운영체제 2024-1

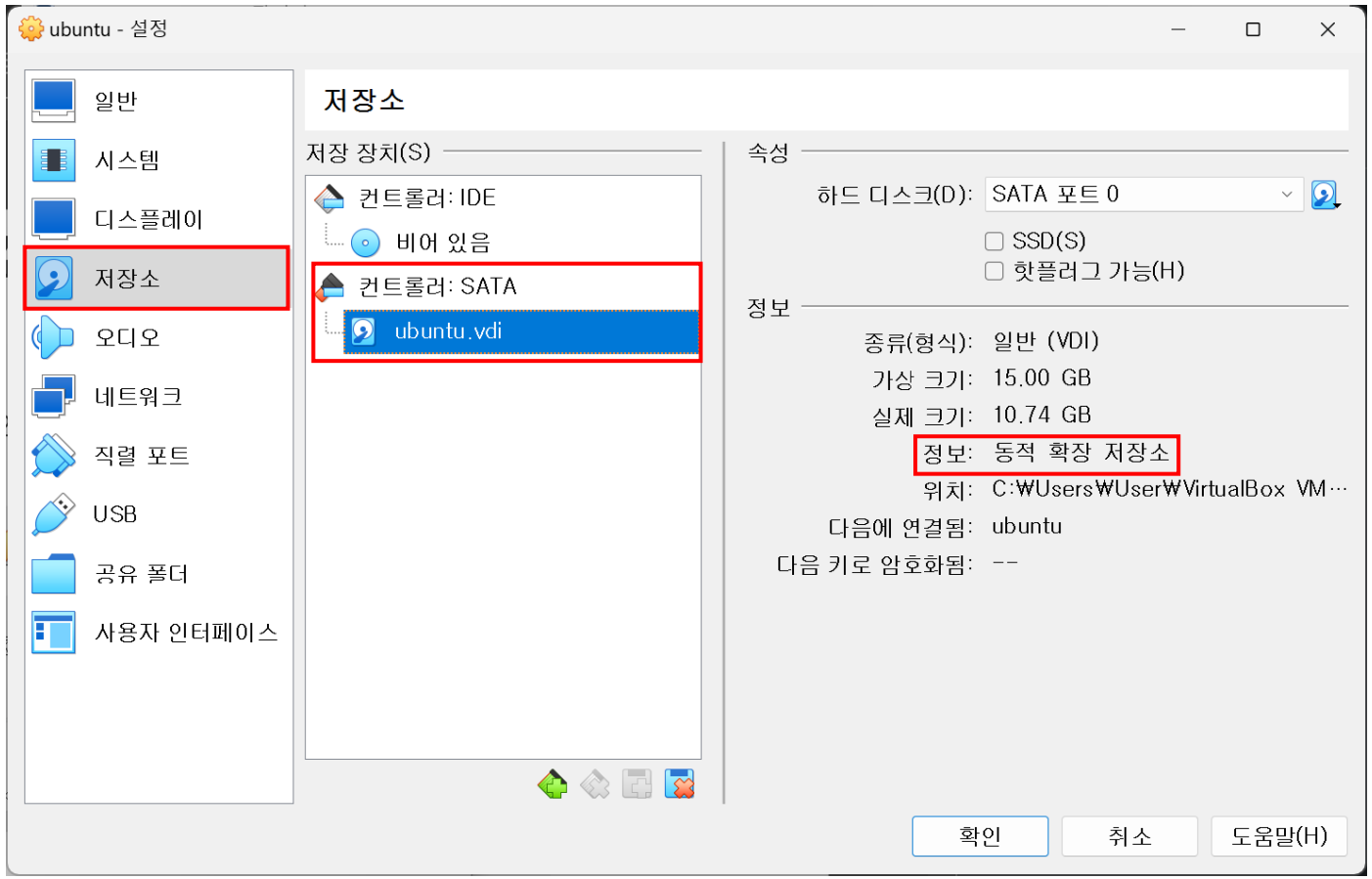
Xv6 – 02. CPU Scheduling

2024년 5월 20일
부산대학교 정보컴퓨터공학부
Prof. 김원석

0. VirtualBox 디스크 크기 조정

