MMAP and Page Fault Handling

SWE3015

Sung-hun Kim

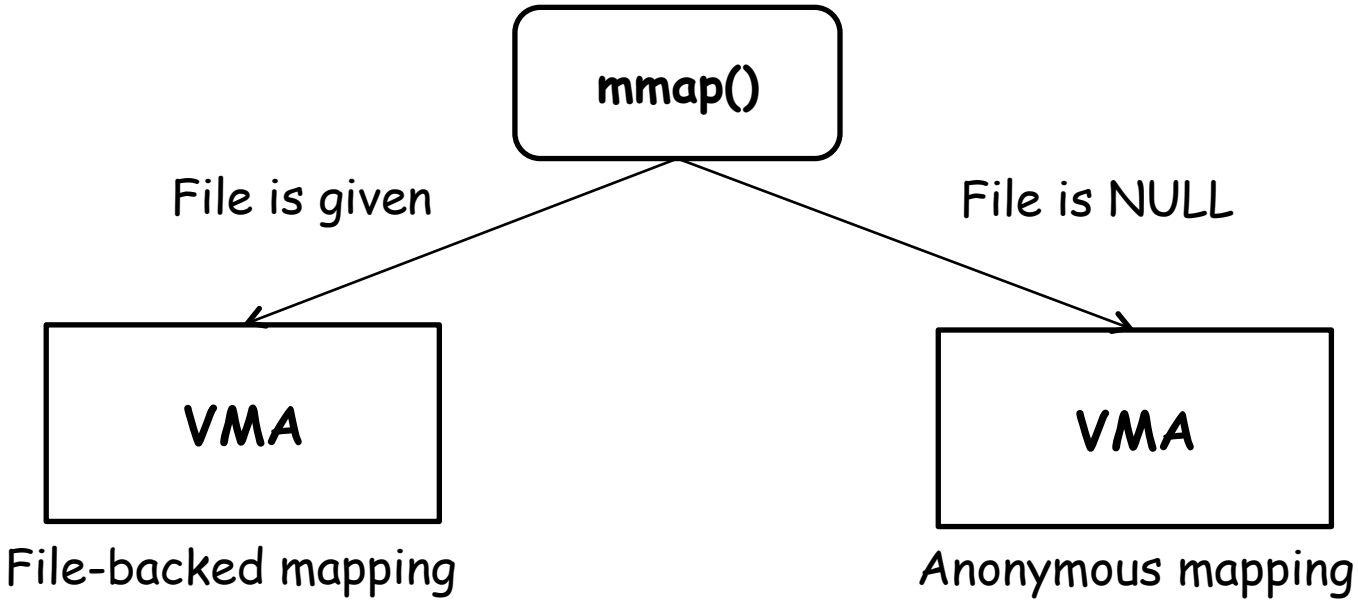
SWE3015: Operating System Project

MMAP

MMAP

- POSIX compliant function
  - Create a new mapping in the virtual address space of calling process

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```



MMAP Calling Path

mmap_pgoff()

vm_mmap_pgoff()

down_write(&mm->mmap_sem);
do_mmap_pgoff()
up_write(&mm->mmap_sem);

mmap_region()

MMAP_REGION(1/2)

- Main function for mmap call
  - Check *mmap* is calling for old mapping

```
vma = vma_merge(args);
if (vma)
    goto out;
```

  - Make a new mapping (vma)

```
vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
```

```
vma->vm_mm = mm;
vma->vm_start = addr;
vma->vm_end = addr + len;
vma->vm_flags = vm_flags;
vma->vm_page_prot = vm_get_page_prot(vm_flags);
vma->vm_pgoff = pgoff;
INIT_LIST_HEAD(&vma->anon_vma_chain);
```

MMAP_REGION(2/2)

- Main function for mmap call (continued)
  - Call file's mmap function

```
if (file) {
    vma->vm_file = get_file(file);
    error = file->f_op->mmap(file, vma);
    //...
}
```

- You should implement mmap of file_operations
  - And configure for newly allocated vma

```
static const struct file_operations fake_fops = {
    .owner  = THIS_MODULE,
    .mmap   = mmap_fake,
};
```

