Virtual Memory in Linux

Abdulhamit MABOÇOĞLU

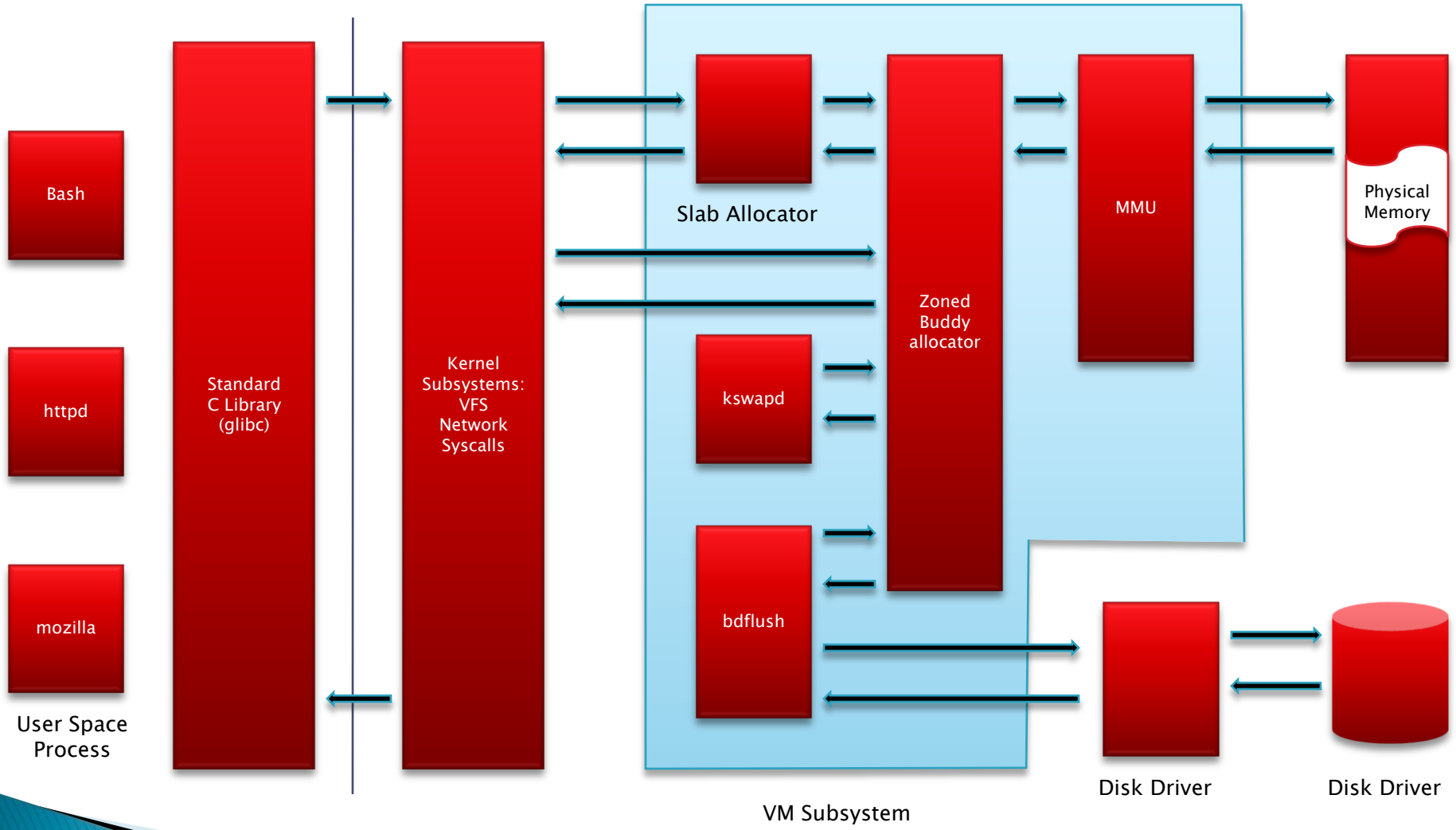
CENG425 System Programming

What is Virtual Memory ?

- **Virtual memory** is an addressing scheme implemented in hardware and software that allows non-contiguous memory to be addressed as if it were contiguous.

- Virtaul Memory (VM) allows an operating system to perform many of its advanced functions, such as process isolation, file caching, and swapping.

What Comprises a VM



High Level Overview of VM Subsystem

MMU

- The Memory Management Unit (MMU) is the hardware base that makes a VM system possible.
- The MMU allows software to reference physical memory by aliased addresses, quite often more than one.

Zoned Buddy Allocator

- The Zoned Buddy Allocator is responsible for the management of page allocations to the entire system.

