory has 8192 pages and these pages are 4096 bytes in size. This leads us to 0x2100000 bytes of memory for xinu. The stack, heap, bss, data and text are within this. The FFS area and the PD-PT area are outside of this 0x210000 bytes of memory. The FFS area has 8192 pages and the PT-PD area has 1024 pages. Text, data and BSS are code dependent and we can't exactly represent them on a picture using a hard value. XINU has variables like etext, edata and ebss that helps us with this. They represent the last address in the text, data and the bss area respectively. Using this we would be able to calculate the size of the region and the starting address of the next region.



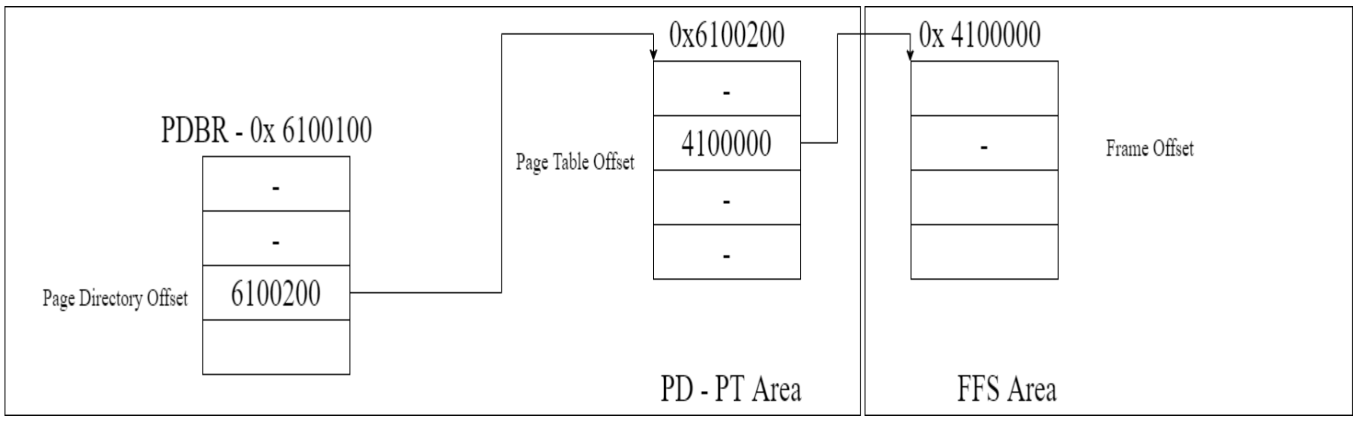
Initialization of page directories and page tables

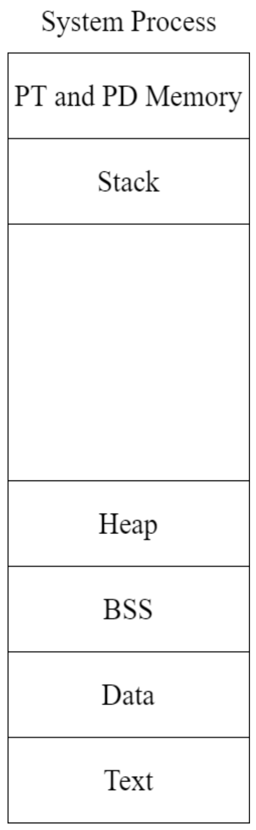
Indicate how page directories and page tables should be initialized. You don't need to show the exact content of page directories and page tables (i.e., the content of each entry). But, you need to indicate how many page directory/table entries you need for each memory area that should be mapped to the processes' virtual space. If different types of processes require different mappings, show them all.

In designing the memory layout and defining the initial content of page directories and page tables, you need to consider the following.

- Before paging is enabled, the OS uses only physical addresses. However, after paging is enabled, all memory accesses are through the virtual address space. You need to map the static segments (TEXT, DATA, etc.) in the virtual address space of all processes.
- User processes cannot map the whole FFS area: they can map only the portion of it that they use (that is, FFS area will be mapped only when `vmalloc` is invoked)
- **If you are taking the course at the graduate level**, in your implementation user processes cannot map the area devoted to page directories and page tables.

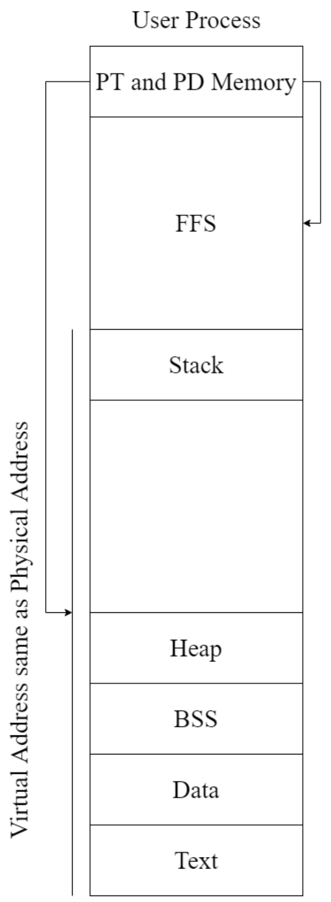
Each process gets a page directory table which is a page of size 4KB. Each page directory can map to 2^{10} page tables, again each page table is a page of size 4KB. So a page directory can map upto 1024 page tables, and each table can map to 1024 physical pages. A pictorial representation of how the page directory and page table is used to get to physical memory location is as shown below. The virtual address can be constructed as [Page Directory Offset][Page Table Offset][Frame Offset]. Page Directory offset lets us go to





Memory Layout for a system process

The memory layout as seen by a system process is shown on the left. The system process has to be mapped only to the xinu memory. Xinu memory takes 8192 pages. The virtual memory should also be equal to the physical memory so that the nothing changes from the perspective of the system process. So we need to have 8 page tables to map to 8192 pages as each page table can map 1024 physical pages. And we need to have 1 page directory for each system process to accommodate the 8 page tables. All the memory in the xinu pages needs to be mapped using the method mentioned above. But as mentioned earlier, the physical address must be equal to the virtual address.



