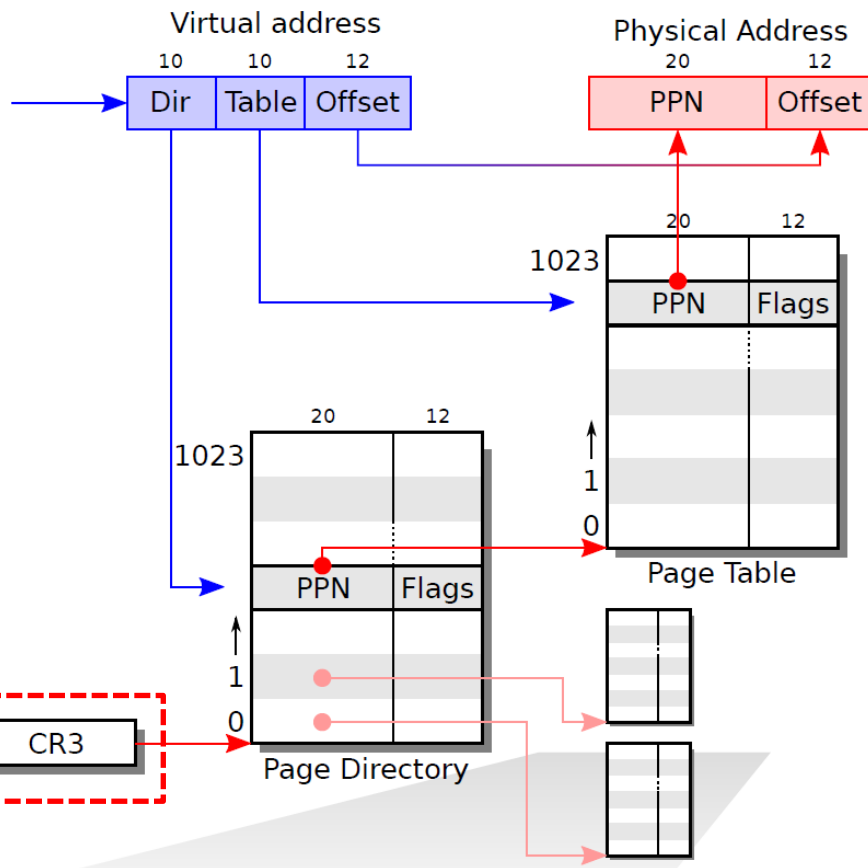


# Project #3: Stack Growth

**Instructor:** Sungyong Ahn

# Page Table Hardware in x86



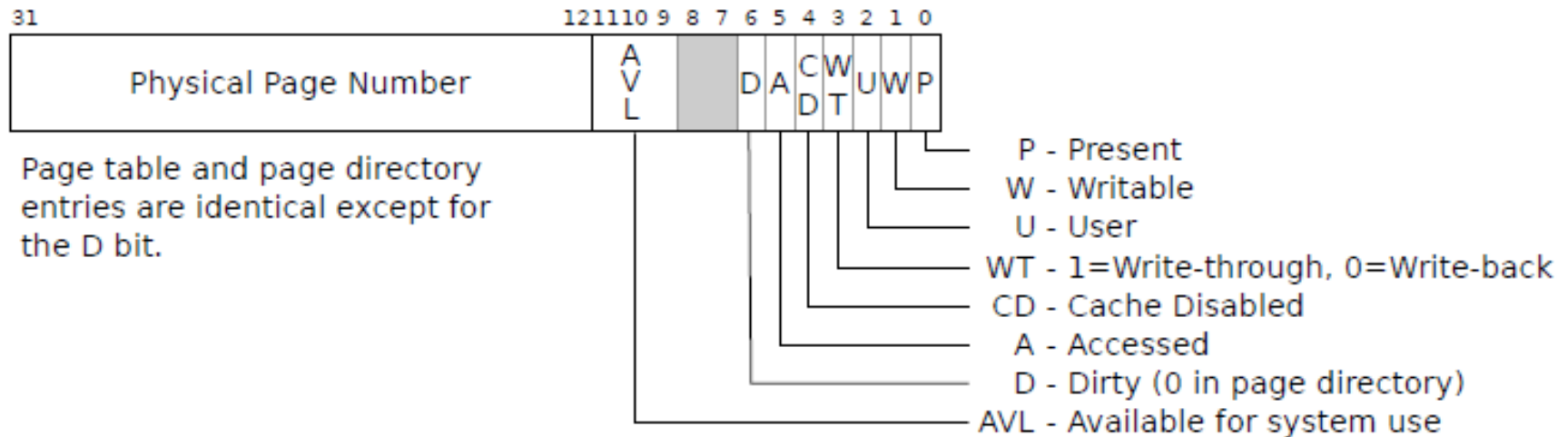
## ■ 32bit address space

- $2^{20}$  (1,048,576) page table entries (PTEs)
- 20-bit physical page number (PPN)

## ■ Two-level page table

- Page directory has 1024 references to page table pages
- Next 10 bits of the virtual address to select a PTE from the page table

