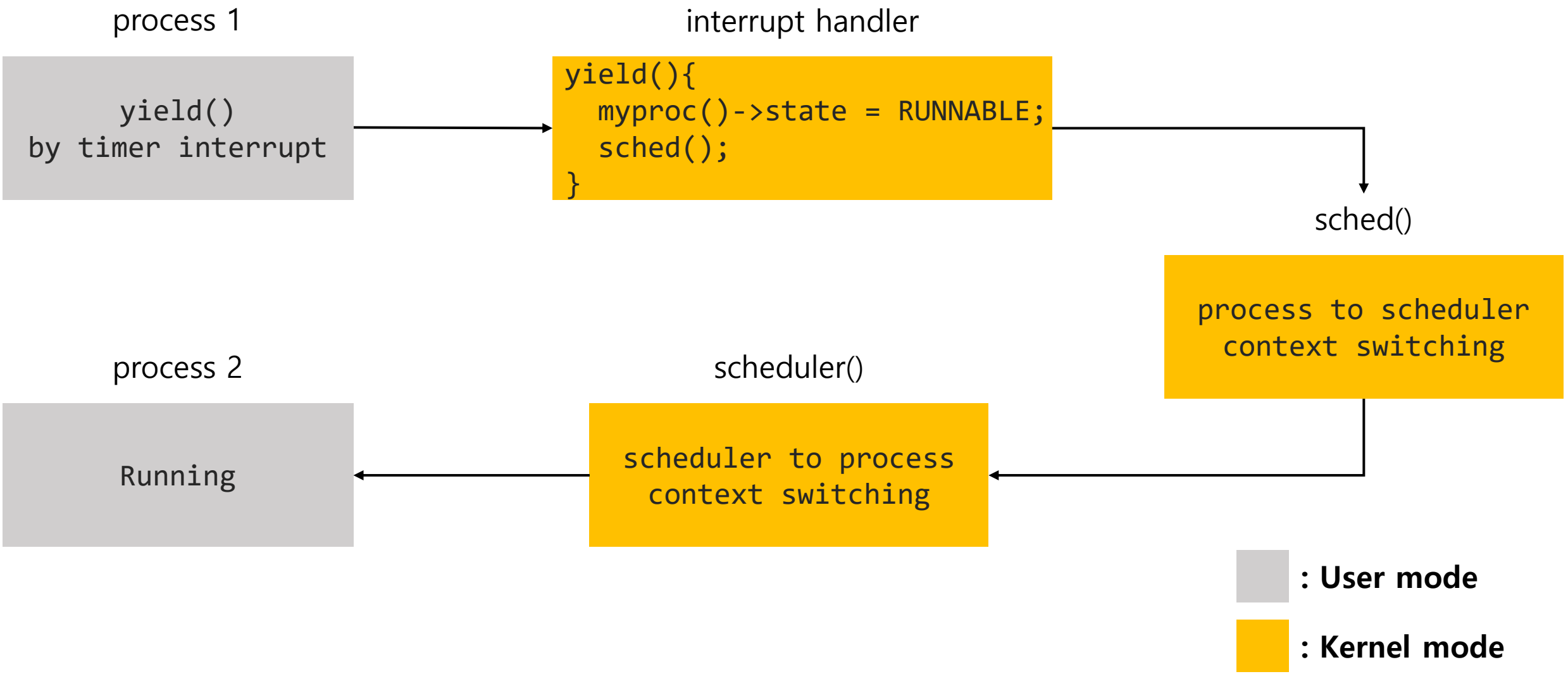
운영체제 2024-1

xv6 – 03. Virtual Memory and Paging

2024년 6월 3일
