

CS 3502

Operating Systems

Project 3 Lab

Kun Suo

Computer Science, Kennesaw State University

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

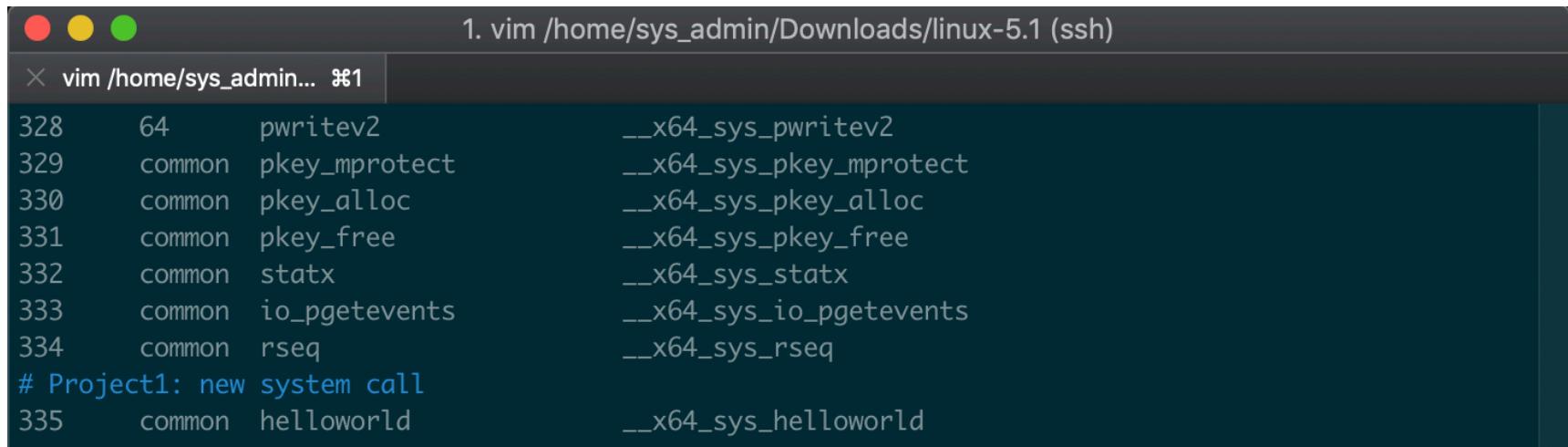
Review how to add system calls

- 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



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are three colored circles (red, yellow, green) and the text "1. vim /home/sys_admin/Downloads/linux-5.1 (ssh)". Below this, the title bar says "vim /home/sys_admin... #1". The main area contains a list of system calls and their corresponding kernel functions:

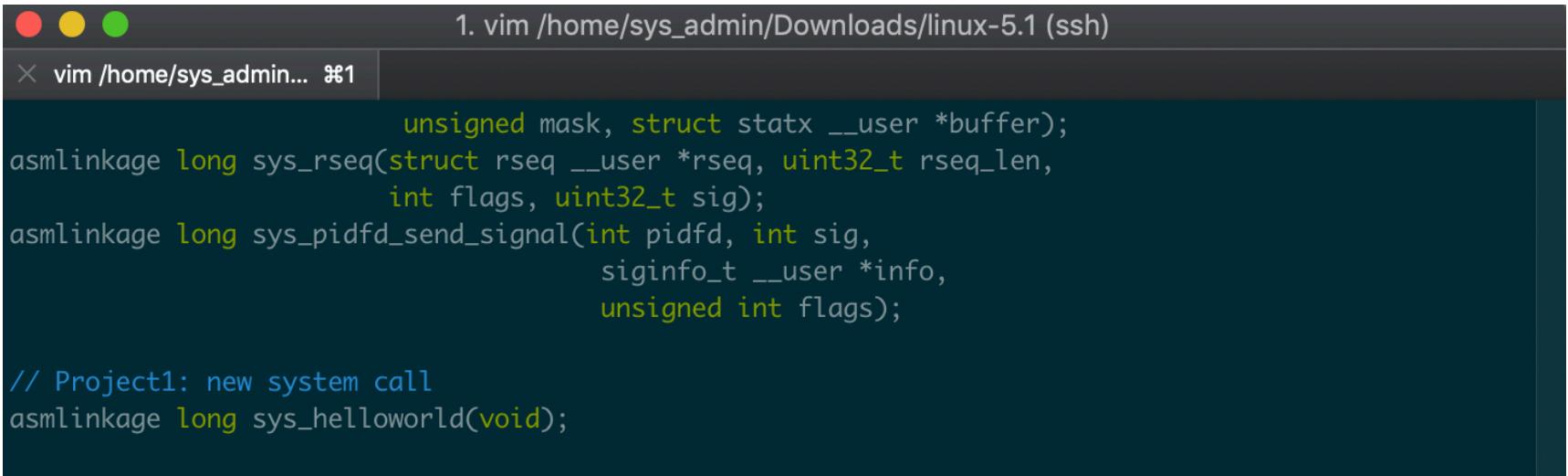
328	64	pwritev2	__x64_sys_pwritev2
329	common	pkey_mprotect	__x64_sys_pkey_mprotect
330	common	pkey_alloc	__x64_sys_pkey_alloc
331	common	pkey_free	__x64_sys_pkey_free
332	common	statx	__x64_sys_statx
333	common	io_pgetevents	__x64_sys_io_pgetevents
334	common	rseq	__x64_sys_rseq
# Project1: new system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

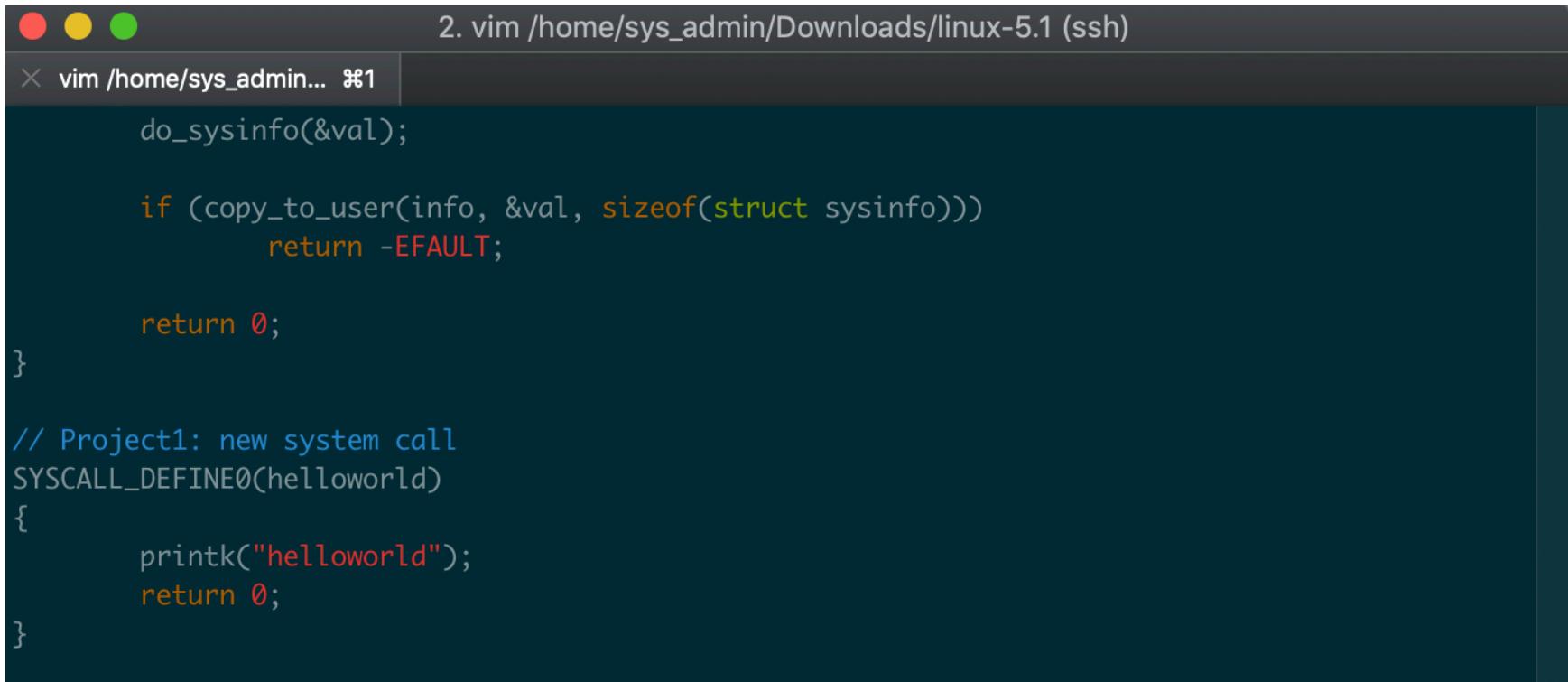
```
unsigned mask, struct statx __user *buffer);  
asmlinkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,  
                         int flags, uint32_t sig);  
asmlinkage long sys_pidfd_send_signal(int pidfd, int sig,  
                                       siginfo_t __user *info,  
                                       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:

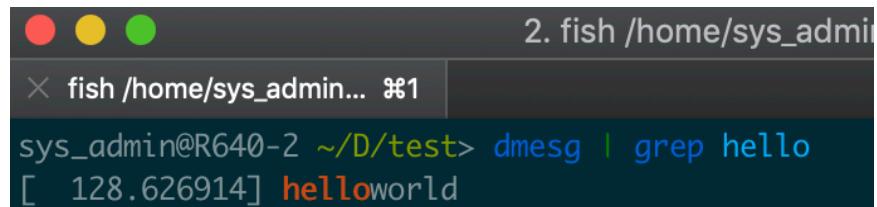
```
*****  
#include <linux/unistd .h>  
#include <sys/syscall .h>  
#include <sys/types .h>  
#include <stdio .h>  
#define __NR_helloworld 335  
  