❖ 가상 머신 디스크 크기 조정

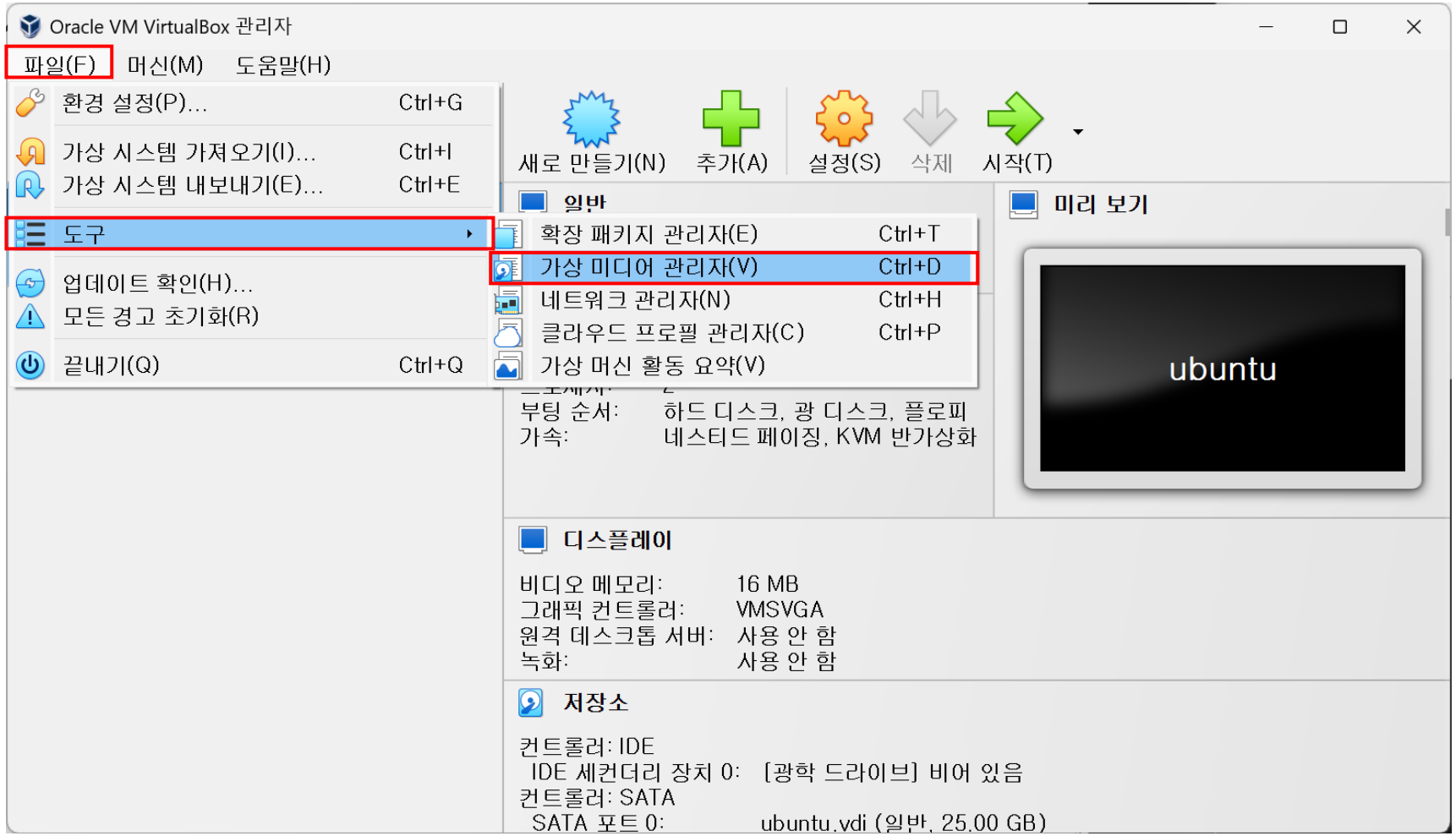
- 가상 머신 설정 > 저장소 > ubuntu.vdi의 정보가 동적 확장 저장소인지 확인
- 동적 확장 저장소가 아니라 고정 크기 저장소라면 가상 머신을 새로 생성해야 함



