ECE391
Computer System Engineering

Lecture 23

Dr. Jian Huang

University of Illinois at Urbana- Champaign

Fall 2018

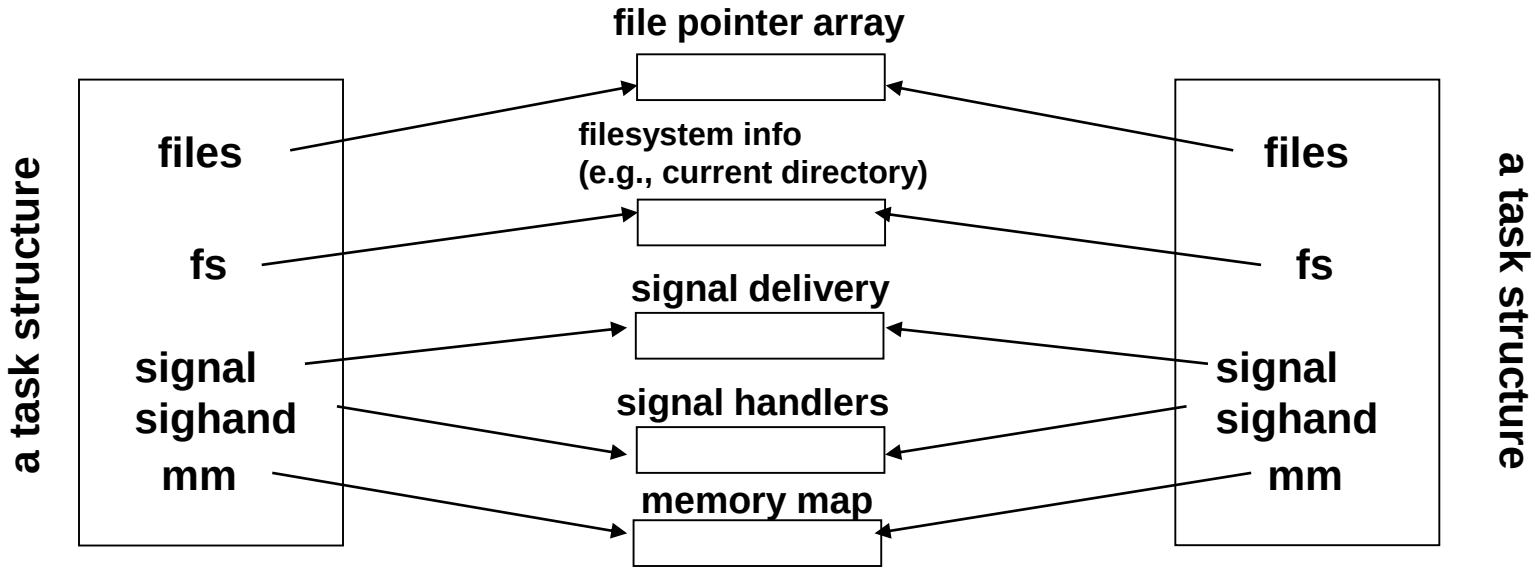
[Some slides of this lecture are borrowed from Sarah Diesburg's COP5641 course at University of Iowa]

Lecture Topics

- Memory maps and regions

What can be shared?

- Some state may be shared between tasks



- Note: each of the shared structures uses reference counts and locks to manage interactions between tasks and to track usage

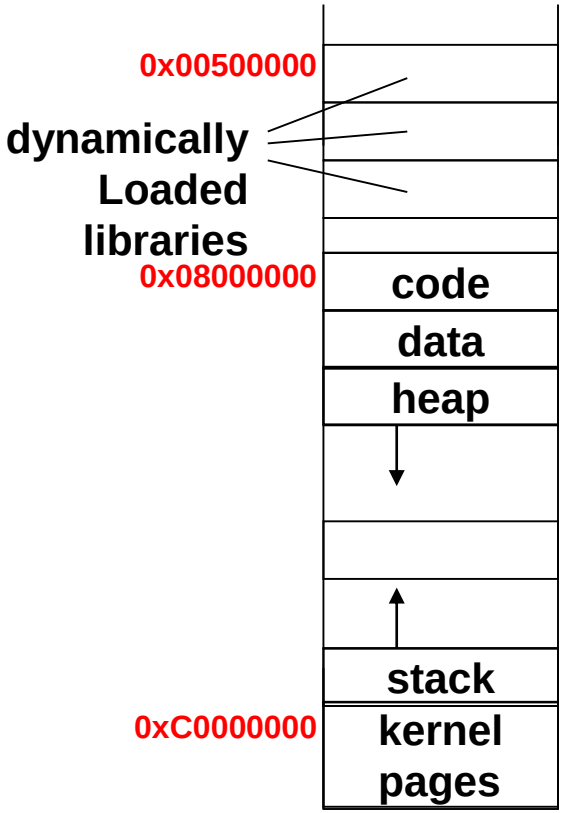
What can be shared?