int main(int argc, char *argv[])  
{  
    syscall (__NR_helloworld) ;  
    return 0 ;  
}  
*****
```

//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).



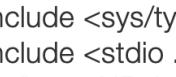
A screenshot of a terminal window titled "fish /home/sys_admin... #1". The command entered is "dmesg | grep hello". The output shows the message "[128.626914] helloworld" at the end of the log.

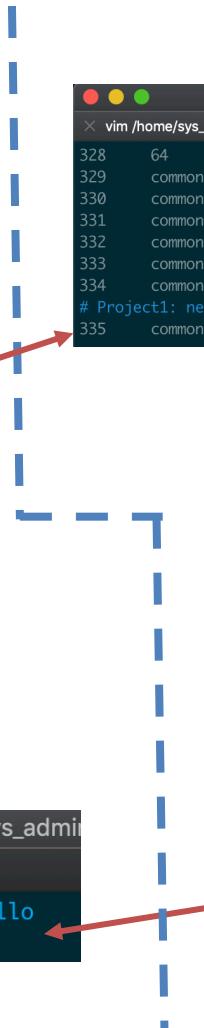
```
2. fish /home/sys_admin... #1  
x fish /home/sys_admin... #1  
sys_admin@R640-2 ~/D/test> dmesg | grep hello  
[ 128.626914] helloworld
```

Put it all together

User space

```
*****  
#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 (ssh)
vim /home/sys_admin... #1
    unsigned mask, struct statx __user *buffer);
asm linkage long sys_rseq(struct rseq __user *rseq, uint32_t rseq_len,
                           int flags, uint32_t sig);
asm linkage long sys_pidfd_send_signal(int pidfd, int sig,
                                       siginfo_t __user *info,
                                       unsigned int flags);

// Project1: new system call
asm linkage long sys_helloworld(void);
```

```
2. vim /home/sys_admin/Downloads/linux

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

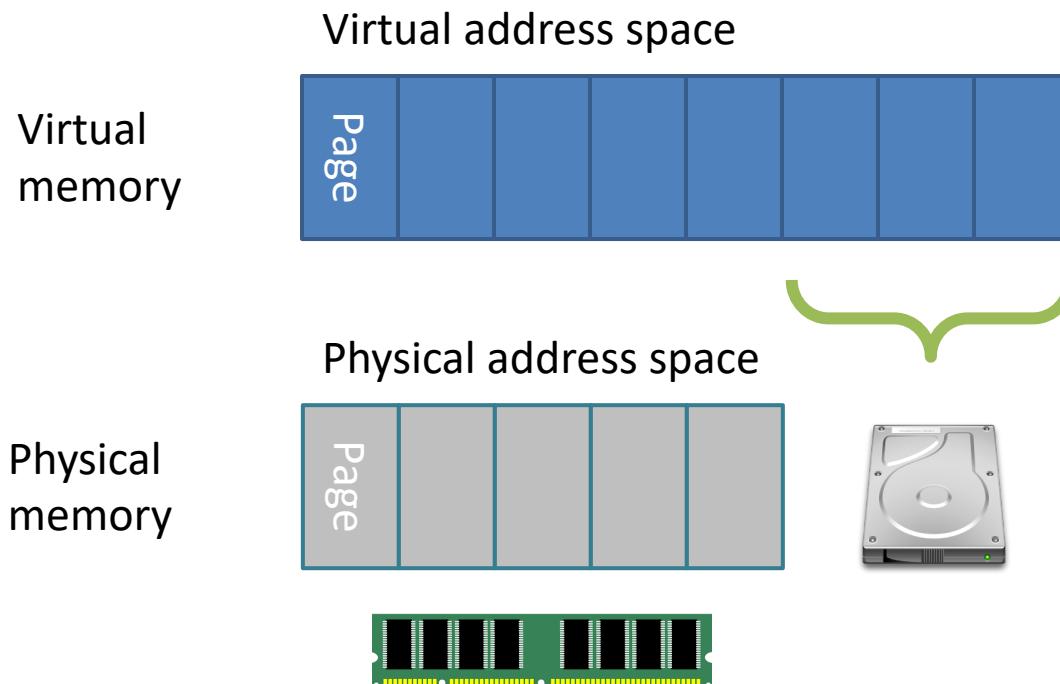
Project 3

- 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.