# Page Table Hardware in x86 (Cont'd)



- PTE\_P indicates whether the PTE is present
- PTE\_W controls whether instructions are allowed to issue writes to the page
- PTE\_U controls whether user programs are allowed to use the page

# Virtual Address Space in xv6

- Layout of the virtual address space and the physical memory.

main.c

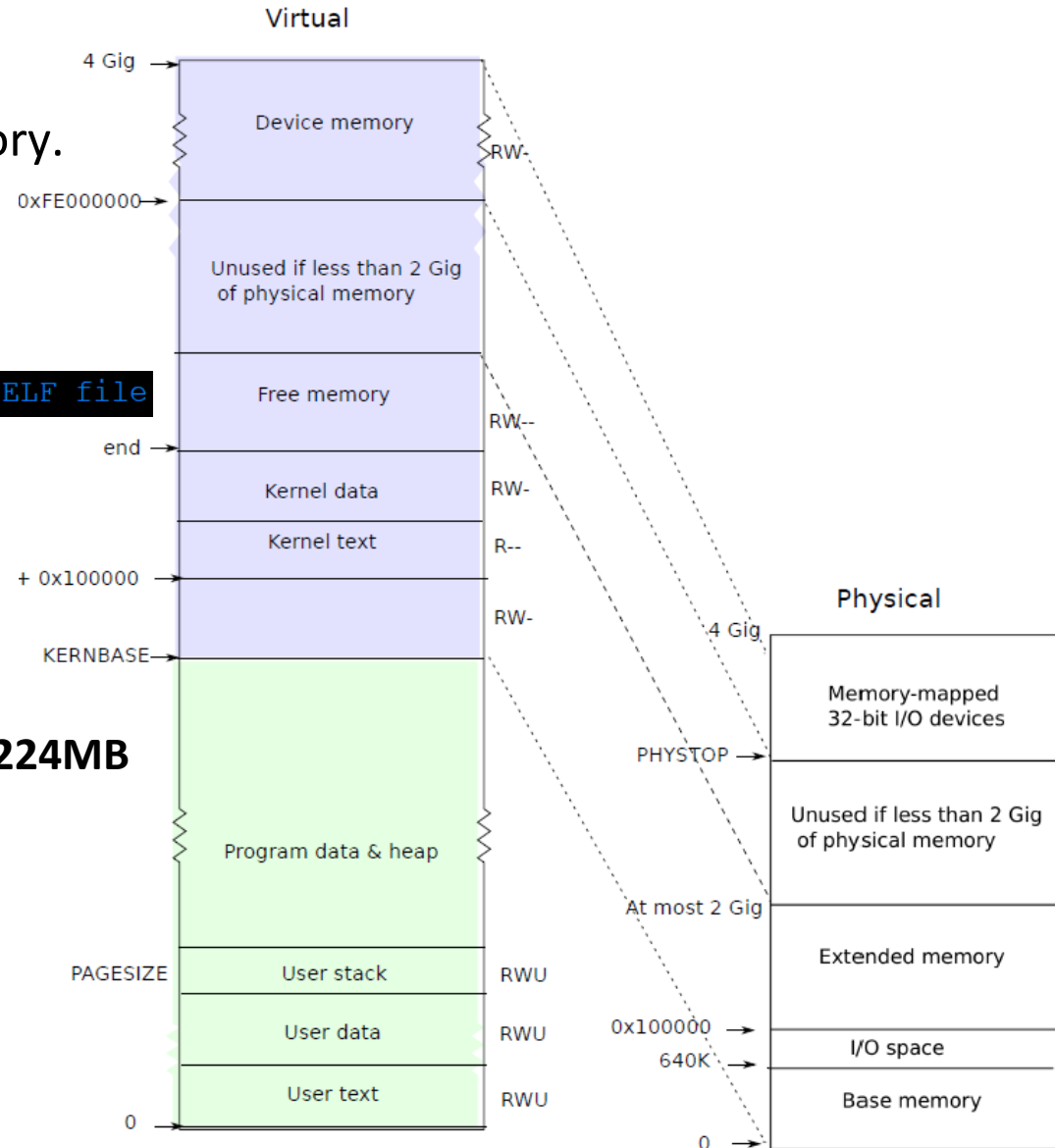
```
extern char end[];
// first address after kernel loaded from ELF file
```

