

운영체제 2024-1

Assignment 01. xv6 System Call

2024년 5월 13일

부산대학교 정보컴퓨터공학부

Prof. 김원석

1. xv6

- ❖ MIT에서 교육 목적으로 개발한 운영체제
- ❖ UNIX V6를 현대의 x86 프로세서 및 RISC-V 시스템을 위해 ANSI C를 사용하여 재구현(Re-implementation)
- ❖ Commentary book on XV6
 - [xv6 – DRAFT as of September 4, 2018 \(mit.edu\)](#)

1. xv6

❖ 실습 내용

- 시스템 콜
- CPU 스케줄링
- 가상메모리와 페이징
- etc

❖ xv6 구동을 위해 VirtualBox를 활용하여 Linux 가상환경을 구축

2. Linux 운영체제 다운로드

- ❖ Ubuntu 22.04 LTS(.iso 파일)를 다운로드
- ❖ [Ubuntu 22.04 LTS Download](#) * 서버 이미지를 사용해도 무방합니다.

Ubuntu 22.04.4 LTS (Jammy Jellyfish)

Select an image

Ubuntu is distributed on three types of images described below.

Desktop image

The desktop image allows you to try Ubuntu without changing your computer at all, and at your option to install it permanently later. This type of image is what most people will want to use. You will need at least 1024MiB of RAM to install from this image.

64-bit PC (AMD64) desktop image

Choose this if you have a computer based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron, EM64T Xeon, Core 2). Choose this if you are at all unsure.

3. VirtualBox 세팅

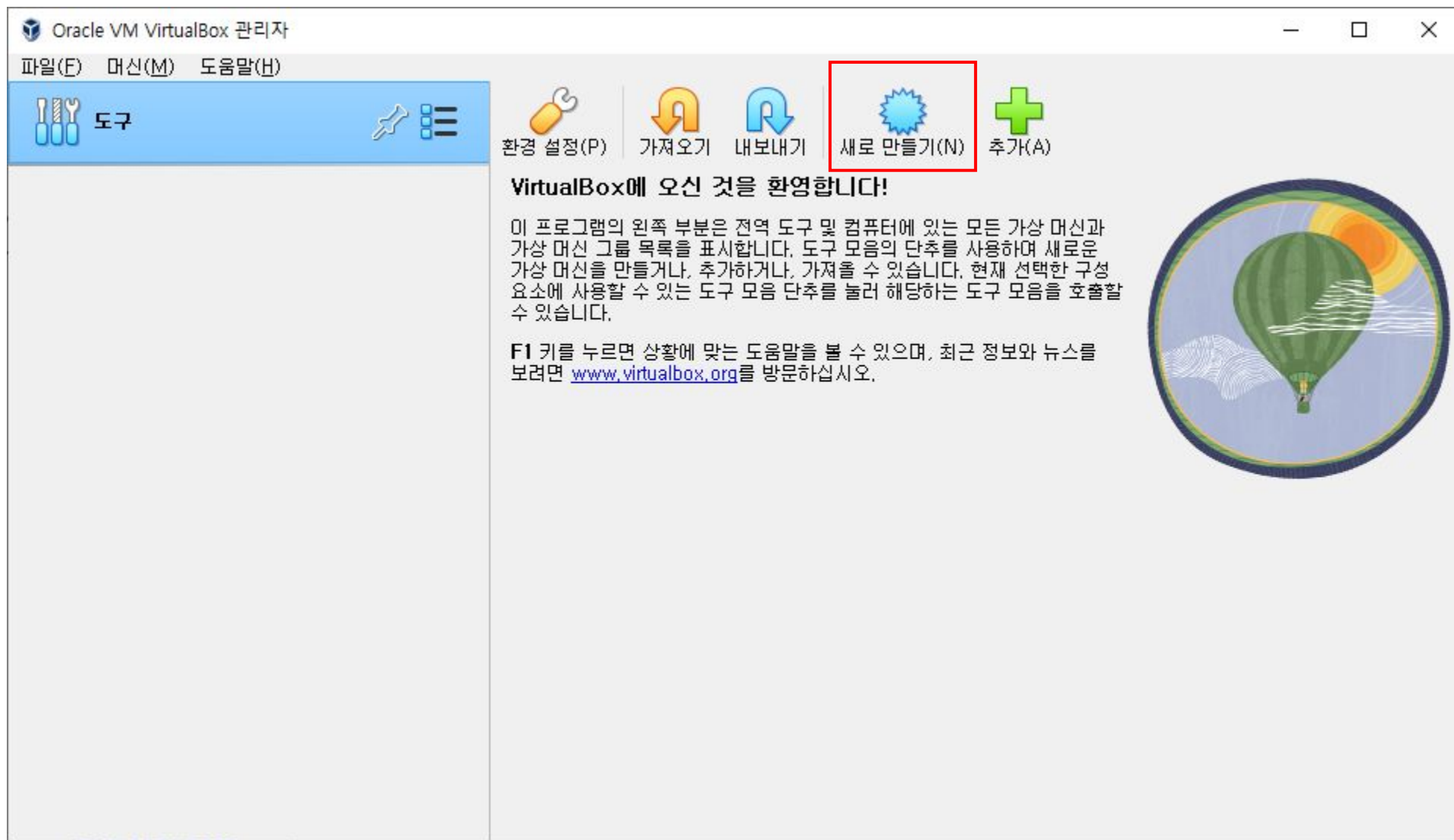
- ❖ 본인 PC의 OS에 맞는 버전의 VirtualBox 다운로드 후 설치
- ❖ [VirtualBox Download](#)