부산대학교 정보컴퓨터공학부
Prof. 김원석

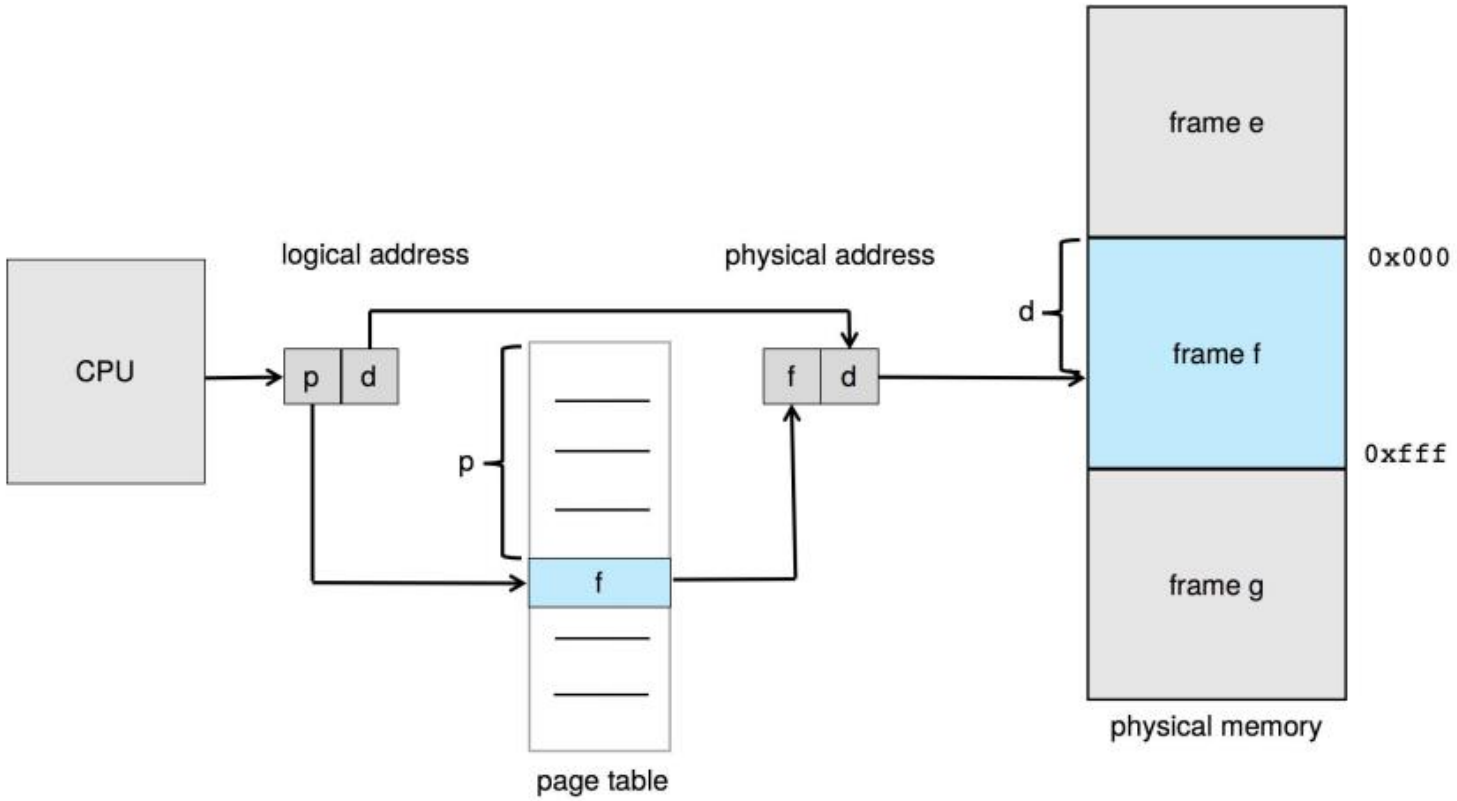
1. xv6에서의 CPU Scheduling (리뷰)



