

CS 3502

Operating Systems

Project 3 Lab

Kun Suo

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

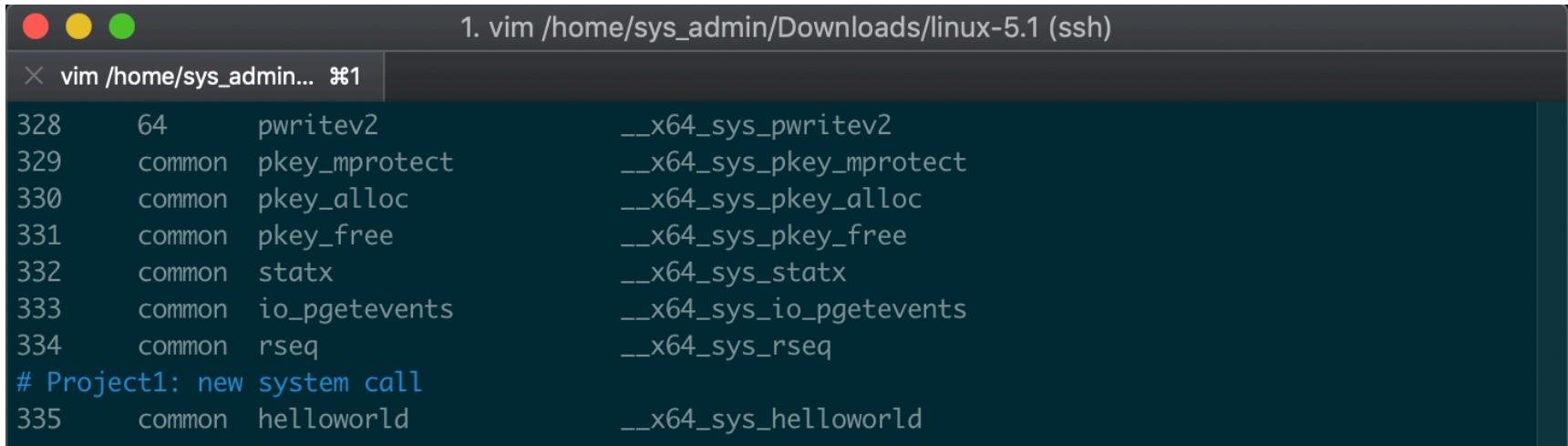
Review how to add system calls

- Step 1: register your system call
- Step 2: declare your system call in the header file
- Step 3: implement your system call
- Step 4: write user level app to call it



Step 1: register your system call

arch/x86/entry/syscalls/syscall_64.tbl



```
1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
vim /home/sys_admin... ⌘1

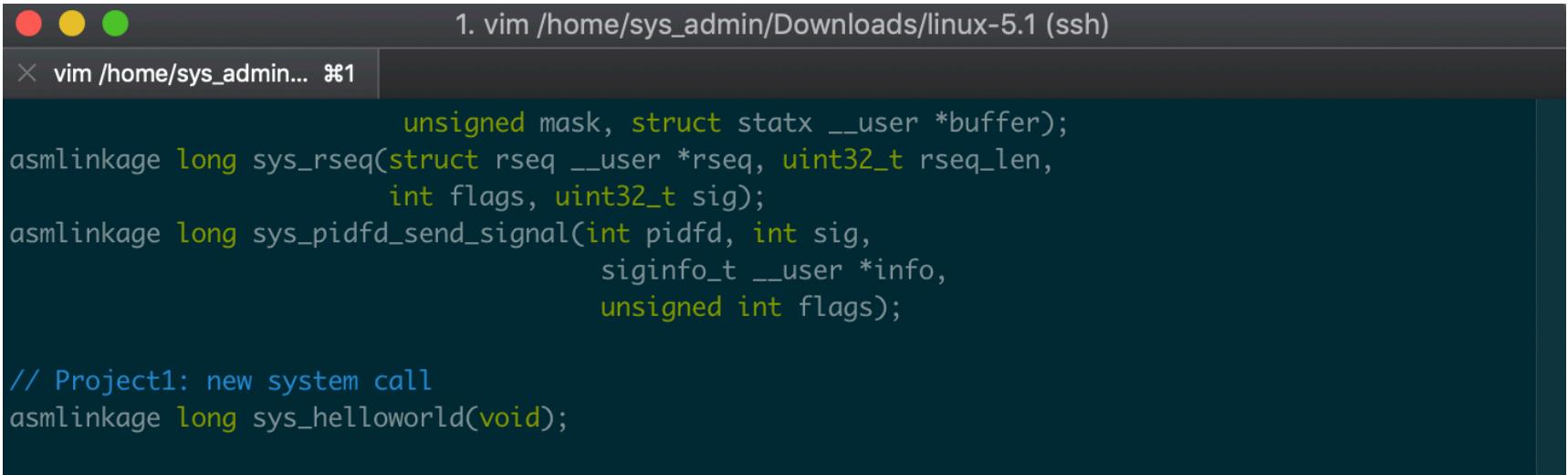
328      64      pwritev2          __x64_sys_pwritev2
329      common   pkey_mprotect    __x64_sys_pkey_mprotect
330      common   pkey_alloc       __x64_sys_pkey_alloc
331      common   pkey_free        __x64_sys_pkey_free
332      common   statx           __x64_sys_statx
333      common   io_pgetevents   __x64_sys_io_pgetevents
334      common   rseq            __x64_sys_rseq
# Project1: new system call
335      common   helloworld      __x64_sys_helloworld
```

https://elixir.bootlin.com/linux/v5.0/source/arch/x86/entry/syscalls/syscall_64.tbl#L346



Step 2: declare your system call in the header file

include/linux/syscalls.h



The screenshot shows a terminal window titled "1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)". The vim editor is displaying the contents of the syscalls.h header file. The code includes several standard Linux system calls like sys_rseq and sys_pidfd_send_signal, followed by a new user-defined system call sys_helloworld.