The binaries are released under the terms of the GPL version 3.

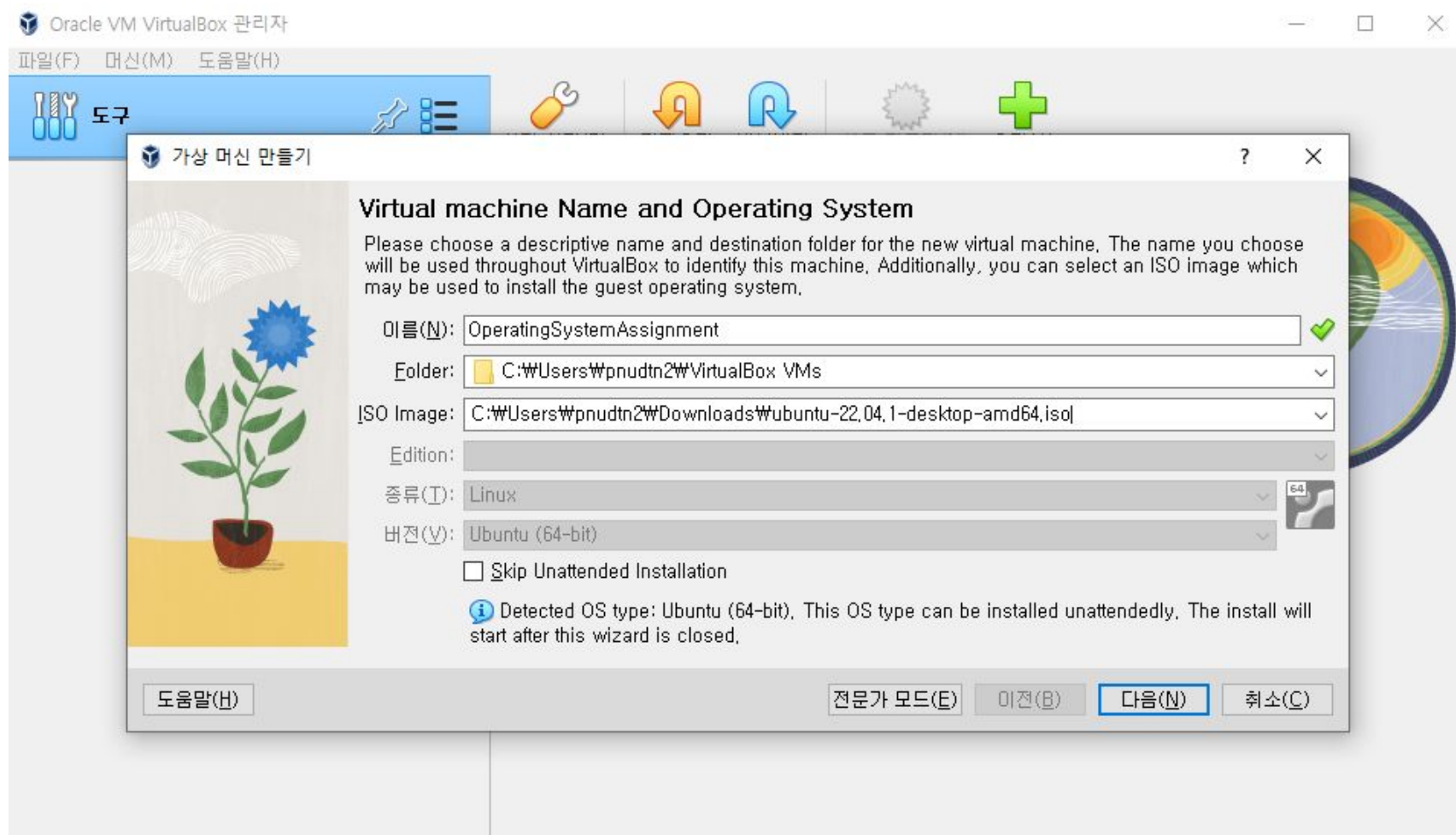
3. VirtualBox 세팅

❖ VirtualBox 실행 후 새로 만들기 클릭



3. VirtualBox 세팅

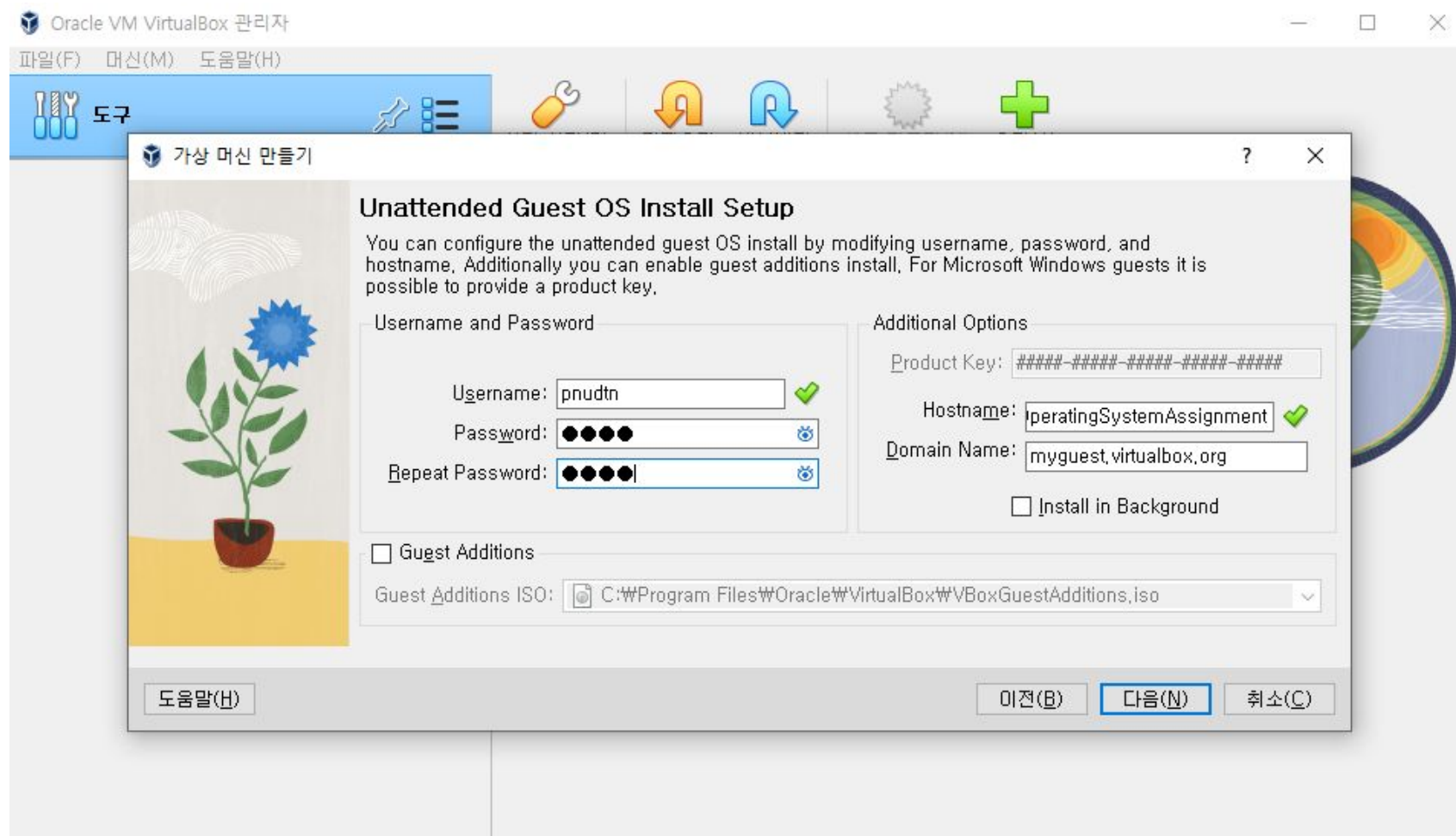
- ❖ 가상 머신의 이름과 가상 머신이 설치될 경로를 임의로 지정하고, 다운로드한 ubuntu iso 파일을 찾아 선택