memlayout.h

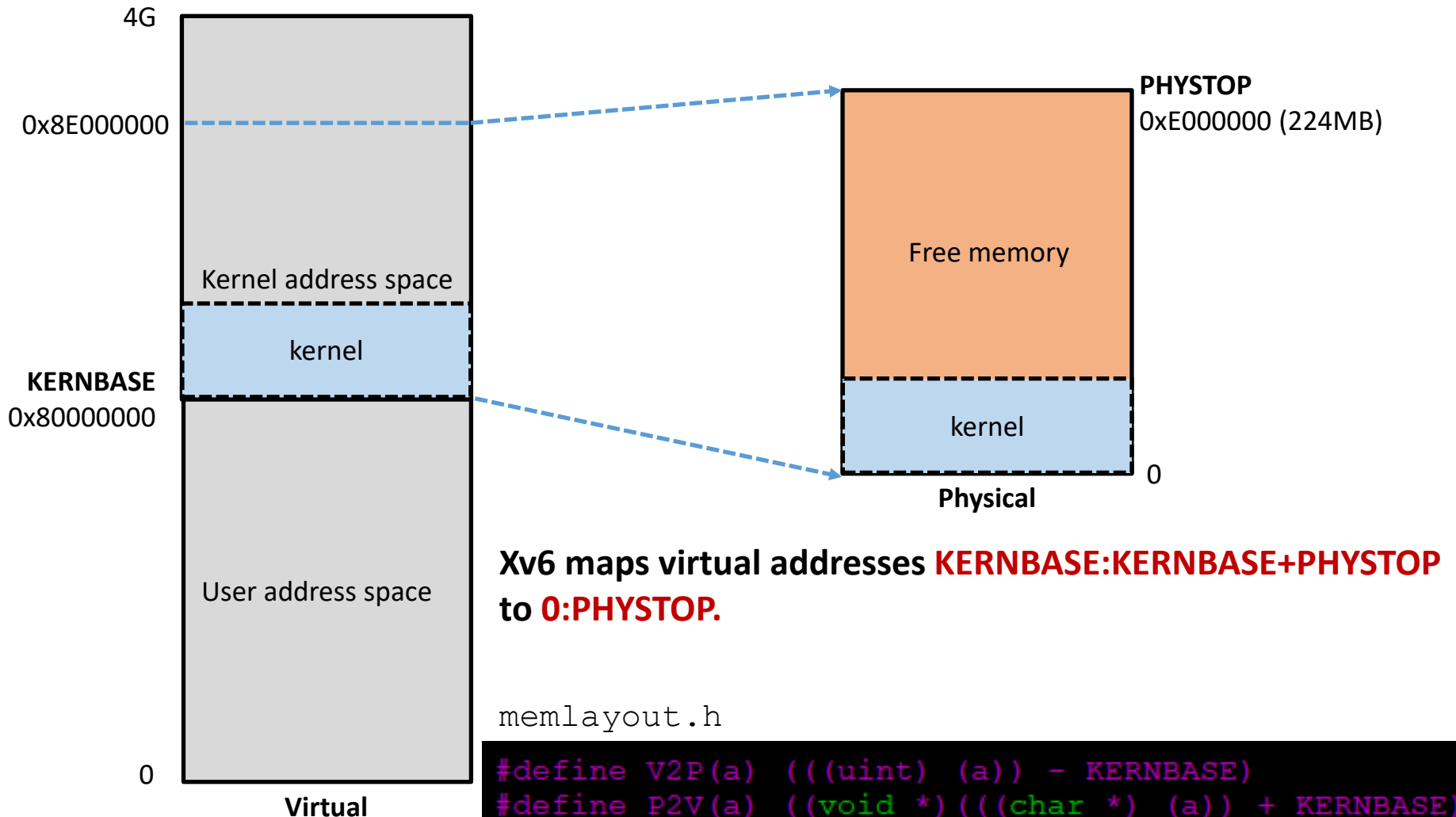
```
#define KERNBASE 0x80000000
#define PHYSTOP 0xE0000000
```

2GB

224MB



# Virtual Address Space in xv6



# proc.h

types.h

```
typedef uint pde_t;
```

```
// Per-process state
struct proc {
    uint sz;                // Size of process memory (bytes)
    pde_t* pgdir;           // Page table
    char *kstack;           // Bottom of kernel stack for this process
    enum procstate state;   // Process state
    int pid;                // Process ID
    struct proc *parent;    // Parent process
    struct trapframe *tf;   // Trap frame for current syscall
    struct context *context; // swtch() here to run process
    void *chan;             // If non-zero, sleeping on chan
    int killed;             // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd;      // Current directory
    char name[16];          // Process name (debugging)
    int nice;
};
```

# main.c

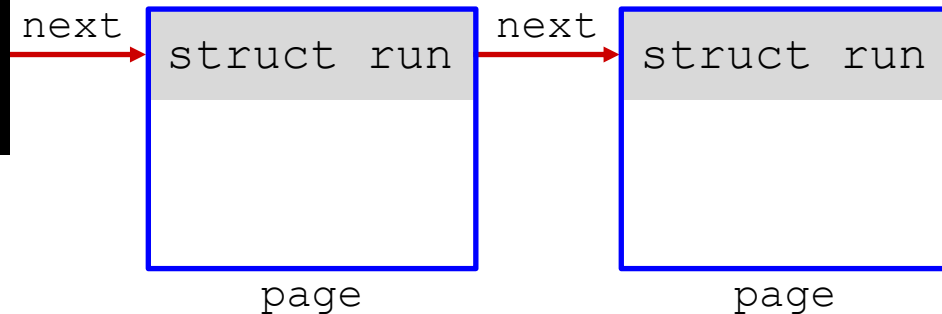
```
int
main(void)
{
    kinit1(end, P2V(4*1024*1024)); // phys page allocator
    kvmalloc(); // kernel page table
    mpinit(); // detect other processors
    lapicinit(); // interrupt controller
    seginit(); // segment descriptors
    picinit(); // disable pic
    ioapicinit(); // another interrupt controller
    consoleinit(); // console hardware
    uartinit(); // serial port
    pinit(); // process table
    tvinit(); // trap vectors
    binit(); // buffer cache
    fileinit(); // file table
    ideinit(); // disk
    startothers(); // start other processors
    kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
    userinit(); // first user process
    mpmain(); // finish this processor's setup
}
```