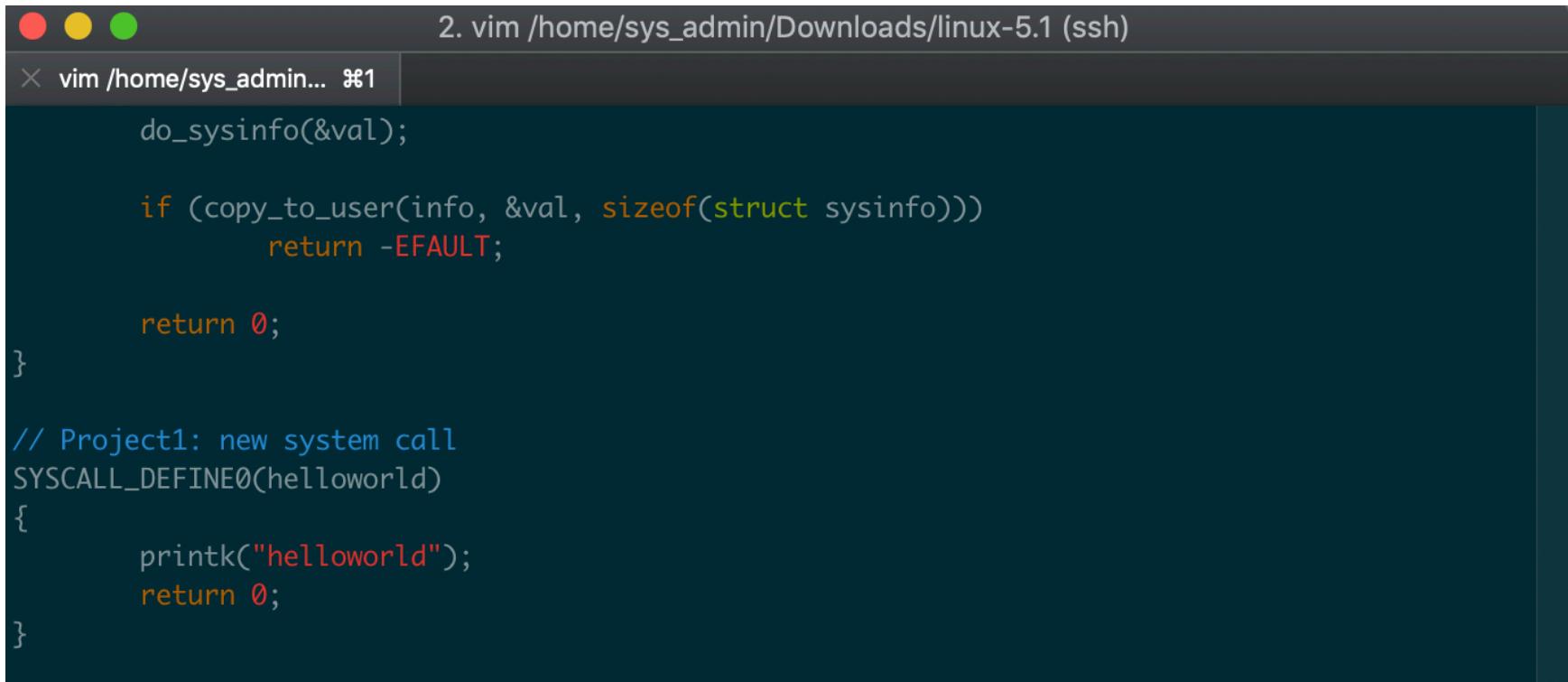
```
unsigned mask, struct statx __user *buffer);  
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,  
                         int flags, uint32_t sig);  
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,  
                                       siginfo_t __user *info,  
                                       unsigned int flags);  
  
// Project1: new system call  
asmlinkage long sys_helloworld(void);
```

<https://elixir.bootlin.com/linux/v5.0/source/include/linux/syscalls.h>



Step 3: implement your system call

kernel/sys.c



```
2. vim /home/sys_admin/Downloads/linux-5.1 (ssh)
vim /home/sys_admin... #1

do_sysinfo(&val);

if (copy_to_user(info, &val, sizeof(struct sysinfo)))
    return -EFAULT;

return 0;
}

// Project1: new system call
SYSCALL_DEFINE0(helloworld)
{
    printk("helloworld");
    return 0;
}
```

<https://elixir.bootlin.com/linux/v5.0/source/kernel/sys.c#L402>



Step 4: write user level app to call it

test_syscall.c:

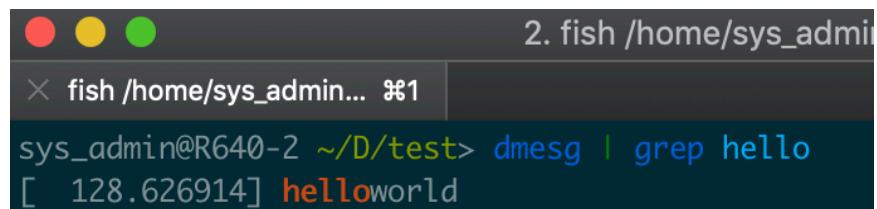
```
*****  
#include <linux/unistd .h>  
#include <sys/syscall .h>  
#include <sys/types .h>  
#include <stdio .h>  
#define __NR_helloworld 335  
  
int main(int argc, char *argv[])  
{  
    syscall (__NR_helloworld) ;  
    return 0 ;  
}  
*****
```

//If syscall needs parameter, then:
//syscall (__NR_helloworld, a, b, c) ;

Compile and execute:

```
$ gcc test_syscall.c -o test_syscall  
$ ./test_syscall
```

The test program will call the new system call and output a helloworld message at the tail of the output of dmesg (system log).