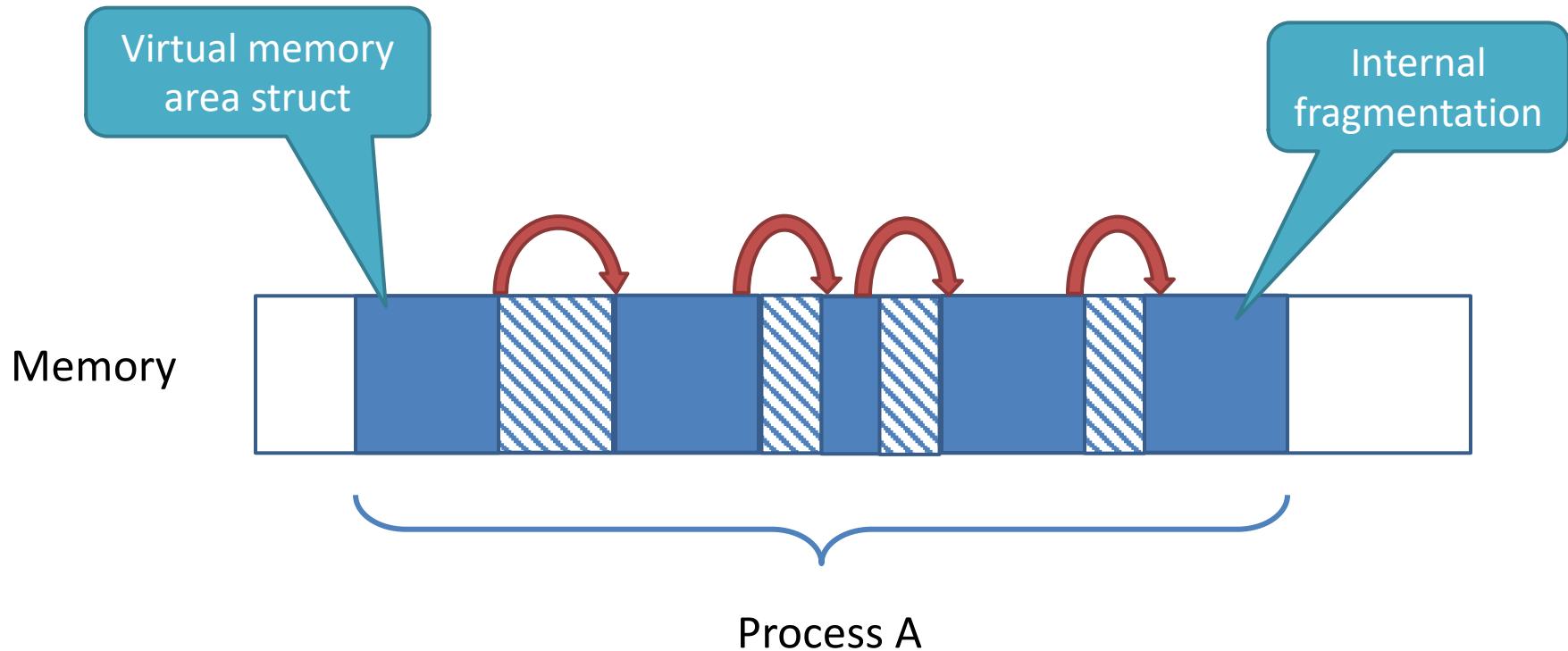


Project 3

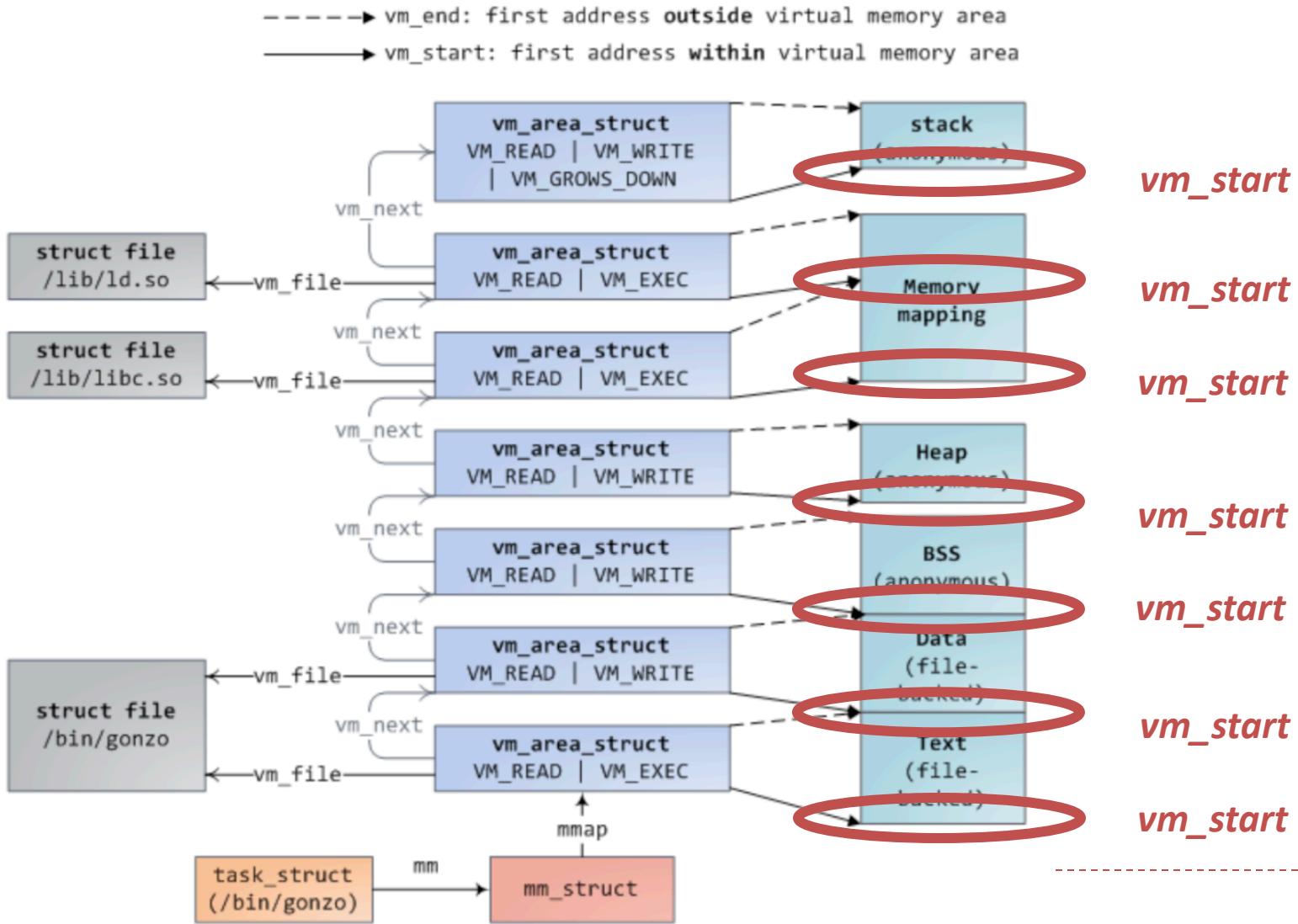
- Virtual memory address could be in memory or on disk.