# Initialization free physical page's list

## ■ kinit1(), kinit2()

### ■ kmem.freelist

```
struct run {  
    struct run *next;  
};  
  
struct {  
    struct spinlock lock;  
    int use_lock;  
    struct run *freelist;  
} kmem;
```





# Creating a page table

```
// Set up kernel part of a page table.
```

```
pde_t*
```

```
setupkvm(void)
```

```
{
```

```
    pde_t *pgdir;
```

```
    struct kmap *k;
```

```
    if((pgdir = (pde_t*)kalloc()) == 0)
```

```
        return 0;
```

```
    memset(pgdir, 0, PGSIZE);
```

```
    if (P2V(PHYSTOP) > (void*)DEVSPACE)
```

```
        panic("PHYSTOP too high");
```

```
    for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
```

```
        if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
                    (uint)k->phys_start, k->perm) < 0) {
```

```
            freevm(pgdir);
```

```
            return 0;
```

```
        }
```

```
    return pgdir;
```

```
}
```

page directory 생성 (1페이지 크기)

주어진 범위만큼의 virtual page number를  
physical frame number로 매핑하는 page table  
entry 생성

# Creating a page table

```
static int
mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
{
    char *a, *last;
    pte_t *pte;

    a = (char*) PGROUNDDOWN((uint)va);
    last = (char*) PGROUNDDOWN(((uint)va) + size - 1);
    for(;;){
        if((pte = walkpgdir(pgdir, a, 1)) == 0)
            return -1;
        if(*pte & PTE_P)
            panic("remap");
        *pte = pa | perm | PTE_P;
        if(a == last)
            break;
        a += PGSIZE;
        pa += PGSIZE;
    }
    return 0;
}
```