A screenshot of a terminal window titled "fish /home/sys_admin... #1". The command entered is "dmesg | grep hello". The output shows the message "[128.626914] helloworld" at the end of the log.

```
2. fish /home/sys_admin... #1  
x fish /home/sys_admin... #1  
sys_admin@R640-2 ~/D/test> dmesg | grep hello  
[ 128.626914] helloworld
```

Put it all together

User space

```
test_syscall.c:  
*****  
#include <linux/unistd.h>  
#include <sys/syscall.h>  
#include <sys/types.h>  
#include <stdio.h>  
#define __NR_helloworld 335  
  
int main(int argc, char *argv[]){  
    syscall (__NR_helloworld);  
    return 0;  
}  
*****
```

Kernel space

```
1. vim /home/sys_admin/Downloads/linux-5.1 (ss1)  
x vim /home/sys_admin... #1  
328 64 pwriterv2 __x64_sys_pwriterv2  
329 common pkey_mprotect __x64_sys_pkey_mprotect  
330 common pkey_alloc __x64_sys_pkey_alloc  
331 common pkey_free __x64_sys_pkey_free  
332 common statx __x64_sys_statx  
333 common io_pgetevents __x64_sys_io_pgetevents  
334 common rseq __x64_sys_rseq  
335 common helloworld __x64_sys_helloworld
```

```
1. vim /home/sys_admin/Downloads/linux-5.1 (ss1)  
x vim /home/sys_admin... #1  
unsigned mask, struct statx __user *buffer);  
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,  
int flags, uint32_t sig);  
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,  
siginfo_t __user *info,  
unsigned int flags);  
  
// Project1: new system call  
asmlinkage long sys_helloworld(void);
```

```
2. vim /home/sys_admin/Downloads/linux-5.1 (ss1)  
x vim /home/sys_admin... #1  
do_sysinfo(&val);  
  
if (copy_to_user(info, &val, sizeof(struct sysinfo)))  
    return -EFAULT;  
  
return 0;  
  
// Project1: new system call  
SYSCALL_DEFINE0(helloworld)  
{  
    printk("helloworld");  
    return 0;  
}
```

```
2. fish /home/sys_admin... #1  
x fish /home/sys_admin... #1  
sys_admin@R640-2 ~$D/test> dmesg | grep hello  
[ 128.626914] helloworld
```



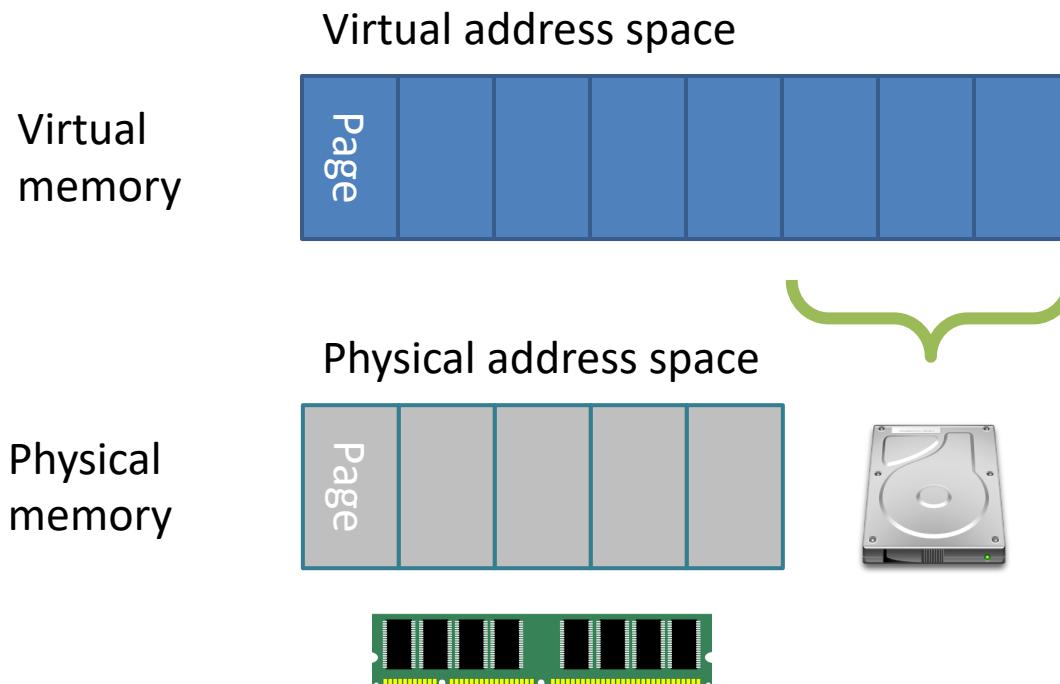
Project 3

- Given the above virtual memory areas used by a process, write a system call *sys_getPageInfo* to report the current status of specific addresses in these virtual memory areas. The system call takes the pid of a process as input and outputs the following information of start address *vm_start* in each *vm_area_struct* that the process has:
 - If the data in this address is in memory or on disk.
 - If the page which this address belongs to has been referenced or not.
 - If the page which this address belongs to is dirty or not.

