CSE 306 Operating Systems
Process Address Space

YoungMin Kwon

Process's Address Space

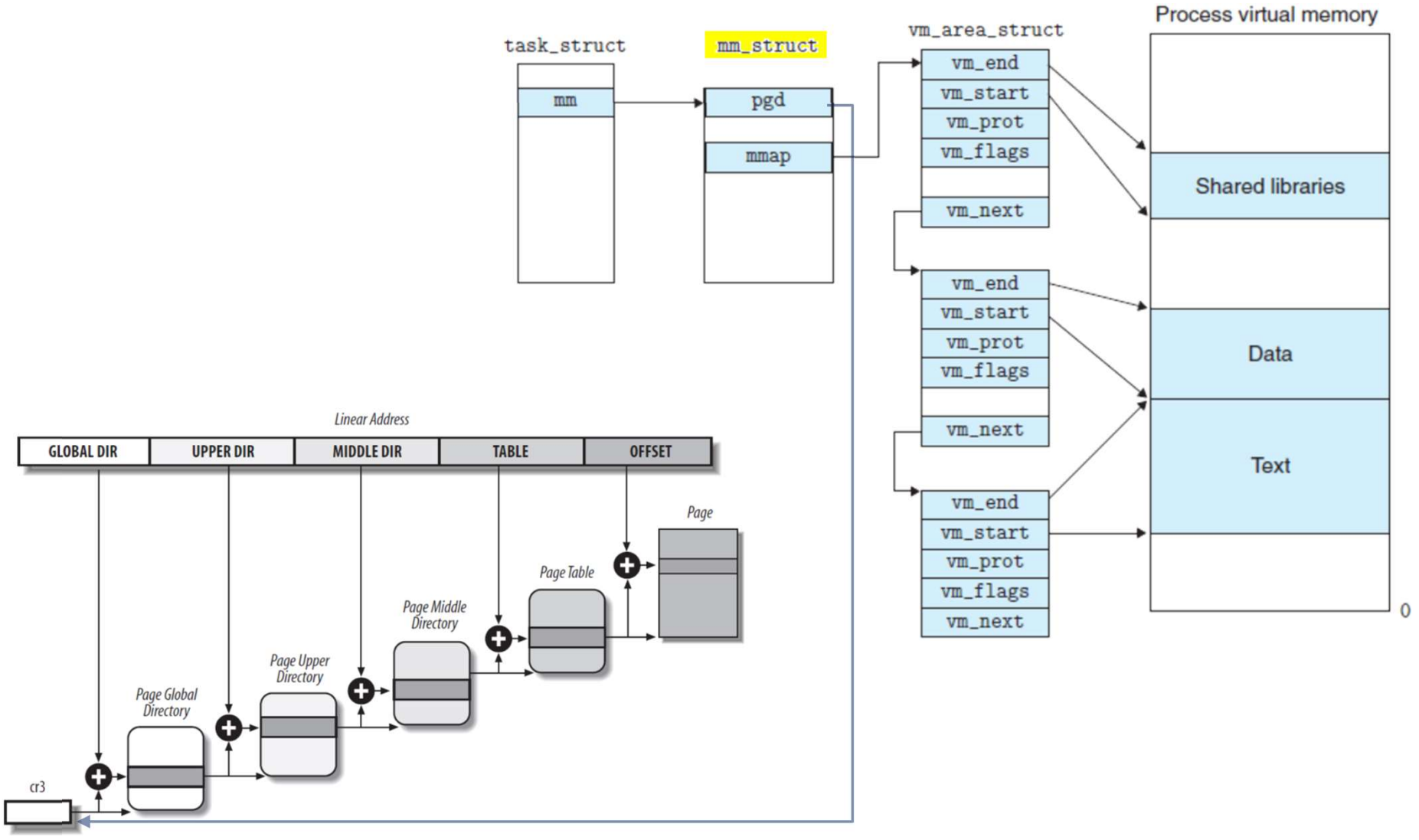
- Address space of a process
  - All linear addresses that the process is allowed to use
- Memory regions
  - Intervals of linear address characterized by an initial linear address, a length, and some access rights
- Memory allocation for user processes
  - On request, a right to use a new range of linear address is given to the process
  - The new interval, called memory region, becomes a part of the process's address space
  - A page frame is allocated later when the process causes a page fault

Memory Descriptor

- Memory descriptor (struct mm_struct)
  - Includes all information about the process address space
- Some fields of struct mm_struct

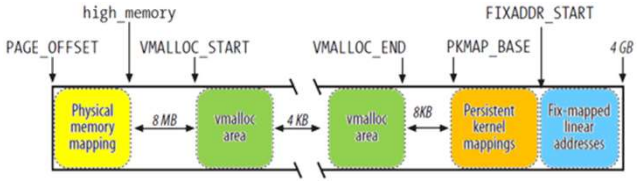
Type	Field	Description
struct vm_area_struct *	mmap	Pointer to the head of the list of memory region objects
struct rb_root	mm_rb	Pointer to the root of the red-black tree of memory region objects
struct vm_area_struct *	mmap_cache	Pointer to the last referenced memory region object
pgd_t *	pgd	Pointer to the Page Global Directory
unsigned long	start_code	Initial address of executable code
unsigned long	end_code	Final address of executable code
unsigned long	start_data	Initial address of initialized data
unsigned long	end_data	Final address of initialized data
unsigned long	start_brk	Initial address of the heap
unsigned long	brk	Current final address of the heap
unsigned long	start_stack	Initial address of User Mode stack

Memory Descriptor



Memory Descriptor of Kernel Threads

- Kernel threads
  - Kernel threads run in Kernel mode and do not use memory regions