원하는 virtual page의 PTE를 반환

찾은 PTE에 physical address 및  
flag를 설정

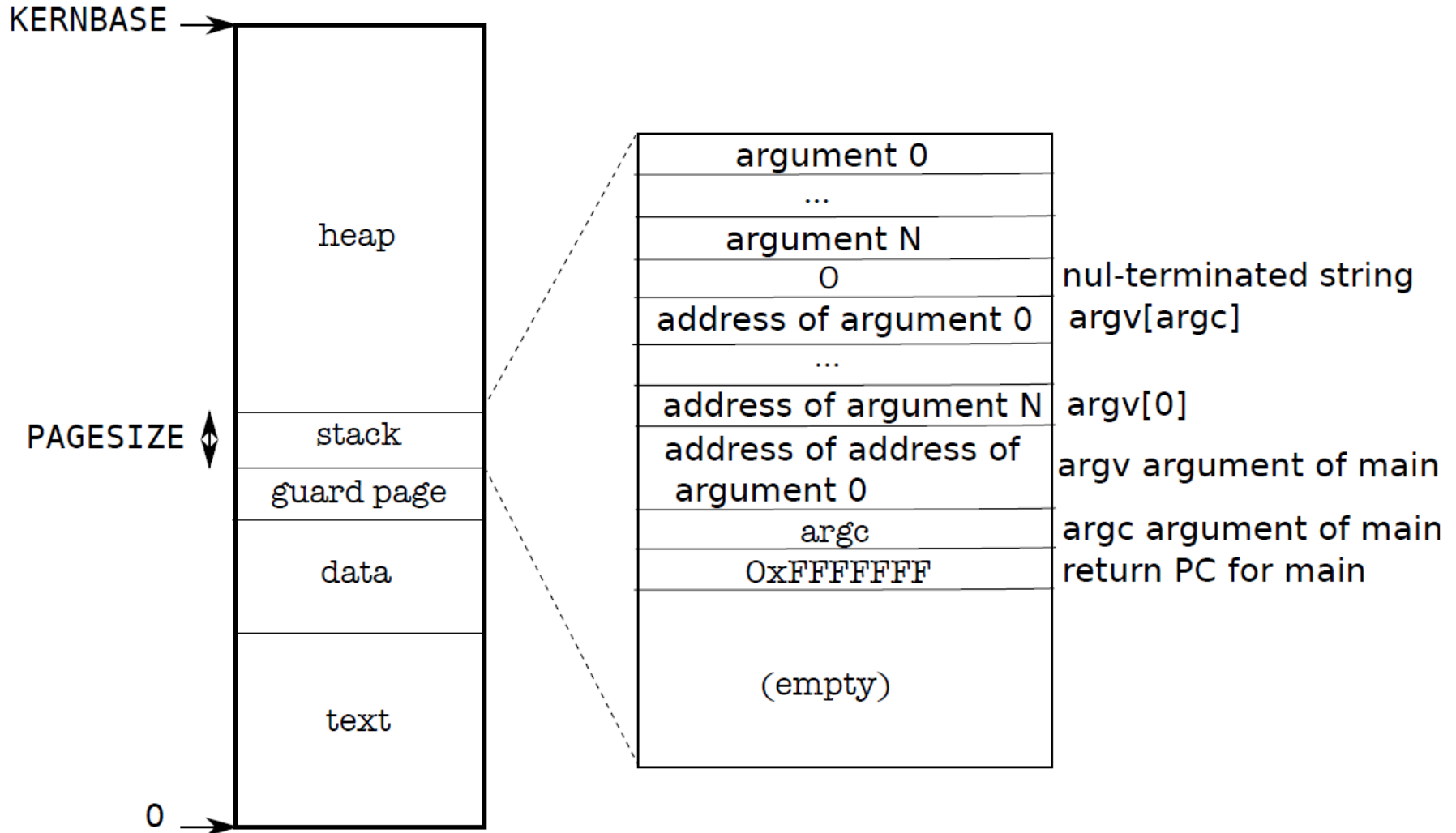
범위에 있는 다음 페이지로 이동

# Creating a page table

```
static pte_t *
walkpgdir(pde_t *pgdir, const void *va, int alloc)
{
    pde_t *pde;
    pte_t *pgtab;

    pde = &pgdir[PDX(va)];
    if(*pde & PTE_P){
        pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
    } else {
        if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
            return 0;
        // Make sure all those PTE_P bits are zero.
        memset(pgtab, 0, PGSIZE);
        // The permissions here are overly generous, but they can
        // be further restricted by the permissions in the page table
        // entries, if necessary.
        *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
    }
    return &pgtab[PTX(va)];
}
```

# User address space in xv6 (exec)



**Figure 2-3.** Memory layout of a user process with its initial stack.

# User address space in xv6

■ `int exec(char *path, char **argv)`

```
// Load program into memory.
sz = 0;
for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){
    if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
        goto bad;
    if(ph.type != ELF_PROG_LOAD)
        continue;
    if(ph.memsz < ph.filesz)
        goto bad;
    if(ph.vaddr + ph.memsz < ph.vaddr)
        goto bad;
    if( (sz = allocuvm(pgdir, sz, ph.vaddr + ph.memsz)) == 0)
        goto bad;
    if(ph.vaddr % PGSIZE != 0)
        goto bad;
    if( (loaduvm(pgdir, (char*)ph.vaddr, ip, ph.off, ph.filesz) < 0)
        goto bad;
}
iunlockput(ip);
end_op();
ip = 0;
```

프로그램을 읽어오는데 필요한 물리 메모리를 할당하고 PTE를 생성한다.

파일로부터 프로그램 이미지를 메모리로 읽어온다.

# User address space in xv6

## ■ `int exec(char *path, char **argv)`

```
// Allocate two pages at the next page boundary.
// Make the first inaccessible. Use the second as the user stack.
sz = PGROUNDUP(sz);
if((sz = allocvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
    goto bad;
clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
sp = sz;
```

- 프로그램 이미지 위에 stack을 위한 페이지 할당
- 두 개의 페이지를 할당하지만 guard page는 사용하지 않음
  - `clearpteu()`는 guard page를 inaccessible하게 만들기 위한 함수

```
curproc->pgdir = pgdir;
curproc->sz = sz;
curproc->tf->eip = elf.entry; // main
curproc->tf->esp = sp;
```

새로 생성된 프로세스 이미지 크기를  
proc 구조체에 저장

# User address space in xv6

## ■ allocvm()

- Allocate page tables and physical memory to grow process

```
int
allocvm(pde_t *pgdir, uint oldsz, uint newsz)
{
    char *mem;
    uint a;

    if(newsz >= KERNBASE)
        return 0;
    if(newsz < oldsz)
        return oldsz;

    a = PGROUNDUP(oldsz);
    for(; a < newsz; a += PGSIZE){
        mem = kalloc();
        if(mem == 0){
            cprintf("allocvm out of memory\n");
            deallocvm(pgdir, newsz, oldsz);
            return 0;
        }
        memset(mem, 0, PGSIZE);
        if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
            cprintf("allocvm out of memory (2)\n");
            deallocvm(pgdir, newsz, oldsz);
            kfree(mem);
            return 0;
        }
    }
    return newsz;
}
```

