进程虚地址空间管理和延迟的内存分配技术

关键数据结构 1
page结构和mem_map数组

- 每个主存页框有一个Page结构
  登记这个主存页框的使用情况
- mem_map是Page结构数组
  系统用主存页框号，查数组得Page结构

Type	Name	Description
unsigned long	flags	Array of flags (see Table 8-2). Also encodes the zone number to which the page frame belongs.
atomic_t	_count	Page frame's reference counter.
atomic_t	_mapcount	Number of Page Table entries that refer to the page frame (-1 if none).
unsigned long	private	Available to the kernel component that is using the page (for instance, it is a buffer head pointer in case of buffer page; see "Block Buffers and Buffer Heads" in Chapter 15). If the page is free, this

.....

内存区域 memory region

- 一个内存区域就是一个逻辑段。在进程的虚地址空间中，每个逻辑段占连续空间，尺寸是整数个逻辑页面。所有逻辑页面的访问权限相同，系统处理这些逻辑页面的方法也相同。

- 内存区域的属性　　有文件的段。　　缺页时，会读文件填充主存页框
  - 代码段　　只读，可执行，有文件
    缺页时，系统会读一个磁盘文件填充分配给代码段的主存页框
  - 数据段　　可读，可写，不可执行，有文件
    第一次缺页时，系统会读一个磁盘文件填充分配给数据段的主存页框
  - mmap的段　　可读，可写，不可执行，有文件
    第一次缺页时，系统会读一个磁盘文件填充分配给代码段的主存页框。mmap的段关闭时，主存页框中的内容刷回磁盘

虚空间的起始地址　　虚空间的结束地址　　文件　　文件中的偏移量

内存区域 memory region

- 内存区域的属性 续　　匿名段，没有文件。
  缺页时，用0填充物理页框
  - BSS段
  - heap
  - 共享内存的段
  - Stack　　VM_GROWSDOWN

虚空间的起始地址　虚空间的结束地址　文件——文件中的偏移量

关键数据结构 2

vm_area_struct 和 内存区域

Type	Field	Description
struct mm_struct *	vm_mm	Pointer to the memory descriptor that owns the region.
unsigned long	vm_start	First linear address inside the region.
unsigned long	vm_end	First linear address after the region.
struct vm_area_struct *	vm_next	Next region in the process list.
pgprot_t	vm_page_prot	Access permissions for the page frames of the region.
unsigned long	vm_flags	Flags of the region.
struct rb_node	vm_rb	Data for the red-black tree (see later in this chapter).
union	shared	Links to the data structures used for reverse mapping (see the section "Reverse Mapping for Mapped Pages" in Chapter 17).