SWE3015: Operating System Project

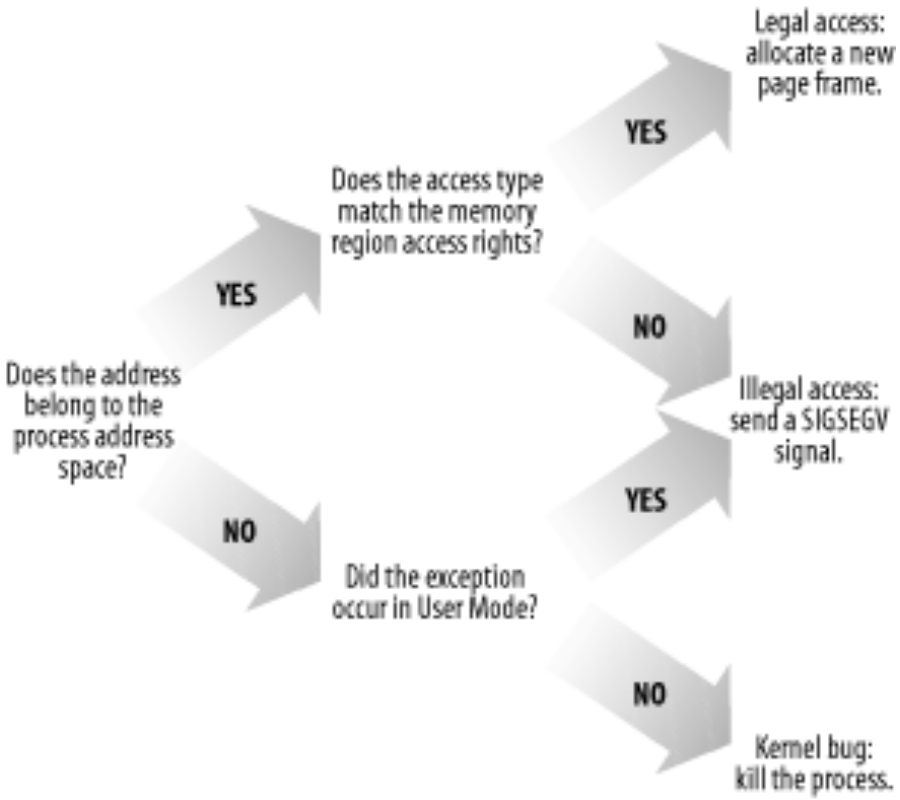
PAGE FAULT HANDLING

Page Fault

- A trap to the software raised by the hardware
  - when a program accesses a page that is mapped in the virtual address space,
  - but not loaded in physical memory
- Typical reasons are…
  - Demand paging
  - Copy on Write
  - User software bug (SIGSEGV)
  - Or even kernel bug (kernel “Oops”)

