Device Driver Training
Session 5

Contents

- Physical & Virtual memory
- Memory management
- Virtual memory organization
- Memory mapping
- Kmalloc allocator
- Page allocators
- Vmalloc allocator
- Memory mapping to device

Physical & Virtual Memory

- The memory management is one of the most important roles of the OS

- Physical memory is memory that actually present in the machine
  - Need is for more memory than existing limited physical memory
  - To overcome this limitation, the most successful way is virtual memory

- Virtual memory is much larger than the physical memory
  - Linux is, of course, a virtual memory system, meaning that the addresses seen by user programs do not directly correspond to the physical addresses used by the hardware
  - Programs can transparently use this though the fact is only limited physical memory available
  - Even a single process can have a virtual address space larger than the system's physical memory

Physical & Virtual Memory cont..

- Memory is divided into discrete units called pages
- Much of the system's internal handling of memory is done on a per page basis
- Page size varies from one architecture to the next, although most systems currently use 4096 byte pages
- The constant PAGE_SIZE (defined in <asm/page.h>) gives the page size on any given architecture
- Kernel and user space work with virtual addresses that are mapped to physical addresses by the memory management hardware
- This mapping is defined by page tables, set up by OS

Memory management features

- Large Address Spaces
- Protection
  - Each process in the system has its own virtual address space completely separate from each other
  - A process running one application cannot affect another
- Memory Mapping
  - The contents of a file are linked directly into the virtual address space of a process
- Fair Physical Memory Allocation
  - Each running process in the system has fair share of the physical memory
- Shared Virtual Memory
  - Dynamic libraries are common example of executing code shared between several processes

Virtual memory organization

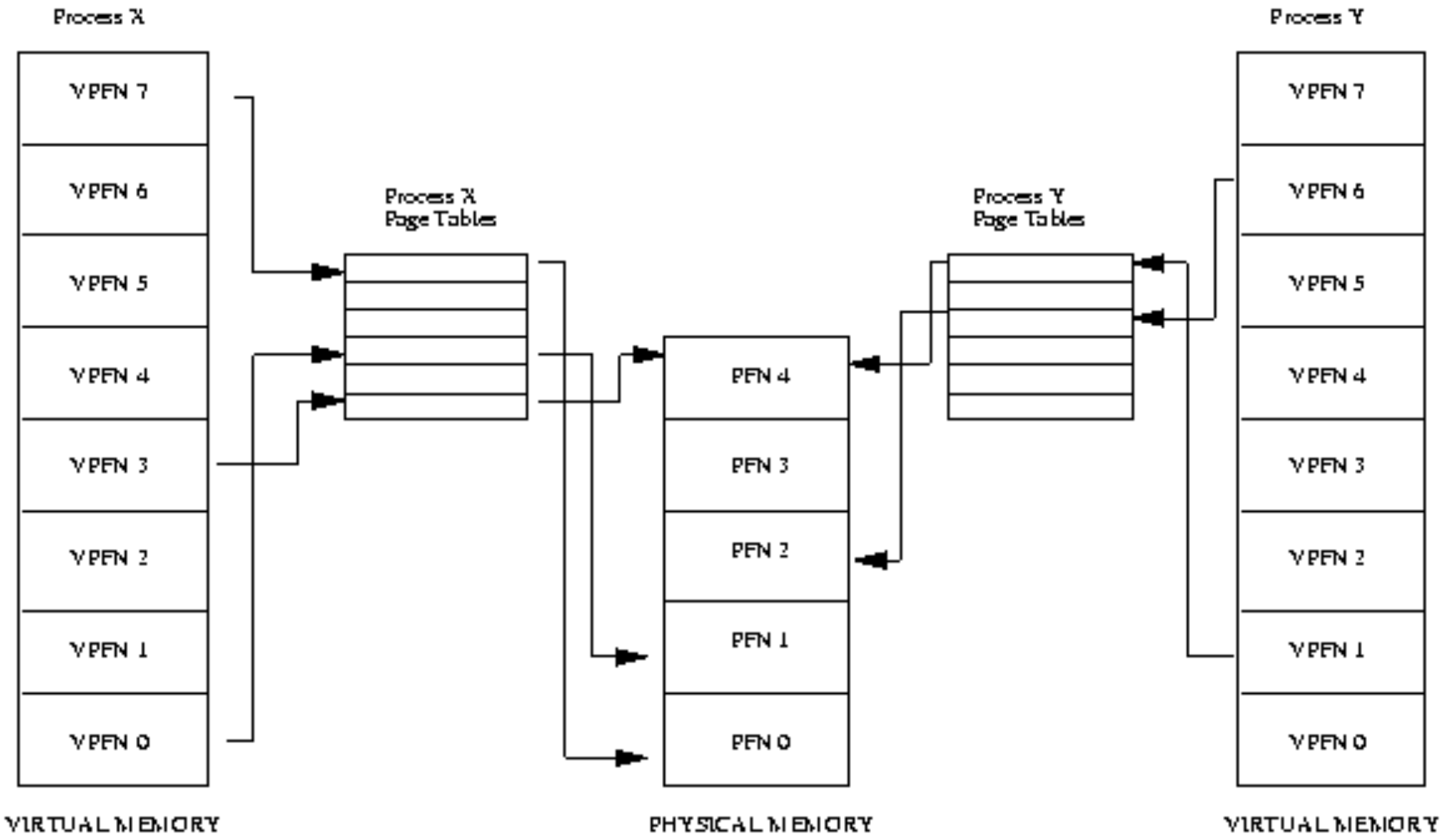
- Kernel space
  - 1GB reserved for kernel-space
  - Contains kernel code and core data structures
  - Most memory can be a direct mapping of physical memory at a fixed offset