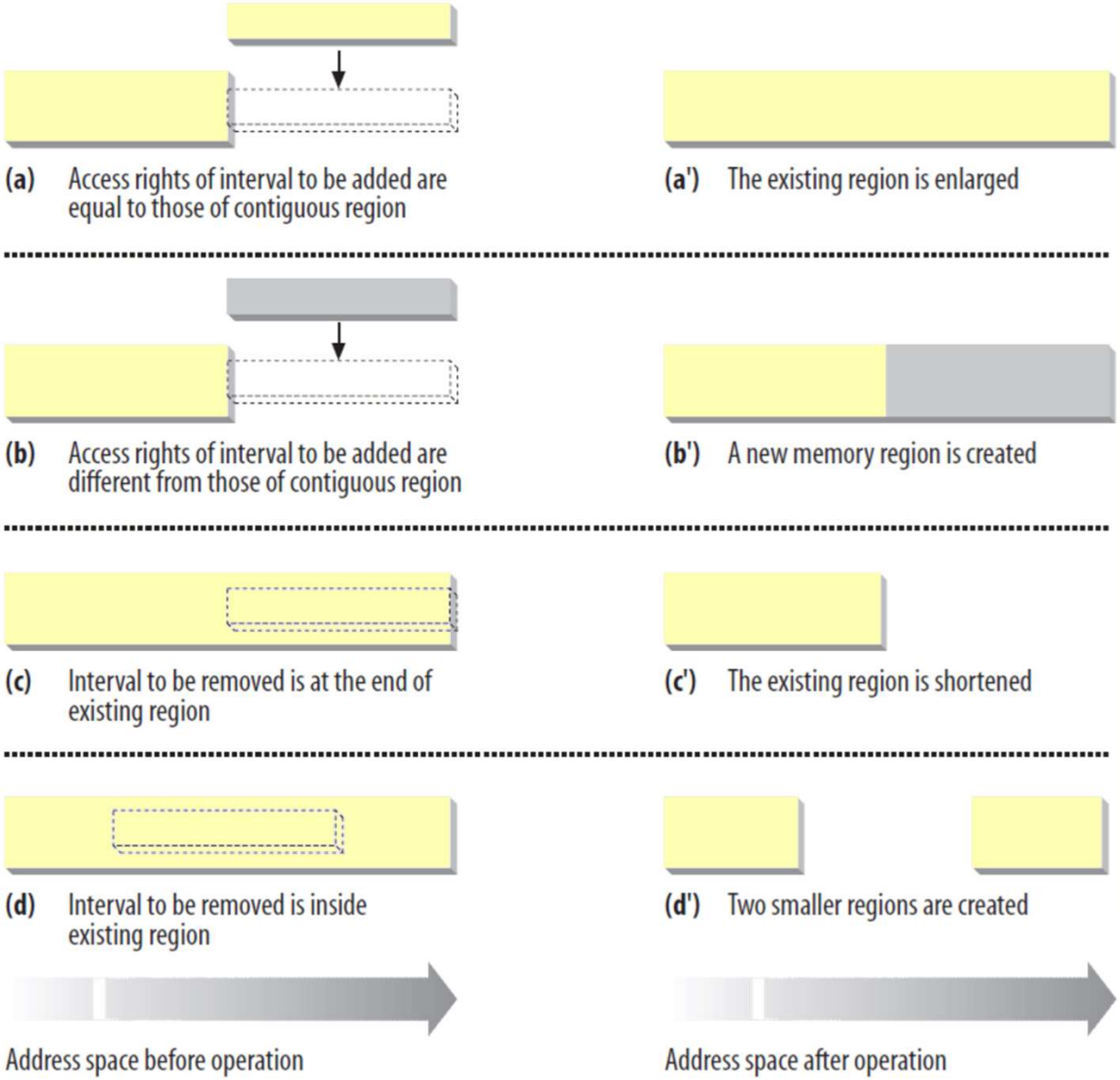
  - Never access linear addresses below TASK_SIZE (0xc0000000)
    - Page Table above TASK_SIZE should always be identical
    - To avoid useless TLB and cache flushes, a kernel thread uses the set of page tables of the last previously running regular process
  - mm field of the process descriptor is NULL
    - active_mm field of the process descriptor points to the mm field of the previously running process

Memory Regions

- Linux manages memory regions by vm_area_struct
- Some fields of struct vm_area_struct