- This code manages lists of physically contiguous pages and maps them into the MMU page tables, so as to provide other kernel subsystems with valid physical address ranges when the kernel requests them (Physical to Virtual Address mapping is handled by a higher layer of the VM).

Zoned Buddy Allocator

- The Buddy Allocator also manages memory zones.

DMA

- This zone consists of the first 16 MB of RAM, from which legacy devices allocate to perform direct memory operations.

NORMAL

- This zone encompasses memory addresses from 16 MB to 1 GB and is used by the kernel for internal data structures as well as other system and user space allocations.

HIGMEM

- This zone includes all memory above 1 GB and is used exclusively for system allocations (file system buffers, user space allocations, etc).

Slab Allocator

- The Slab Allocator provides a more usable front end to the Buddy Allocator for those sections of the kernel which require memory in sizes that are more flexible than the standard 4 KB page.

- The Slab Allocator allows other kernel components to create caches of memory objects of a given size.

Slab Allocator

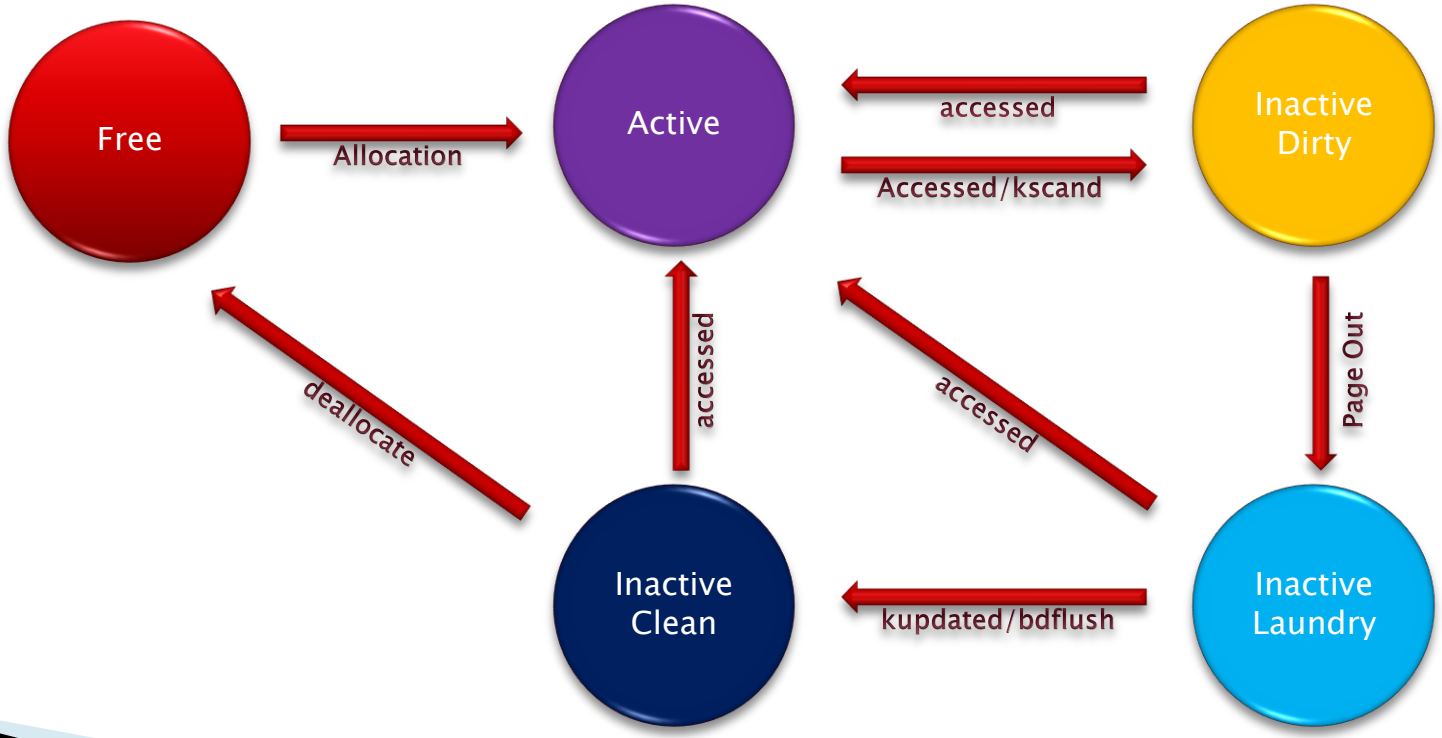
- The Slab Allocator is responsible for placing as many of the cache's objects on a page as possible and monitoring which objects are free and which are allocated.
- The Slab Allocator may only allocate from the DMA and NORMAL zones.