0. VirtualBox 디스크 크기 조정

❖ 가상 머신 디스크 크기 조정

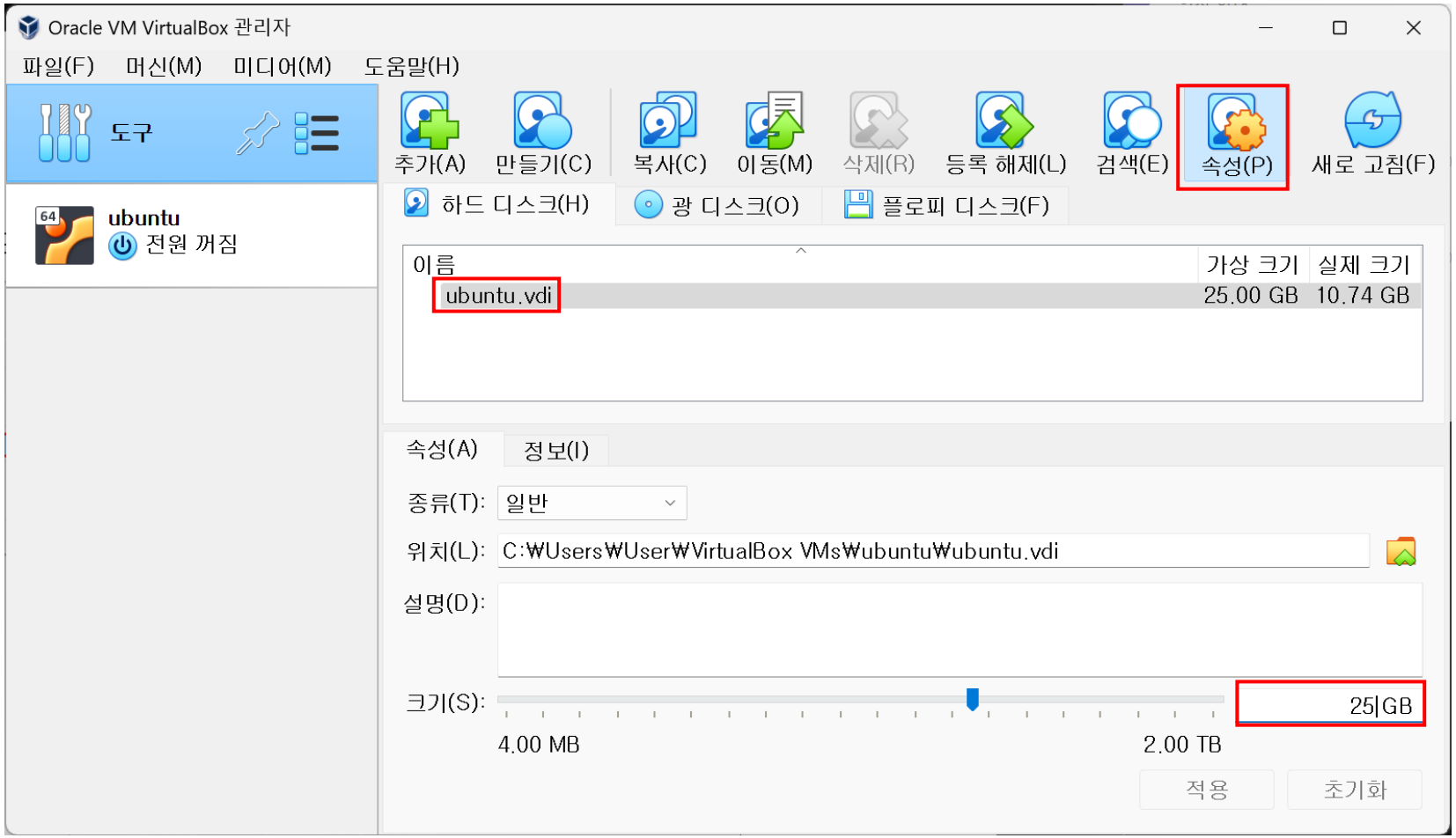
- 파일 > 도구 > 가상 미디어 관리자



0. VirtualBox 디스크 크기 조정

❖ 가상 머신 디스크 크기 조정

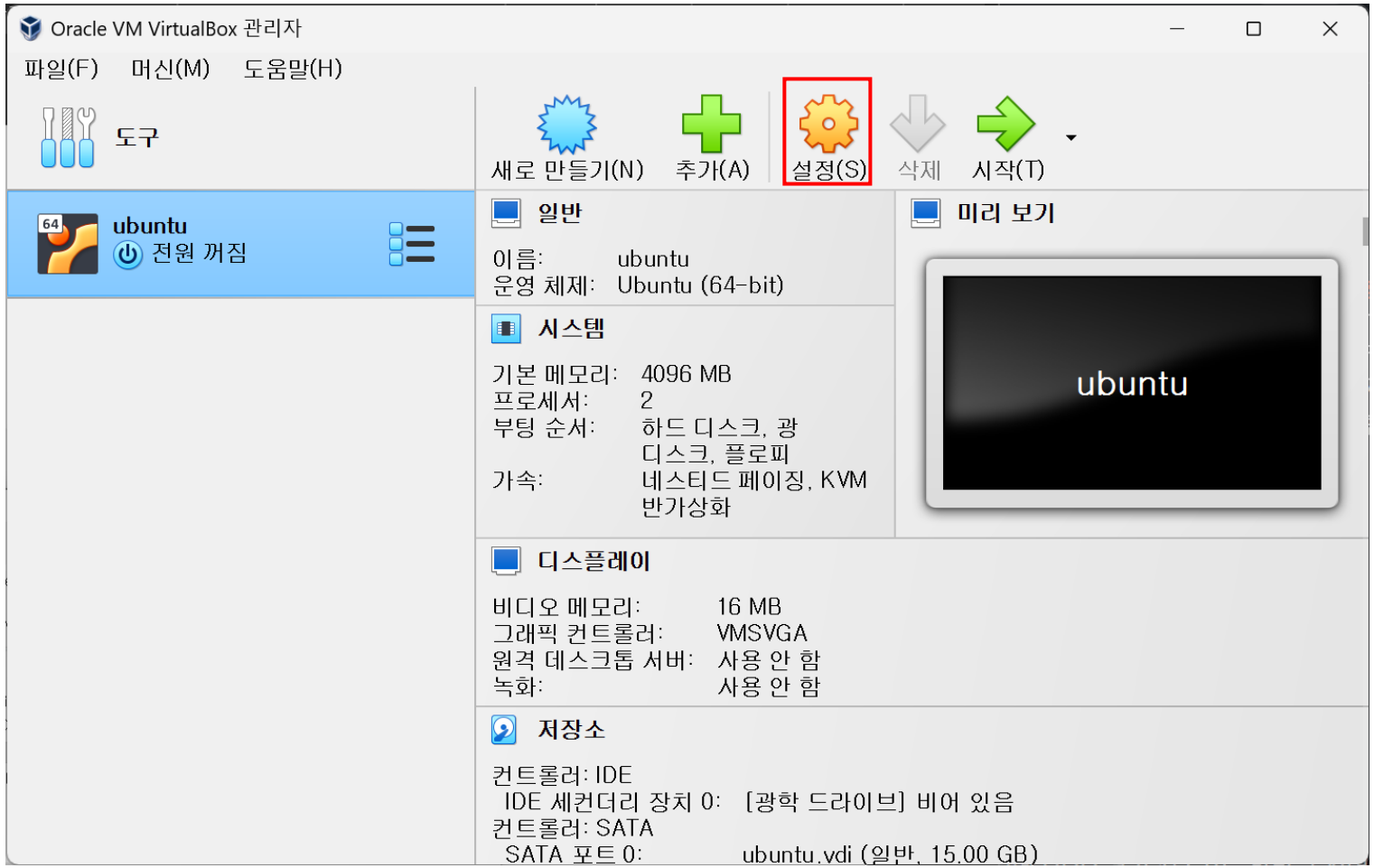