2. xv6에서의 Virtual Memory

❖ x86 프로세서는 메모리의 virtual address를 통해 physical address에 접근

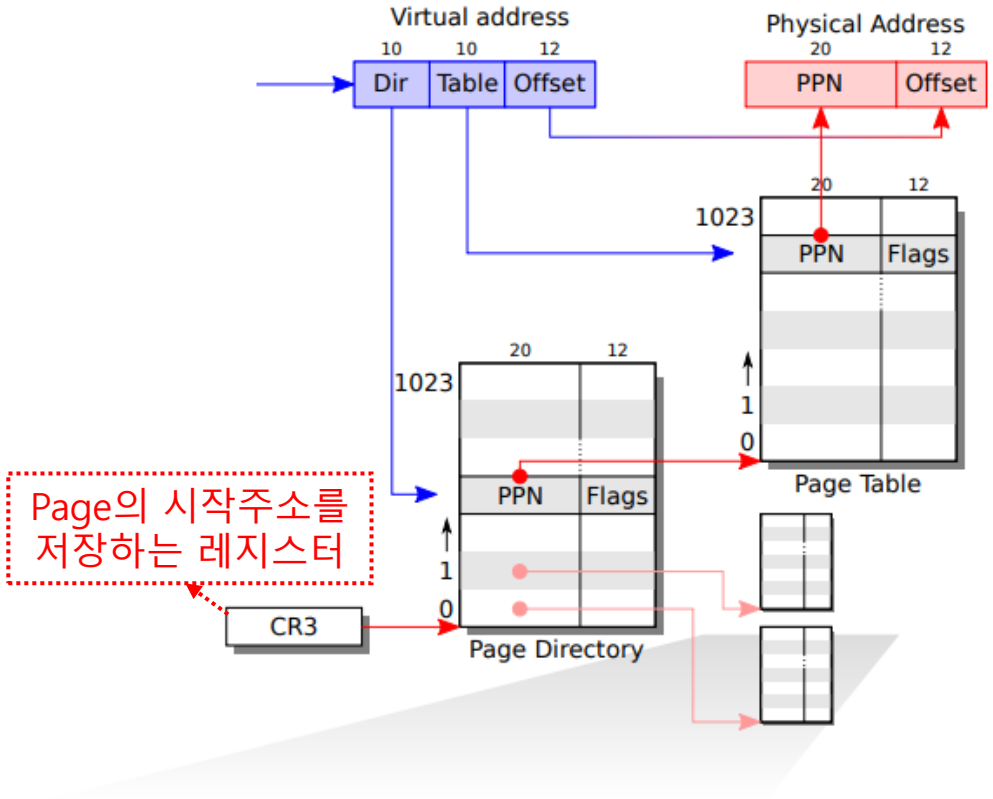
- xv6는 x86 아키텍처를 사용
- Page table entry는 해당 페이지가 할당된 물리 메모리 프레임의 base address를 포함
- 오프셋을 이용하여 프레임 내의 정확한 물리 주소를 찾을 수 있음

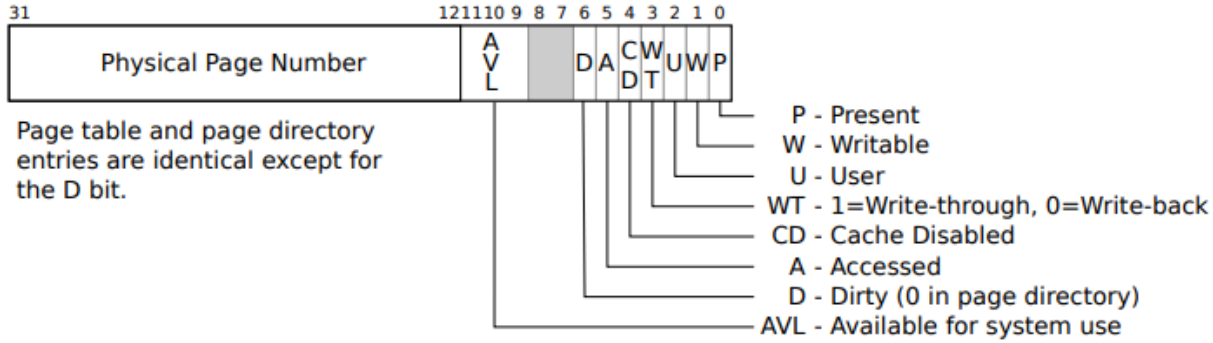


2. xv6에서의 Virtual Memory

❖ Two-Level Paging

- xv6는 32-bit 기반 OS이므로 주소체계도 그에 따름
  - 앞의 20 bits는 PTE(Page Table Entry)의 Index에 할당
  - 뒤의 10 bits는 Offset에 할당
- 20 bits를 모두 하나의 페이지 테이블에 할당한다면 프로세스마다 2^{20}개의 PTE를 갖는 페이지 테이블이 만들어짐
  - 각 페이지 테이블은 4 MB의 크기를 갖게 되고 이는 현실적으로 너무 큼
- 따라서 실제로는 오른쪽과 같이 Level을 분리하여 Page Directory와 Page Table로 구분
  - Page Directory와 Page Table은 각각 2^{10}개의 PTE를 가짐
  - Page Table은 필요할 시 새로 할당하여 메모리를 절약