3. VirtualBox 세팅

❖ 해당 가상 머신에서 사용할 계정명과 비밀번호를 입력

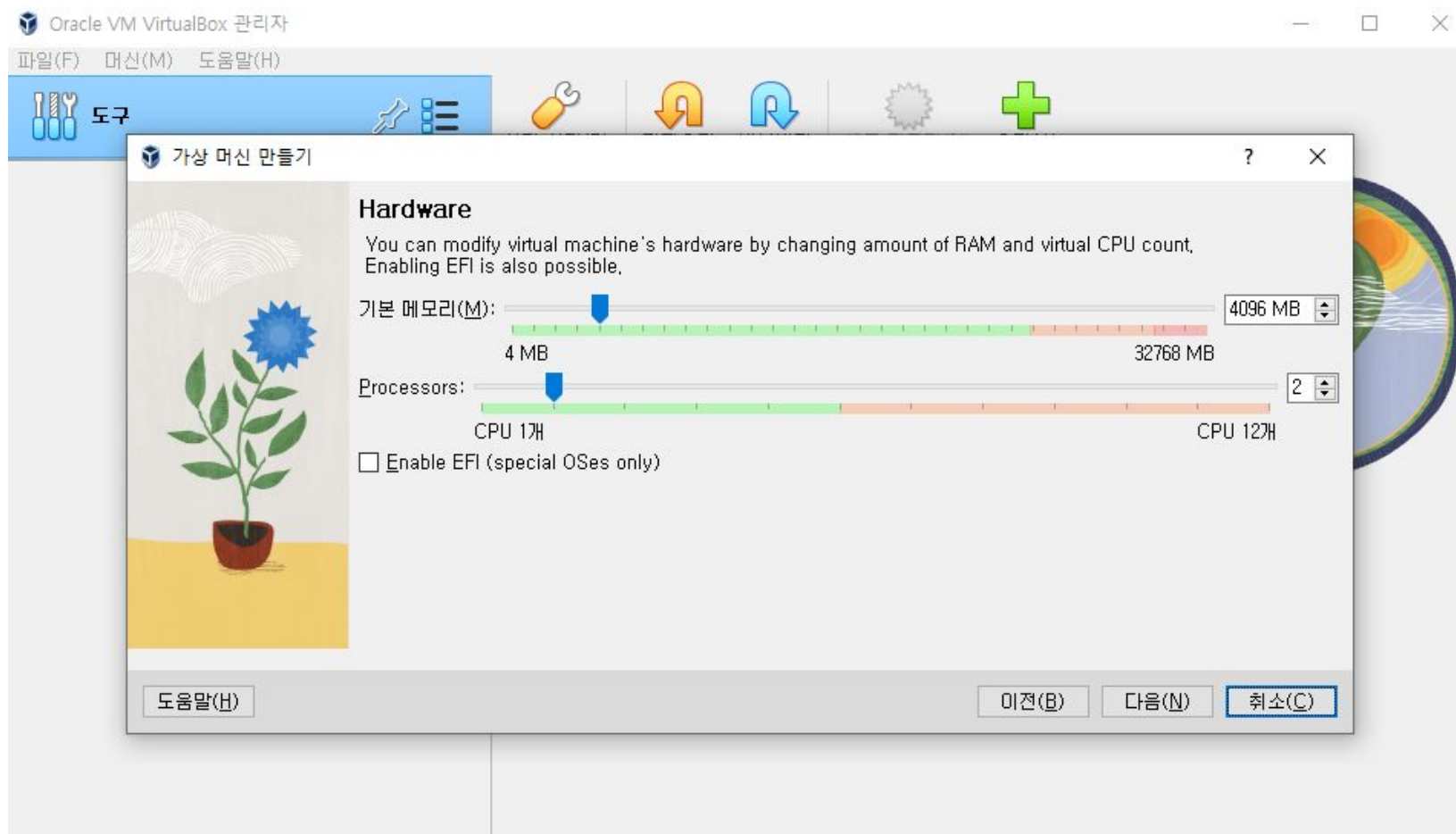
- 계정명과 비밀번호는 유실되지 않도록 반드시 메모



3. VirtualBox 세팅