- ubuntu.vdi > 속성 아래 크기를 25GB로 조정 후 적용



0. VirtualBox 공유 폴더 설정

❖ 가상 머신과 실제 OS 간 파일 공유

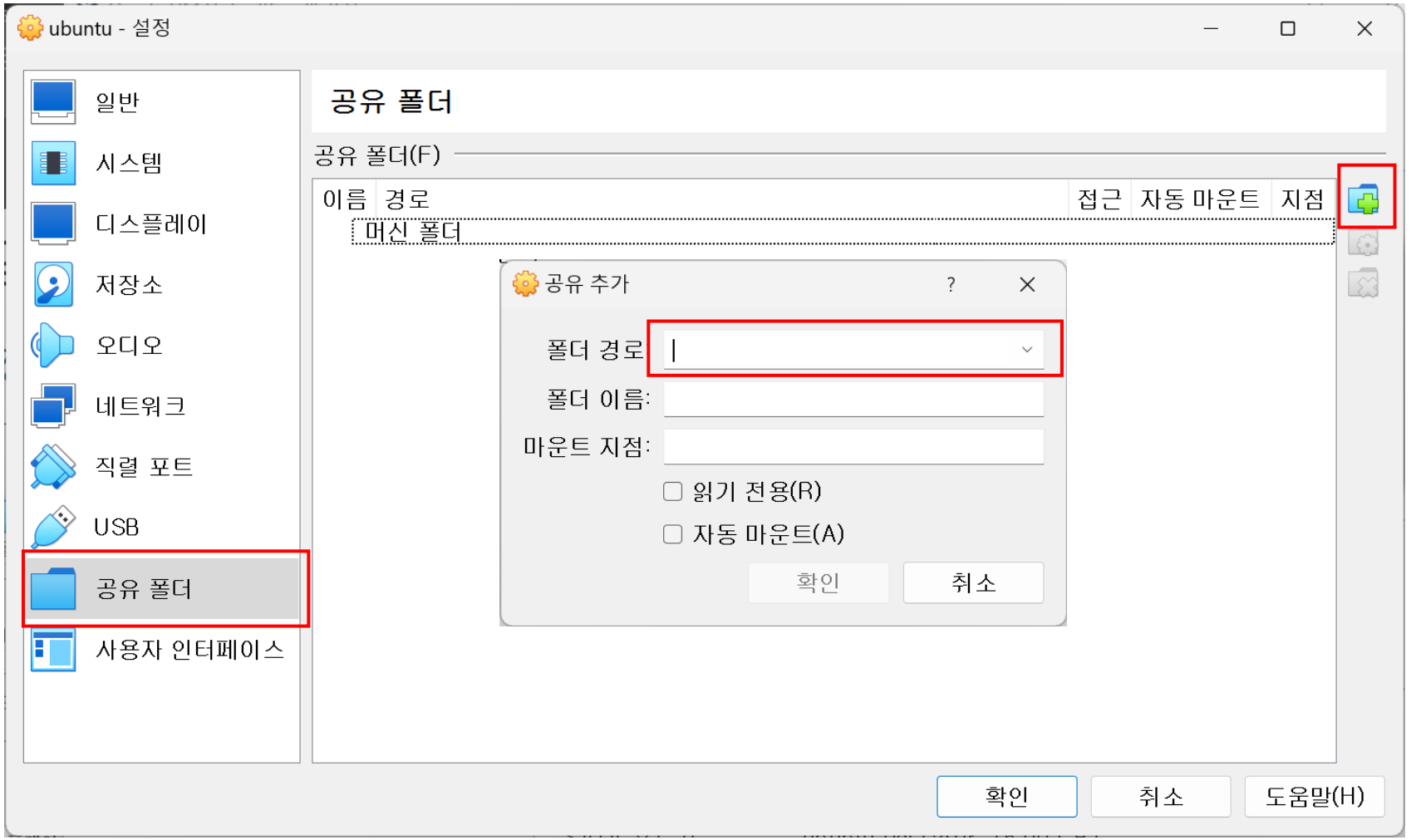
- 가상 머신 내에서 PLATO에 파일을 업로드하려 할 때 안 되는 문제가 있음