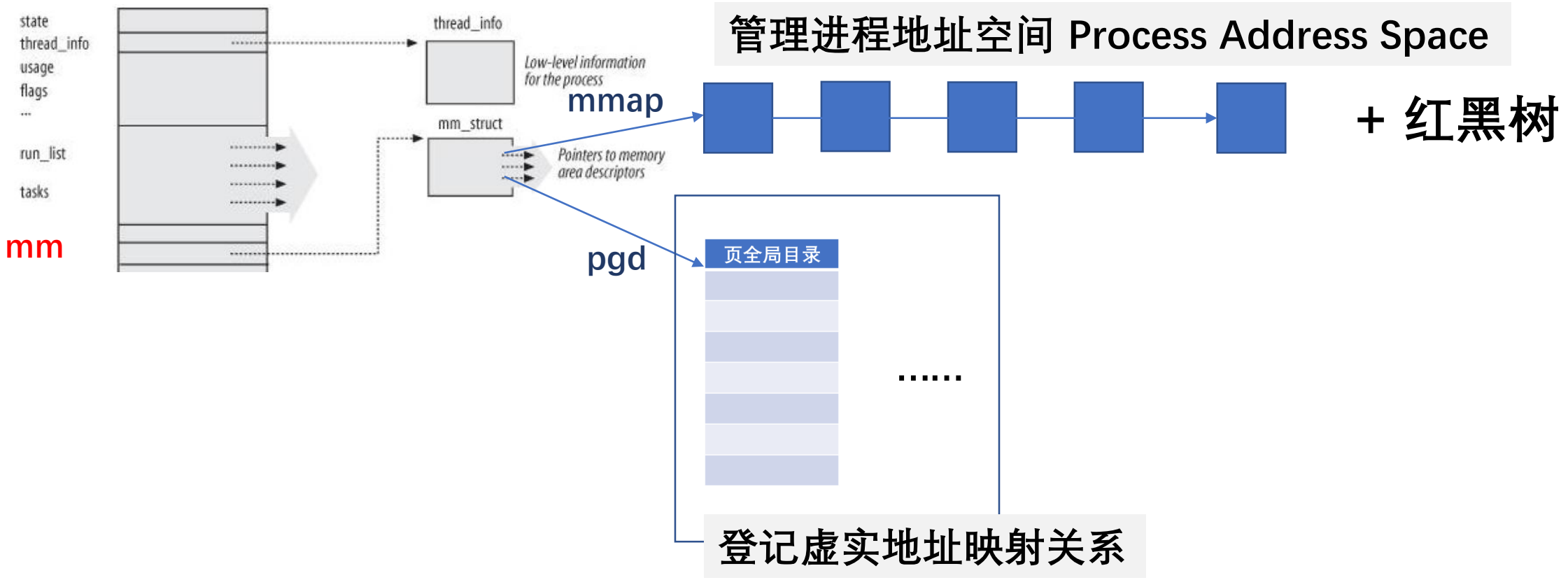
关键数据结构 2
vm_area_struct 和 内存区域

struct list_head	anon_vma_node	Pointers for the list of anonymous memory regions (see the section "Reverse Mapping for Anonymous Pages" in Chapter 17).
struct anon_vma *	anon_vma	Pointer to the anon_vma data structure (see the section "Reverse Mapping for Anonymous Pages" in Chapter 17).
struct vm_operations_struct*	vm_ops	Pointer to the methods of the memory region.
unsigned long	vm_pgoff	Offset in mapped file (see Chapter 16). For anonymous pages, it is either zero or equal to vm_start/PAGE_SIZE (see Chapter 17).
struct file *	vm_file	Pointer to the file object of the mapped file, if any.
void *	vm_private_data	Pointer to private data of the memory region.

关键数据结构 2
vm_area_struct中的vm_ops

Method	Description
open	Invoked when the memory region is added to the set of regions owned by a process.
close	Invoked when the memory region is removed from the set of regions owned by a process.
nopage	Invoked by the Page Fault exception handler when a process tries to access a page not present in RAM whose linear address belongs to the memory region (see the later section "Page Fault Exception Handler").
populate	Invoked to set the page table entries corresponding to the linear addresses of the memory region (prefaulting). Mainly used for non-linear file memory mappings.

关键数据结构3　内存描述符mm_struct中的mmap和vm_area_struct 链表



16.2.1. Memory Mapping Data Structures

A memory mapping is represented by a combination of the following data structures :

- The inode object associated with the mapped file
- The `address_space` object of the mapped file
- A file object for each different mapping performed on the file by different processes
- A `vm_area_struct` descriptor for each different mapping on the file
- A page descriptor for each page frame assigned to a memory region that maps the file

Figure 16-2 illustrates how the data structures are linked. On the left side of the image we show the inode, which identifies the file. The `i_mapping` field of each inode object points to the `address_space` object of the file. In turn, the `page_tree` field of each `address_space` object points to the radix tree of pages belonging to the address space (see the section "The Radix Tree" in Chapter 15), while the `i_mmap` field points to a second tree called the radix priority search tree (PST) of memory regions belonging to the address space. The main use of PST is for performing "reverse mapping," that is, for identifying quickly all processes that share a given page. We'll cover in detail PSTs in the next chapter, because they are used for page frame reclaiming. The link between file objects relative to the same file and the inode is established by means of the `f_mapping` field.