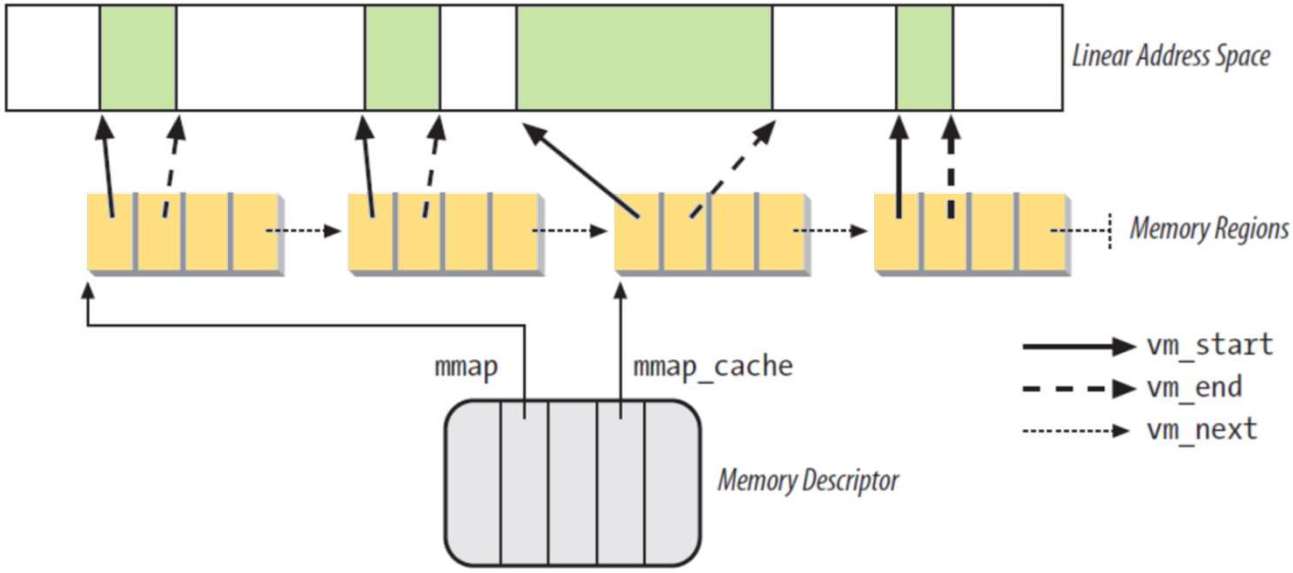
Type	Field	Description
struct mm_struct *	vm_mm	Pointer to the memory descriptor that owns the region.
unsigned long	vm_start	First linear address inside the region.
unsigned long	vm_end	First linear address after the region.
struct vm_area_struct *	vm_next	Next region in the process list.
pgprot_t	vm_page_prot	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).
struct vm_operations_struct*	vm_ops	Pointer to the methods of the memory region.

Adding or Removing a Memory Region



Kernel tries to merge regions if their access rights match

Memory Region Data Structures

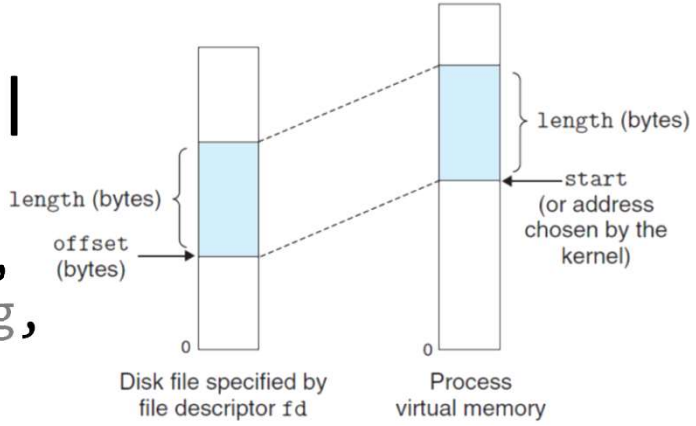


- Two structures for memory regions:
  - Linked list: to scan the entire memory regions
  - RB tree: to quickly find a memory region

Memory Region Handling

- To allocate a linear address interval

```
unsigned long do_mmap(struct file *file,
    unsigned long addr, unsigned long len,
    unsigned long prot, unsigned long flag,
    unsigned long offset);
```



  - If adjacent intervals have the same access rights, merge with them
  - Otherwise, creates a new linear address interval
    - If file and offset are valid, maps the file at offset for length len.
    - addr: address to start the search from
    - prot: access permissions

Memory Region Handling

- do_mmap() (case when a new vma is created)
  - vm_area_struct is allocated from vm_area_cachep slab cache
  - New vma is added to the linked list and rbtree
  - total_vm field is updated
  - Return the initial address of the newly created address interval

Memory Region Handling

- To release a linear address interval

```
int do_munmap(struct mm_struct *mm,
              unsigned long start,  size_t len);
```

  - Removes an address interval from a specific process address space
  - mm specifies the address space
  - Starting from start, len byte will be removed from the address space

Memory Region Handling

- Helper functions
  - find_vma(): find the closest region to a given address
    - The region's end address is larger than the address
  - find_vma_intersection(): find a region that overlaps a given interval
  - get_unmapped_area(): find a free interval

```
struct vm_area_struct *find_vma(struct mm_struct *mm, unsigned long addr) {
…
    /*check mmap_cache first*/
    vma = mm->mmap_cache;
    if (vma && vma->vm_start <= addr && addr < vma->vm_end)
        return vma;
…
    /* search the rb tree */
    rb_node = mm->mm_rb.rb_node;
    vma = NULL;
    while (rb_node) {
        vma_tmp = rb_entry(rb_node, struct vm_area_struct, vm_rb);
        if (addr < vma_tmp->vm_end) {
            vma = vma_tmp;                    //addr may not be in any regions
            if (vma_tmp->vm_start <= addr) //if addr is in this region
                break;
            rb_node = rb_node->rb_left;
        }
        else
            rb_node = rb_node->rb_right;
    }
    if (vma)
        mm->mmap_cache = vma; //update the mmap_cache
    return vma;
}
```

```
struct vm_area_struct *find_vma_intersection(
    struct mm_struct * mm,
    unsigned long start_addr,
    unsigned long end_addr)
{
…
    vma = find_vma(mm, start_addr);
    if (vma && end_addr <= vma->vm_start)
        vma = NULL;
    return vma;
}
```