- The task structure also includes a tgid (thread group id) field
  - same for all threads in address space
  - takes the place of the traditional Unix process id
- Remember the *do_fork* call ?
  - the *clone_flags* argument of do_fork controls sharing
  - between the new task and the task that made the call to do_fork

CLONE_FILES	use the same file pointer array
CLONE_FS	use the same file system info
CLONE_SIGHAND	use the same signal handlers
CLONE_VM	use the same memory map
CLONE_PARENT	copy the parent (new sibling instead of child)
CLONE_THREAD	in the same thread group as caller task (implies signal sharing, VM, SIGHAND)

Memory Maps

- How can we abstract a program's memory map?
  - mostly empty space for any program
  - but can be abstracted as a set of contiguous regions
- Use a structure to track:
  mm_struct (linux/sched.h)
- Use another structure for each region:
  vm_area_struct (linux/mm.h)
- Things in the mm_struct include
  - linkage, statistics, shortcuts
  - synchronization, some other stuff



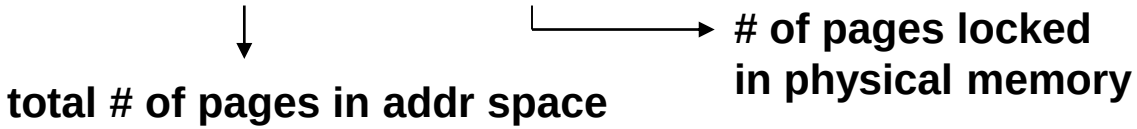
Memory Maps

```
/* mm_struct linkage */
struct vm_area_struct* mmap;
                                /* ordered linked list of regions */

rb_root   mm_rb;                /* red-black tree of regions (for
                                speed/scalability of # of regions) */

pgd_t*    pgd;                  /* page global directory pointer */
atomic_t  mm_users;             /* # of threads/LWPs */
atomic_t  mm_count;             /* # of refs. including temp. use
                                by kernel threads */
```

Memory Maps

- Example statistics

```
int map_count                    /* # of regions */
unsigned long total_vm, locked_vm;
```



`get_mm_rss` macro returns resident set size

(# of pages currently in phys. mem; sum of two fields)

Memory Maps

- Shortcuts

```
unsigned long start_code, end_code;     /* code  */
unsigned long start_data, end_data;     /* data  */
unsigned long start_brk,  brk;          /* heap  */
unsigned long start_stack;              /* stack */
unsigned long arg_start,  arg_end;      /* args  */
unsigned long env_start,  env_end;      /* env.  */

struct vm_area_struct* mmap_cache;      /* last region
                                           referenced */
```