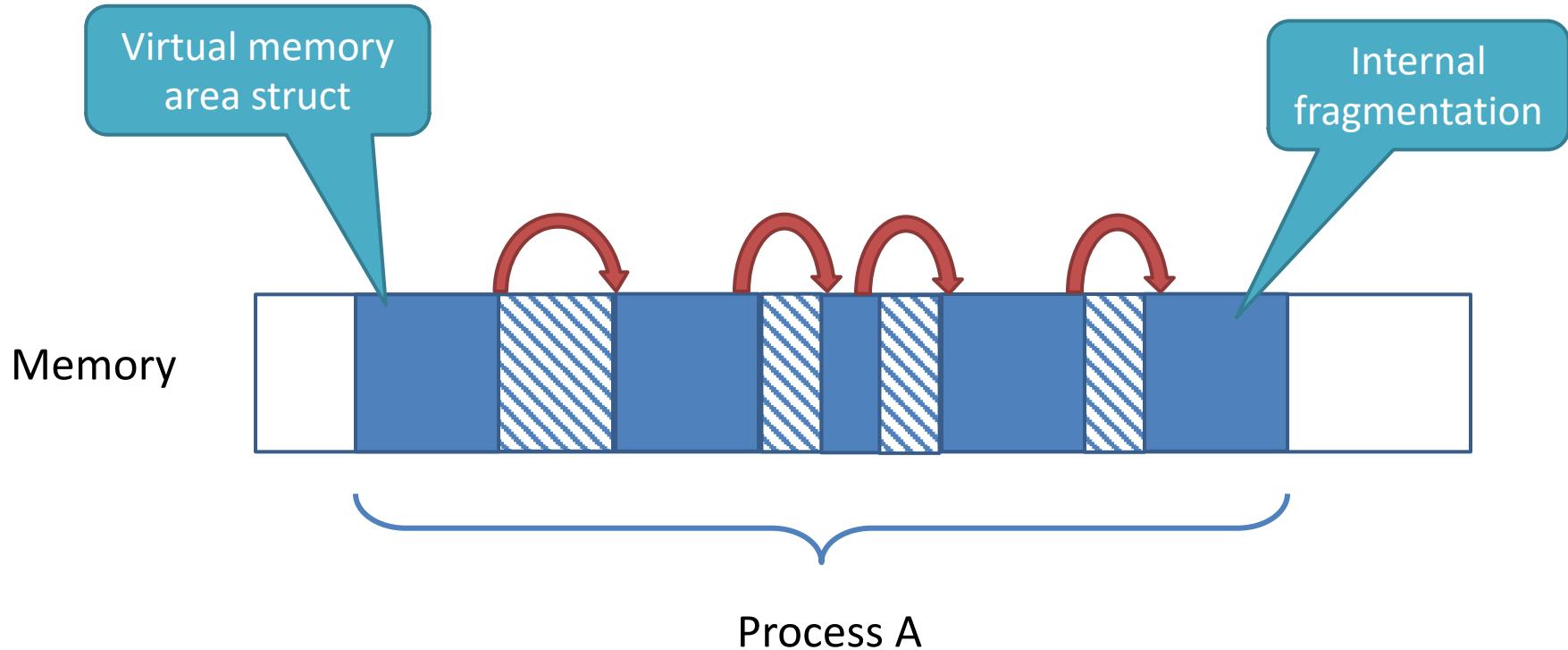


Project 3

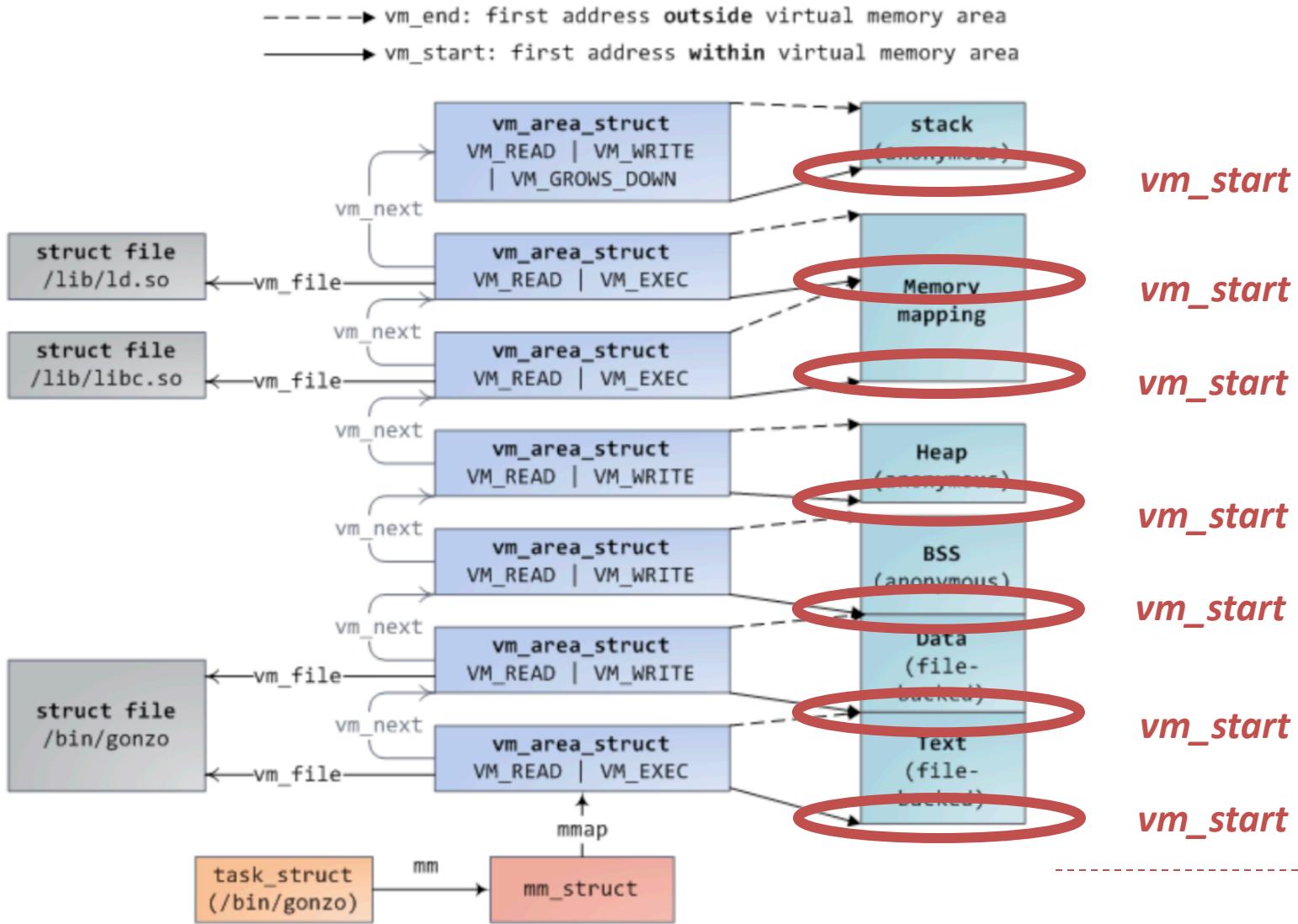
- Virtual memory address could be in memory or on disk.