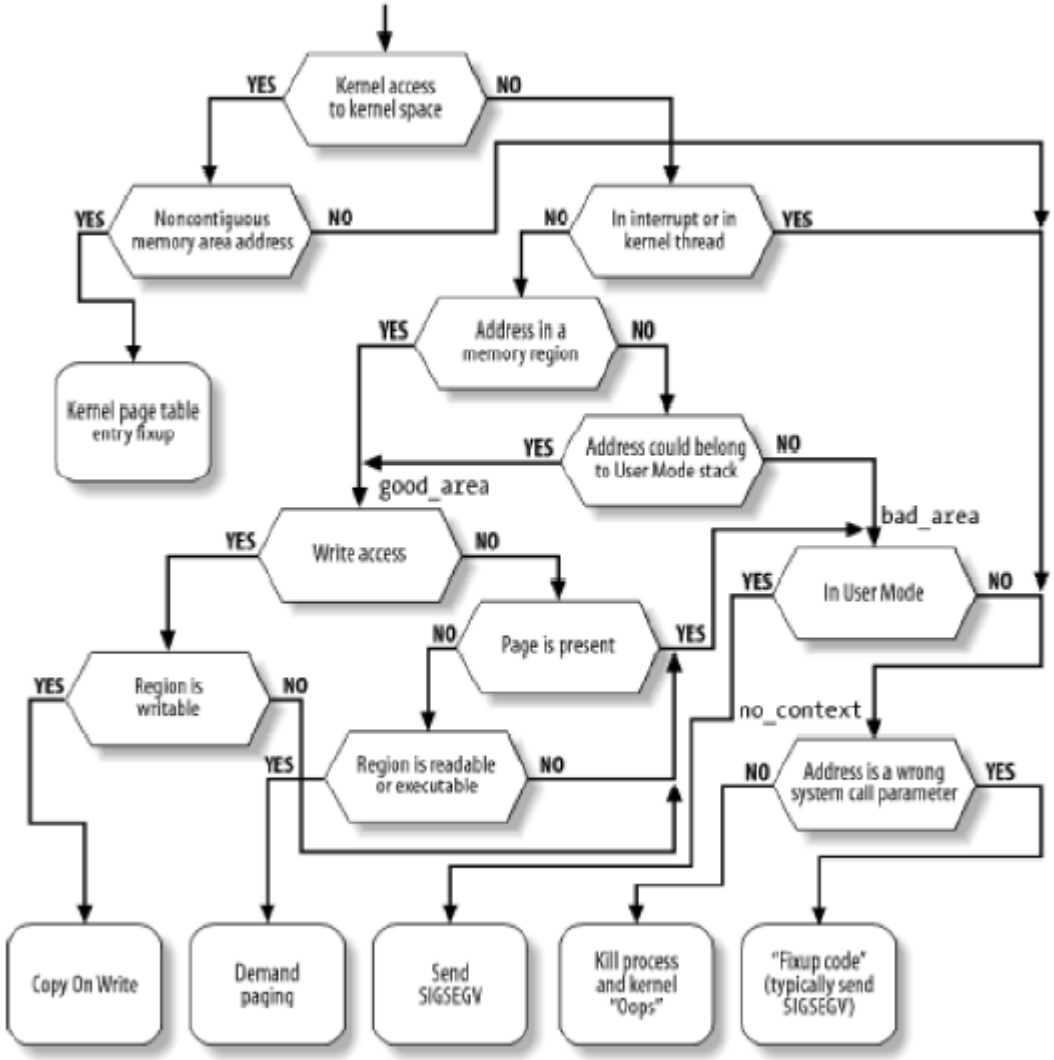
Page Fault Handling

- <arch/x86/mm/fault.c> do_page_fault()



Page Fault Handling

- <arch/x86/mm/fault.c> do_page_fault()



Page Fault Handler Calling Path

- arch/x86/mm/fault.c

do_page_fault()

__do_page_fault()

find_vma()
handle_mm_fault()

Page Fault Handler Calling Path

- mm/memory.c

handle_mm_fault()

__handle_mm_fault()

pte_offset_map(), handle_pte_fault()

do_linear_fault()

do_[read | cow | linear]_fault()

__do_fault()

vma->vm_ops->fault()

__handle_mm_fault

- Find pte
  - pgd_offset(), pud_alloc(), pmd_alloc()
  - pte_offset_map() -> find location of pte in pmd
- Handle fault
  - handle_pte_fault(mm, vma, address, pte, pmd, flags)

do_[read|cow|shared]_fault

- Call pre-define fault handler
  - __do_fault(vma, address, pgoff, flags, &fault_page);
- Make *pte* entry
  - Acquire lock for *pmd*
  - Make *pte*
    - pte = pte_offset_map_lock(mm, pmd, address, &ptl);
    - Get the location of faulted page table entry
  - Insert *pte*
    - do_set_pte(vma, address, fault_page, pte, true, false);

__do_fault

- Key data structure: vm_fault

```
struct vm_fault vmf;

vmf.virtual_address = (void __user *)(address & PAGE_MASK);
vmf.pgoff = pgoff;
vmf.flags = flags;
vmf.page = NULL;

ret = vma->vm_ops->fault(vma, &vmf);

//…

*page = vmf.page;
```

Questions