❖ 가상 머신의 메모리 크기와 프로세서 수를 본인 PC의 사양에 맞게 조정

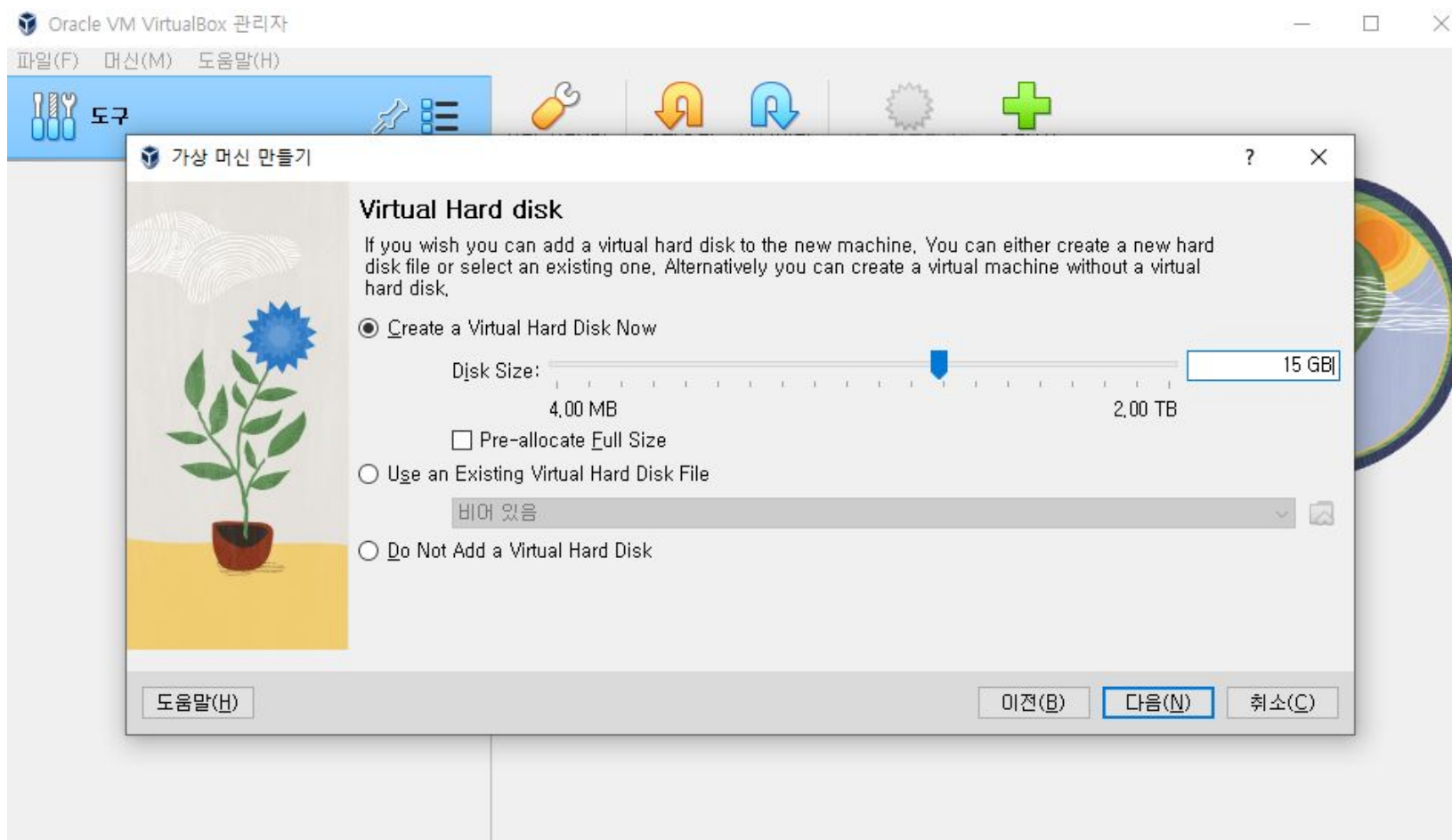
- Ubuntu 22.04 LTS 버전은 4GB의 메모리와 2 코어를 권장