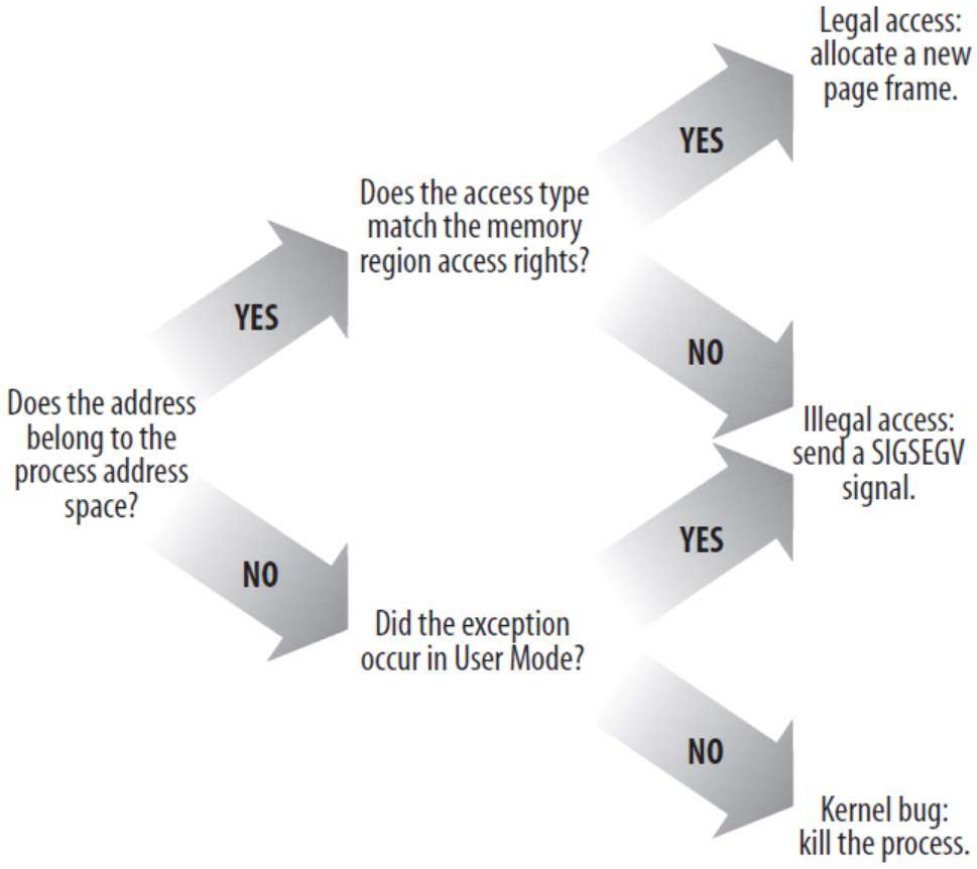
```
unsigned long arch_get_unmapped_area(struct file*, unsigned long addr,
    unsigned long len, unsigned long pgoff, unsigned long flags) {
…
    if (len > TASK_SIZE)  //3GB: user space size
        return -ENOMEM;
    addr = (addr + 0xfff) & 0xfffff000;  //align to 4KB page boundary

    if (addr && addr + len <= TASK_SIZE) {//check the given addr
        vma = find_vma(current->mm, addr);
        if (!vma || addr + len <= vma->vm_start)
            return addr;
    }

    start_addr = addr = mm->free_area_cache; //search
    for (vma = find_vma(current->mm, addr); ; vma = vma->vm_next) {
        if (addr + len > TASK_SIZE) {
            if (start_addr == (TASK_SIZE / 3 + 0xfff) & 0xfffff000)
                return -ENOMEM;
            start_addr = addr = (TASK_SIZE / 3 + 0xfff) & 0xfffff000;
            vma = find_vma(current->mm, addr);
        }
        if (!vma || addr + len <= vma->vm_start) {
            mm->free_area_cache = addr + len;
            return addr;
        }
        addr = vma->vm_end;
    }
…
}
```

Page Fault Exception Handler

- Linux page fault handler must distinguish
  - Programming error
  - Legitimate memory reference that has not been allocated yet

Page Fault Exception Handler

- Overall scheme

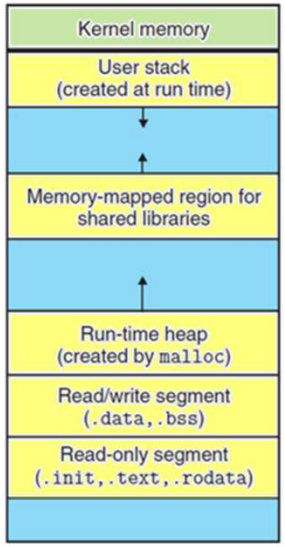


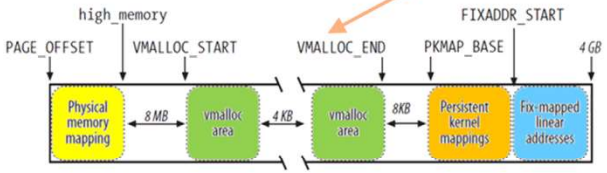
Page Fault Exception Handler

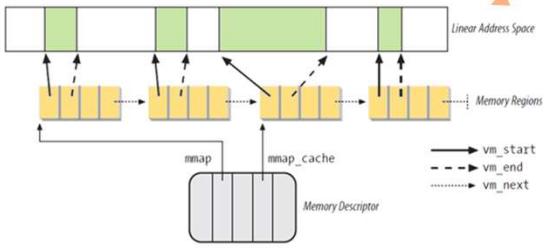
```
void do_page_fault(struct pt_regs *regs,
                   unsigned long error_code)
```