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. */

```

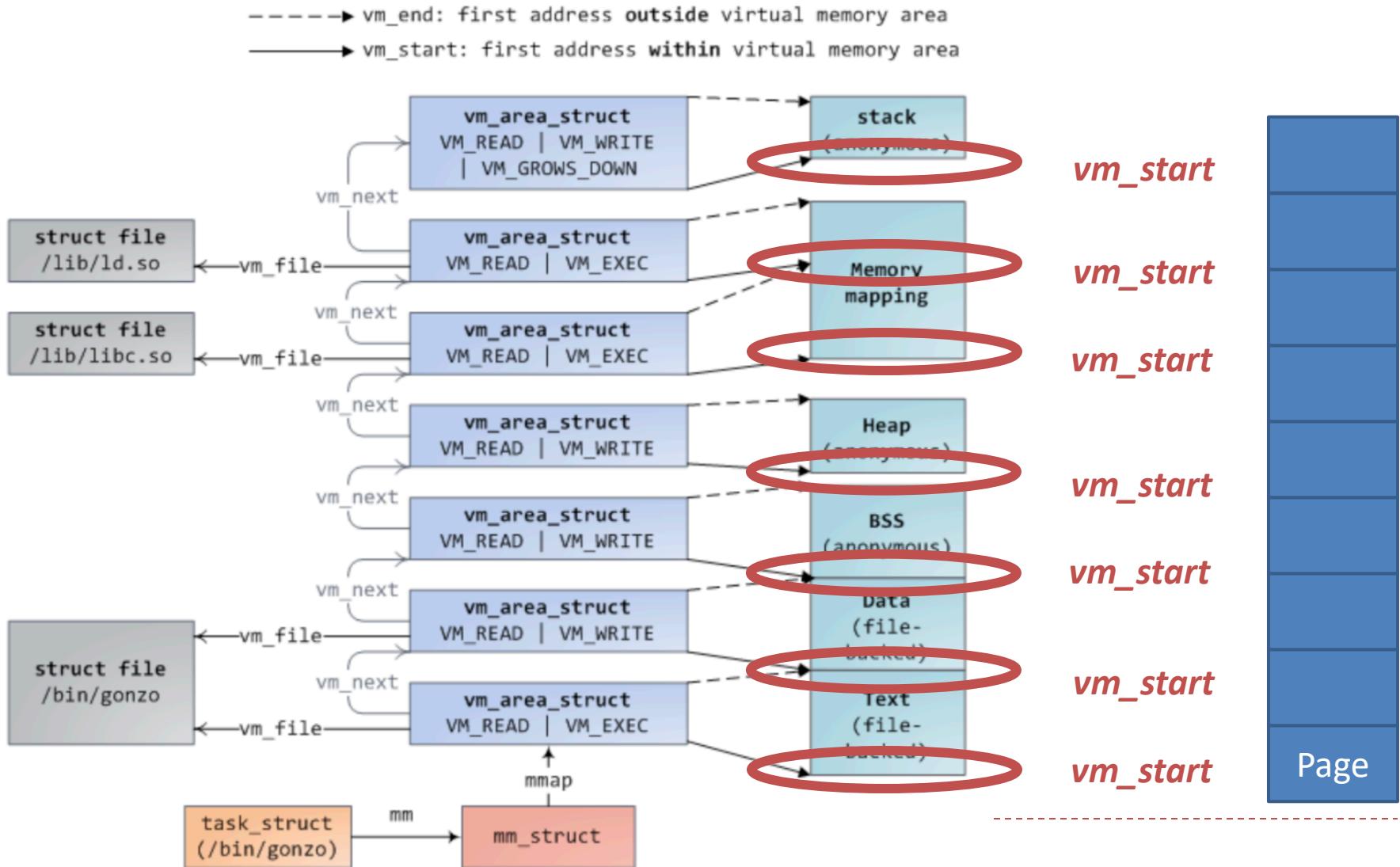


Project 3

- 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

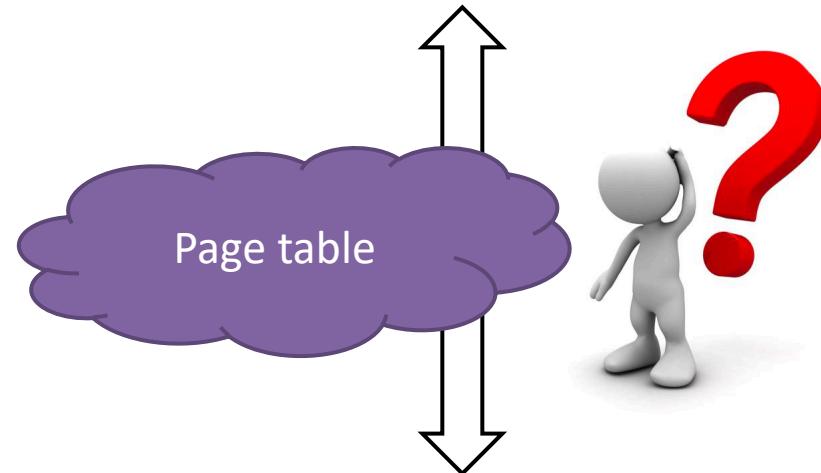
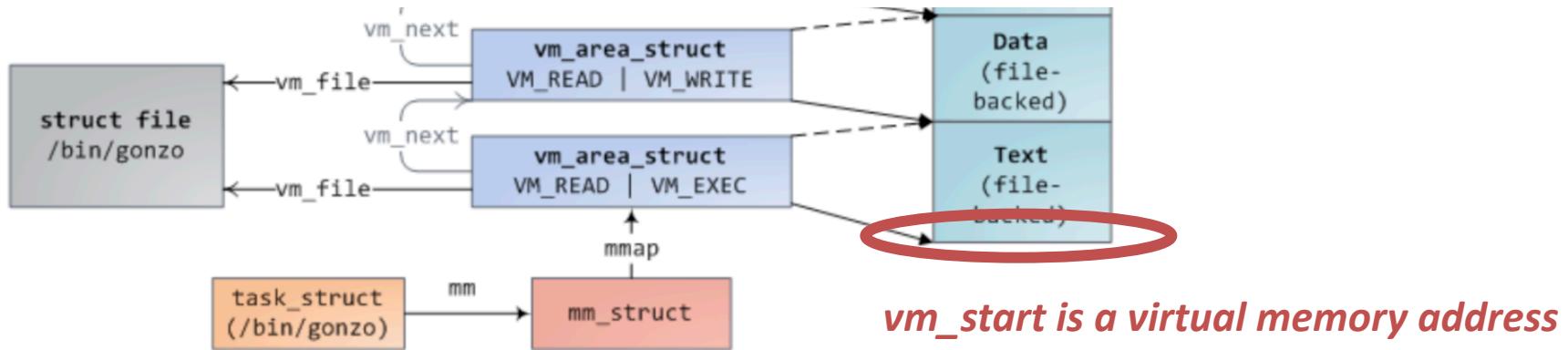
Table: Page Table Entry Status Bits

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

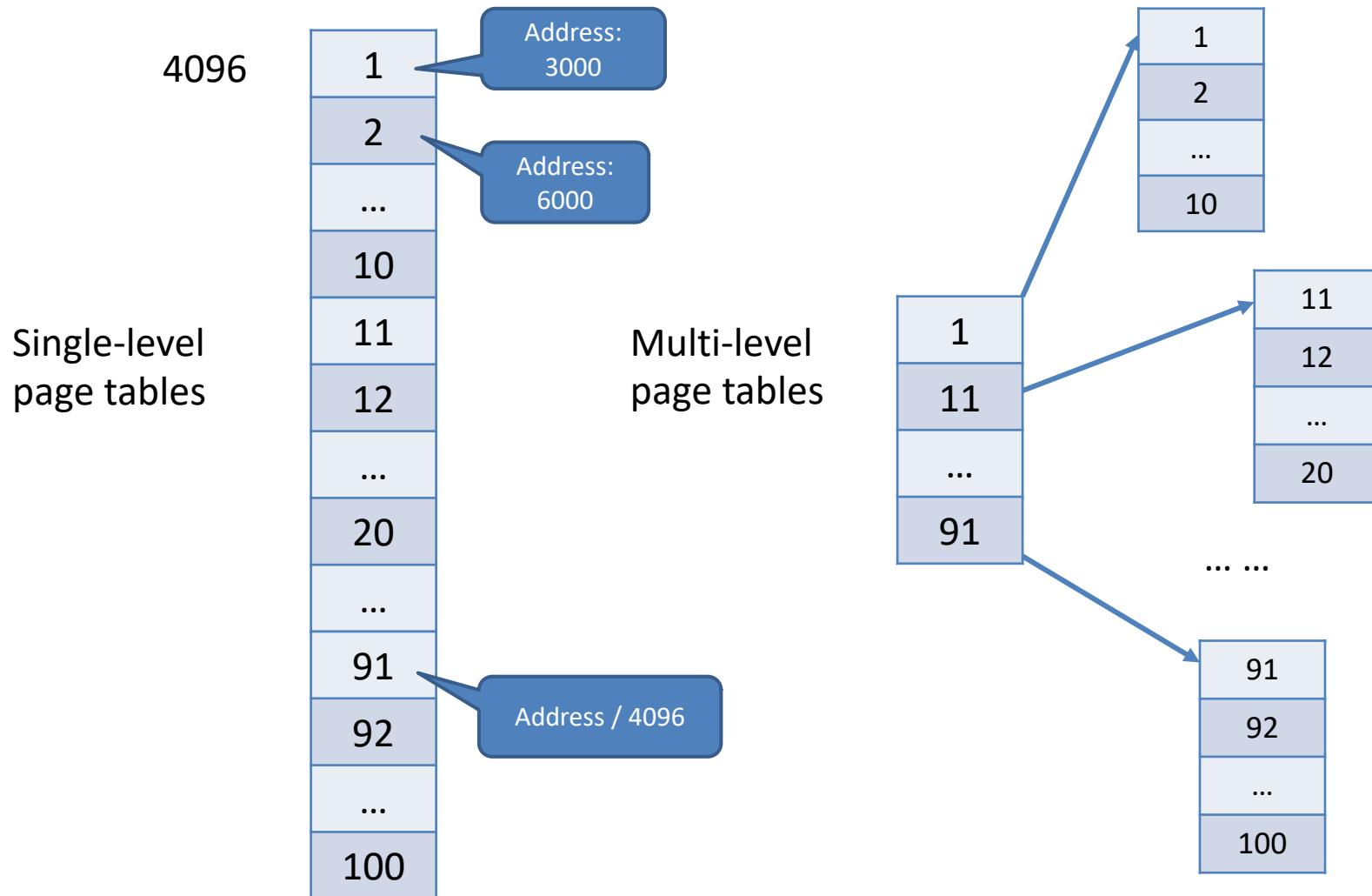


`typedef struct { pteval_t pte; } pte_t;`

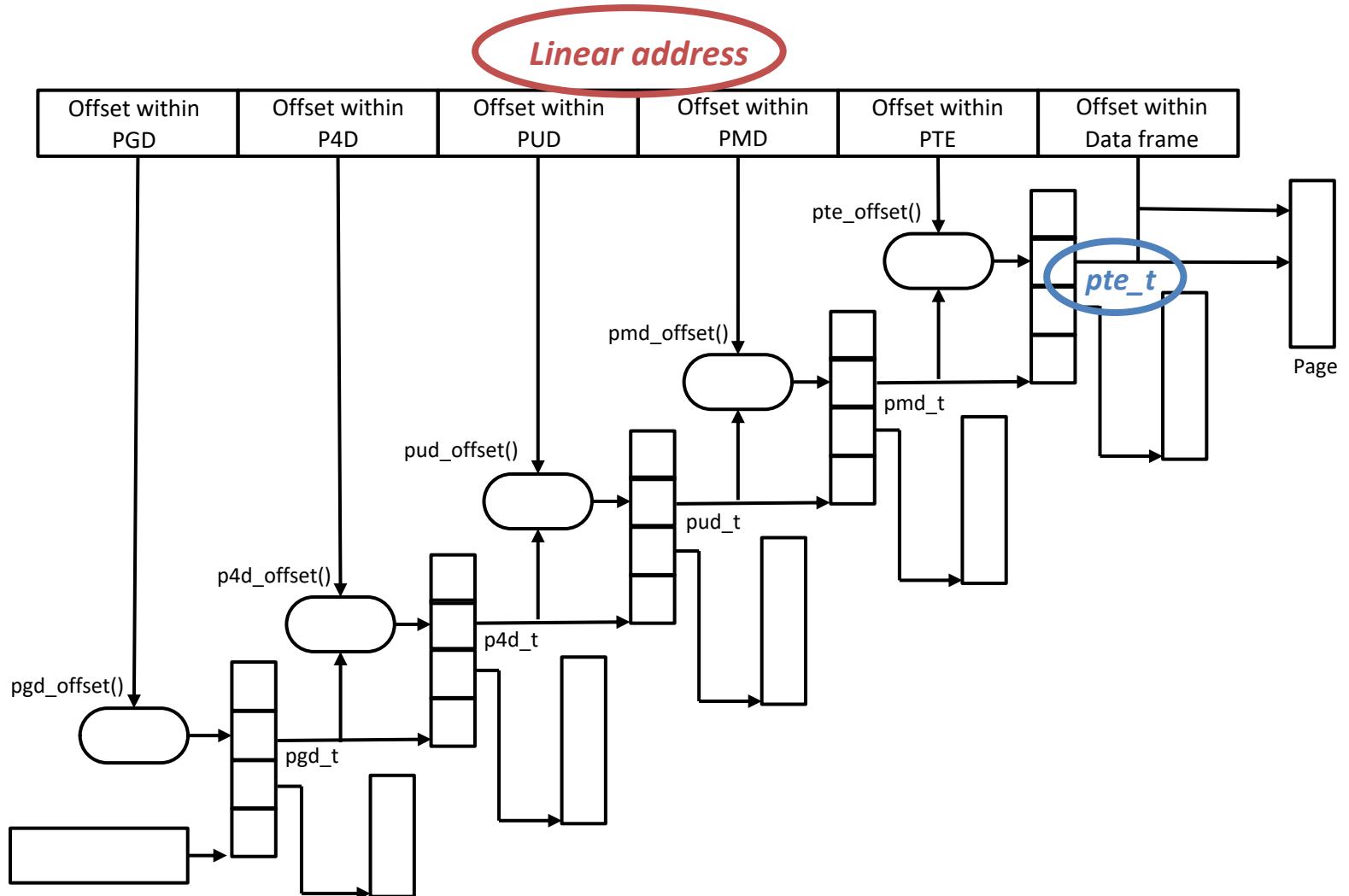
pte_t is a memory page entry



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);
}

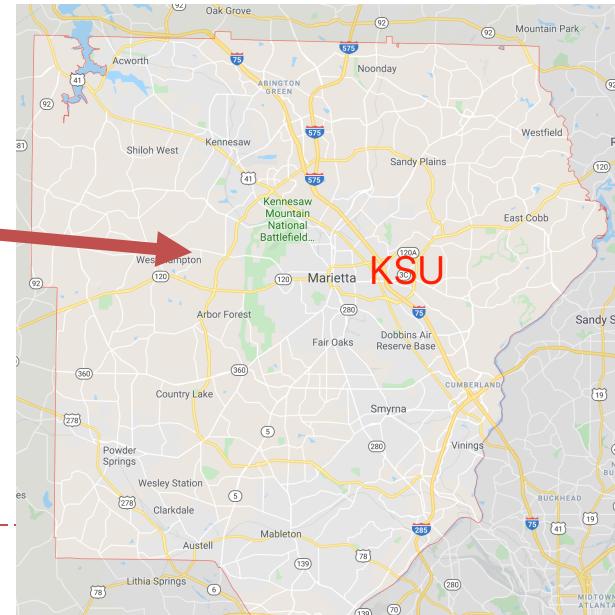
/* Find an entry in the second-level page table.. */
static inline pmd_t *pmd_offset(pud_t *pud, unsigned long address)
{
    return (pmd_t *)pud_page_vaddr(*pud) + pmd_index(address);
}

static inline pte_t *pte_offset_kernel(pmd_t *pmd, unsigned long address)
{
    return (pte_t *)pmd_page_vaddr(*pmd) + pte_index(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

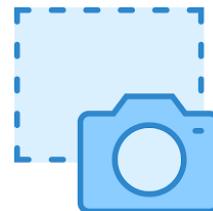
Submission