3. VirtualBox 세팅

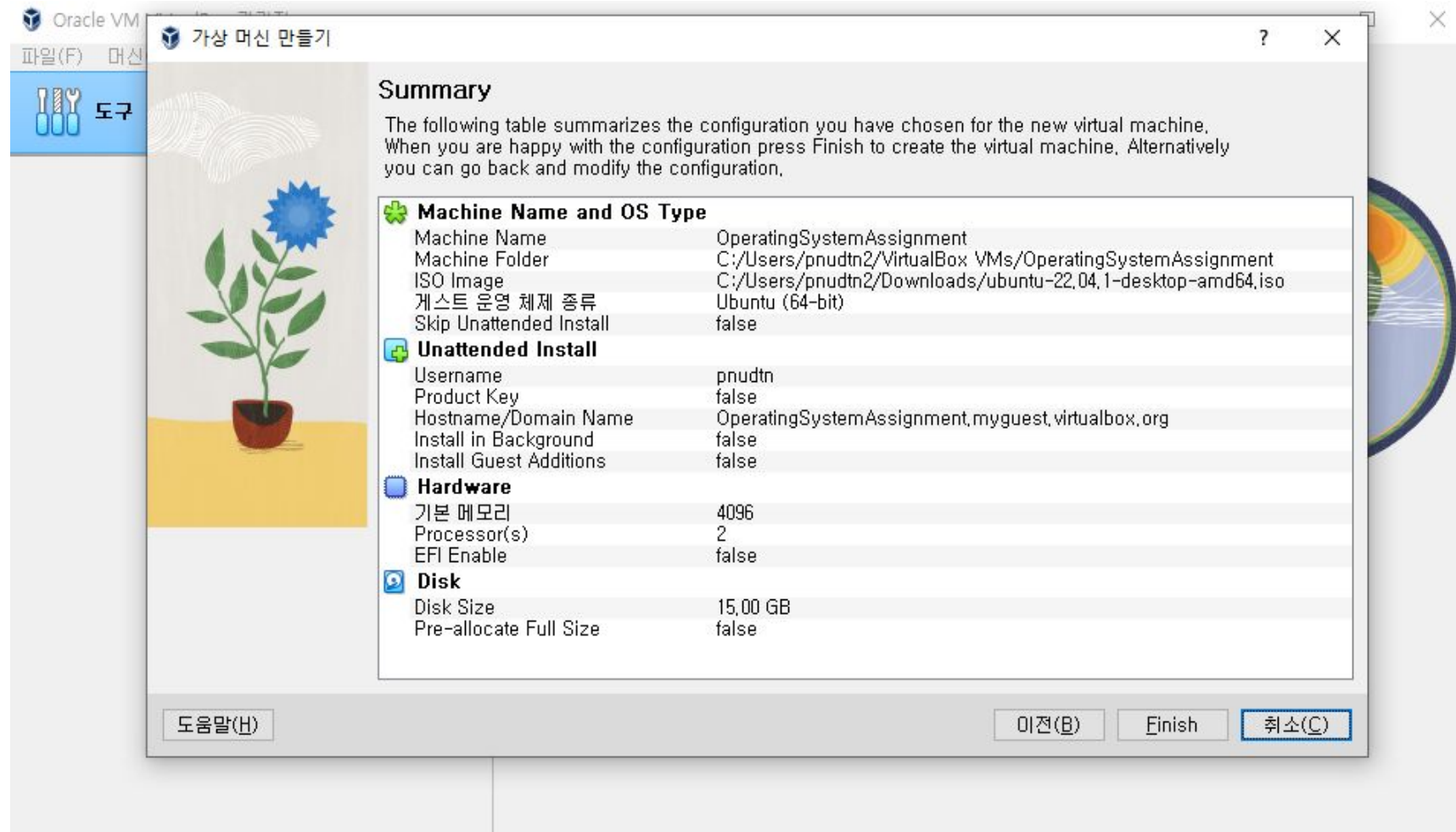
❖ 가상 머신의 하드 디스크 크기를 조정

- 가상 머신을 본 교과목의 과제 용도로만 사용시 15 GB 정도면 충분



3. VirtualBox 세팅

- ❖ 세팅이 정상적으로 되었는지 확인하고 'Finish' 버튼을 클릭하면 자동으로 가상 머신이 설치됨



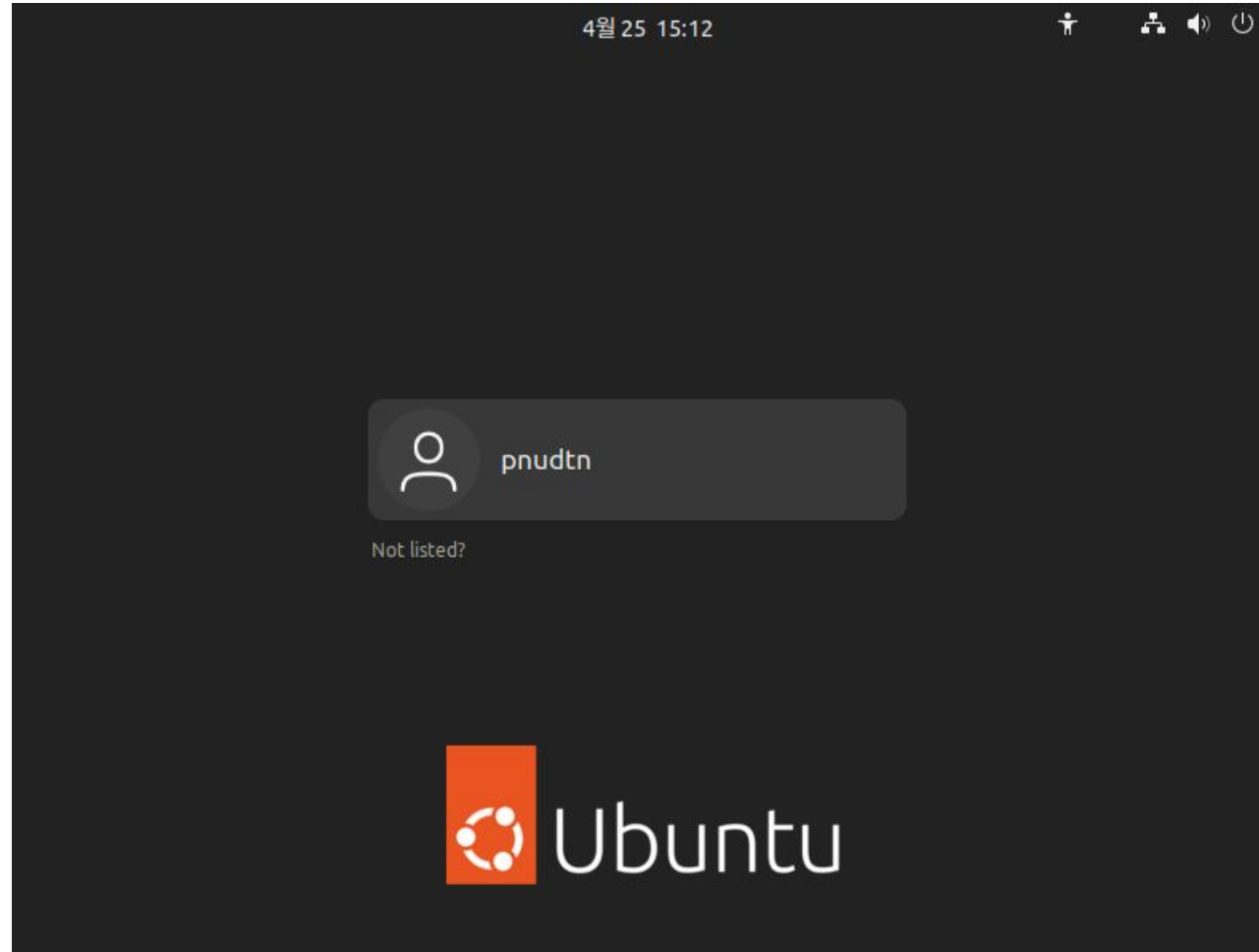
3. VirtualBox 세팅

- ❖ 설치된 가상 머신을 시작하면 몇 분간 추가 세팅을 진행함



4. xv6 설치 및 빌드