# User address space in xv6

## ■ sbrk system call

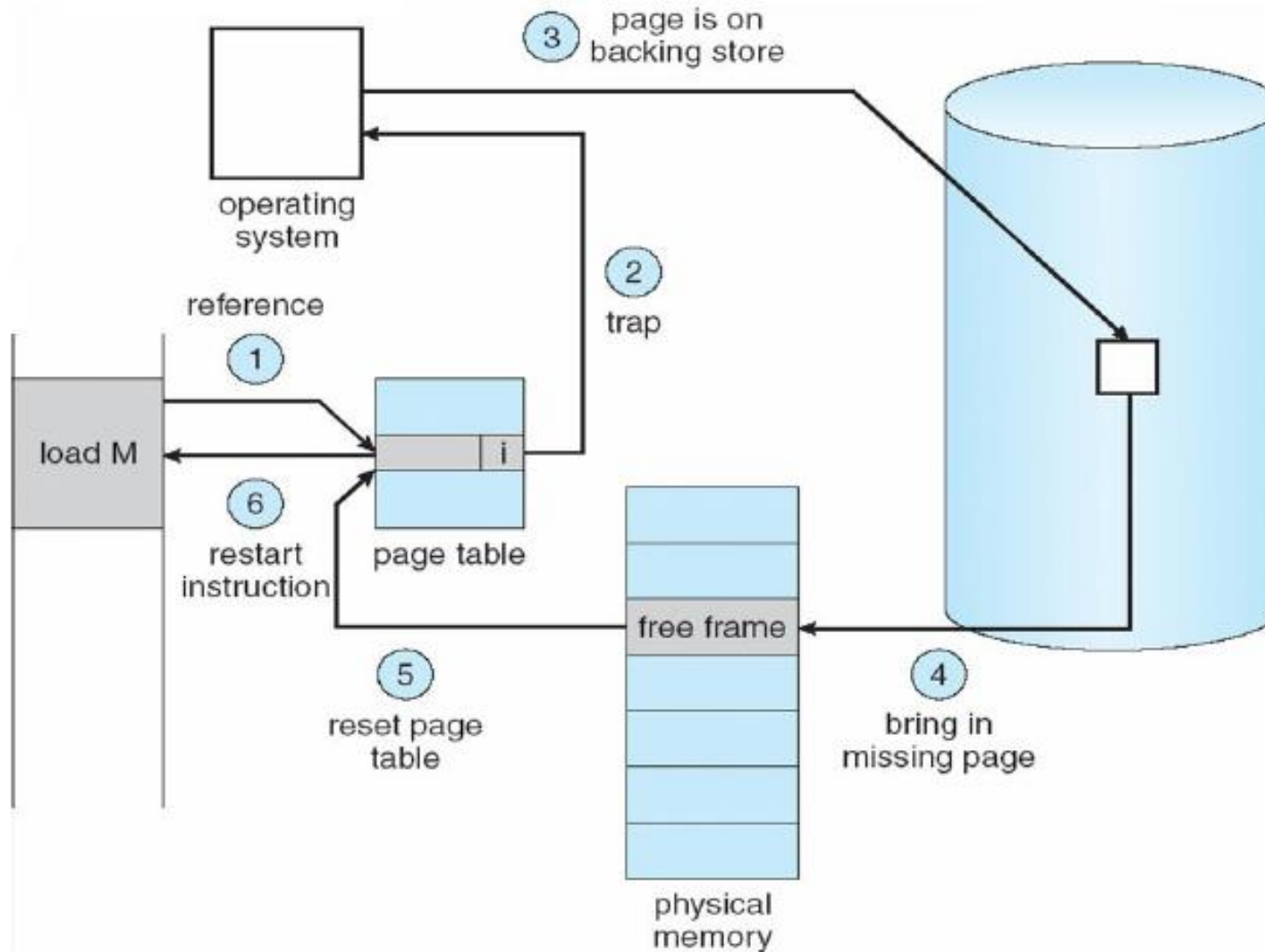
- Shrink or grow process's memory (Heap)
- `growproc()`

```
// Grow current process's memory by n bytes.
// Return 0 on success, -1 on failure.
int
growproc(int n)
{
    uint sz;
    struct proc *curproc = myproc();

    sz = curproc->sz;
    if(n > 0){
        if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
            return -1;
    } else if(n < 0){
        if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
            return -1;
    }
    curproc->sz = sz;
    switchuvm(curproc);
    return 0;
}
```



# Handling Page Faults



# Page Fault Exception in Intel x86

## ■ Conditions

- There is no translation for the linear address
- There is a translation for the linear address, but its access rights do not permit the access

■ **CR2** stores the linear address that caused a page fault

■ **CR3** stores the physical address of Page directory

■ Processor triggers interrupt **14** (page fault)

# Page fault handler in xv6

- If page fault occurs, “trapno” of trapframe automatically filled with T\_PGFLT and call trap function in trap.c

```
// x86 trap and interrupt constants.

// Processor-defined:
#define T_DIVIDE      0      // divide error
#define T_DEBUG      1      // debug exception
#define T_NMI        2      // non-maskable interrupt
#define T_BRKPT      3      // breakpoint
#define T_OFLOW      4      // overflow
#define T_BOUND      5      // bounds check
#define T_ILLOP      6      // illegal opcode
#define T_DEVICE      7      // device not available
#define T_DBLFLT      8      // double fault
// #define T_COPROC    9      // reserved (not used since 486)
#define T_TSS        10     // invalid task switch segment
#define T_SEGNP      11     // segment not present
#define T_STACK      12     // stack exception
#define T_GPFLT      13     // general protection fault
#define T_PGFLT      14     // page fault
// #define T_RES       15     // reserved
#define T_FPERR      16     // floating point error
#define T_ALIGN      17     // alignment check
```