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

CS3502_D2Lname.zip



Kernel code



Screenshot

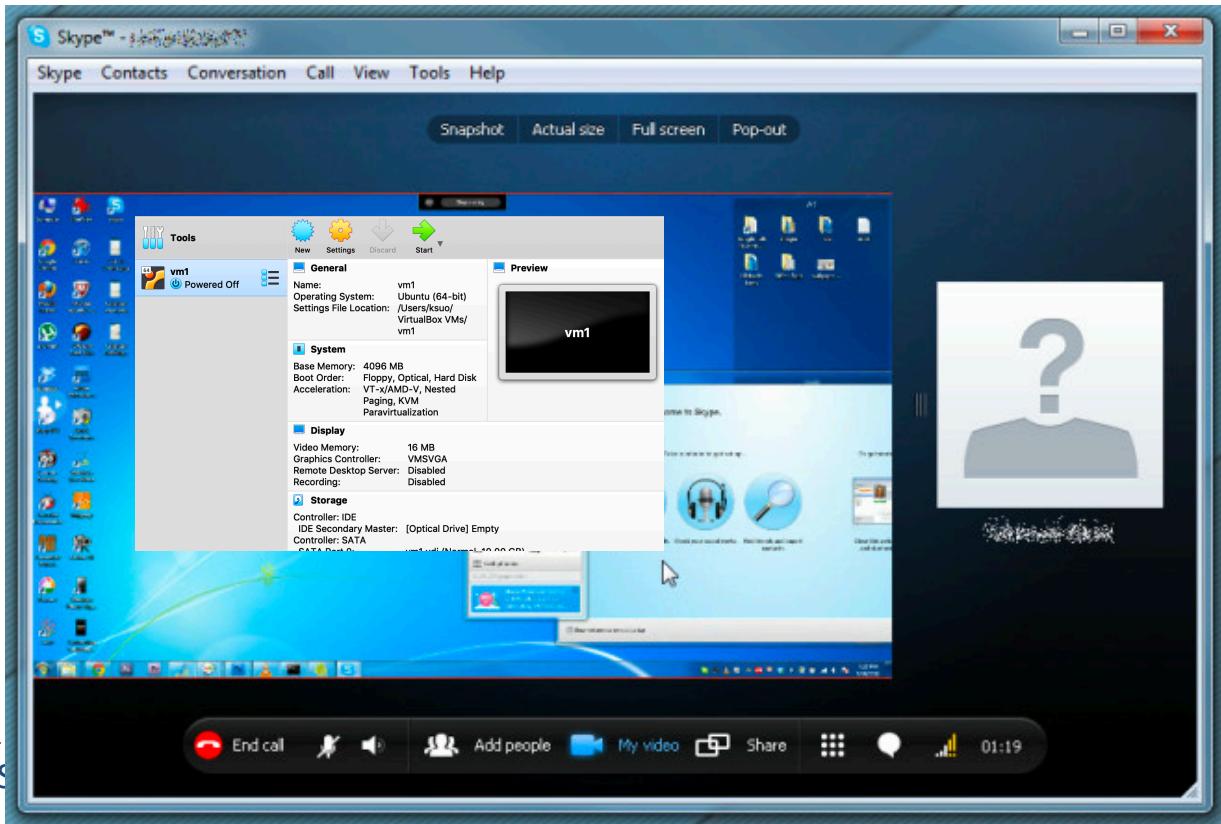


Report

User code

Questions

- T/Th 2-3pm, J-318
- Skype ID: *suokun.nju*, share home screen, only voice, no video, no remote control and privacy issue. No time or location limitation.



Assignment 1

- Write a system call *sys_getmemInfo* (e.g., follow the steps in Project 1) to report statistics of a process's virtual address space. You can use *printk* in your system call and collect the information in *dmesg* command. The system call should take a process ID as input and outputs the following information about the process:
 1. Each virtual memory area's access permissions.
 2. The names of files mapped to these virtual memory areas.
 3. The total size of the process's virtual address space.



Assignment 1

- Write a system call *sys_getmemInfo* (e.g., follow the steps in Project 1) to report **statistics of a process's virtual address space**. You can use *printk* in your system call and collect the information in *dmesg* command. The system call should take a process ID as input and outputs the following information about the process:
 1. Each virtual memory area's access permissions.
 2. The names of files mapped to these virtual memory areas.
 3. The total size of the process's virtual address space.



Revisit Process Data Structure

- Process Control Block (PCB)
 - State
 - Identifiers
 - Scheduling info
 - File system
 - Virtual memory
 - Process specific context
 - ...

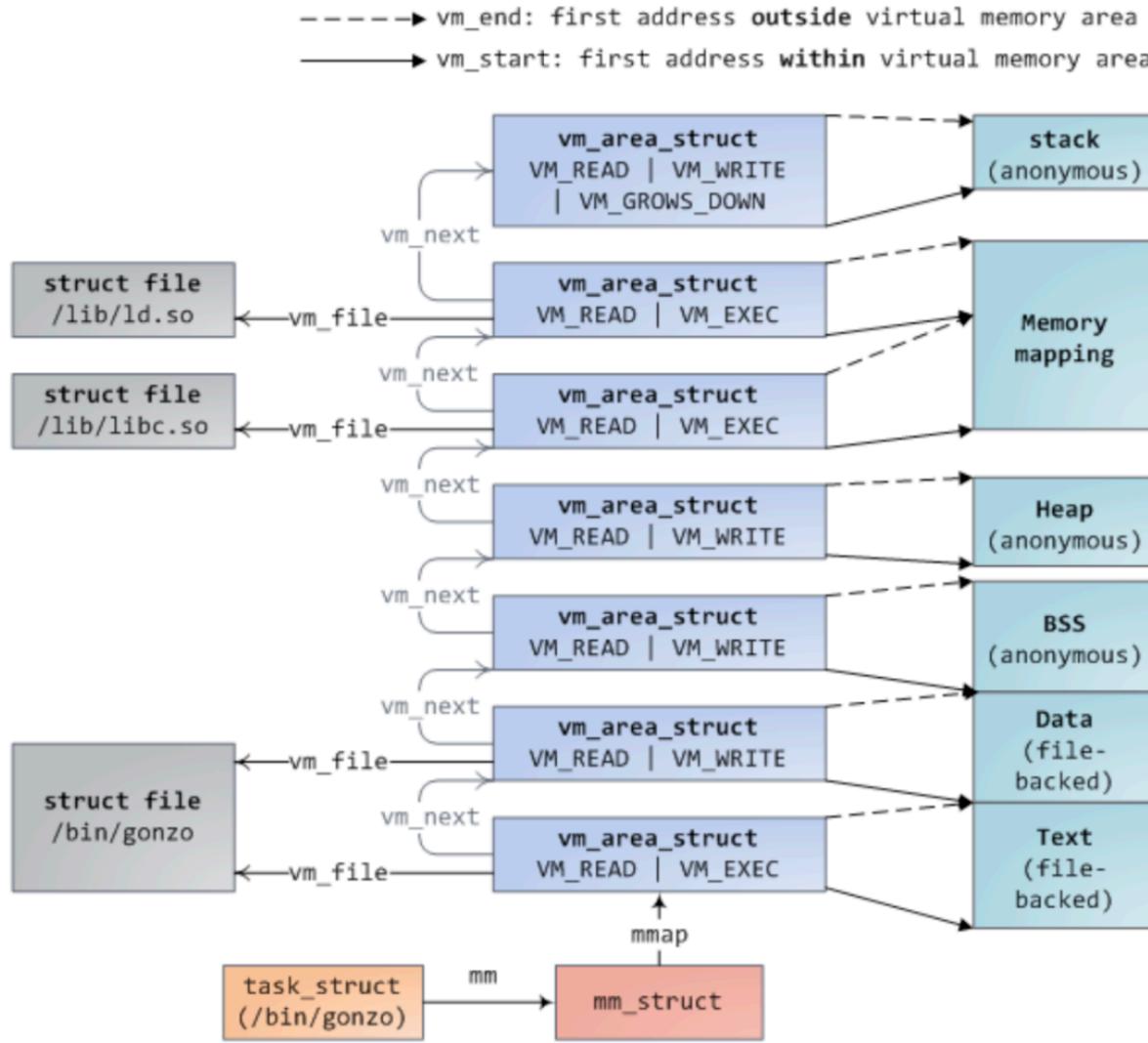
task_struct

<https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L637>

<https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L740>



Revisit Process Data Structure



Revisit Process Data Structure

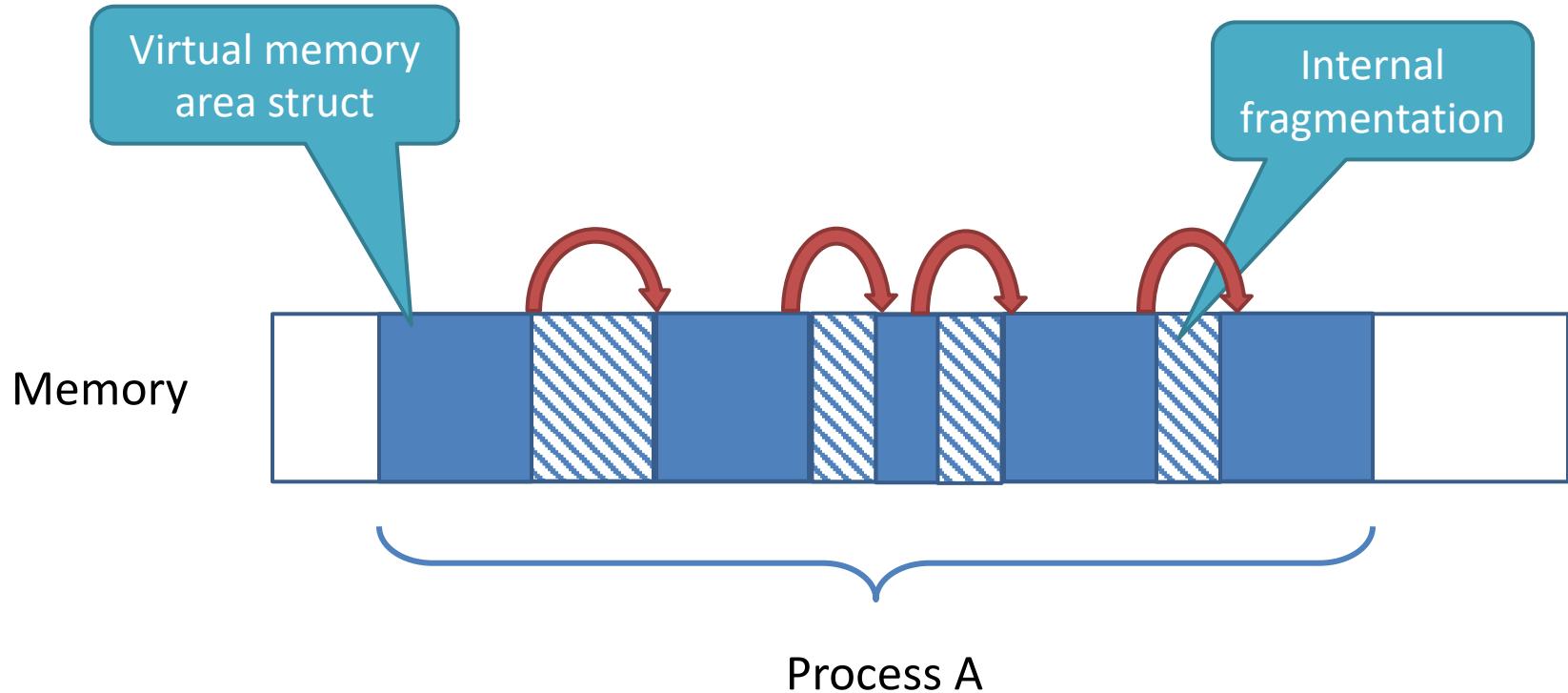
```
struct task_struct {
#define CONFIG_THREAD_INFO_IN_TASK
/*
 * For reasons of header soup (see current_thr
 * must be the first element of task_struct.
 */
struct thread_info          thread_info;
#endif
/* -1 unrunnable, 0 runnable, >0 stopped: */
volatile long                state;