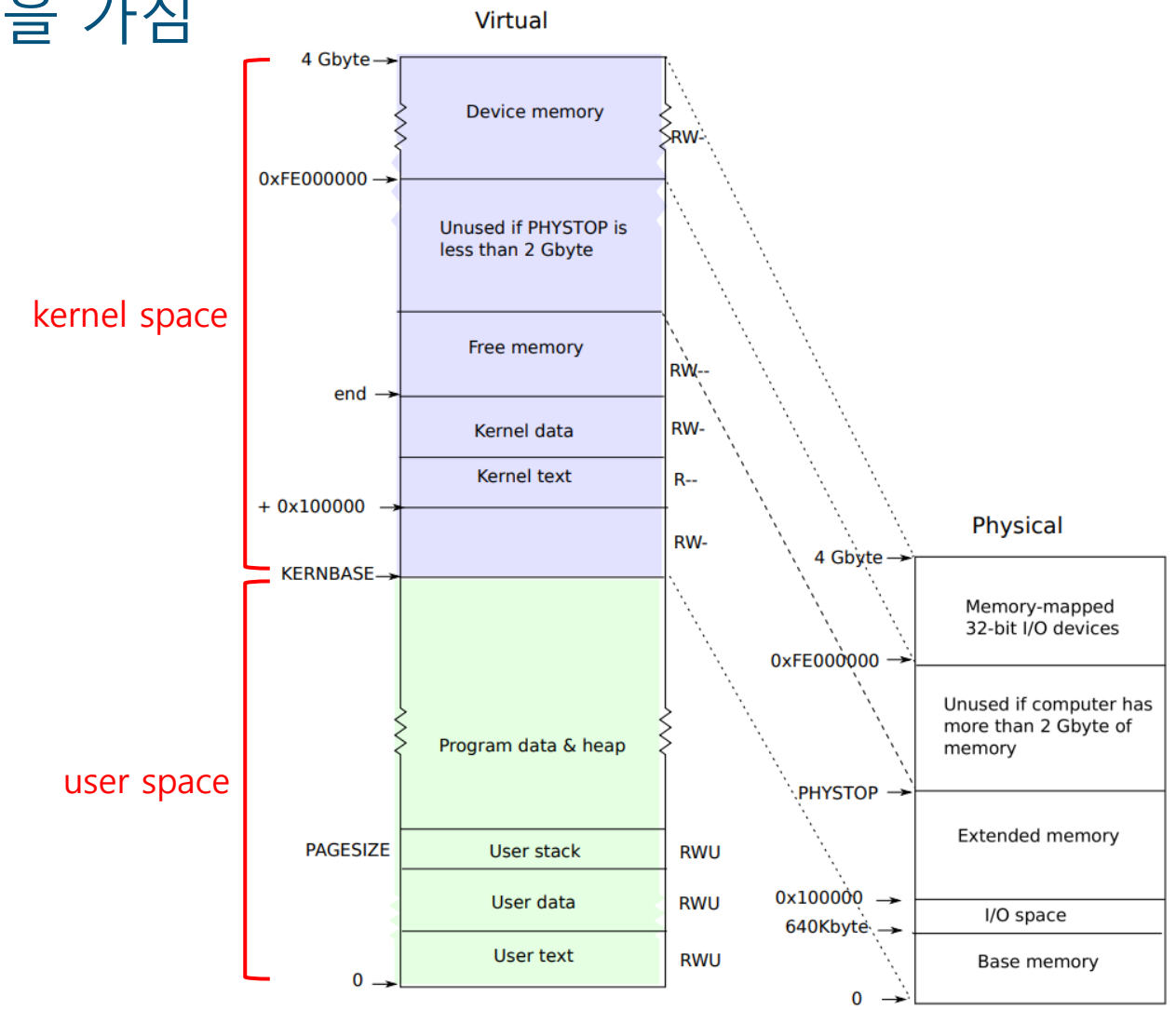


2. xv6에서의 Virtual Memory

❖ xv6의 프로세스는 고유의 가상 주소 공간을 가짐

- 오른쪽과 같이 user space와 kernel space로 구분
- KERNBASE을 기점으로 최대 2GB씩 할당
- memlayout.h에 KERNBASE, PHYSTOP 등의 상수와 V2P, P2V 등의 매크로 함수가 정의되어 있음

```
C memlayout.h > ...
1   // Memory layout
2
3   #define EXTMEM  0x100000            // Start of extended memory
4   #define PHYSTOP 0xE000000           // Top physical memory
5   #define DEVSPACE 0xFE000000         // Other devices are at high addresses
6