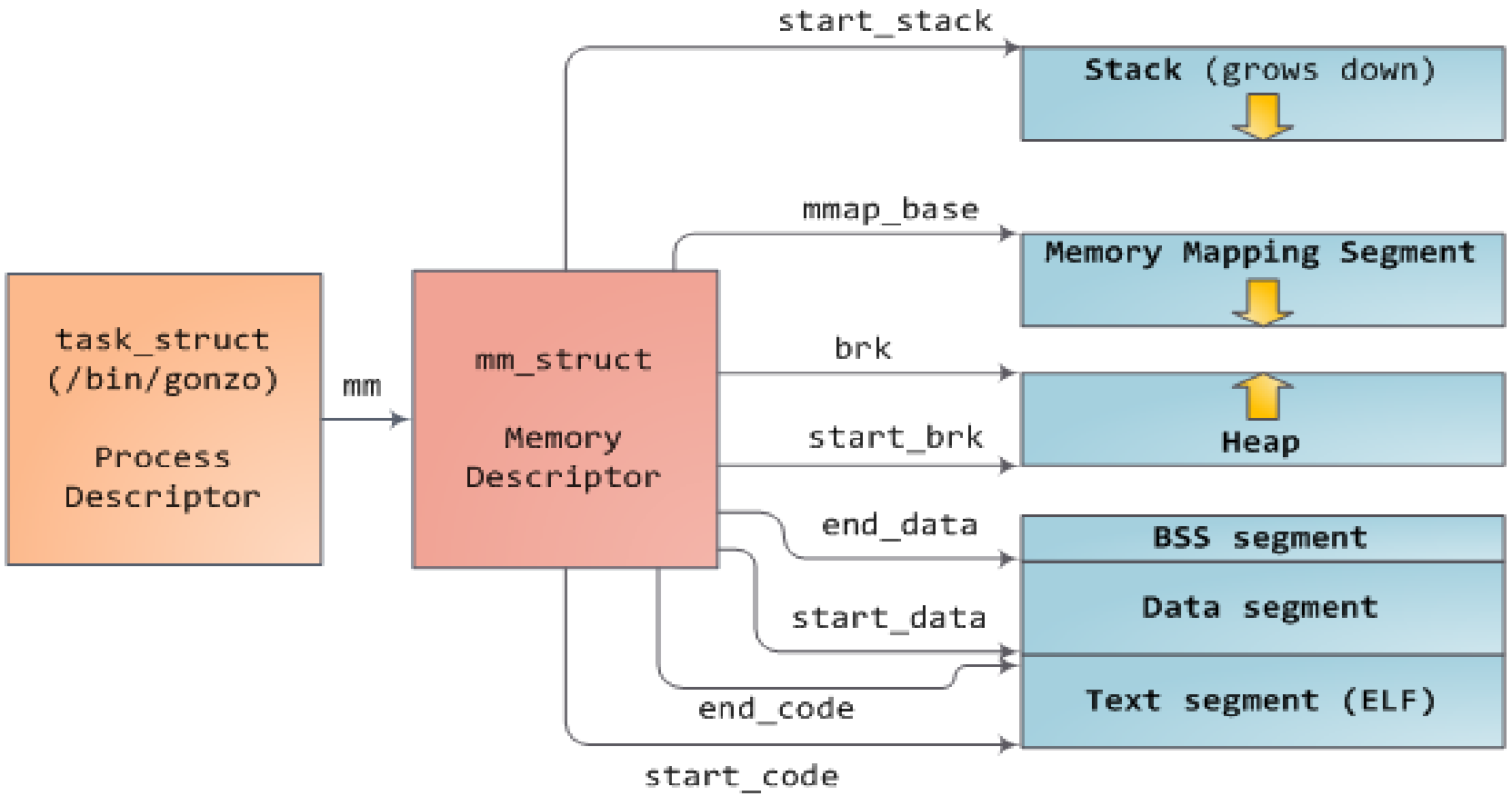
Memory Maps

- Synchronization

```
struct rw_semaphore mmap_sem; /* r/w sem. for regions */
spinlock_t page_table_lock;   /* page table spin lock */
```

- Other stuff includes
  - more shortcuts
  - pointers for link list of memory maps
  - architecture-specific MMU context