Kernel Threads

- The last component in the VM subsystem are the kernel threads: kscand, kswapd, kupdated, and bdflush.
- These tasks are responsible for the recovery and management of in use memory.
- The active tasks in the kernel related to VM usage are responsible for attempting to move pages out of RAM

The Life of a Page

- All of the memory managed by the VM is labeled by a state.



VM Page State Machine

The Life of a Page

- Dependent on the current needs of the system, the VM may transfer pages from one state to the next, according to the state machine.

- Using these states, the VM can determine what is being done with a page by the system at a given time and what actions the VM may take on the page.

The States

FREE

- All pages available for allocation begin in this state. This indicates to the VM that the page is not being used for any purpose and is available for allocation.

ACTIVE

- Pages which have been allocated from the Buddy Allocator enter this state. It indicates to the VM that the page has been allocated and is actively in use by the kernel or a user process

INACTIVE DIRTY

- This state indicates that the page has fallen into disuse by the entity which allocated it and thus is a candidate for removal from main memory.

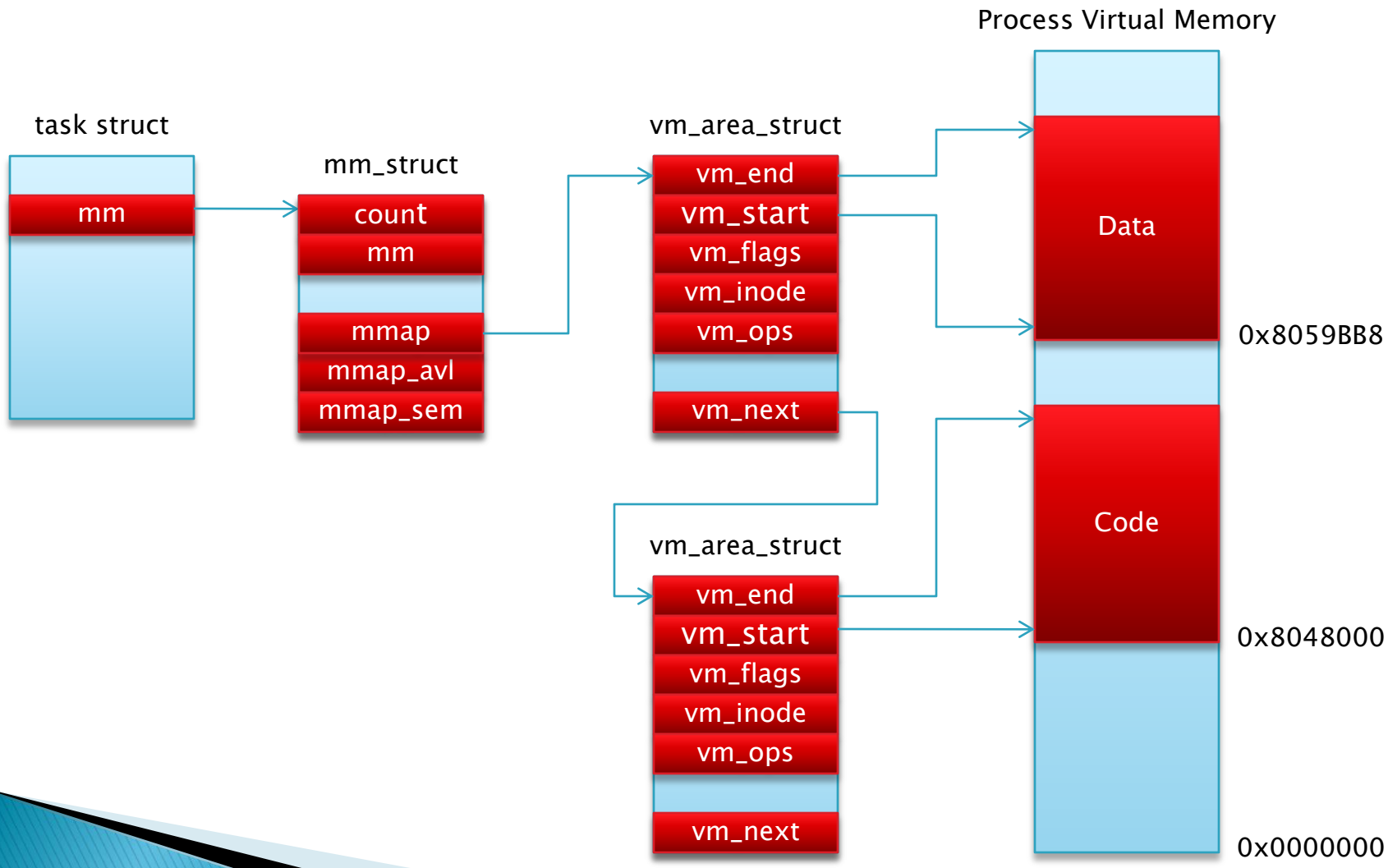
INACTIVE LAUNDERED

- This is an interim state in which those pages which have been selected for removal from main memory enter while their contents are being moved to disk.