- User space processes
  - 3GB mapping available for each user space process
  - Process code and data (program, stack, ...)
  - Not necessarily mapped to physical memory

Memory mapping

- Processor always access memory either to fetch instructions or to fetch and store data
- As mentioned, memory is divided in pages and each of these pages is given a unique number called page frame number
- For the page size of 4K, Virtual address is made up of 2 fields:
  - 12 least significant bits are the offset
  - The remaining, higher bits indicate the page number
  - Each time processor encounters a virtual address, it must extract offset and virtual PFN
  - Processor must translate virtual PFN to physical one and then access the location at the correct offset into physical page
- To do this, processor uses page table

Memory mapping cont..



Memory mapping cont…

- Page table maps virtual pages of each process into physical pages in memory
  - Diagram shows virtual address spaces of process X & Y with their own page tables
  - Process X's virtual PFN 0 is mapped into memory in physical PFN 1
  - Process Y's virtual PFN 1 is mapped into memory in physical PFN 4
  - Pages of virtual memory do not have to be present in physical memory in any particular order
  - Assuming a page size of 0x2000 bytes, and address of 0x2194 in process Y's virtual address space, then the processor would translate that address into offset 0x194 into virtual PFN 1
  - As process Y's physical PFN for Virtual PFN 1 is 4, thus physical address for virtual address 0x2194 would be 0x8194

Kernel memory allocators

- Kernel memory allocators like kmalloc, vmalloc allocate physical pages
- Kernel allocated memory cannot be swapped out
- Fault handling not required for kernel memory
- Most kernel memory allocation functions also return a kernel virtual address to be used within the kernel space

Kmlloc & kfree

- The kmalloc allocator is the general purpose memory allocator in the Linux kernel
- The allocated area is physically contiguous
- Easy to learn due to similarity with malloc
- Should be used as the primary allocator unless there is a strong reason to use another one
  - void *kmalloc(size_t size, int flags);
    - Size – size of the block to be allocated
    - Flags – controls the behavior of kmalloc in a number of ways
- kfree - Frees an allocated area
  - void kfree(const void *objp);
- Defined in <linux/slab.h>

Kmalloc flags

- GFP_ATOMIC
  - Used to allocate memory from interrupt handlers and other code outside of a process context. Never sleeps
- GFP_KERNEL
  - Normal allocation of kernel memory. May sleep
- GFP_DMA
  - Allocates memory in an area of the physical memory usable for DMA transfers
- All the flags are defined in <linux/gfp.h>

Kmalloc flags cont..