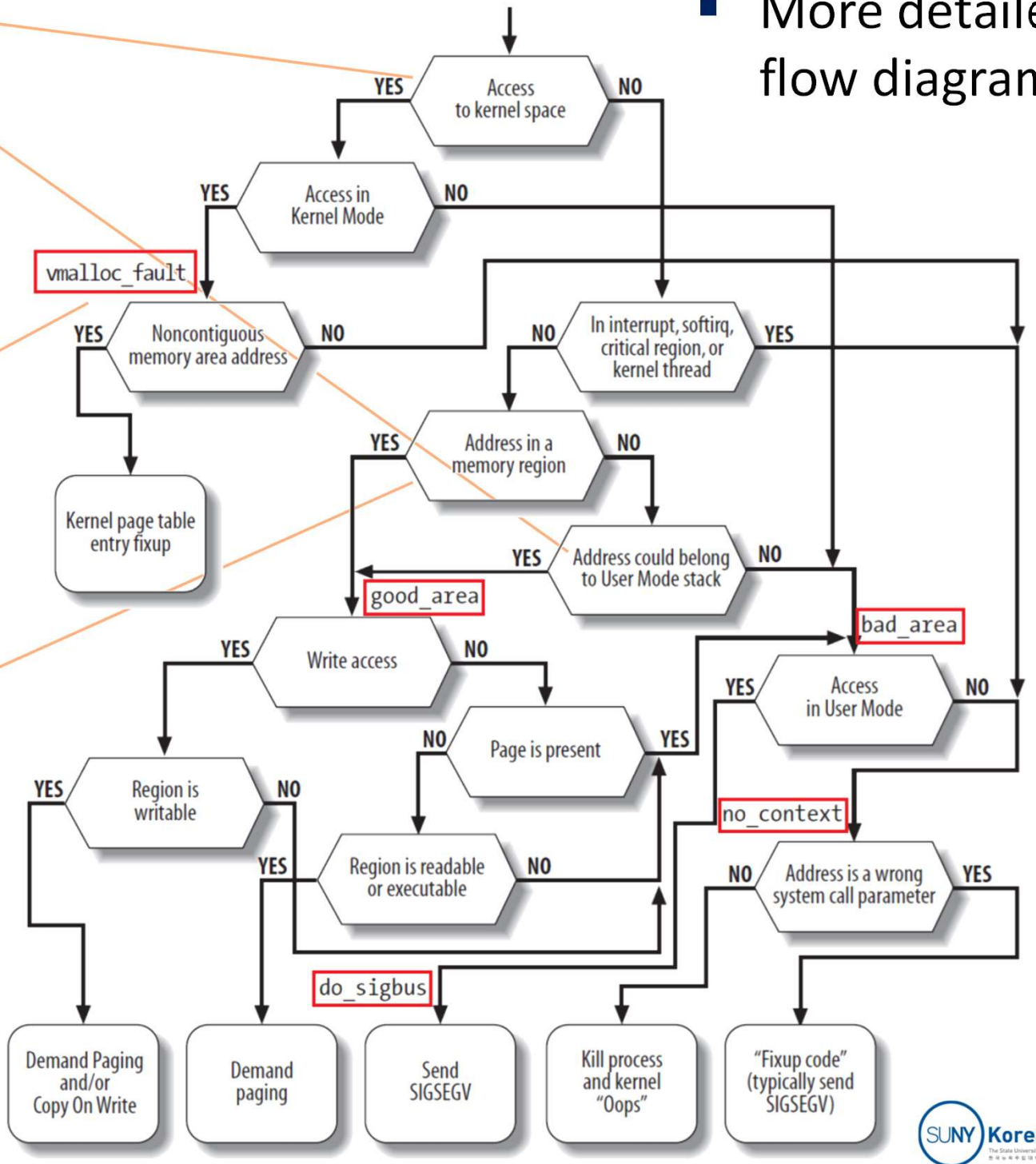
- vmalloc_fault, good_area, bad_area, and no_context are labels within do_page_fault()
- regs: containing the registers when the exception occurred
- error_code:
  - Bit 0: if clear, access to a page that is not present;
    if set, invalid access right
  - Bit 1: if clear, read or execute access;
    if set, write access
  - Bit 2: if clear, in Kernel mode;
    if set, in User mode

- More detailed flow diagram









do_page_fault()

```
/* Access to kernel space? */
info.si_code = SEGV_MAPERR;
if (address >= TASK_SIZE) {
    if (!(error_code & 0x101))
        goto vmalloc_fault;
    goto bad_area_nosemaphore;
}
```

Invalid access right

User mode

```
/* interrupt handler, soft irq, critical region
   or kernel thread? */
if (in_atomic() || !tsk->mm)
    goto bad_area_nosemaphore;
```

do_page_fault()

```
/* address in a memory region? */
vma = find_vma(tsk->mm, address);
if (!vma)
    goto bad_area;
if (vma->vm_start <= address)
    goto good_area;

/* address belongs to a user mode stack? */
if (!(vma->vm_flags & VM_GROWSDOWN))
    goto bad_area;
if (   error_code & 0x100                //user mode?
    && address < regs->esp - 32)    //too far from esp?
    goto bad_area;
if (expand_stack(vma, address))     //can extend the stack?
    goto bad_area;
goto good_area;
```

User mode

Assignment 5

- In this assignment, we explore the process address space of the current process
  - Implement all TODOs to finish implementing mm_regions system call
  - Submit mm_regions.c