Revisit Process Data Structure



Project 3



Revisit Process Data Structure

```
struct task_struct {
#define CONFIG_THREAD_INFO_IN_TASK
/*
 * For reasons of header soup (see current_thr
 * must be the first element of task_struct.
 */
struct thread_info          thread_info;
#endif
/* -1 unrunnable, 0 runnable, >0 stopped: */
volatile long                state;

struct sched_info             sched_info;
struct list_head               tasks;
#ifdef CONFIG_SMP
struct plist_node              pushable_tasks;
struct rb_node                pushable_dl_tasks;
#endif
struct mm_struct              *mm;
struct mm_struct              *active_mm;
```

```
struct mm_struct {
    struct vm_area_struct *mmap;
    struct rb_root mm_rb;
    u32 vmacache_seqnum;
#define CONFIG_MMU
    unsigned long (*get_unmapped_area) (st
                                         unsigned long
                                         unsigned long
                                         unsigned long
                                         unsigned long mmap_base;
                                         unsigned long mmap_legacy_base;
                                         unsigned long task_size;
                                         unsigned long highest_vm_end;
                                         pgd_t * pgd;
                                         atomic_t mm_users;
                                         atomic_t mm_count;
                                         atomic_long_t nr_ptes;
#endif
#ifdef CONFIG_PGTABLE_LEVELS > 2
```



Revisit Process Data Structure

```
struct mm_struct {
    struct vm_area_struct *mmap;
    struct rb_root mm_rb;
    u32 vmacache_seqnum;
#ifdef CONFIG_MMU
    unsigned long (*get_unmapped_area) (st
        unsigned long
        unsigned long
#endif
    unsigned long mmap_base;
    unsigned long mmap_legacy_base;
    unsigned long task_size;
    unsigned long highest_vm_end;
    pgd_t *pgd;
    atomic_t mm_users;
    atomic_t mm_count;
    atomic_long_t nr_ptes;
#if CONFIG_PGTABLE_LEVELS > 2
    /* The first cache line has the info for VMA tree walking. */
    unsigned long vm_start; /* Our start address within vm_mm. */
    unsigned long vm_end;   /* The first byte after our end address
                           within vm_mm. */
    /* linked list of VM areas per task, sorted by address */
    struct vm_area_struct *vm_next, *vm_prev;
    struct rb_node vm_rb;
    /*
     * Largest free memory gap in bytes to the left of this VMA.
     * Either between this VMA and vma->vm_prev, or between one of the
     * VMAs below us in the VMA rbtree and its ->vm_prev. This helps
     * get_unmapped_area find a free area of the right size.
     */
    unsigned long rb_subtree_gap;
    /* Second cache line starts here. */
    struct mm_struct *vm_mm; /* The address space we belong to. */
    pgprot_t vm_page_prot; /* Access permissions of this VMA. */
    unsigned long vm_flags; /* Flags, see mm.h. */

```

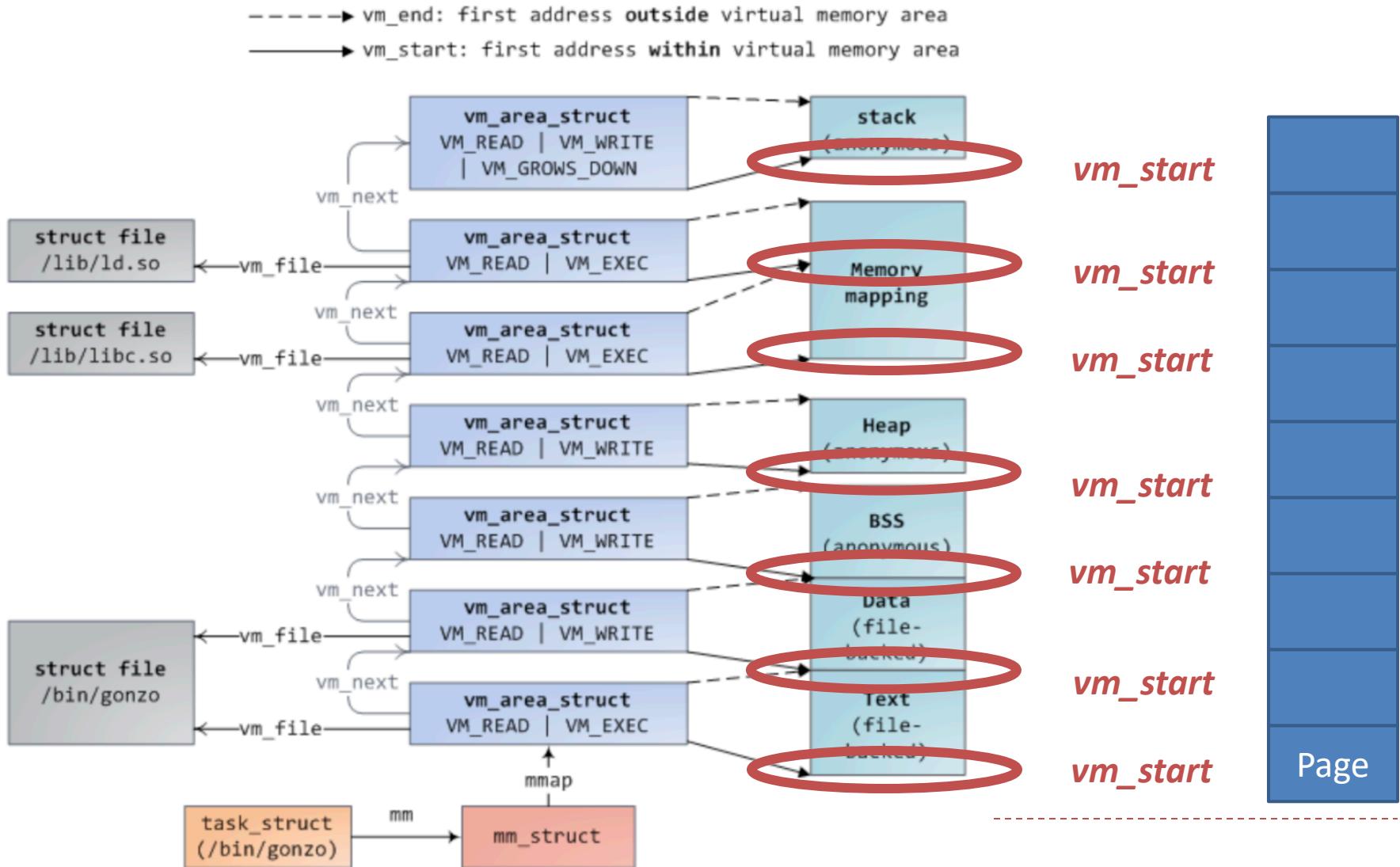


Project 3

- Given the above virtual memory areas used by a process, write a system call `sys_getPageInfo` to report the current status of specific addresses in these virtual memory areas. The system call takes the **pid of a process as input** and **outputs** the following information of **start address `vm_start`** in each **`vm_area_struct`** that the process has:
 - If the data in this address is in memory or on disk.**
 - If the page which this address belongs to has been referenced or not.**
 - If the page which this address belongs to is dirty or not.**