0. VirtualBox 공유 폴더 설정

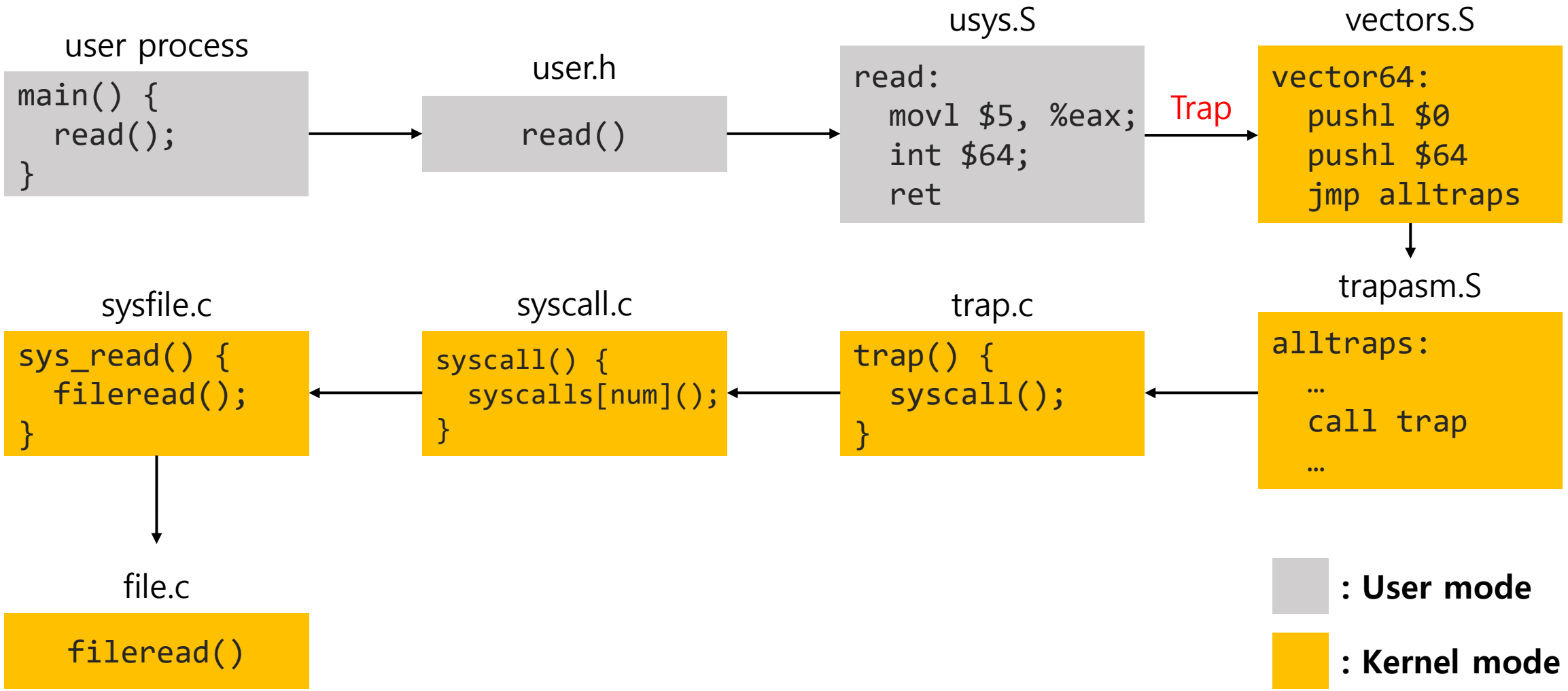
❖ 가상 머신과 실제 OS 간 파일 공유

- 원하는 경로 지정 후 저장

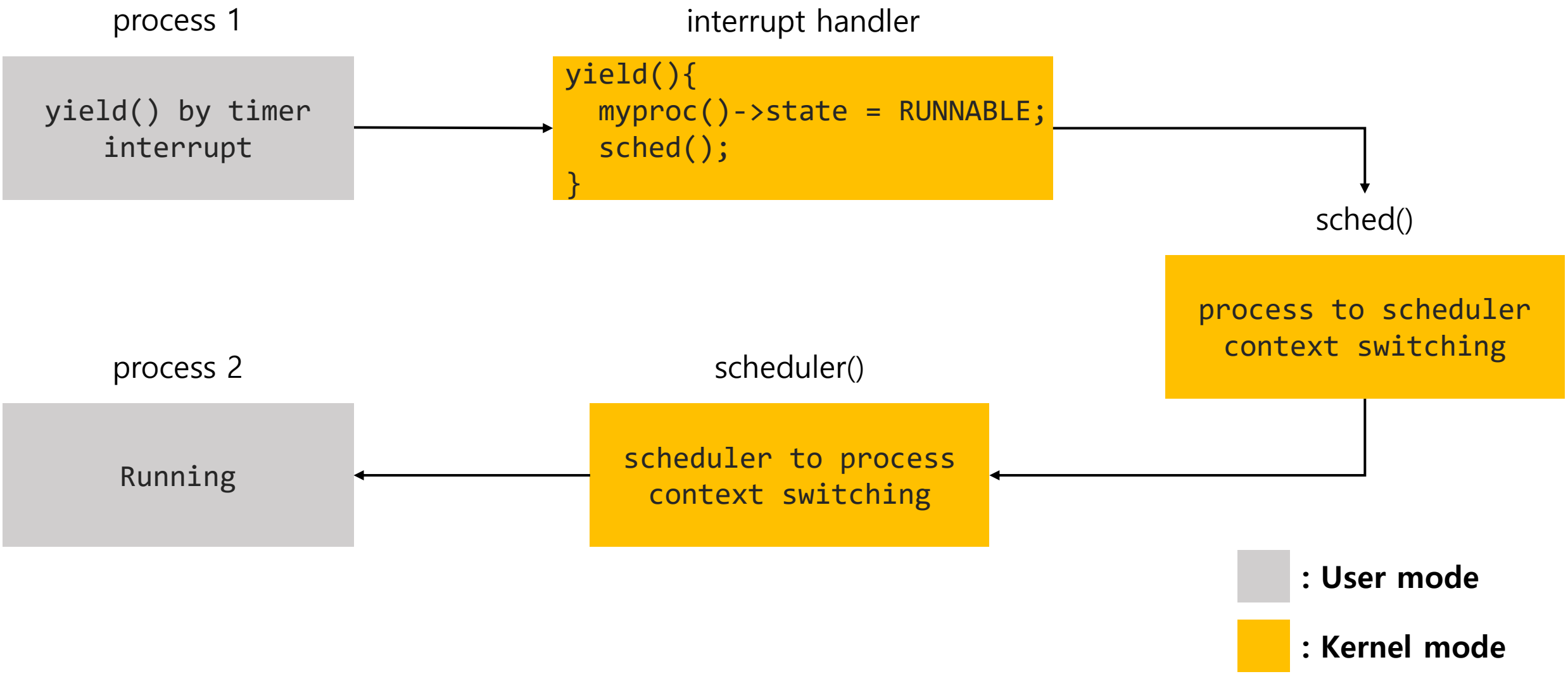


1. xv6에서의 시스템 콜 처리 과정 (리뷰)

❖ Example: read system call



1. xv6에서의 CPU Scheduling (개요)



1. xv6에서의 CPU Scheduling

❖ proc.h 내에 context switching을 위한 여러 자료형이 정의되어 있음

- 각 CPU에 대한 정보를 저장하는 구조체
- 각 프로세스에 대한 정보를 저장하는 구조체(PCB)
- 프로세스의 상태를 나타내는 열거형 변수
- 컨텍스트 스위치를 위한 레지스터를 저장하는 구조체

proc.h > ...

```
27  struct context {
28    uint edi;
29    uint esi;
30    uint ebx;
31    uint ebp;
32    uint eip;
33  };
34
35  enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

proc.h > ...