- Due date 6/4/2024

```c
//mm_regions.c
#include <linux/syscalls.h>
#include <linux/string.h>
#include <linux/slab.h>
#include <linux/mm_types.h>
#include <linux/fs.h>
#include <linux/dcache.h>
#include <linux/uaccess.h>

#define ONFALSEGOTO(exp, res, label) {\
    if(!(exp)) {\
        (res);\
        goto label;\
    }\
}

//page table walk for user address space
pte_t* getpte(struct mm_struct *mm, unsigned long addr) {
    pgd_t *pgd;
    p4d_t *p4d;
    pud_t *pud;
    pmd_t *pmd;
    pte_t *pte;

    //TODO: return pte for the addr or NULL if it is not available
}
```

```c
static char* area(struct mm_struct *mm, unsigned long addr,
                                        unsigned long flags) {
    char *code  = "code ";
    char *data  = "data ";
    char *heap  = "heap ";
    char *stack = "stack";
    char *none  = "     ";

    //TODO: using mm->start_code, mm->end_code, ... and VM_GROWSDOWN
    //      for the flags, identify the area to which addr belongs.

    return none;
}

static char* str_pte_flags(pte_t *pte) {
    static char flags[] = "----";

    //TODO: update flags[0] and flags[1] to 'w'/'-' and 'x'/'-' respectively
    //      depending on whether the pte is writable or executable.

    return flags;
}
```

```c
static char* str_page_flags(struct page *page) {
    static char flags[] = "---000-      ";

    //TODO: update flags[0] to 'r' or '-' based on whether the page is recently
    //      referenced
    //      update flags[1] to 'd' or '-' based on whether the page is dirty
    //      update flags[2] to 's' or '-' based on whether the page is part of slab
    //      update flags[3..5] to the reference count number of the page
    //      update flags[7..12] to the page frame number (in hexadecimal) of page

    return flags;
}

static char* str_vma_flags(struct vm_area_struct *vma) {
    static char flags[] = "----";

    //TODO: update flags[0] to 'r' or '-' based on whether the vm area is readable
    //      update flags[1] to 'w' or '-' based on whether the vm area is writable
    //      update flags[2] to 'x' or '-' based on whether the vm area is executable
    //      update flags[3] to 'l' or '-' based on whether the vm area is locked

    return flags;
}
```

```
static int fillbuf(struct mm_struct *mm, char *buf, int buflen) {
    char tmp[100];
    int len, left = buflen;
    struct vm_area_struct *vma;
    pte_t *pte;
    struct page *page;
    //TODO: set vma to the first vm area

    while(vma) {
        len = sprintf(tmp, "0x%lx-0x%lx: %s, ",
            vma->vm_start, vma->vm_end, area(mm, vma->vm_start, vma->vm_flags));
        if (len >= left) break;
        strcpy(buf, tmp); buf += len; left -= len;

        len = sprintf(tmp, "vma flags: %s, ", str_vma_flags(vma));
        if (len >= left) break;
        strcpy(buf, tmp); buf += len; left -= len;

        pte = getpte(mm, vma->vm_start); //virt_to_page works only for kernel area
        if (pte != NULL)
            len = sprintf(tmp, "pte flags: %s, ", str_pte_flags(pte));
        else
            len = sprintf(tmp, "pte flags: NA  , ");
        if (len >= left) break;
        strcpy(buf, tmp); buf += len; left -= len;
…
```

```
…
        page = (pte != NULL) ? pte_page(*pte) : NULL;
        if (page != NULL)
            len = sprintf(tmp, "page flag: %s, ", str_page_flags(page));
        else
            len = sprintf(tmp, "page flag: NA           , ");
        if (len >= left) break;
        strcpy(buf, tmp); buf += len; left -= len;

        if (vma->vm_file)
            len = sprintf(tmp, "file: %s\n", vma->vm_file->f_path.dentry->d_iname);
        else
            len = sprintf(tmp, "file: none\n");
        if (len >= left) break;
        strcpy(buf, tmp); buf += len; left -= len;

        //TODO: set vma to the next vm area
    }

    return buflen - left + 1;
}
```

```c
SYSCALL_DEFINE2(mm_regions, char*, buf, int, buflen) {
    long err = 0;
    char *kbuf = NULL;
    int len;

    ONFALSEGOTO(buflen <= 4096, err = -EINVAL, out);

    //TODO: implement mm_regions

out:
    if(kbuf)
        kfree(kbuf);

    return err;
}
```

```c
//mm_regions_user.c
//user space program
//
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>
#include "wrapper.h"

//a global variable
int g_i = 1;

void* alloc(void *addr, int prot) {
    return mmap(addr, 1, prot, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
}

void area_info() {
    char buf[4096];
    long res;
    res = mm_regions(buf, sizeof(buf));
    if (res) {
        printf("error: mm_regions: %ld\n", res);
        exit(0);
    }
    printf("%s\n", buf);
}
```

```
void test() {
    char *p = (void*)alloc(NULL, PROT_READ | PROT_WRITE);
    char *q = (void*)alloc(NULL, PROT_WRITE);
    char *r = (void*)alloc(NULL, PROT_READ);

    printf("parent:\np(rw) = %p\nq(w)  = %p\nr(r)  = %p\n", p, q, r);
    area_info();

    printf("parent: p[0] = 1;\n");
    p[0] = 1;
    area_info();

    sleep(1);
    if (fork() == 0) {
        printf("child: p[0] = q[0] = 1;\n");
        p[0] = q[0] = 1;
        area_info();
        exit(0);
    }

    sleep(1);
    printf("parent: p[0] = r[0];\n");
    p[0] = r[0];
    area_info();
}
```

```c
int main() {
    test();
}
```

```
parent:
p(rw) = 0x7f26a98bf000
q(w)  = 0x7f26a9892000
r(r)  = 0x7f26a9891000