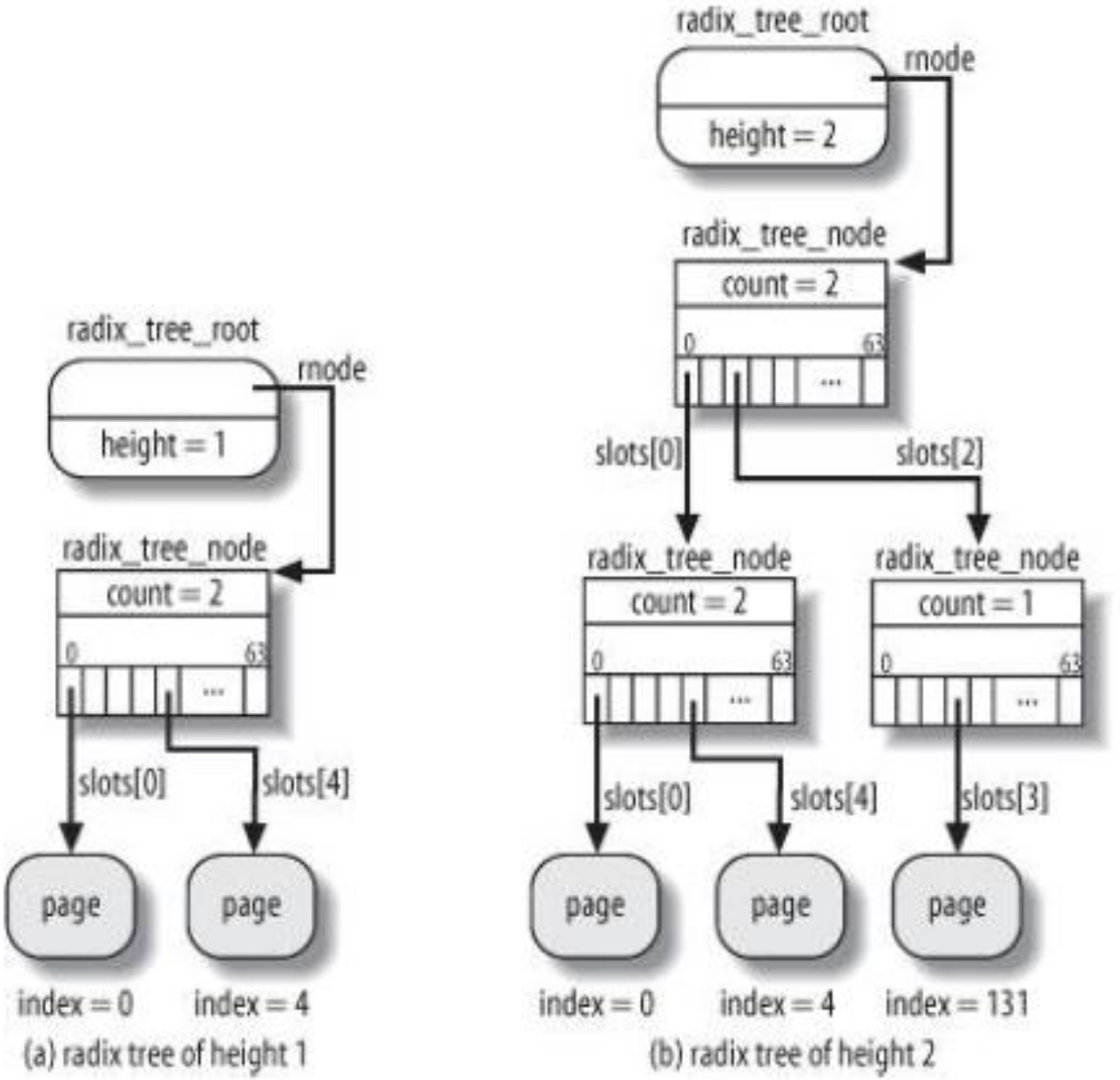
Each memory region descriptor has a `vm_file` field that links it to the file object of the mapped file (if that field is null, the memory region is not used in a memory mapping). The position of the first mapped location is stored into the `vm_pgoff` field of the memory region descriptor; it represents the file offset as a number of page-size units. The length of the mapped file portion is simply the length of the memory region, which can be computed from the `vm_start` and `vm_end` fields.