```
1   // Per-CPU state
2   struct cpu {
3     uchar apicid;               // Local APIC ID
4     struct context *scheduler;  // swtch() here to enter scheduler
5     struct taskstate ts;        // Used by x86 to find stack for interrupt
6     struct segdesc gdt[NSEGS];  // x86 global descriptor table
7     volatile uint started;      // Has the CPU started?
8     int ncli;                   // Depth of pushcli nesting.
9     int intena;                 // Were interrupts enabled before pushcli?
10    struct proc *proc;          // The process running on this cpu or null
11  };
```

proc.h > ...

```
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52  };
```

1. xv6에서의 CPU Scheduling

❖ proc.c 내에 context switching을 위한 프로세스 테이블이 구현되어 있음

- NPROC은 프로세스의 최대 개수를 지정하는 상수로, param.h에 64로 정의되어 있음

```
proc.c > ...
10    struct {
11      struct spinlock lock;
12      struct proc proc[NPROC];
13    } ptable;
```

실행 대기상태의 프로세스를 탐색

선택된 프로세스를 CPU에 할당하고 상태를 running으로 바꾼 후 컨텍스트 스위칭

프로세스 테이블에 대한 타 프로세스의 접근을 제한

```
proc.c > ...
322   void
323   scheduler(void)
324   {
325     struct proc *p;
326     struct cpu *c = mycpu();
327     c->proc = 0;
328
329     for(;;){
330       // Enable interrupts on this processor.
331       sti();
332
333       // Loop over process table looking for process to run.
334       acquire(&ptable.lock);
335       for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
336         if(p->state != RUNNABLE)
337           continue;
338
339         // Switch to chosen process.  It is the process's job
340         // to release ptable.lock and then reacquire it
341         // before jumping back to us.
342         c->proc = p;
343         switchuvm(p);
344         p->state = RUNNING;
345
346         swtch(&(c->scheduler), p->context);
347         switchkvm();
348
349         // Process is done running for now.
350         // It should have changed its p->state before coming back.
351         c->proc = 0;
352       }
353       release(&ptable.lock);
354
355     }
356   }
```

❖ 스케줄러 또한 구현되어 있음

- Round-Robin 알고리즘
  : 각 프로세스는 순서대로 일정 시간 CPU를 선점 후 ready queue의 가장 끝으로 돌아감

1. xv6에서의 CPU Scheduling

❖ (참고) 컨텍스트 스위칭

- switchhuvm: Task State Segment(TSS)와 H/W 페이지 테이블을 선택된 프로세스가 저장하고 있는 정보로 전환
- swtch: CPU의 context(scheduler)를 저장하고, 선택된 프로세스의 context를 불러옴으로서 스위칭
- switchkvm: 프로세스의 실행이 중단/종료되었을 때 H/W 페이지 테이블을 커널 영역에 대한 테이블로 전환
- 각 함수의 상세 동작은 코멘터리북 62p 참고
  - xv6 - DRAFT as of September 4, 2018 (mit.edu)

```
339     // Switch to chosen process.  It is the process's job
340     // to release ptable.lock and then reacquire it
341     // before jumping back to us.
342     c->proc = p;
343     switchuvm(p);
344     p->state = RUNNING;
345
346     swtch(&(c->scheduler), p->context);
347     switchkvm();
```

swtch.S

```
1   # Context switch
2   #
3   #   void swtch(struct context **old, struct context *new);
4   #
5   # Save the current registers on the stack, creating
6   # a struct context, and save its address in *old.
7   # Switch stacks to new and pop previously-saved registers.
8
9   .globl swtch
10  swtch:
11    movl 4(%esp), %eax
12    movl 8(%esp), %edx
13
14    # Save old callee-saved registers
15    pushl %ebp
16    pushl %ebx
17    pushl %esi
18    pushl %edi
19
20    # Switch stacks
21    movl %esp, (%eax)
22    movl %edx, %esp
23
24    # Load new callee-saved registers
25    popl %edi
26    popl %esi
27    popl %ebx
28    popl %ebp
29    ret
```

proc.h > ...

```
27  struct context {
28    uint edi;
29    uint esi;
30    uint ebx;
31    uint ebp;
32    uint eip;
33  };
```

1. xv6에서의 CPU Scheduling

❖ xv6는 내부적으로 설정된 타이머에 따라 주기적으로 타이머 인터럽트 발생

- 타이머 인터럽트 발생 시 trap.c 내의 trap 함수에서 yield 함수를 호출

trap.c > ...

```
103     // Force process to give up CPU on clock tick.
104     // If interrupts were on while locks held, would need to check nlock.
105     if(myproc() && myproc()->state == RUNNING &&
106        tf->trapno == T_IRQ0+IRQ_TIMER)
107       yield();
```

- proc.c 내에 구현된 yield 함수는 현재 CPU에서 실행 중인 프로세스를 강제로 준비 상태로 바꾸고 sched 함수를 호출

proc.c > ...