7   // Key addresses for address space layout (see kmap in vm.c for layout)
8   #define KERNBASE 0x80000000         // First kernel virtual address
9   #define KERNLINK (KERNBASE+EXTMEM)  // Address where kernel is linked
10
11  #define V2P(a) (((uint) (a)) - KERNBASE)
12  #define P2V(a) ((void *)(((char *) (a)) + KERNBASE))
```



2. xv6에서의 Virtual Memory

❖ sbrk 시스템 콜

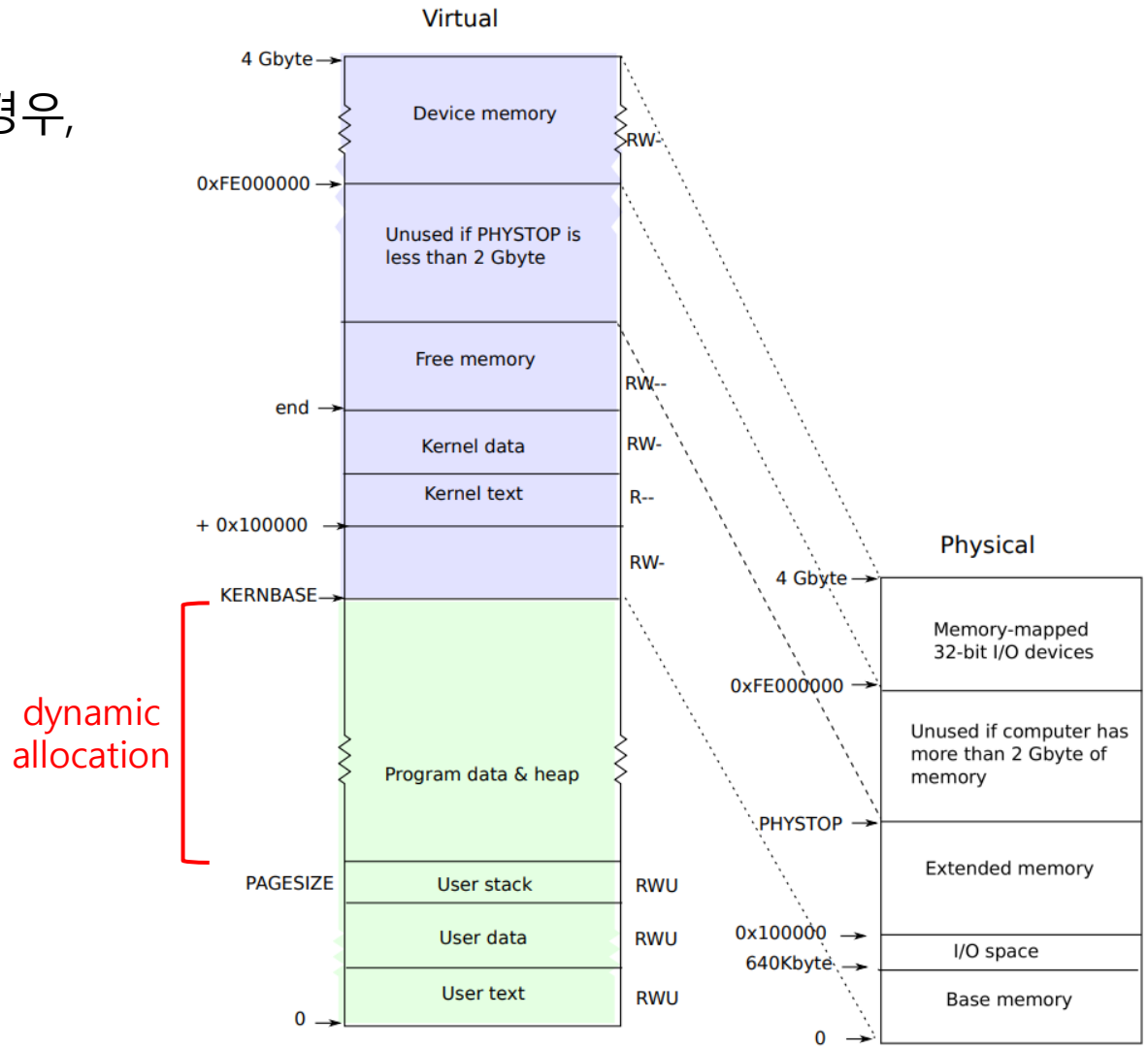
- 프로세스가 malloc 등의 함수로 메모리를 동적 할당할 경우, sbrk 시스템 콜이 호출되어 현재 실행 중인 프로세스의 heap 사이즈를 조절
- 프로세스를 위한 물리 메모리를 할당하고 page를 통해 물리 주소와 가상 주소를 매핑