- **GFP_KERNEL** - means that the allocation is performed on behalf of a process running in kernel space.
  - Using GFP_KERNEL means that kmalloc can put the current process to sleep waiting for a page when called in low-memory situations.
  - A function that allocates memory using GFP_KERNEL must, therefore, be reentrant and cannot be running in atomic context.
  - While the current process sleeps, the kernel takes proper action to locate some free memory, either by flushing buffers to disk or by swapping out memory from a user process.
- **GFP_ATOMIC -** sometimes kmalloc is called from outside a process's context
  - This type of call can happen, for interrupt handlers, tasklets, and kernel timers
  - In this case, the current process should not be put to sleep, and the driver should use this flag

Vmalloc allocator

- vmalloc() allocator can be used to obtain virtually contiguous memory zones, but not physically contiguous
- Although the pages are not consecutive in physical memory the kernel sees them as a contiguous range of addresses
- vmalloc returns 0 (the NULL address) if an error occurs, otherwise, it returns a pointer to a linear memory area of size at least size
- API in include/linux/vmalloc.h
  - void *vmalloc(unsigned long size);
  - void vfree(void *addr);
- One minor drawback of vmalloc is that it can't be used in atomic context because, internally, it uses kmalloc(GFP_KERNEL) to acquire storage for the page tables, and therefore could sleep

Vmalloc allocator cont..

- The right time to use vmalloc is when we are allocating memory for a large sequential buffer that exists only in software
- It's important to note that vmalloc has more overhead than other allocation mechanism like__get_free_pages, because it must both retrieve the memory and build the page tables
- Therefore, it doesn't make sense to call vmalloc to allocate just one page
- An example of a function in the kernel that uses vmalloc is the create_module system call, which uses vmalloc to get space for the module being created
- Code and data of the module are later copied to the allocated space using copy_from_user
- In this way, the module appears to be loaded into contiguous memory

Page allocator

- If a module needs to allocate big chunks of memory, it is usually better to use a page- oriented technique
- The allocated area is virtually contiguous, but also physically contiguous
- To allocate pages, the following functions are available:
  - get_zeroed_page(unsigned int flags);
    - Returns a pointer to a new page and fills the page with zeros
  - __get_free_page(unsigned int flags);
    - Similar to get_zeroed_page, but doesn't clear the page
    - Flags same as kmalloc

Page allocator

- __get_free_pages(unsigned int flags, unsigned int order);
  - Allocates and returns a pointer to the first byte of a memory area that is potentially several (physically contiguous) pages long but doesn't zero the area
  - Order is the base-two logarithm of the number of pages you are requesting or freeing (i.e., log2*N*).
    - For example, order is 0 if you want one page and 3 if you request eight pages.
    - If order is too big (no contiguous area of that size is available), the page allocation fail

- To free pages, followig functions are available:
  - void free_page(unsigned long addr);
  - void free_pages(unsigned long addr, unsigned long order);

Memory mapping to device

- Memory mapping can be implemented to provide user programs with direct access to device memory
- Mapping a device means associating a range of user space addresses to device memory
- Whenever the program reads or writes in the assigned address range, it is actually accessing the device
- For example, mapping implemented for X server, it allows quick and easy access to the video card's memory
- For a performance critical application like this, direct access makes a large difference
- Linux provides this by mmap abstraction

Mmap cont..

- Not every device lends itself to the mmap abstraction; for instance, for serial ports and other stream oriented devices
- Another limitation of mmap is that mapping is PAGE_SIZE grained
- The kernel can manage virtual addresses only at the level of page tables; therefore, the mapped area must be a multiple of PAGE_SIZE and must live in physical memory starting at an address that is a multiple of PAGE_SIZE
- The kernel forces size granularity by making a region slightly bigger if its size isn't a multiple of the page size
- The mmap method is part of the file_operations structure and is invoked when the mmap system call is issued
- With mmap, the kernel performs a good deal of work before the actual method is invoked, and, therefore, the prototype of the method is quite different from that of the system call

Thank you