Memory Descriptor



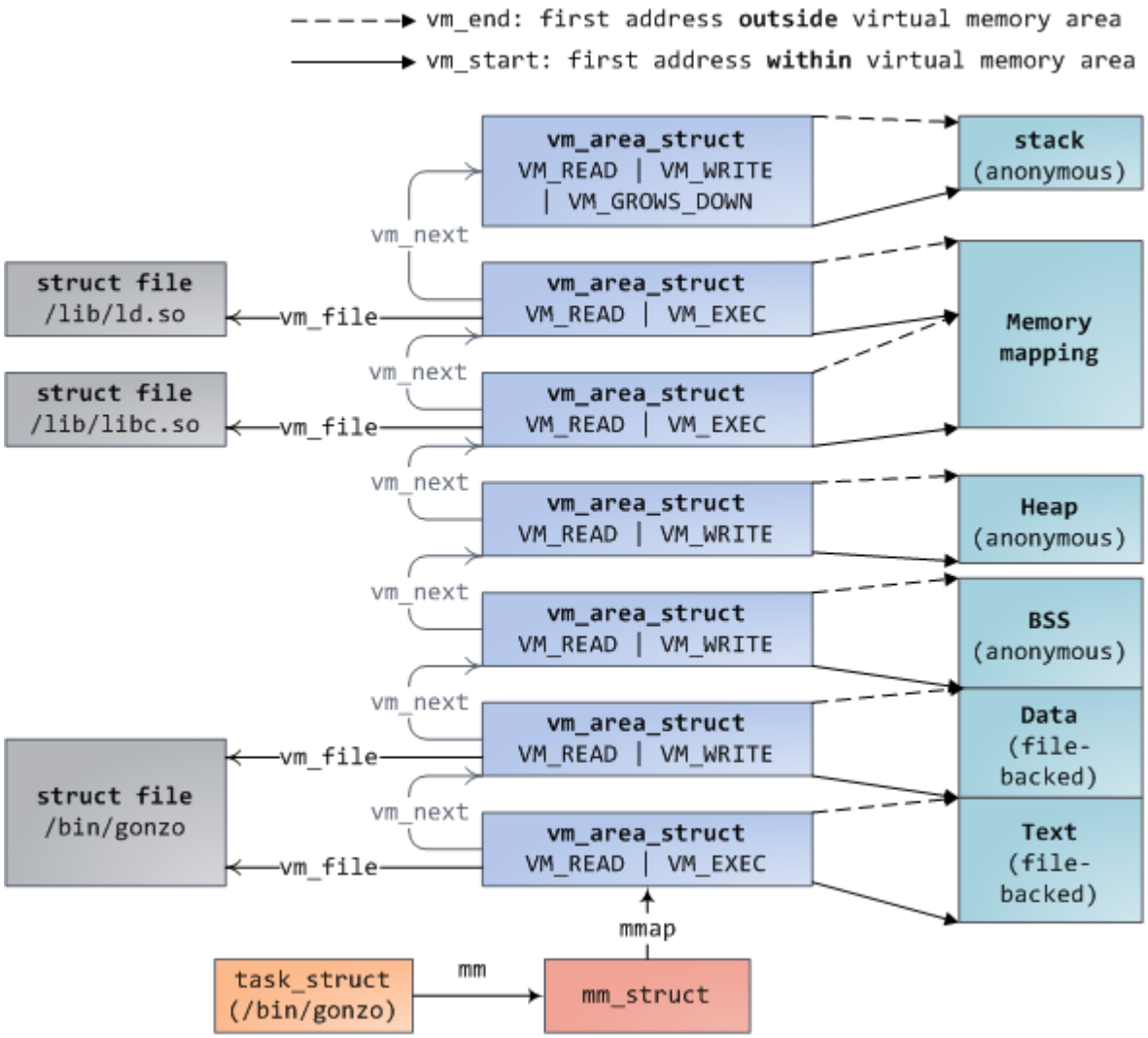
Process Memory Map

- struct mm_struct - contains list of process' VMAs, page tables, etc.
- accessible via current-> mm
- The threads of a process share one struct mm_struct object