sysproc.c > ...

```
45  int
46  sys_sbrk(void)
47  {
48    int addr;
49    int n;
50
51    if(argint(0, &n) < 0)
52      return -1;
53    addr = myproc()->sz;
54    if(growproc(n) < 0)
55      return -1;
56    return addr;
57  }
```

proc.c > ...

```
158  // Grow current process's memory by n bytes.
159  // Return 0 on success, -1 on failure.
160  int
161  growproc(int n)
162  {
163    uint sz;
164    struct proc *curproc = myproc();
165
166    sz = curproc->sz;
167    if(n > 0){
168      if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
169        return -1;
170    } else if(n < 0){
171      if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
172        return -1;
173    }
174    curproc->sz = sz;
175    switchuvm(curproc);
176    return 0;
177  }
```



2. xv6에서의 Virtual Memory

❖ growproc 함수는 아래의 allocuvm 함수와 deallocuvm 함수를 통해 메모리 사이즈를 조절하고 테이블을 갱신

vm.c > ...

```c
// Allocate page tables and physical memory to grow process from oldsz to
// newsz, which need not be page aligned.  Returns new size or 0 on error.
int
allocuvm(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("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

vm.c > ...

```c
// Deallocate user pages to bring the process size from oldsz to
// newsz.  oldsz and newsz need not be page-aligned, nor does newsz
// need to be less than oldsz.  oldsz can be larger than the actual
// process size.  Returns the new process size.
int
deallocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  pte_t *pte;
  uint a, pa;

  if(newsz >= oldsz)
    return oldsz;

  a = PGROUNDUP(newsz);
  for(; a  < oldsz; a += PGSIZE){
    pte = walkpgdir(pgdir, (char*)a, 0);
    if(!pte)
      a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
    else if((*pte & PTE_P) != 0){
      pa = PTE_ADDR(*pte);
      if(pa == 0)
        panic("kfree");
      char *v = P2V(pa);
      kfree(v);
      *pte = 0;
    }
  }
  return newsz;
}
```

3. xv6 assignment.3

- `sbrk()` 시스템 콜에서 memory allocation 제거
- Lazy allocation

3-0. 과제 수행 전 필수 이행사항

❖ xv6-public-hw2 디렉토리를 복제

- 디렉토리 이름을 xv6-public-hw3로 하여 복제

```
$ sudo cp -r xv6-public-hw2 xv6-public-hw3
```

- 이후 수행할 과제는 모두 xv6-public-hw3 디렉토리의 파일을 사용하여 진행
  - 원본 디렉토리(xv6-public-hw2)와 비교하여 수정 로그를 얻기 위함

3-1. sbrk() 시스템 콜에서 memory allocation 제거

❖ sbrk() 시스템 콜 호출 시 물리 메모리 영역에 메모리를 즉시 할당하지 않도록 수정

- 수정한 sbrk() 시스템 콜은 프로세스의 크기를 n만큼 증가시키고 수정 이전 크기를 반환해야 한다.
  이때, 메모리를 할당하지 않아야 하므로 growproc()을 호출하지 않도록 수정한다.
- echo 명령어로 메모리 할당을 하지 못하게 되었는지 확인한다.
  - 늘어난 heap 영역을 참조하는 page가 없으므로 해당 메모리에 접근 시 page fault가 발생한다(정상).
    addr 0xc004는 page fault가 발생한 원인이 된 virtual memory를 뜻한다.

sysproc.c > ...