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



Revisit Process Data Structure

```
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
#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
```

```
struct vm_area_struct {
    /* 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
     * VMA 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. */
}
```



1. Each virtual memory area's access permissions

```
struct vm_area_struct {
    /* 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;
    pgprot_t vm_page_prot;
    unsigned long vm_flags;
} /* The address space we belong to. */
   /* Access permissions of this VMA. */
   /* Flags, see mm.h. */
```

typedef struct pgprot { pgprotval_t pgprot; } pgprot_t;

typedef unsigned long pgprotval_t;

printk("%d", permission);

printk("%lu", permission);

```
graph LR; A[pgprot_t] --> B[pgprotval_t]; B --> C[pgprotval_t]
```



2. The names of files mapped to these virtual memory areas

```
struct vm_area_struct {
    /* 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;

    /* Information about our backing store: */
    unsigned long vm_pgoff;          /* Offset (within vm_file) in PAGE_SIZE
                                    units, *not* PAGE_CACHE_SIZE */
    /* File we map to (can be NULL). */
    /* was vm_pte (shared mem) */

    struct file * vm_file;
    void * vm_private_data;

#ifndef CONFIG_MMU
    struct vm_region *vm_region;     /* NOMMU mapping region */
#endif
#ifdef CONFIG_NUMA
    struct mempolicy *vm_policy;     /* NUMA policy for the VMA */
#endif
};
```

```
printf("%s", vas->vm_file->
        f_path.dentry->d_name.name);
```

```
struct file {
    union {
        struct llist_node fu_llist;
        struct rcu_head fu_rcuhead;
    } fu;
    struct path f_path;
    struct inode *f_inode;
};

struct path {
    struct vfsmount *mnt;
    struct dentry *dentry;
};

struct dentry {
    /* RCU lookup touched fields */
    unsigned int d_flags;
    seqcount_t d_seq;
    struct hlist_node d_hash;
    struct dentry *d_parent;
    struct qstr d_name;
    struct inode *d_inode;
};

struct qstr {
    union {
        struct {
            HASH_LEN_DECLARE;
        };
        u64 hash_len;
    };
    const unsigned char *name;
};
```

2. The names of files mapped to these virtual memory areas

```
struct vm_area_struct {  
    /* 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;  
  
    /* Information about our backing store: */  
    unsigned long vm_pgoff;          /* Offset (within vm_file) in PAGE_SIZE  
                                    units, *not* PAGE_CACHE_SIZE */  
    struct file * vm_file;           /* File we map to (can be NULL). */  
    void * vm_private_data;          /* was vm_pte (shared mem) */  
  
#ifndef CONFIG_MMU  
    struct vm_region *vm_region;  
#endif  
#ifdef CONFIG_NUMA  
    struct mempolicy *vm_policy;     /* NUMA policy for the VMA */  
#endif  
};
```

fs = vas->vm_file;
if(fs != NULL){
 printk("The file name mapped to
this vma is %s\n", fs-
 >f_path.dentry->d_name.name);
}
} else {
 printk("The file name mapped to
this vma is NULL\n");
}
}

It could be null!



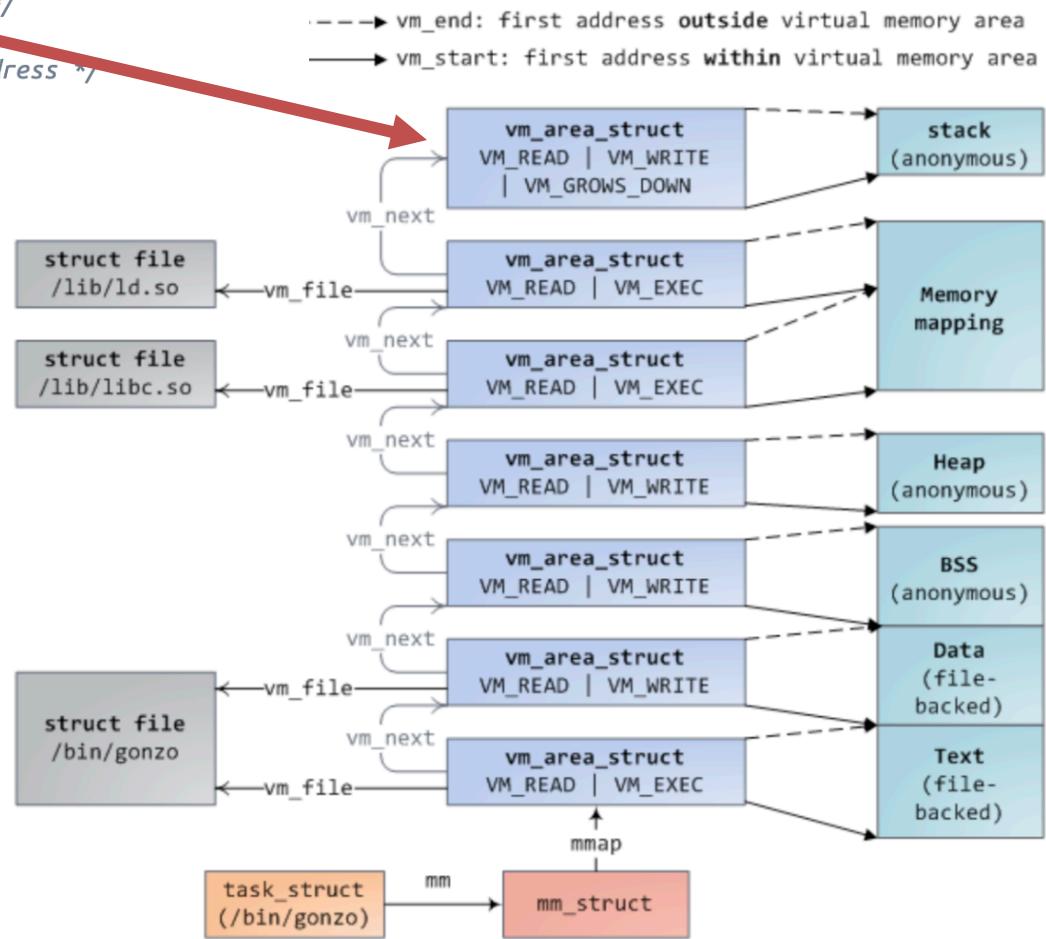
3. The total size of the process's virtual address space

```
struct vm_area_struct {
    /* 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
                                    we can see in 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;
```



Assignment 1

- Write a system call *sys_getmemInfo* (e.g., follow the steps in Project 1) to report statistics of a process's virtual address space. You can use *printk* in your system call and collect the information in *dmesg* command.
- Input: pid
- Output:
 - Each virtual memory area's access permissions.
 - The names of files mapped to these virtual memory areas.
 - The total size of the process's virtual address space.