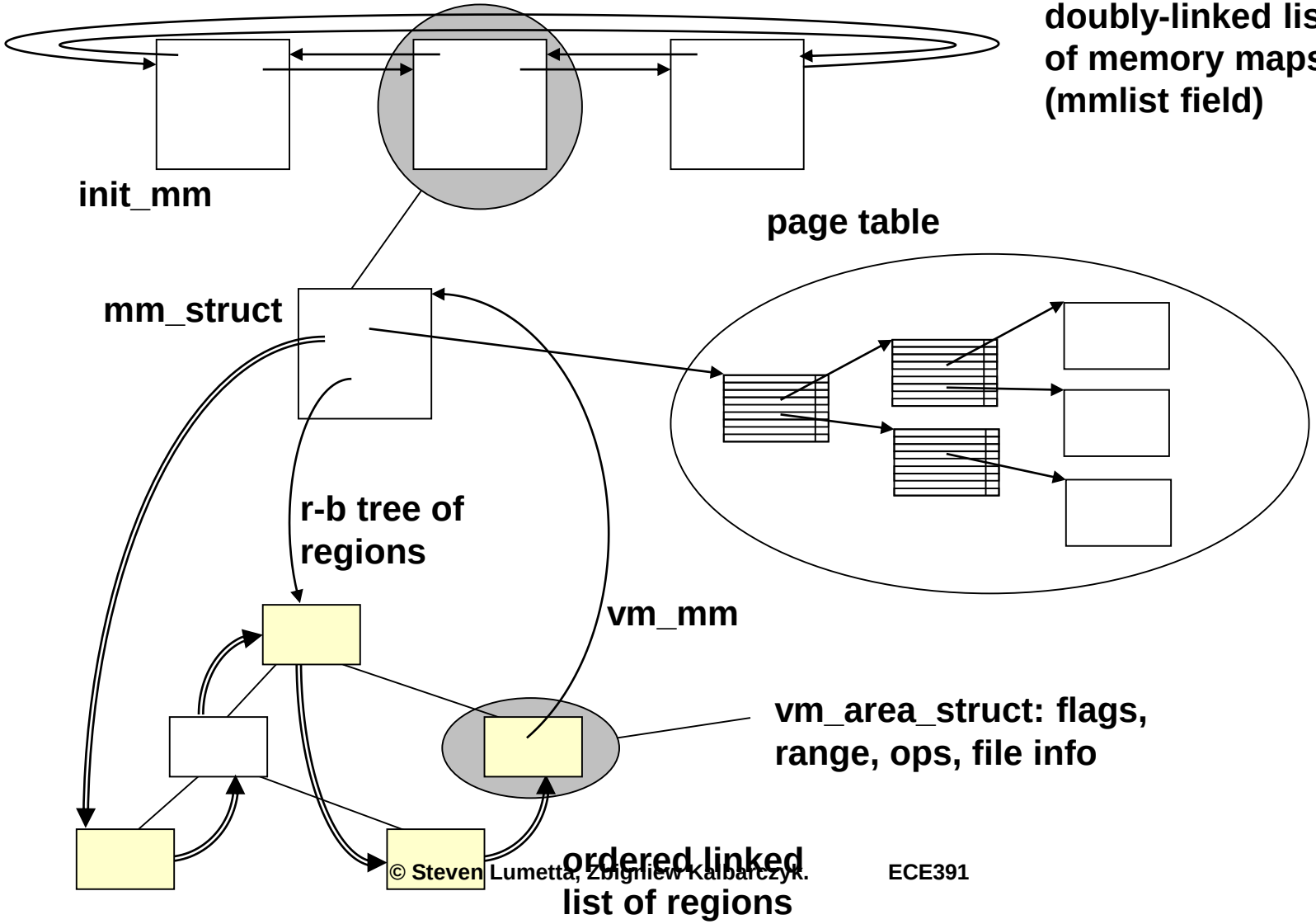
Virtual Memory Regions

- A range of contiguous VM is represented by an object of type struct vm_area_struct.
- Used by kernel to keep track of memory mappings of processes
- Each is a contract to handle the VMem→PMem mapping for a given range of addresses
- Some kinds of areas:
  - Stack, memory mapping segment, heap, BSS, data, text

Virtual Memory Area Mapping Descriptor



Memory Map Organization



Memory Map Organization

- Now let's look at the region abstraction
  - the vm_area_struct
  - (take a look at /proc/<pid>/maps, or /proc/self/maps, for a list in human-readable form)

- partial vm_area_struct contents

```
struct mm_struct* vm_mm;          /* owning memory map */
unsigned long vm_start, vm_end;   /* virt. addr. range */
struct vm_area_struct* vm_next;   /* linked list       */
rb_node vm_rb;                    /* red-black linkage */
```

Memory Map Organization

```
/* for mmaps */
struct file* vm_file;    /* file pointer (or NULL)  */
unsigned long vm_pgoff; /* offset of first page
                           in file's address space */
pgprot_t vm_page_prot;  /* access perms in PTE form
                          (for creating PTEs/PDEs)   */
unsigned long vm_flags; /* access perms + others
                          (checked in all other code) */

/* operations jump table */
struct vm_operations_struct* vm_ops;
void* vm_private_data;          /* up to you... */
```

Example operations in ***vm_ops***

// called when the memory region is added

```
void (*open) (struct vm_area_struct* area);
```

// called once per process using the region

```
void (*close) (struct vm_area_struct* area);
```