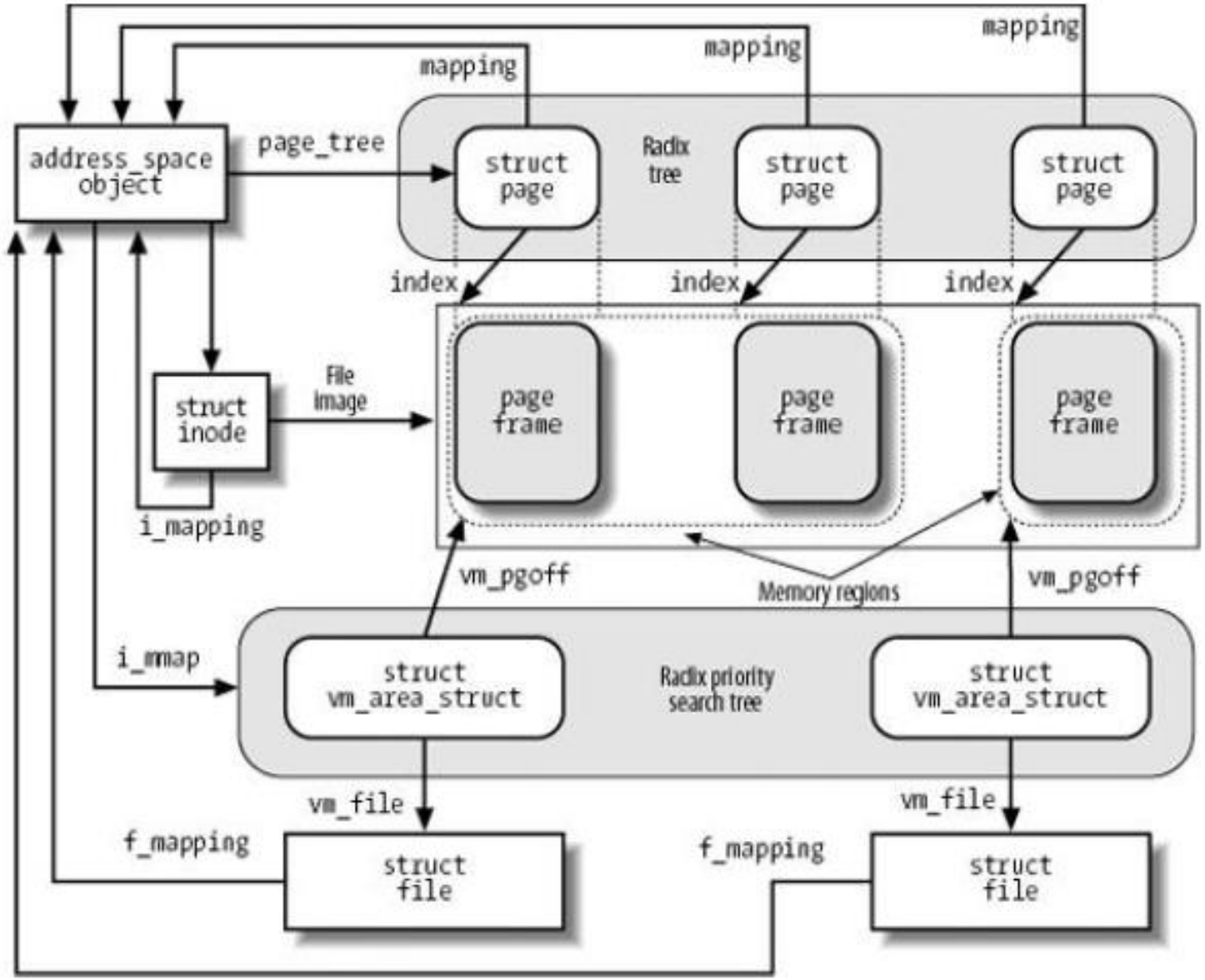
address对象

Type	Field	Description
struct inode *	host	Pointer to the inode hosting this object, if any
struct radix_tree_root	page_tree	Root of radix tree identifying the owner's pages
spinlock_t	tree_lock	Spin lock protecting the radix tree