- ❖ 아래의 화면이 나올 때 까지 대기 후 6p에서 설정한 비밀번호로 로그인



4. xv6 설치 및 빌드

❖ 로그인된 유저 계정에 관리자 권한을 부여

- 터미널에 아래의 명령어를 입력하여 유저 계정을 관리자 그룹에 추가하고 sudo 명령어를 사용할 수 있도록 함

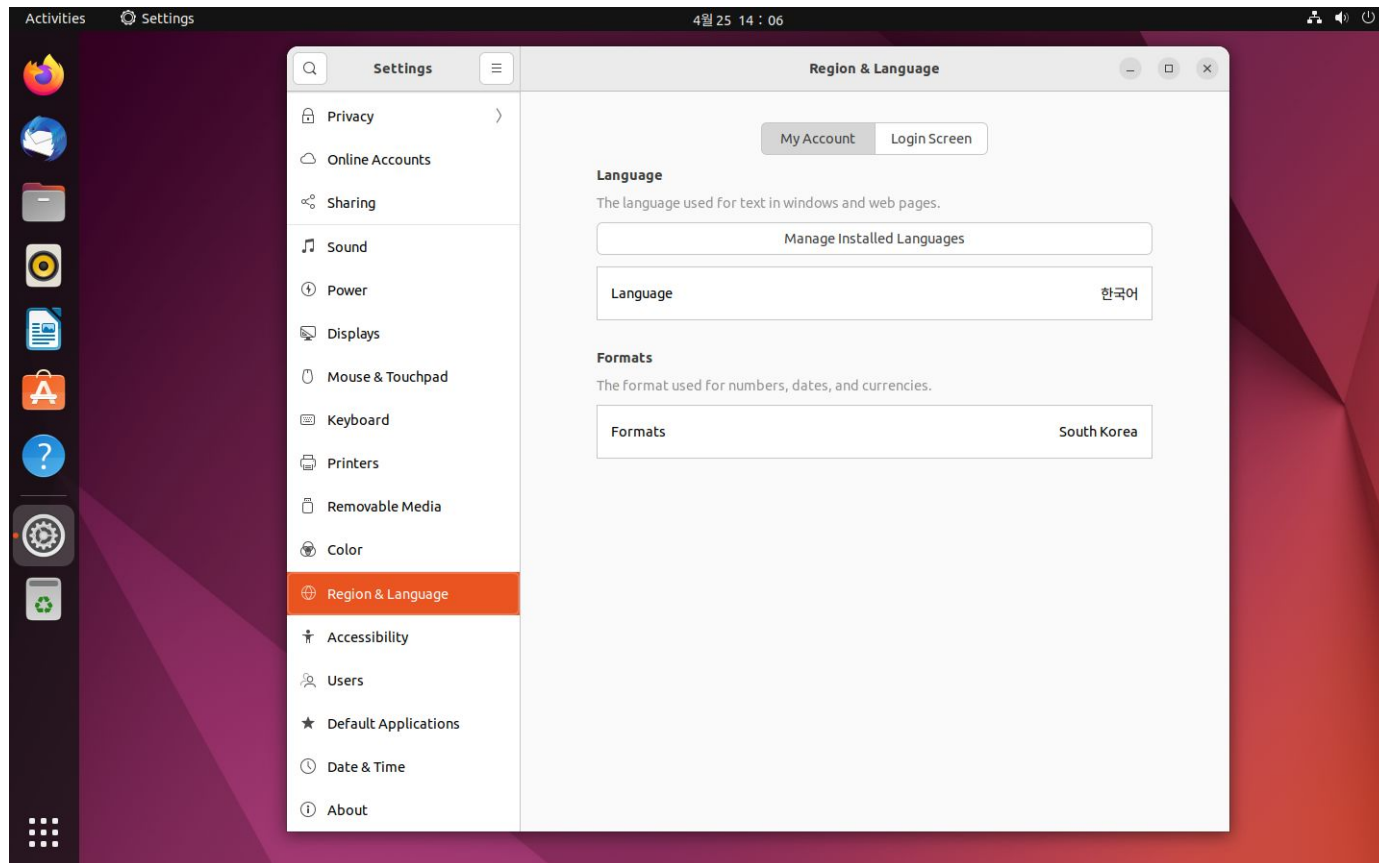
```
$ sudo su
$ usermod -aG sudo {username}
$ grep "sudo" /etc/group      # 해당 명령어로 유저 계정이 sudo 그룹에 추가되었는지 확인
$ su {username}
```

```
pnudtn@OperatingSystemAssignment:~$ sudo su
[sudo] password for pnudtn: 설정한 패스워드 입력
root@OperatingSystemAssignment:/home/pnudtn# usermod -aG sudo pnudtn
root@OperatingSystemAssignment:/home/pnudtn# grep "sudo" /etc/group
sudo:x:27:pnudtn
root@OperatingSystemAssignment:/home/pnudtn# su pnudtn
pnudtn@OperatingSystemAssignment:~$ S
```