Project 3



Project 3

Table: Page Table Entry Status Bits

Bit	Function
<code>_PAGE_PRESENT</code>	Page is resident in memory and not swapped out.
<code>_PAGE_ACCESSED</code>	Set if the page is referenced.
<code>_PAGE_DIRTY</code>	Set if the page is written to.

```
static inline int pte_present(pte_t a)
{
    return pte_flags(a) & (_PAGE_PRESENT | _PAGE_PROTNONE);
}

static inline int pte_young(pte_t pte)
{
    return pte_flags(pte) & _PAGE_ACCESSED;
}

static inline int pte_dirty(pte_t pte)
{
    return pte_flags(pte) & _PAGE_DIRTY;
```

<https://elixir.bootlin.com/linux/latest/source/arch/x86/include/asm/pgtable.h>



Project 3

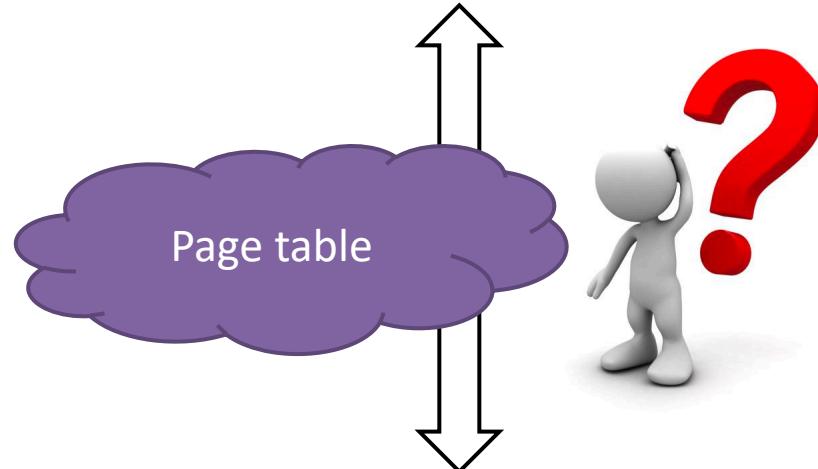
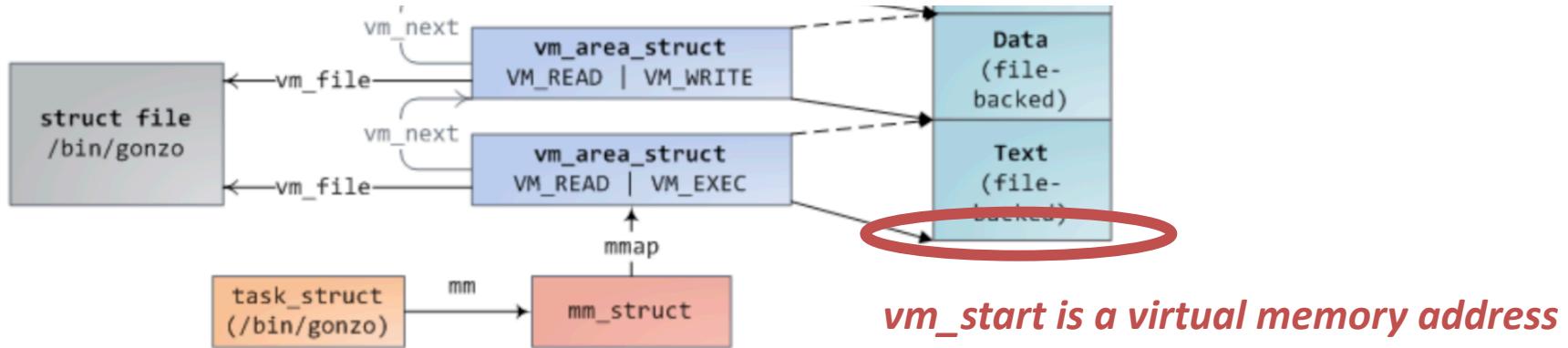
Table: Page Table Entry Status Bits

Bit	Function
<code>_PAGE_PRESENT</code>	Page is resident in memory and not swapped out.
<code>_PAGE_ACCESSED</code>	Set if the page is referenced.
<code>_PAGE_DIRTY</code>	Set if the page is written to.

- The function `pte_present` is used for judging a page is in memory or disk. If the return value is 1, the page is in the memory, otherwise it is in the disk.
- The function `pte_young` is used for judging a page is referenced or not. If the return value is 1, the page has been referenced, otherwise it has not been referenced yet.
- The function `pte_dirty` is used for judging a page is dirty or not. If the return value is 1, the page is dirty, otherwise it is not dirty.