0x55816d35c000-0x55816d35d000: code , vma flags: r---, pte flags: ----, page flag: ---002-1078e9, file: a.out
0x55816d35d000-0x55816d35e000: code , vma flags: r-x-, pte flags: -x--, page flag: ---002-1078ea, file: a.out
0x55816d35e000-0x55816d35f000:      , vma flags: r---, pte flags: ----, page flag: ---002-1078eb, file: a.out
0x55816d35f000-0x55816d360000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c8b, file: a.out
0x55816d360000-0x55816d361000: data , vma flags: rw--, pte flags: w---, page flag: ---001-125bf5, file: a.out
0x55816e44c000-0x55816e46d000: heap , vma flags: rw--, pte flags: w---, page flag: ---002-133bfc, file: none
0x7f26a968e000-0x7f26a96b3000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a96b3000-0x7f26a982b000:      , vma flags: r-x-, pte flags: -x--, page flag: r--092-13593b, file: libc-2.31.so
0x7f26a982b000-0x7f26a9875000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9875000-0x7f26a9876000:      , vma flags: ----, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9876000-0x7f26a9879000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c85, file: libc-2.31.so
0x7f26a9879000-0x7f26a987c000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1257bf, file: libc-2.31.so
0x7f26a987c000-0x7f26a9882000:      , vma flags: rw--, pte flags: w---, page flag: ---002-1078e7, file: none
0x7f26a9891000-0x7f26a9892000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7f26a9892000-0x7f26a9893000:      , vma flags: -w--, pte flags: NA  , page flag: NA           , file: none
0x7f26a9893000-0x7f26a9894000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359ed, file: ld-2.31.so
0x7f26a9894000-0x7f26a98b7000:      , vma flags: r-x-, pte flags: -x--, page flag: r--087-1359ec, file: ld-2.31.so
0x7f26a98b7000-0x7f26a98bf000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359cf, file: ld-2.31.so
0x7f26a98bf000-0x7f26a98c0000:      , vma flags: rw--, pte flags: NA  , page flag: NA           , file: none
0x7f26a98c0000-0x7f26a98c1000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c84, file: ld-2.31.so
0x7f26a98c1000-0x7f26a98c2000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1078e5, file: ld-2.31.so
0x7f26a98c2000-0x7f26a98c3000:      , vma flags: rw--, pte flags: w---, page flag: ---001-125475, file: none
0x7ffc851b5000-0x7ffc851d6000: stack, vma flags: rw--, pte flags: NA  , page flag: NA           , file: none
0x7ffc851f9000-0x7ffc851fc000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7ffc851fc000-0x7ffc851fd000:      , vma flags: r-x-, pte flags: -x--, page flag: ---089-1358b7, file: none
```

```
parent: p[0] = 1;
0x55816d35c000-0x55816d35d000: code , vma flags: r---, pte flags: ----, page flag: ---002-1078e9, file: a.out
0x55816d35d000-0x55816d35e000: code , vma flags: r-x-, pte flags: -x--, page flag: ---002-1078ea, file: a.out
0x55816d35e000-0x55816d35f000:      , vma flags: r---, pte flags: ----, page flag: ---002-1078eb, file: a.out
0x55816d35f000-0x55816d360000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c8b, file: a.out
0x55816d360000-0x55816d361000: data , vma flags: rw--, pte flags: w---, page flag: ---001-125bf5, file: a.out
0x55816e44c000-0x55816e46d000: heap , vma flags: rw--, pte flags: w---, page flag: ---002-133bfc, file: none
0x7f26a968e000-0x7f26a96b3000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a96b3000-0x7f26a982b000:      , vma flags: r-x-, pte flags: -x--, page flag: r--092-13593b, file: libc-2.31.so
0x7f26a982b000-0x7f26a9875000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9875000-0x7f26a9876000:      , vma flags: ----, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9876000-0x7f26a9879000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c85, file: libc-2.31.so
0x7f26a9879000-0x7f26a987c000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1257bf, file: libc-2.31.so
0x7f26a987c000-0x7f26a9882000:      , vma flags: rw--, pte flags: w---, page flag: ---002-1078e7, file: none
0x7f26a9891000-0x7f26a9892000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7f26a9892000-0x7f26a9893000:      , vma flags: -w--, pte flags: NA  , page flag: NA           , file: none
0x7f26a9893000-0x7f26a9894000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359ed, file: ld-2.31.so
0x7f26a9894000-0x7f26a98b7000:      , vma flags: r-x-, pte flags: -x--, page flag: r--087-1359ec, file: ld-2.31.so
0x7f26a98b7000-0x7f26a98bf000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359cf, file: ld-2.31.so
0x7f26a98bf000-0x7f26a98c0000:      , vma flags: rw--, pte flags: w---, page flag: ---002-133c59, file: none
0x7f26a98c0000-0x7f26a98c1000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c84, file: ld-2.31.so
0x7f26a98c1000-0x7f26a98c2000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1078e5, file: ld-2.31.so
0x7f26a98c2000-0x7f26a98c3000:      , vma flags: rw--, pte flags: w---, page flag: ---001-125475, file: none
0x7ffc851b5000-0x7ffc851d6000: stack, vma flags: rw--, pte flags: NA  , page flag: NA           , file: none
0x7ffc851f9000-0x7ffc851fc000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7ffc851fc000-0x7ffc851fd000:      , vma flags: r-x-, pte flags: -x--, page flag: ---089-1358b7, file: none
```

```
child: p[0] = q[0] = 1;