```
45  int
46  sys_sbrk(void)
47  {
48    int addr;
49    int n;
50
51    if(argint(0, &n) < 0)
52      return -1;
53    addr = myproc()->sz;
54    if(growproc(n) < 0)
55      return -1;
56    return addr;
57  }
```

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ echo osAssignment
pid 3 sh: trap 14 err 6 on cpu 0 eip 0x1240 addr 0xc004--kill proc
$
```

traps.h > ...

```
18  #define T_PGFLT         14      // page fault
```

3-2. Lazy page allocation

❖ page fault 발생 시 프로세스를 종료하지 않고 물리 주소에 메모리 동적 할당 및 페이지를 매핑하도록 구현

- trap.c 내의 코드를 수정하여 user space로부터 발생한 page fault에 대응하도록 한다.
  - 새로 할당된 physical memory's page를 fault가 발생한 주소에 매핑한 후, user space로 복귀하여 프로세스를 계속 실행한다.
  - 이때, "`pid 3 sh: trap 14`"를 출력한 cprintf 직전에 코드를 추가해야 한다.
- page fault는 default에서 처리되고 있으며 pid, name, trap이 발생한 가상 메모리 주소 등을 출력한다.

trap.c > trap(trapframe *)

```
83    //PAGEBREAK: 13
84    default:
85      if(myproc() == 0 || (tf->cs&3) == 0){
86        // In kernel, it must be our mistake.
87        cprintf("unexpected trap %d from cpu %d eip %x (cr2=0x%x)\n",
88                tf->trapno, cpuid(), tf->eip, rcr2());
89        panic("trap");
90      }
91      // In user space, assume process misbehaved.
92      cprintf("pid %d %s: trap %d err %d on cpu %d "
93              "eip 0x%x addr 0x%x--kill proc\n",
94              myproc()->pid, myproc()->name, tf->trapno,
95              tf->err, cpuid(), tf->eip, rcr2());
96      myproc()->killed = 1;
97    }
```

3-2. Lazy page allocation

❖ vm.c의 allocuvm() 함수를 참고하여 메모리 할당 및 page에 매핑(trap.c)

- PGROUNDUP
  - 파라미터에서 PGSIZE의 배수를 갖는 주소로 올림하여 이동하는 함수
  - 과제 수행을 위해 page fault를 발생시키는 virtual address를 PGROUNDDOWN() 함수를 사용하여 page boundary로 내린다.
- break 또는 return을 사용하여 cprintf와 proc->killed = 1이 실행되는 것을 방지한다.

C vm.c > ...

```
219  // Allocate page tables and physical memory to grow process from oldsz to
220  // newsz, which need not be page aligned.  Returns new size or 0 on error.
221  int
222  allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
223  {
224    char *mem;
225    uint a;
226
227    if(newsz >= KERNBASE)
228      return 0;
229    if(newsz < oldsz)
230      return oldsz;
231
232    a = PGROUNDUP(oldsz);
233    for(; a < newsz; a += PGSIZE){
234      mem = kalloc();
```

3-2. Lazy page allocation

❖ mappages()를 호출할 수 있도록 수정

- vm.c의 mappages() 정의에서 static int를 int로 수정한다.

```
 GNU nano 6.2                                                    vm.c
int
mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
{
  char *a, *last;
  pte_t *pte;
```

- trap.c에서 mappages() 함수를 호출할 수 있도록 trap.c 내에 다음 구문을 추가한다.
  - int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm);