Assignment 1: user test program 1

```
1 #include <linux/unistd.h>
2 #include <sys/syscall.h>
3 #include <sys/types.h>
4 #include <stdio.h>
5 #include <unistd.h>
6
7 #define __NR_getmemInfo 338
8
9
10 int main(int argc, char *argv[])
11 {
12     int no = getpid();
13     syscall (__NR_getmemInfo, no);
14     return 0;
15 }
```

- One test program just calls the new system call and report the calling process's statistics.

System call getpid()
will return the pid of
current process.

Assignment 1: expected output of user test program 1

```
fish /home/ksuo/b/.../1810
```

User input is 1810

```
[ 6557.143619] Find the task 1810
```

1. Virtual memory area access permission

```
[ 6557.143620] The number 0 vma's access permission is 37,  
[ 6557.143621] The file name mapped to this vma is test1.o  
[ 6557.143622] The number 1 vma's access permission is 9223372036854775845,  
[ 6557.143622] The file name mapped to this vma is test1.o  
[ 6557.143623] The number 2 vma's access permission is 9223372036854775845  
[ 6557.143623] The file name mapped to this vma is test1.o  
[ 6557.143624] The number 3 vma's access permission is 37,  
[ 6557.143624] The file name mapped to this vma is libc-2.27.so  
[ 6557.143624] The number 4 vma's access permission is 288,  
[ 6557.143625] The file name mapped to this vma is libc-2.27.so  
[ 6557.143625] The number 5 vma's access permission is 9223372036854775845,  
[ 6557.143626] The file name mapped to this vma is libc-2.27.so  
[ 6557.143626] The number 6 vma's access permission is 9223372036854775845,  
[ 6557.143627] The file name mapped to this vma is libc-2.27.so  
[ 6557.143627] The number 7 vma's access permission is 9223372036854775845,  
[ 6557.143627] The file name mapped to this vma is NULL  
[ 6557.143628] The number 8 vma's access permission is 37,  
[ 6557.143628] The file name mapped to this vma is ld-2.27.so  
[ 6557.143629] The number 9 vma's access permission is 9223372036854775845,  
[ 6557.143629] The file name mapped to this vma is NULL  
[ 6557.143630] The number 10 vma's access permission is 9223372036854775845,  
[ 6557.143630] The file name mapped to this vma is ld-2.27.so  
[ 6557.143631] The number 11 vma's access permission is 9223372036854775845,  
[ 6557.143631] The file name mapped to this vma is ld-2.27.so  
[ 6557.143631] The number 12 vma's access permission is 9223372036854775845,  
[ 6557.143632] The file name mapped to this vma is NULL  
[ 6557.143632] The number 13 vma's access permission is 9223372036854775845,  
[ 6557.143633] The file name mapped to this vma is NULL  
[ 6557.143633] The number 14 vma's access permission is 9223372036854775845,  
[ 6557.143633] The file name mapped to this vma is NULL  
[ 6557.143634] The number 15 vma's access permission is 37,  
[ 6557.143634] The file name mapped to this vma is NULL  
[ 6557.143635] The size of the process's virtual address space is 4476928
```

2. Virtual memory area mapped files

3. Total virtual memory area size

Assignment 1: user test program 2

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <sys/types.h>
4 #include <sys/syscall.h>
5 #include <unistd.h>
6
7 #define __NR_getmemInfo 338
8 #define NUM_THREADS 3
9
10 void *myThread(void *threadid)
11 {
12     int no = syscall(SYS_gettid);
13     printf("This is a pthread. Pid is %d\n", no);
14     syscall(__NR_getmemInfo, no);
15 }
16
17
18 int main(int argc, char *argv[])
19 {
20     pthread_t threads[NUM_THREADS];
21     int t, rc;
22
23     printf("This is a process. Pid is %d\n", getpid());
24
25     for(t=0; t<NUM_THREADS; t++){
26         rc = pthread_create(&threads[t], NULL, (void*)myThread, NULL);
27
28         if(rc){
29             printf("Error\n");
30             return 1;
31         }
32     }
33
34     for(t=0; t<NUM_THREADS; t++){
35         pthread_join(threads[t], NULL);
36     }
37
38     printf("%s", "program is over.\n");
39     pthread_exit(NULL);
40     return 0;
41 }
```

System call
SYS_gettid() returns
the pid of each thread

- The other test program should create multiple threads and report information about individual threads.

Assignment 1: expected output of user test program 2

```
ksuo@ksuo-VirtualBox ~/hw3> ./test2.o
This is a process. Pid is 2007
This is a pthread. Pid is 2008
This is a pthread. Pid is 2010
This is a pthread. Pid is 2009
program is over.
```

User side

Kernel dmesg

```
[ 6859.138227] Find the task 2009
[ 6859.138229] The number 0 vma's access permission is 37,
[ 6859.138230] The file name mapped to this vma is test2.o
[ 6859.138231] The number 1 vma's access permission is 9223372036854775845,
[ 6859.138232] The number 2 vma's access permission is 9223372036854775845,
[ 6859.138233] The number 3 vma's access permission is 9223372036854775845,
[ 6859.138234] The number 4 vma's access permission is 288,
[ 6859.138235] The number 5 vma's access permission is 9223372036854775845,
[ 6859.138236] The number 6 vma's access permission is 288,
[ 6859.138237] The number 7 vma's access permission is 9223372036854775845,
[ 6859.138238] The number 8 vma's access permission is 9223372036854775845,
[ 6859.138239] The number 9 vma's access permission is 288,
[ 6859.138240] The number 10 vma's access permission is 9223372036854775845,
[ 6859.138241] The number 11 vma's access permission is 9223372036854775845,
[ 6859.138242] The number 12 vma's access permission is 9223372036854775845,
[ 6859.138243] The number 13 vma's access permission is 37,
[ 6859.138244] The number 14 vma's access permission is 288,
[ 6859.138245] The number 15 vma's access permission is 9223372036854775845,
[ 6859.138246] The number 16 vma's access permission is 9223372036854775845,
[ 6859.138247] The number 17 vma's access permission is 9223372036854775845,
[ 6859.138248] The number 18 vma's access permission is 37,
[ 6859.138249] The number 19 vma's access permission is 9223372036854775845,
[ 6859.138250] The number 20 vma's access permission is 9223372036854775845,
```

Reference

- If you forget how to print out process related information, please review our project 1

The screenshot shows a course page from Kennesaw State University's LMS. The top navigation bar includes links for Course Home, Content, Discussions, Assignments, Quizzes, Other, Classlist, and Grades. Below the navigation, the breadcrumb trail shows the user has navigated from the Table of Contents through ppt to Project1-Reference-Code. The main title is "Project1-Reference-Code". A file card displays "Project1-Reference-Code.zip" (4.82 KB, last modified Sep 29, 2019 at 11:27 AM) with a "Download" button.

Operating Systems Section 03 Fall S...

Course Home Content Discussions Assignments Quizzes Other ▾ Classlist Grades

Table of Contents > ppt > Project1-Reference-Code

Project1-Reference-Code

Project1-Reference-Code.zip
4.82 KB - Last Modified Sep 29, 2019
11:27 AM

Download

Submission

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