```
384   // Give up the CPU for one scheduling round.
385   void
386   yield(void)
387   {
388     acquire(&ptable.lock);  //DOC: yieldlock
389     myproc()->state = RUNNABLE;
390     sched();
391     release(&ptable.lock);
392   }
```

1. xv6에서의 CPU Scheduling

❖ proc.c 내의 sched 함수는 swtch 함수를 이용해 컨텍스트 스위칭

- 현재 실행 중인 프로세스의 context를 저장하고 CPU의 scheduler를 불러옴
  - eip(Program Counter)를 이용해 scheduler가 중단된 지점부터 실행 가능

proc.c > ...

```
358  // Enter scheduler.  Must hold only ptable.lock
359  // and have changed proc->state. Saves and restores
360  // intena because intena is a property of this
361  // kernel thread, not this CPU. It should
362  // be proc->intena and proc->ncli, but that would
363  // break in the few places where a lock is held but
364  // there's no process.
365  void
366  sched(void)
367  {
368    int intena;
369    struct proc *p = myproc();
370
371    if(!holding(&ptable.lock))
372      panic("sched ptable.lock");
373    if(mycpu()->ncli != 1)
374      panic("sched locks");
375    if(p->state == RUNNING)
376      panic("sched running");
377    if(readeflags()&FL_IF)
378      panic("sched interruptible");
379    intena = mycpu()->intena;
380    swtch(&p->context, mycpu()->scheduler);
381    mycpu()->intena = intena;
382  }
```

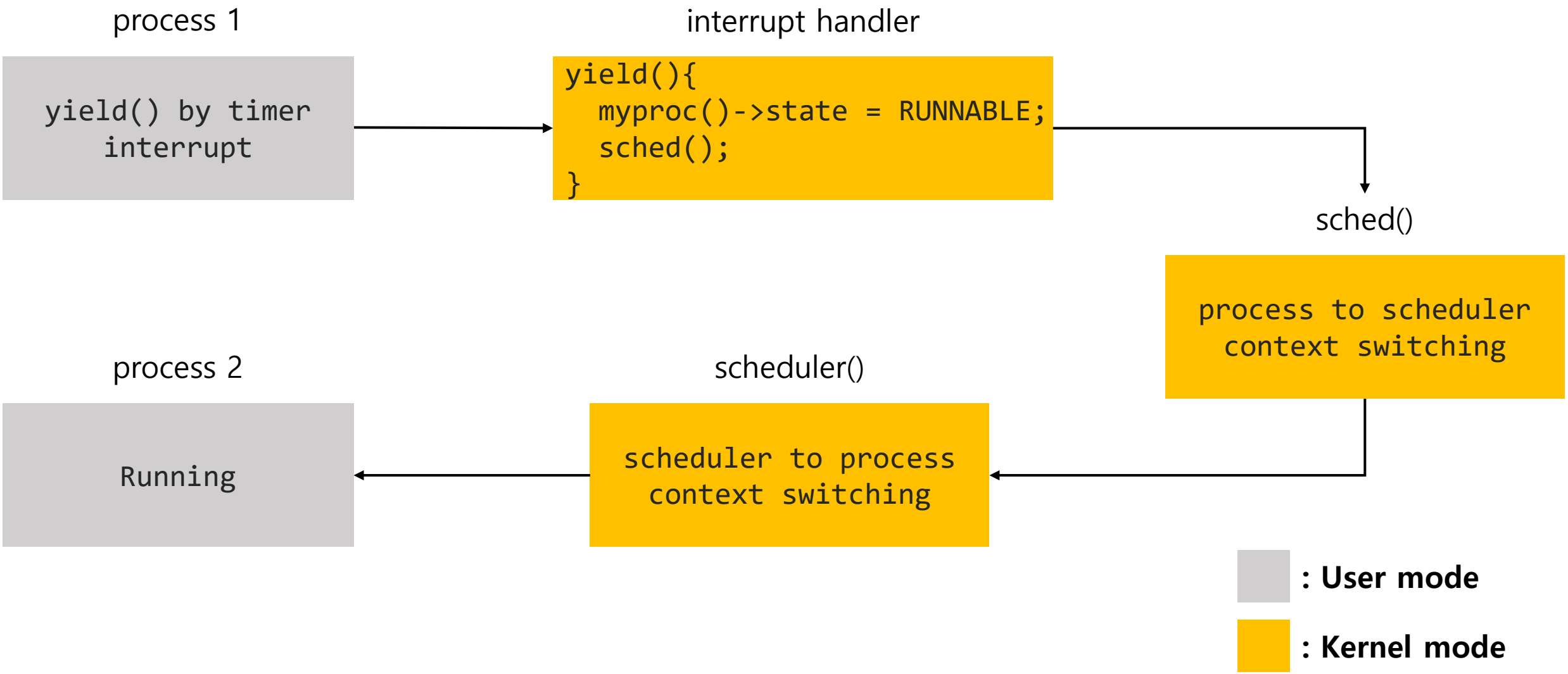
proc.h > ...

```
27   struct context {
28     uint edi;
29     uint esi;
30     uint ebx;
31     uint ebp;
32     uint eip;
33   };
```

proc.c > ...