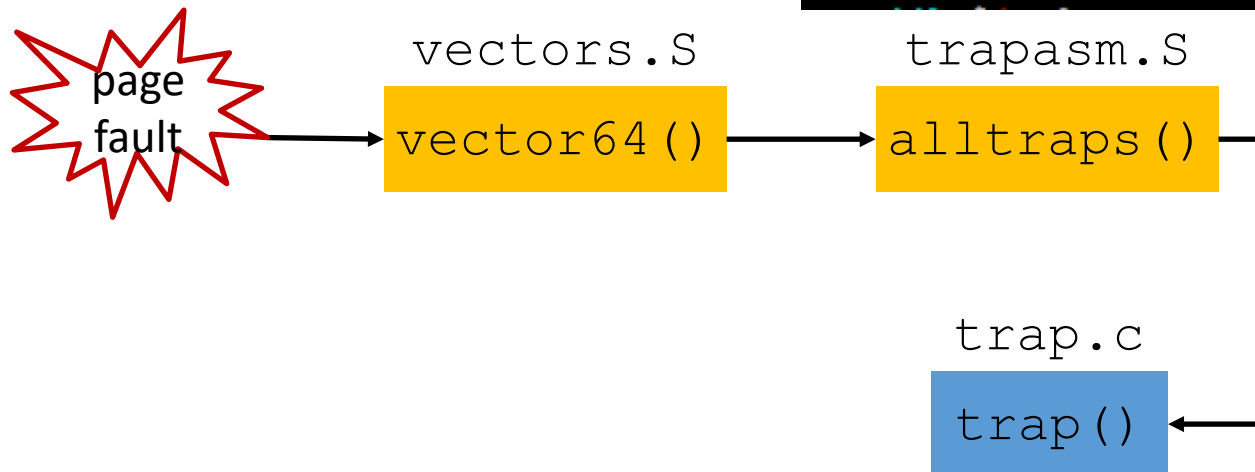
# Page fault handler in xv6

```
.globl vector14
vector14:
    pushl $14
    jmp alltraps
```

```
.globl alltraps
alltraps:
    # Build trap frame.
    pushl %ds
    pushl %es
    pushl %fs
    pushl %gs
    pushal

    # Set up data segments.
    movw $(SEG_KDATA<<3), %ax
    movw %ax, %ds
    movw %ax, %es

    # Call trap(tf), where tf=%esp
    pushl %esp
    call trap
```



# Page fault handler in xv6

## ■ trap.c

- trap()
- You have to make your “own” page fault handler!
- Currently, implemented as below...
  - rcr2() -> page fault address

```
//PAGEBREAK: 13
default:
    if(myproc() == 0 || (tf->cs&3) == 0){
        // In kernel, it must be our mistake.
        cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
            tf->trapno, cpuid(), tf->eip, rcr2());
        panic("trap");
    }
    // In user space, assume process misbehaved.
    cprintf("pid %d %s: trap %d err %d on cpu %d "
        "eip 0x%x addr 0x%x--kill proc\n",
        myproc()->pid, myproc()->name, tf->trapno,
        tf->err, cpuid(), tf->eip, rcr2());
    myproc()->killed = 1;
}
```

# Project #3. Stack growth

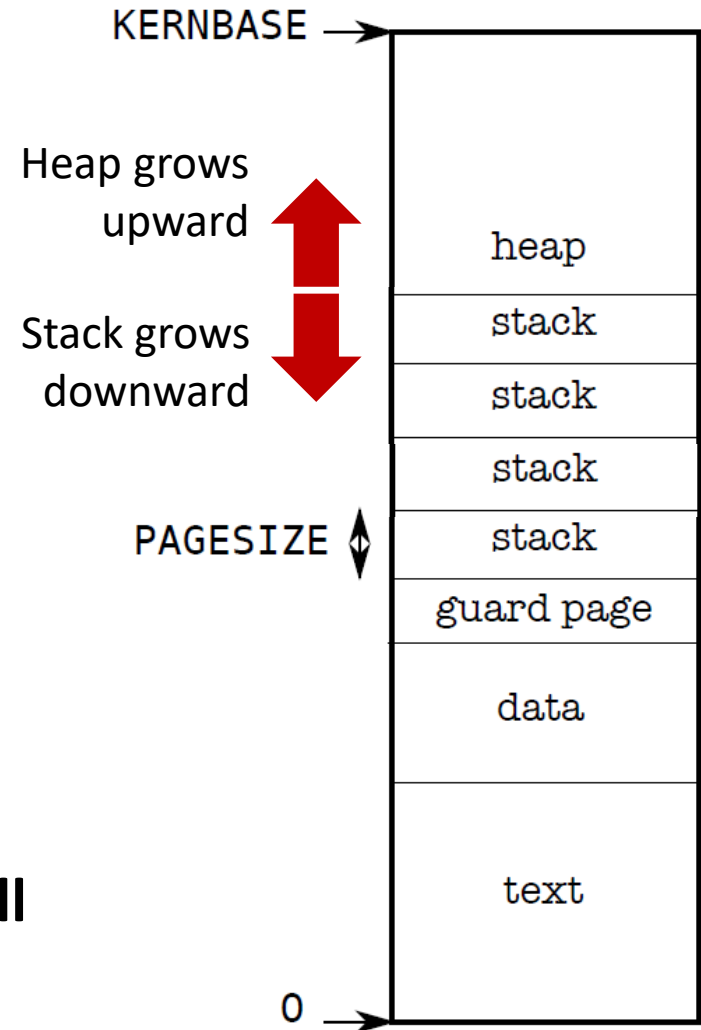
## ■ Initial size of stack

- Prepare 1 page initially
- Can be grow up to 4 pages

## ■ Growth of stack

- Implementation of page fault handler
- New page should be allocated to this process if current stack is full