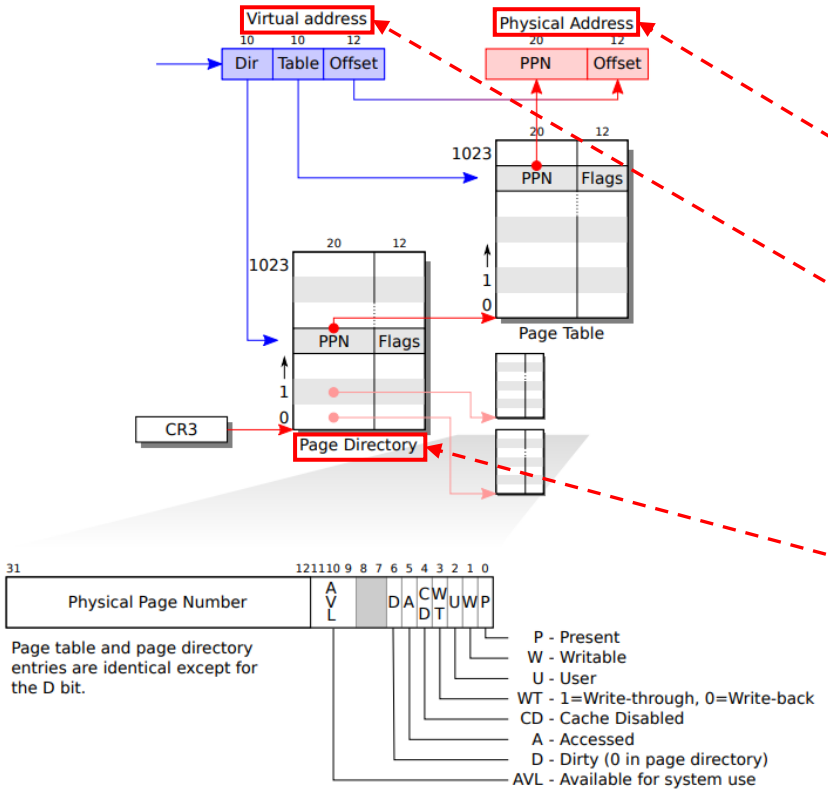
```
 GNU nano 6.2                                                    trap.c
int mappages(pdt_t *pgdir, void *va, uint size, uint pa, int perm);
```

- fault가 page fault인지 확인하기 위해 trap() 내에서 tf->trapno가 T_PGFLT인지 비교한다.

- mappages()
  - page table에 새로운 page table entry를 만들어 가상 주소와 물리 주소를 매핑

```
C vm.c > ...
240       memset(mem, 0, PGSIZE);
241       if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
242         cprintf("allocuvm out of memory (2)\n");
243         deallocuvm(pgdir, newsz, oldsz);
244         kfree(mem);
245         return 0;
246       }
247     }
248     return newsz;
249   }
```

3-2. Lazy page allocation



vm.c > ...

```
219  // Allocate page tables and physical memory to grow process from oldsz to
220  // newsz, which need not be page aligned.  Returns new size or 0 on error.
221  int
222  allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
223  {
224    char *mem;
225    uint a;
226
227    if(newsz >= KERNBASE)
228      return 0;
229    if(newsz < oldsz)
230      return oldsz;
231
232    a = PGROUNDUP(oldsz);
233    for(; a < newsz; a += PGSIZE){
234      mem = kalloc();
235      if(mem == 0){
236        cprintf("allocuvm out of memory\n");
237        deallocuvm(pgdir, newsz, oldsz);
238        return 0;
239      }
240      memset(mem, 0, PGSIZE);
241      if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
242        cprintf("allocuvm out of memory (2)\n");
243        deallocuvm(pgdir, newsz, oldsz);
244        kfree(mem);
245        return 0;
246      }
247    }
248    return newsz;
249  }
```

3-2. Lazy page allocation

❖ 옳게 구현했을 시 아래와 같이 echo가 다시 정상적으로 동작

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ echo osAssignment
osAssignment
$
```

4. 제출

❖ 제출

- 아래의 명령어를 통해 원본 디렉토리(xv6-public-hw2)와 비교한 수정 로그(hw3.patch)를 생성
  - 비교하는 두 디렉토리는 반드시 make clean 명령어로 불필요한 파일을 제거한 상태여야 함

```
~/xv6-public-hw3$ sudo make clean          # 해당 명령어로 불필요한 파일을 먼저 제거
~/xv6-public-hw3$ cd ..
~$ sudo diff -utrN xv6-public-hw2 xv6-public-hw3 > hw3.patch
```

- 아래의 명령어를 통해 hw3.patch파일과 xv6-public-hw3 디렉토리를 함께 압축

```
~$ sudo zip -r 학번-hw3.zip hw3.patch ./xv6-public-hw3
```

- 압축 폴더(학번-hw3.zip)를 PLATO에 업로드