*//*called when page not mapped is accessed

```
void struct page* (*nopage)
     (struct vm_area_struct* area,
      unsigned long address, int* type);
```

Used for on demand paging - pages loaded only lazily into the physical memory (i.e., when needed)

Flags Meaning

VM_READ, VM_WRITE, VM_EXEC, VM_SHARED
mapped as readable, writable, executable, and shared by >1 program

VM_MAYREAD	VM_READ can be set
VM_MAYWRITE	VM_WRITE can be set
VM_MAYEXEC	VM_EXEC can be set
VM_MAYSHARE	VM_SHARED can be set
VM_LOCKED	locked in physical memory
VM_IO	I/O memory
VM_GROWSUP	can expand towards higher addresses (heap)
VM_GROWSDOWN	can expand towards lower addresses (stack)

struct vm_area_struct

- Represents how a region of virtual memory is mapped
- Members include:
  - vm_start, vm_end - limits of VMA in virtual address space
  - vm_page_prot - permissions (p = private, s = shared)
  - vm_pgoff - of memory area in the file (if any) mapped

struct vm_area_struct

- vm_file - the struct file (if any) mapped
- provides (indirect) access to:
  - major, minor - device of the file
  - inode - inode of the file
  - image - name of the file
- vm_flags - describe the area, e.g.,
  - VM_IO - memory-mapped I/O region will not be included in core dump
  - VM_RESERVED - cannot be swapped
- vm_ops - dispatching vector of functions/methods on this object
- vm_private_data - may be used by the driver

struct vm_area_struct

- void *open (struct vm_area_struct *area);
  - allows initialization, adjusting reference counts, etc.;
  - invoked only for additional references, after mmap(), like fork()
- void *close (struct vm_area_struct *area);
  - allows cleanup when area is destroyed;
  - each process opens and closes exactly once
- int fault (struct vm_area_struct *vma, struct vm_fault *vmf);
  - general page fault handler;

Example of Memory Maps

```
1. 08048000-0804c000 r-x 4   2296660     /bin/cat
2. 0804c000-0804d000 rw- 1   2296660     /bin/cat
3. 09d82000-09da3000 rw- 33  0 [heap]
4. b7de0000-b7f18000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f18000-b7f19000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26  13238871    /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2   13238871    /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21  0 [stack]
```

```
1. 08048000-0804c000 r-x 4   121274994   /usr/bin/tac
2. 0804c000-0804d000 rw- 1   121274994   /usr/bin/tac
3. 08642000-08663000 rw- 33  0 [heap]
4. b7e56000-b7f8e000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26  13238871    /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2   13238871    /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21  0 [stack]
```

memory maps for two programs: cat and tac

Example of Memory Maps

```
1. 08048000-0804c000
2. 0804c000-0804d000 rw- 1   2296660     /bin/cat
3. 09d82000-09da3000 rw- 33  0 [heap]
4. b7de0000-b7f18000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f18000-b7f19000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26  13238871    /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2   13238871    /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21  0 [stack]
```

Mapped region starting and ending virtual address

```
1. 08048000-0804c000 r-x 4   121274994   /usr/bin/tac
2. 0804c000-0804d000 rw- 1   121274994   /usr/bin/tac
3. 08642000-08663000 rw- 33  0 [heap]
4. b7e56000-b7f8e000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26  13238871    /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2   13238871    /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21  0 [stack]
```

memory maps for two programs: cat and tac

Example of Memory Maps

```
1. 08048000-0804c000 r-x
2. 0804c000-0804d000 rw- 1   2296660     /bin/cat
3. 09d82000-09da3000 rw- 33  0 [heap]
4. b7de0000-b7f18000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f18000-b7f19000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26  13238871    /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2   13238871    /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21  0 [stack]
```