## ■ When stack pointer reaches guard page, occurs stack overflow and kill process



# Page Fault Handler Implementation

```
trap.c void
trap(struct trapframe *tf)
{
    if(tf->trapno == T_SYSCALL) {
        if(myproc()->killed)
            exit();
        myproc()->tf = tf;
        syscall();
        if(myproc()->killed)
            exit();
        return;
    }

    switch(tf->trapno) {
        case T_PGFLT:
            pagefault();
            break;
        case T_IRQ0 + IRQ_TIMER:
            if(cpuid() == 0) {
                acquire(&tickslock);
                ticks++;
                wakeup(&ticks);
                release(&tickslock);
            }
            lapiceoi();
            break;
    }
}
```

# Page Fault Handler Implementation

■ `int exec(char *path, char **argv)`

```
// Allocate two pages at the next page boundary.  
// Make the first inaccessible. Use the second as the user stack.  
sz = PGROUNDUP(sz);  
if((sz = allocvm(pgdir, sz, sz + 2*PGSIZE)) == 0)  
    goto bad;  
clearpteu(pgdir, (char*)(sz - 2*PGSIZE));  
sp = sz;
```

- 가상주소공간에 4개의 stack page 공간 확보
  - 실제로는 하나의 physical page만 할당
  - 나머지 virtual page는 page entry는 생성하되 physical page는 할당하지 않음.
- `allocvm()`을 변형해 stack 공간 할당을 위한 함수 구현 (ex: `allocvm_stack`)



# Page Fault Handler Implementation

## ■ vm.c

- void pagefault(void) 구현
  - rcr2() 를 호출해 page fault가 발생한 virtual address 결정
  - virtual address가 valid한 값인지 확인
    - 현재 stack의 top의 다음 page에 속한 주소인지 확인
    - 4 stack pages의 범위를 벗어나는지 확인
  - Print out “[Pagefault] Invalid access!” when approaching abnormal address
  - Print out “[Pagefault] Allocate new page!” if page-fault handler is executed normally
  - 새로운 페이지를 할당하고 page table에 매핑한다.
  - lcr3(V2P(pde\_t)) – reloading cr3

# Page Fault Handler Implementation

## ■ vm.c

### ■ void pagefault(void) 구현

```
//PAGEFAULT
void pagefault(void)
{
    ...
    addr = rcr2();
    ...
    if( addr is valid ) {
        1 physical page alloc
        ...
        lcr3(V2P(myproc()->pgdir));
        cprintf("[Pgaefault]Aococate new page!\n");
    }
    else {
        cprintf("[Pagefault]Invalid access!\n");
        myproc()->killed = 1;
    }
}
```

# Page Fault Handler Implementation

## ■ vm.c : copyvm()

- Fork() 함수에서 호출
- parent의 page table을 child process로 복사하는 함수
- Stack에 4개의 page가 할당 되었지만 physical page는 할당되지 않았기 때문에 panic 발생
- physical page가 할당되지 않은 상태에서도 'panic'이 발생하지 않도록 수정

```
pde_t*
copyvm(pde_t *pgdir, uint sz)
{
    pde_t *d;
    pte_t *pte;
    uint pa, i, flags;
    char *mem;

    if((d = setupkvm()) == 0)
        return 0;
    for(i = 0; i < sz; i += PGSIZE){
        if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
            panic("copyvm: pte should exist");
        if(!(*pte & PTE_P))
            panic("copyvm: page not present");
        pa = PTE_ADDR(*pte);
        flags = PTE_FLAGS(*pte);
        if((mem = kalloc()) == 0)
            goto bad;
```

# Project #3. Template Code

- Download **xv6-pnu-3.tar.gz** from PLATO
- Modifications
  - `freemem()` system call
    - Return the number of free pages in `kmem.freelist`

# Submission

- Compress your xv6 folder as **StudentID-3.tar.gz**

- `$tar -czvf StudentID-3.tar.gz ./xv6-pnu-3`
- Please command `$make clean` before compressing

- Submit your `tar.gz` file through **PLATO**

- Due date: **5/31 (SUN), 23:59**

- Late submission penalty

- -25% penalty of total mark per day

- **PLEASE DO NOT COPY !!**

- YOU WILL GET F IF YOU COPIED

# Tips

- Reading xv6 commentary will help you a lot
  - <https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf>
    - The line numbers in this book refer to the source booklet below
    - Reading **Chap. 2 “Page tables”** of xv6-commentary will help your project
  - <https://pdos.csail.mit.edu/6.828/2018/xv6/xv6-rev11.pdf>