INACTIVE CLEAN

- Pages in this state have been laundered. This means that the contents of the page are in sync with the backed up data on disk.

Process Virtual Memory



Process Virtual Memory

- In any given time period a process will not have used all of the code and data contained within its virtual memory.
- When the process attempts to access the code or data the system hardware will generate a page fault and hand control to the Linux kernel to fix things up.
- Therefore, for every area of virtual memory in the process's address space Linux needs to know where that virtual memory comes from and how to get it into memory so that it can fix up these page faults.

Page Faults

- First, the CPU presents the desired address to the MMU.
- However, the MMU has no translation for this address. So, it interrupts the CPU, and causes software known as a page fault handler to be executed.

Page Faults

- The page fault handler then determines what must be done to resolve this page fault. It can:
- Find where the desired page resides on disk and read it in (this is normally the case if the page fault is for a page of code)
- Determine that the desired page is already in RAM (but not allocated to the current process) and direct the MMU to point to it
- Point to a special page containing nothing but zeros and later allocate a page only if the page is ever written to (this is called a *copy on write* page)
- Get it from somewhere else (more on this later)
- While the first three actions are relatively straightforward, the last one is not.

Process Virtual Memory

- When a process allocates virtual memory, Linux does not actually reserve physical memory for the process.

- Instead, it describes the virtual memory by creating a new vm_area_struct data structure. This is linked into the process's list of virtual memory.

Process Virtual Memory

- When the process attempts to write to a virtual address within that new virtual memory region then the system will page fault.
- Linux looks to see if the virtual address referenced is in the current process's virtual address space. If it is, Linux creates the appropriate PTEs and allocates a physical page of memory for this process.
- The code or data may need to be brought into that physical page from the file system or from the swap disk. The process can then be restarted at the instruction that caused the page fault and, this time as the memory physically exists, it may continue.

What is Swap ?

- The kernel will write the contents of a currently unused block of memory to the hard disk so that the memory can be used for another purpose.
- When the original contents are needed again, they are read back into memory.
- The part of the hard disk that is used as virtual memory is called the *swap space*

Swapping

- Pages from a process are swapped
- The process becomes run able and attempts to access a swapped page
- The page is faulted back into memory
- A short time later, the page is swapped out again

Creating a swap space

- A swap file is an ordinary file; it is in no way special to the kernel.
- The only thing that matters to the kernel is that it has no holes, and that it is prepared for use with **mkswap**.

Creating a swap space

- One good way to create the swap file without holes is through the following command:

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

Creating a swap space

- After you have created a swap file or a swap partition, you need to write a signature to its beginning;

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480
bytes
$
```

Using a swap space

- An initialized swap space is taken into use with **swapon**.

- This command tells the kernel that the swap space can be used. The path to the swap space is given as the argument, so to start swapping on a temporary swap file one might use the following command.

```
$ swapon /extra-swap
$
```

Using a swap space

- Swap spaces can be used automatically by listing them in the /etc/fstab file.

```
/dev/hda8       none    swap    sw      0       0
/swapfile       none    swap    sw      0       0
```

- The startup scripts will run the command **swapon -a**, which will start swapping on all the swap spaces listed in **/etc/fstab**. Therefore, the **swapon** command is usually used only when extra swap is needed.

Using a swap space

- You can monitor the use of swap spaces with **free**. It will tell the total amount of swap space used.

```
$ free
                 total       used       free     shared
 buffers
Mem:             15152      14896        256      12404       2528
 -/+ buffers:               12368       2784
Swap:            32452       6684      25768
$
```

References

- The Linux System Administrator's Guide, Chapter 7. Memory Management, http://www.faqs.org/docs/linux_admin/index.html
- Linux Knowledge Base and Tutorial, Process Virtual Memory, http://www.linux-tutorial.info/modules.php?name=Tutorial&pageid=322
- Red Hat Magazine, Understanding Virtual Memory, http://www.redhat.com/magazine/001nov04/features/vm
- Red Hat Linux 8.0: The Official Red Hat Linux System Administration Primer, Chapter 4. Physical and Virtual Memory, http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/admin-primer/s1-memory-virt-details.html