```
322  void
323  scheduler(void)
324  {
325    struct proc *p;
326    struct cpu *c = mycpu();
327    c->proc = 0;
328
329    for(;;){
330      // Enable interrupts on this processor.
331      sti();
332
333      // Loop over process table looking for process to run.
334      acquire(&ptable.lock);
335      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
336        if(p->state != RUNNABLE)
337          continue;
338
339        // Switch to chosen process.  It is the process's job
340        // to release ptable.lock and then reacquire it
341        // before jumping back to us.
342        c->proc = p;
343        switchuvm(p);
344        p->state = RUNNING;
345
346        swtch(&(c->scheduler), p->context);
347        switchkvm();
348
349        // Process is done running for now.
350        // It should have changed its p->state before coming back.
351        c->proc = 0;
352      }
353      release(&ptable.lock);
354
355    }
356  }
```

1. xv6에서의 CPU Scheduling (요약)



2. xv6 assignment 02

❖ yield, setnice, getnice 시스템 콜 구현

- nice(niceness)
  - 프로세스간 우선순위를 표현하는 값, 작을 수록 높은 우선순위를 가짐
  - 리눅스 운영체제에서는 [–20, 20) 내의 값을 가짐, default = 0
  - xv6에서는 구현되어 있지 않으므로 이를 직접 구현
    - nice는 [0, 40) 내의 값을 갖고 default 값을 20으로 설정
    - fork 시스템 콜을 통해 자식 프로세스 생성 시 부모 프로세스의 nice를 상속받도록 구현

❖ 우선순위 스케줄링 알고리즘 구현

2-0. 과제 수행 전 필수 이행사항

❖ xv6-public 디렉토리를 복제

- 디렉토리 이름을 xv6-public-hw2로 하여 복제

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

```
pnudtn@xv6:~$ ls
Desktop    Downloads  Pictures  Templates  hw1.zip  xv6-public
Documents  Music      Public    Videos     snap
pnudtn@xv6:~$ sudo cp -r xv6-public xv6-public-hw2
pnudtn@xv6:~$ ls
Desktop    Downloads  Pictures  Templates  hw1.zip  xv6-public
Documents  Music      Public    Videos     snap     xv6-public-hw2
pnudtn@xv6:~$
```

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

2-1. yield, setnice, getnice 시스템 콜 구현

❖ int yield(void)

- proc.c에 구현되어 있는 yield 함수를 user process에서도 직접 호출할 수 있도록 시스템 콜을 구현

❖ int setnice(int pid, int nice)

- 프로세스(pid)의 nice 값을 전달한 인자 값으로 설정
- 인자로 전달한 pid를 가진 프로세스가 존재하지 않거나, nice가 [0,40) 범위 밖에 있으면 -1을 반환
- 정상 동작 시 0 반환
- acquire(&ptable.lock), release(&ptable.lock)를 활용하여 시스템콜 처리 도중 ptable에 대한 타 프로세스의 접근 제한

❖ int getnice(int pid)

- 프로세스(pid)의 nice를 반환
- 인자로 받은 pid를 가진 프로세스가 존재하지 않으면 -1을 반환

2-2. 우선순위 스케줄링 알고리즘 구현

❖ nice value가 낮은 프로세스가 우선적으로 CPU에 할당

- 타이머 인터럽트에 의해 주기적으로 할당 해제되지 않도록 수정
- 가장 작은 nice를 가진 프로세스가 여러 개일 경우 FCFS(First Come First Served) 알고리즘을 적용하여 그중 가장 먼저 RUNNABLE 상태가 되었던 프로세스를 RUNNING 상태로 전환
  - 프로세스가 생성된 순서가 아님에 주의
- setnice 시스템 콜에 의해 프로세스의 nice가 변경되었을 경우 현재 실행 중인 프로세스를 CPU에서 할당 해제하고 새로 할당할 프로세스를 다시 탐색해야 함

2-3. Makefile 수정 및 알고리즘 테스트

❖ Makefile 파일 수정

- xv6는 멀티 프로세싱을 지원하고 기본적으로 코어 개수가 2개로 설정되어 있음
  - 스케줄링 알고리즘의 테스트를 위해 코어 개수를 1로 수정

Makefile

```
223   ifndef CPUS
224   CPUS := 1
225   endif
```

- 알고리즘 검증을 위해 테스트 프로그램 3개(hw2_1.c, hw2_2.c, hw2_3.c)를 제공
  - 빌드 과정에 포함되도록 우측 그림과 같이 수정

Makefile

```
168   UPROGS=\
169       _cat\
170       _echo\
171       _forktest\
172       _grep\
173       _init\
174       _kill\
175       _ln\
176       _ls\
177       _mkdir\
178       _rm\
179       _sh\
180       _stressfs\
181       _usertests\
182       _wc\
183       _zombie\
184       _hw1\
185       _hw2_1\
186       _hw2_2\
187       _hw2_3\
```

2-3. Makefile 수정 및 알고리즘 테스트

❖ 테스트 프로그램을 통한 스케줄링 알고리즘 검증

- 아래의 그림과 같이 테스트 프로그램 실행 시 Step이 1부터 오름차순으로 출력되어야 함

```
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
$ hw2_1
##### Step 1 #####
##### Step 2 #####
PID 3 exited
##### Step 3 #####
PID 5 exited
##### Step 4 #####
PID 4 exited
$
```

3. 제출

❖ 제출

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

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

- 아래의 명령어를 통해 hw2.patch파일과 xv6-public-hw2 디렉토리를 함께 압축

```
~$ sudo zip -r 학번-hw2.zip hw2.patch ./xv6-public-hw2
```

- 압축 폴더(학번-hw2.zip)를 PLATO에 업로드

- 과제 관련 문의는 교수님이 아니라 조교 메일(inohzzang@pusan.ac.kr)로 문의 바랍니다.