Project 3

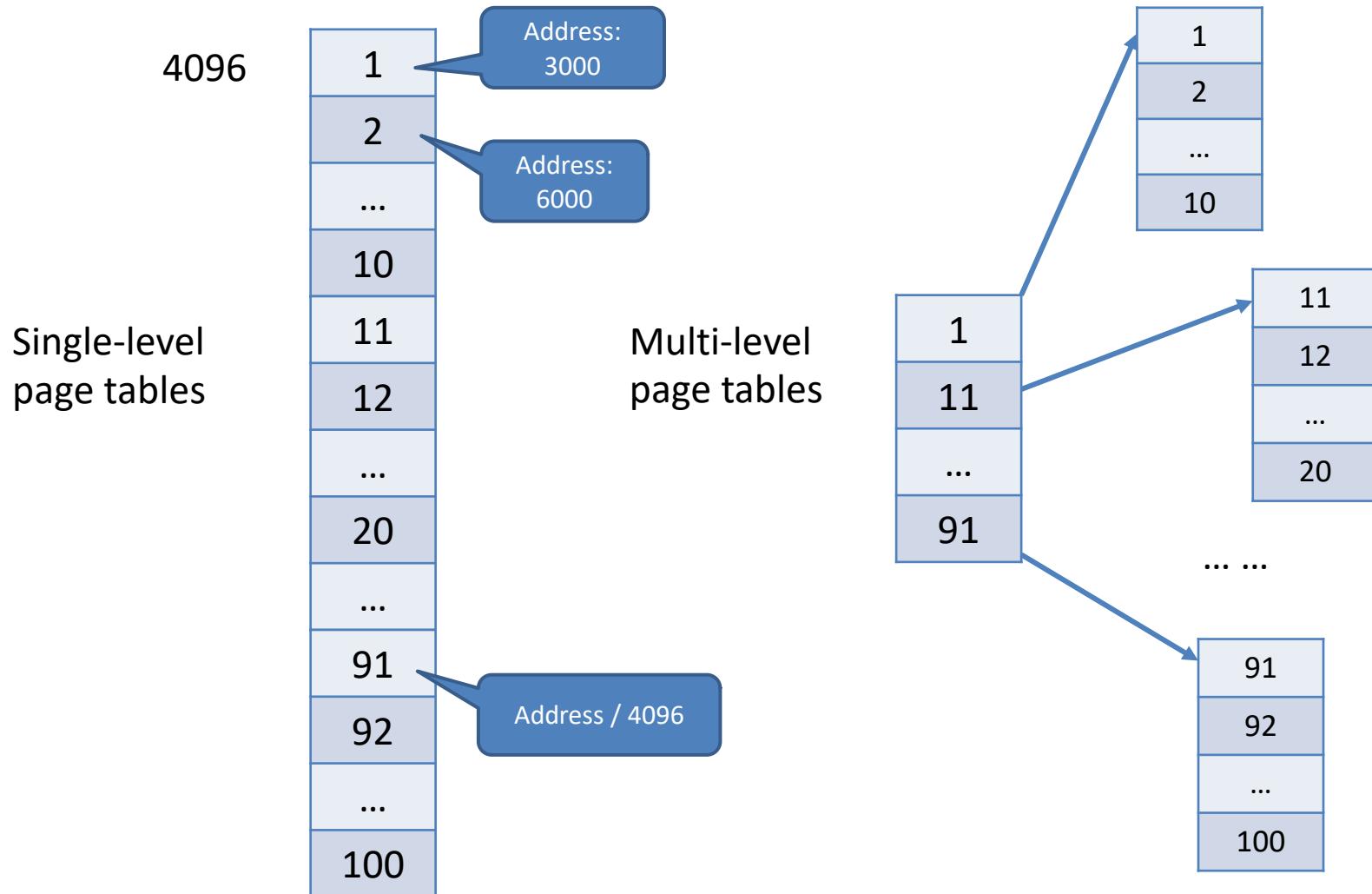


`typedef struct { pteval_t pte; } pte_t;`

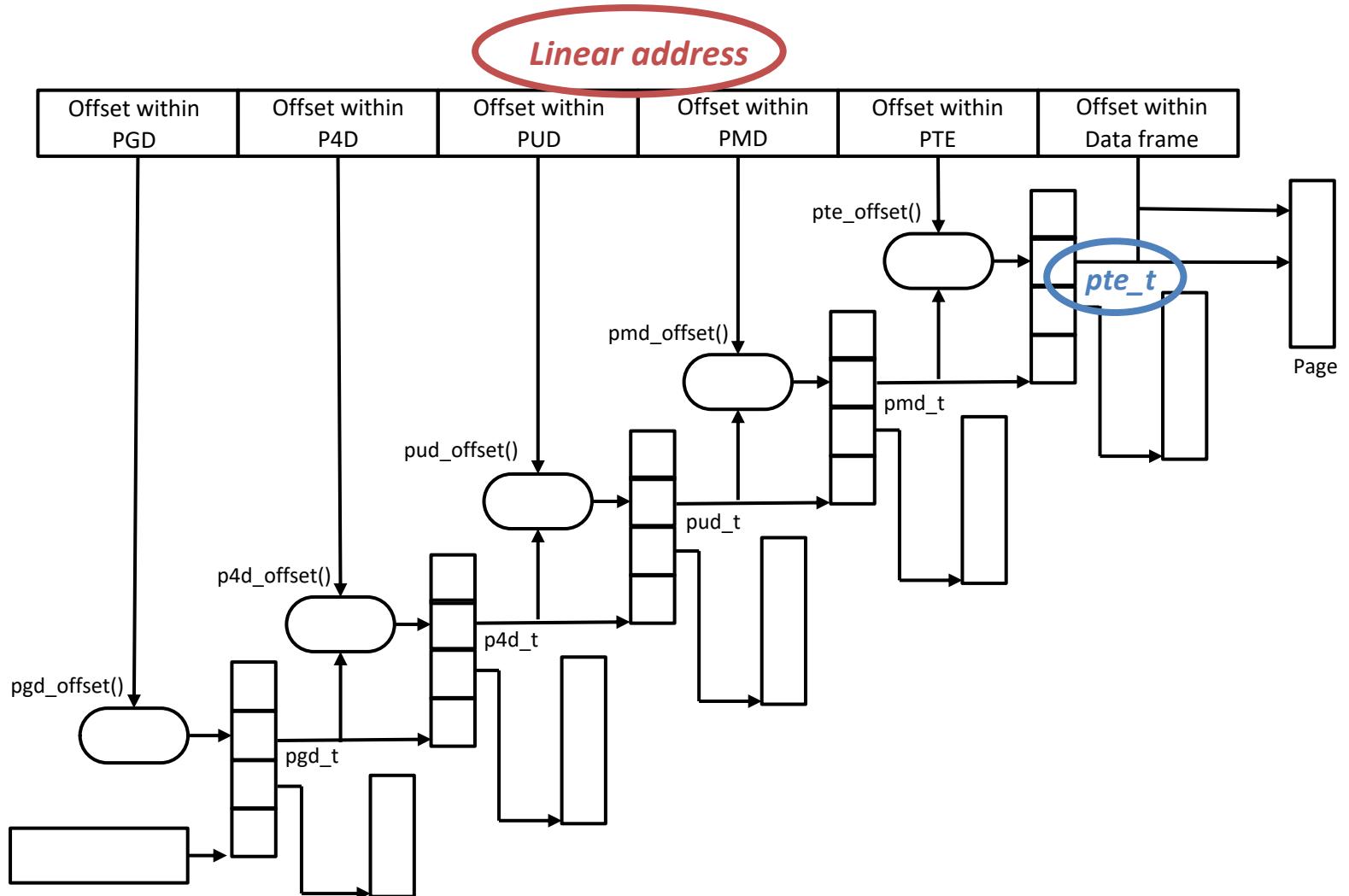
pte_t is a memory page entry



Project 3: revisit memory address



Project 3: revisit memory address



Virtual address

```
/*
 * a shortcut to get a pgd_t in a given mm
 */
#define pgd_offset(mm, address) pgd_offset_pgd((mm)->pgd, (address))

/* to find an entry in a page-table-directory. */
static inline p4d_t *p4d_offset(pgd_t *pgd, unsigned long address)
{
    if (!pgtable_l5_enabled())
        return (p4d_t *)pgd;
    return (p4d_t *)pgd_page_vaddr(*pgd) + p4d_index(address);
}

/* Find an entry in the third-level page table.. */
static inline pud_t *pud_offset(p4d_t *p4d, unsigned long address)
{
    return (pud_t *)p4d_page_vaddr(*p4d) + pud_index(address);
}

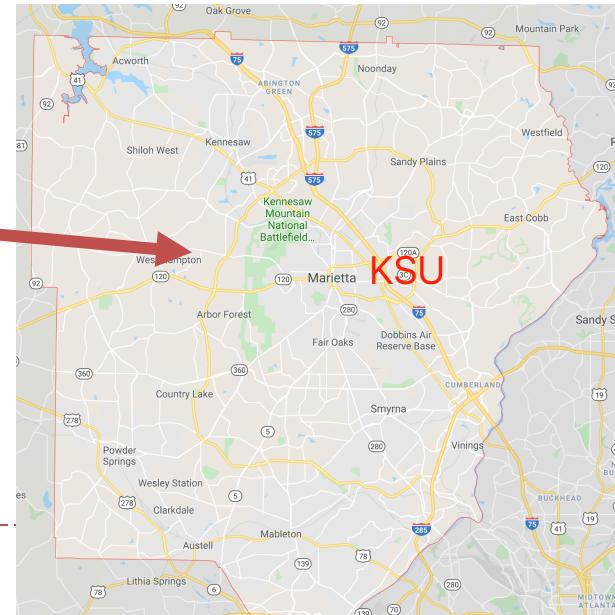
/* Find an entry in the second-level page table.. */
static inline pmd_t *pmd_offset(pud_t *pud, unsigned long address)
{
    return (pmd_t *)pud_page_vaddr(*pud) + pmd_index(address);
}

static inline pte_t *pte_offset_kernel(pmd_t *pmd, unsigned long address)
```

<https://elixir.bootlin.com/linux/latest/source/arch/x86/include/asm/pgtable.h>



Just like we search on maps



Project 3: user test program

```
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define __NR_getPageInfo 339

/*
 * Execution: test using process pid=123
 * $ ./assignment2-user1.o 123
 */

int main(int argc, char *argv[])
{
    int pid = 0;

    pid = atoi(argv[1]);    //pid the process
    syscall(__NR_getPageInfo, pid);

    return 0;
}
```

- The system call takes the pid of a process as input and outputs the following information of start address *vm_start* in each *vm_area_struct* that the process has:
 - Memory or disk?
 - Referenced or not?
 - Dirty or not?

Input is the
user process id