0x55816d35c000-0x55816d35d000: code , vma flags: r---, pte flags: NA  , page flag: NA           , file: a.out
0x55816d35d000-0x55816d35e000: code , vma flags: r-x-, pte flags: -x--, page flag: ---003-1078ea, file: a.out
0x55816d35e000-0x55816d35f000:      , vma flags: r---, pte flags: ----, page flag: ---003-1078eb, file: a.out
0x55816d35f000-0x55816d360000:      , vma flags: r---, pte flags: ----, page flag: ---002-133c8b, file: a.out
0x55816d360000-0x55816d361000: data , vma flags: rw--, pte flags: ----, page flag: ---002-125bf5, file: a.out
0x55816e44c000-0x55816e46d000: heap , vma flags: rw--, pte flags: w---, page flag: ---001-1078ee, file: none
0x7f26a968e000-0x7f26a96b3000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a96b3000-0x7f26a982b000:      , vma flags: r-x-, pte flags: -x--, page flag: r--093-13593b, file: libc-2.31.so
0x7f26a982b000-0x7f26a9875000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9875000-0x7f26a9876000:      , vma flags: ----, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9876000-0x7f26a9879000:      , vma flags: r---, pte flags: ----, page flag: ---002-133c85, file: libc-2.31.so
0x7f26a9879000-0x7f26a987c000:      , vma flags: rw--, pte flags: ----, page flag: ---002-1257bf, file: libc-2.31.so
0x7f26a987c000-0x7f26a9882000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1078e4, file: none
0x7f26a9891000-0x7f26a9892000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7f26a9892000-0x7f26a9893000:      , vma flags: -w--, pte flags: w---, page flag: ---001-1078f0, file: none
0x7f26a9893000-0x7f26a9894000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: ld-2.31.so
0x7f26a9894000-0x7f26a98b7000:      , vma flags: r-x-, pte flags: NA  , page flag: NA           , file: ld-2.31.so
0x7f26a98b7000-0x7f26a98bf000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: ld-2.31.so
0x7f26a98bf000-0x7f26a98c0000:      , vma flags: rw--, pte flags: w---, page flag: ---002-1078f1, file: none
0x7f26a98c0000-0x7f26a98c1000:      , vma flags: r---, pte flags: ----, page flag: ---002-133c84, file: ld-2.31.so
0x7f26a98c1000-0x7f26a98c2000:      , vma flags: rw--, pte flags: w---, page flag: ---001-133d32, file: ld-2.31.so
0x7f26a98c2000-0x7f26a98c3000:      , vma flags: rw--, pte flags: ----, page flag: ---002-125475, file: none
0x7ffc851b5000-0x7ffc851d6000: stack, vma flags: rw--, pte flags: NA  , page flag: NA           , file: none
0x7ffc851f9000-0x7ffc851fc000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7ffc851fc000-0x7ffc851fd000:      , vma flags: r-x-, pte flags: NA  , page flag: NA           , file: none
```

```
parent: p[0] = r[0];
0x55816d35c000-0x55816d35d000: code , vma flags: r---, pte flags: ----, page flag: ---002-1078e9, file: a.out
0x55816d35d000-0x55816d35e000: code , vma flags: r-x-, pte flags: -x--, page flag: r--002-1078ea, file: a.out
0x55816d35e000-0x55816d35f000:      , vma flags: r---, pte flags: ----, page flag: r--002-1078eb, file: a.out
0x55816d35f000-0x55816d360000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c8b, file: a.out
0x55816d360000-0x55816d361000: data , vma flags: rw--, pte flags: ----, page flag: ---001-125bf5, file: a.out
0x55816e44c000-0x55816e46d000: heap , vma flags: rw--, pte flags: w---, page flag: ---001-133bfc, file: none
0x7f26a968e000-0x7f26a96b3000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a96b3000-0x7f26a982b000:      , vma flags: r-x-, pte flags: -x--, page flag: r--092-13593b, file: libc-2.31.so
0x7f26a982b000-0x7f26a9875000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9875000-0x7f26a9876000:      , vma flags: ----, pte flags: NA  , page flag: NA           , file: libc-2.31.so
0x7f26a9876000-0x7f26a9879000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c85, file: libc-2.31.so
0x7f26a9879000-0x7f26a987c000:      , vma flags: rw--, pte flags: ----, page flag: ---001-1257bf, file: libc-2.31.so
0x7f26a987c000-0x7f26a9882000:      , vma flags: rw--, pte flags: w---, page flag: ---001-1078e7, file: none
0x7f26a9891000-0x7f26a9892000:      , vma flags: r---, pte flags: ----, page flag: ---001-1360df, file: none
0x7f26a9892000-0x7f26a9893000:      , vma flags: -w--, pte flags: NA  , page flag: NA           , file: none
0x7f26a9893000-0x7f26a9894000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359ed, file: ld-2.31.so
0x7f26a9894000-0x7f26a98b7000:      , vma flags: r-x-, pte flags: -x--, page flag: r--087-1359ec, file: ld-2.31.so
0x7f26a98b7000-0x7f26a98bf000:      , vma flags: r---, pte flags: ----, page flag: r--087-1359cf, file: ld-2.31.so
0x7f26a98bf000-0x7f26a98c0000:      , vma flags: rw--, pte flags: w---, page flag: ---001-133c59, file: none
0x7f26a98c0000-0x7f26a98c1000:      , vma flags: r---, pte flags: ----, page flag: ---001-133c84, file: ld-2.31.so
0x7f26a98c1000-0x7f26a98c2000:      , vma flags: rw--, pte flags: ----, page flag: ---001-1078e5, file: ld-2.31.so
0x7f26a98c2000-0x7f26a98c3000:      , vma flags: rw--, pte flags: ----, page flag: ---001-125475, file: none
0x7ffc851b5000-0x7ffc851d6000: stack, vma flags: rw--, pte flags: NA  , page flag: NA           , file: none
0x7ffc851f9000-0x7ffc851fc000:      , vma flags: r---, pte flags: NA  , page flag: NA           , file: none
0x7ffc851fc000-0x7ffc851fd000:      , vma flags: r-x-, pte flags: -x--, page flag: ---089-1358b7, file: none
```