Permissions: read, write execute

```
1. 08048000-0804c000 r-x 4   121274994   /usr/bin/tac
2. 0804c000-0804d000 rw- 1   121274994   /usr/bin/tac
3. 08642000-08663000 rw- 33  0 [heap]
4. b7e56000-b7f8e000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26  13238871    /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2   13238871    /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21  0 [stack]
```

memory maps for two programs: cat and tac

Example of Memory Maps

```
1. 08048000-0804c000 r-x 4   Number of pages in the map (decimal)
2. 0804c000-0804d000 rw- 1    2296660     /bin/cat
3. 09d82000-09da3000 rw- 33   0 [heap]
4. b7de0000-b7f18000 r-x 312  13238948    /lib/libc-2.7.so
5. b7f18000-b7f19000 r-- 1    13238948    /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2    13238948    /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26   13238871    /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2    13238871    /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21   0 [stack]
```

```
1. 08048000-0804c000 r-x 4    121274994   /usr/bin/tac
2. 0804c000-0804d000 rw- 1    121274994   /usr/bin/tac
3. 08642000-08663000 rw- 33   0 [heap]
4. b7e56000-b7f8e000 r-x 312  13238948    /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1    13238948    /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2    13238948    /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26   13238871    /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2    13238871    /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21   0 [stack]
```

memory maps for two programs: cat and tac

Example of Memory Maps

```
1. 08048000-0804c000 r-x 4   2296660
2. 0804c000-0804d000 rw- 1   2296660
3. 09d82000-09da3000 rw- 33  0 [heap]
4. b7de0000-b7f18000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f18000-b7f19000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26  13238871    /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2   13238871    /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21  0 [stack]
```

i-node of the file used to provide the data (if any)

```
1. 08048000-0804c000 r-x 4   121274994   /usr/bin/tac
2. 0804c000-0804d000 rw- 1   121274994   /usr/bin/tac
3. 08642000-08663000 rw- 33  0 [heap]
4. b7e56000-b7f8e000 r-x 312 13238948    /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1   13238948    /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2   13238948    /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26  13238871    /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2   13238871    /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21  0 [stack]
```

memory maps for two programs: cat and tac

Example of Memory Maps

```
1. 08048000-0804c000 r-x 4   2296660   /bin/cat
2. 0804c000-0804d000 rw- 1   2296660   /bin/cat
3. 09d82000-09da3000 rw- 33  0 [heap]
4. b7de0000-b7f18000 r-x 312 13238948  /lib/libc
5. b7f18000-b7f19000 r-- 1   13238948  /lib/libc-2.7.so
6. b7f19000-b7f1b000 rw- 2   13238948  /lib/libc-2.7.so
7. b7f26000-b7f40000 r-x 26  13238871  /lib/ld-2.7.so
8. b7f40000-b7f42000 rw- 2   13238871  /lib/ld-2.7.so
9. bf8e9000-bf8fe000 rw- 21  0 [stack]
```

the file used to provide the data (if any)

```
1. 08048000-0804c000 r-x 4   121274994  /usr/bin/tac
2. 0804c000-0804d000 rw- 1   121274994  /usr/bin/tac
3. 08642000-08663000 rw- 33  0 [heap]
4. b7e56000-b7f8e000 r-x 312 13238948   /lib/libc-2.7.so
5. b7f8e000-b7f8f000 r-- 1   13238948   /lib/libc-2.7.so
6. b7f8f000-b7f91000 rw- 2   13238948   /lib/libc-2.7.so
7. b7f9c000-b7fb6000 r-x 26  13238871   /lib/ld-2.7.so
8. b7fb6000-b7fb8000 rw- 2   13238871   /lib/ld-2.7.so
9. bf99f000-bf9b4000 rw- 21  0 [stack]
```

memory maps for two programs: cat and tac

Example: How much memory would be used by the ***cat*** *and* ***tac*** *processes together?*

- Assume that all pages are memory resident (not swapped out to disk) and do not include memory taken up by page tables or kernel data structures.
- Cat process uses
  - 402 pages in memory
- Tac process uses
  - 402 pages in memory
- Dynamic libraries
  - 343 pages in memory
    - Solution: **2 x 402 –343 = 461**

Use of Memory Mapping By Device Drivers

- A device driver is likely to use memory mapping for two main purposes:
  1. To provide user-level access to device memory and/or control registers
     - For example, so an Xserver process can access the graphics controller directly
  2. To share access between user and device/kernel I/O buffers, to avoid copying between DMA/kernel buffers and userspace

The mmap() Interfaces

- User-level API function:
  - void *mmap (caddr_t start, size_t len, int prot, int flags, int fd, off_t offset);
- Driver-level file operation:
  - int (*mmap) (struct file *filp, struct vm_area_struct *vma);