Project 3: expected output of user test program

```
fish /home/ksuo/linux-5.1-modified$ [ 218.906507] Find the task 1675  
[ 218.906509] This is 1 vm_area_struct  
[ 218.906509] the page is in the memory!  
[ 218.906510] the page is not referenced!  
[ 218.906510] the page is not modified!  
[ 218.906511] This is 2 vm_area_struct  
[ 218.906511] the page is in the memory!  
[ 218.906511] the page is not referenced!  
[ 218.906512] the page is not modified!  
[ 218.906512] This is 3 vm_area_struct  
[ 218.906513] the page is in the memory!  
[ 218.906513] the page is not referenced!  
[ 218.906513] the page is not modified!  
[ 218.906514] This is 4 vm_area_struct  
[ 218.906514] the page is in the memory!  
[ 218.906515] the page is not referenced!  
[ 218.906515] the page is not modified!  
[ 218.906515] This is 5 vm_area_struct  
[ 218.906516] the page is in the memory!  
[ 218.906516] the page is not referenced!  
[ 218.906516] the page is not modified!  
[ 218.906517] This is 6 vm_area_struct  
[ 218.906517] the page is in the memory!  
[ 218.906518] the page is not referenced!  
[ 218.906518] the page is not modified!  
[ 218.906519] This is 7 vm_area_struct  
[ 218.906519] the page is not in memory!  
[ 218.906519] the page is not referenced!  
[ 218.906520] the page is not modified!  
[ 218.906520] This is 8 vm_area_struct  
[ 218.906521] the page is in the memory!  
[ 218.906521] the page is not referenced!  
[ 218.906521] the page is not modified!
```

User input is 1675

1. The data in this address is in memory or not

2. The page which this address belongs to has been referenced or not

3. The page which this address belongs to is dirty or not

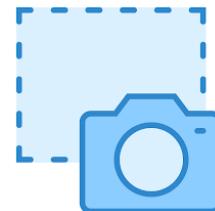
Submission

Submit your assignment zip file through D2L using the appropriate assignment link.

CS3502_D2Lname.zip



Kernel code



Screenshot



Report

User code