Memory Layout for User Process

As mentioned earlier, we need to map the physical address space of xinu memory to virtual memory using PD and PT. These virtual addresses and the physical addresses must be equal. More than this, the user processes can allocate memory from the FFS space. Memory is allocated in pages. We need to have two free memory lists, one for PT-PD memory and one for FFS, to make sure that a user process can't access PT-PD memory. The FFS area once allocated, needs to be mapped using the method as mentioned earlier. However, the physical address need not be equal to the virtual address. As we call vmalloc from the user process, we map only a particular portion of the FFS area (The portion that is allotted for the user process). More on this is explained in the following sections.

System Initialization

Where is paging enabled and how? (see hints)

- Paging is enabled at the end of sysinit() in initialize.c by setting the bit 31 of the register CR0. This can be done by calling the function enable_paging present in control_reg.c

Process Creation

How do you need to modify process creation to support paging?

All processes, user or system gets a page directory. The number of page tables initially allocated to a process also depends on the type of process. Another important point here is that we use a getmem function to allocate memory from the shared heap structure and vmalloc to make the user process ready (give it a virtual address) to allocate memory from the FFS area using lazy allocation policy. Both functions take the number of bytes to be allocated as arguments, however, getmem operates on the granularity of bytes whereas vmalloc operates on pages. PD and PT are allocated only in the vmalloc, create and vcreate function.

The following functions need to be done by create function:

- Allocate a page directory, and page tables to map all the xinu memory to their virtual addresses.
- Store the address of the page directory in the PCB. So when we context switch to a new process, we can change the CR3 register to make sure that we get to the correct page directory and page table.

The following functions need to be done by vcreate function:

- Allocate a page directory, and page tables to map all the xinu memory to their virtual addresses.
- Store the address of the page directory in the PCB. So when we context switch to a new process, we can change the CR3 register to make sure that we get to the correct page directory and page table.
- Each user process needs to get a virtual heap apart from the shared heap memory.

Process Termination

How do you need to modify process termination to support paging?

- The virtual heap, page table and page directories are released using freemem(char *blkaddr, uint32 nbytes).
- The free lists of the PT&PD area and FFS are updated with the respective chunks of memory freed.

Context Switch

What should be done at context switch to support paging?

- The old processes' CR3 should be saved in the PCB and the new processes' PDBR should be retrieved from the PCB and moved on to CR3.
- Two more function arguments should be added to ctxsw: Pointer to the where the old processes CR3 will be stored, &oldproc->pdbr and the place from where the new processes' pdbr will be &newproc->pdbr.
- Contents can be loaded from and to the CR3 register using MOV instruction.

Heap allocation, deallocation and access

What should be done at heap allocation, deallocation and when the heap is accessed? Remember that you need to implement lazy allocation in your code.

At heap allocation:

- Heap allocation happens when a user process calls vmalloc to allocate nbytes of memory from the virtual heap.
- Since the policy is lazy allocation, no physical memory is allocated.
- The PDE and PTE are initialized, i.e. the valid bits for the number of pages being allocated are set to 1 and present is set to 0.

At heap deallocation:

- Heap deallocation happens when vfree is invoked by a user process. The function takes as argument the pointer of the FFS frame to be released and the number of bytes to be released. It releases nbytes starting from the FFS pointer.
- The valid bits and present bits in the PTE and PDE of the FFS are cleared if needed.
- The free lists are updated with the number of bytes freed.

During heap access:

- If heap is being accessed for the first time, the page fault handler is triggered because valid is set and present is clear.
- When the page fault handler sees that valid is set and present is clear, a new FFS is allocated from the free list. Page Table Entry is updated with the Physical Frame Number and its present bit is set.
- If the heap is not being accessed for the first time, the chunk of memory that is being accessed has a valid address. Since the global TLBs are invalidated everytime CR3 is loaded, address

translation is done to do a page table lookup. If the valid bit of either the Page Directory Entry or the Page Table Entry is not set, it is a segmentation fault.

Page Fault Handler Design

1. In which circumstances will the hardware raise a page fault?

   - Hardware raises a page fault if either the valid bit or the present bit of the entry being looked up are not set.

2. What operations should be performed by the page fault handler depending on the circumstances under which it is invoked?

   From 1, page fault handler is invoked in two scenarios:
   - If the valid bit of the entry being looked up is not set.
     - If this is the case, an entry that has not been allocated is being looked up. It is a segmentation fault. A Segmentation Fault is reported and the processes doing the memory access is killed.
   - If the valid bit is set but the present bit is not set.
     - In this case, the entry is being accessed for the first time. The OS will allocate a free frame of FFS from the free list and the Page Table Entry is updated with the Physical Frame Number and the present bit is set.