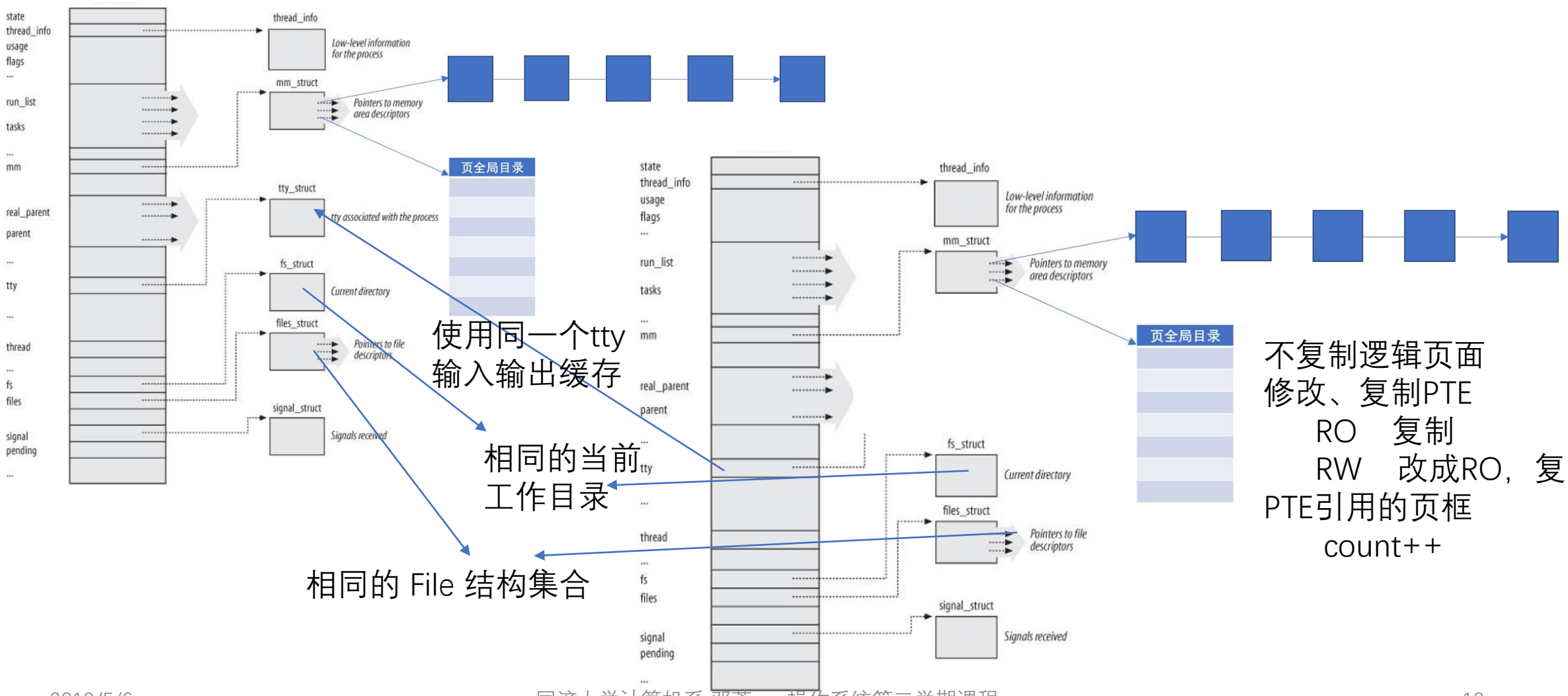
15.1.2 radix tree

+ inode中的i_addr数组

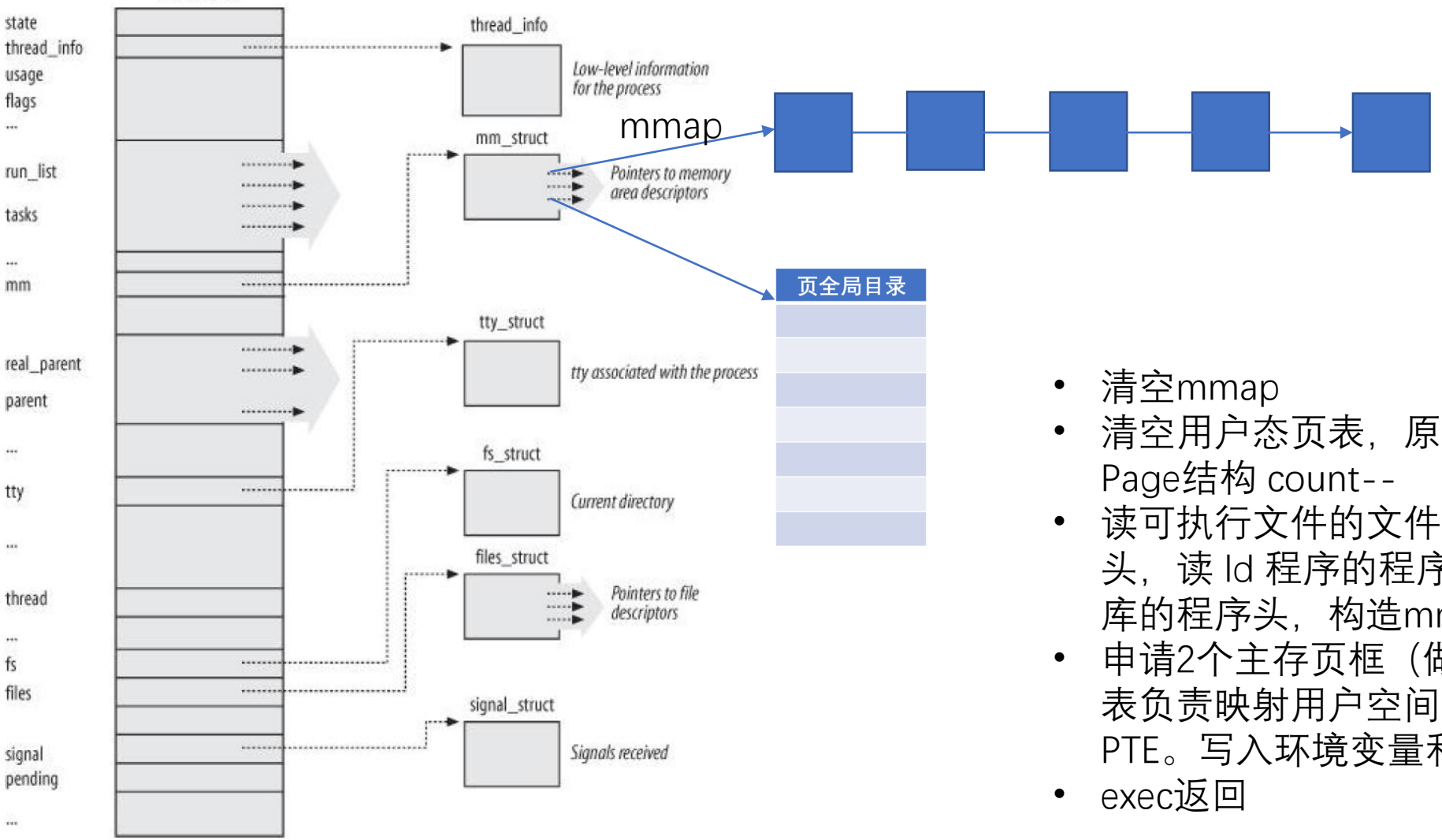



mapping
mapping
mapping
address_space object
page_tree
struct page
Radix tree
struct page
struct page
index
index
index
struct inode
File image
page frame
page frame
page frame
i_mapping
vm_pgoff
Memory regions
vm_pgoff
i_mmap
struct vm_area_struct
Radix priority search tree
struct vm_area_struct
vm_file
vm_file
f_mapping
struct file
f_mapping
struct file

进程创建 fork


state
thread_info
usage
flags
...
run_list
tasks
...
mm
real_parent
parent
...
tty
...
thread
...
fs
files
signal
pending
...
thread_info
Low-level information for the process
mm_struct
Pointers to memory area descriptors
页全局目录
tty_struct
tty associated with the process
fs_struct
Current directory
files_struct
Pointers to file descriptors
signal_struct
Signals received
使用同一个tty
输入输出缓存
相同的当前
工作目录
相同的 File 结构集合
不复制逻辑页面
修改、复制PTE
RO 复制
RW 改成RO，复制
PTE引用的页框
count++

exec



- 清空mmap
- 清空用户态页表，原来页表映射的所有Page结构 count--
- 读可执行文件的文件头，读标准C库的程序头，读 ld 程序的程序头，读所有动态链接库的程序头，构造mmap链表
- 申请2个主存页框（做初始用户栈），写页表负责映射用户空间最后2个逻辑页面的PTE。写入环境变量和命令行参数
- exec返回