4. xv6 설치 및 빌드

❖ 터미널이 열리지 않는 버그 발생 시

- Settings의 Region & Language 탭에서 Language를 “English (United States)”에서 다른 언어(ex. 한국어)로 변경하면 해결됨
- 이후에 다시 “English (United States)”로 되돌려도 무관함



4. xv6 설치 및 빌드

❖ sudo 명령어를 사용할 수 없는 경우

- `username` is not in the sudoers file. 이라는 로그가 출력될 경우, 다음과 같은 방법으로 해결

```
$ su
# vi /etc/sudoers
```

파일 내에 다음 구문 입력
`username ALL=(ALL:ALL) ALL`

입력 후 Esc를 누르고, `:wq!`로 저장

```
user@ubuntu:~$ sudo su
[sudo] password for user:
user is not in the sudoers file. This i
user@ubuntu:~$ su
Password:
root@ubuntu:/home/user# vi /etc/sudoers
```

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
user    ALL=(ALL:ALL) ALL
# Members of the admin group may gain root privileges
%admin   ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

:wq!
```


4. xv6 설치 및 빌드

❖ git 설치 후 xv6 설치

```
$ sudo apt install git
```

```
$ sudo git clone https://github.com/mit-pdos/xv6-public.git
```

```
pnudtn@OperatingSystemAssignment:~$ sudo apt install git
[sudo] password for pnudtn: 설정한 패스워드 입력
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  systemd-hwe-hwdb
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb
  git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 318 not upgraded.
Need to get 4,121 kB of archives.
After this operation, 20.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y Y 입력

pnudtn@OperatingSystemAssignment:~$ sudo git clone https://github.com/mit-pdos/xv6-public.git
Cloning into 'xv6-public'...
remote: Enumerating objects: 13990, done.
remote: Total 13990 (delta 0), reused 0 (delta 0), pack-reused 13990
Receiving objects: 100% (13990/13990), 17.18 MiB | 8.33 MiB/s, done.
Resolving deltas: 100% (9537/9537), done.
```

4. xv6 설치 및 빌드

❖ xv6 실행을 위해 qemu, gcc 컴파일러 등을 추가 설치

```
$ sudo apt-get install build-essential qemu gcc-multilib  
$ sudo apt-get install qemu-system-x86
```

- QEMU (Quick EMUlator)
 - 오픈소스 에뮬레이터
 - x86 하드웨어 환경을 소프트웨어적으로 구현하여 xv6를 구동할 수 있게 함

xv6

QEMU

Ubuntu 22.04 Operating System

4. xv6 설치 및 빌드

- ❖ xv6-public 디렉토리에 설치된 것을 ls 명령어로 확인할 수 있음

```
$ ls
$ cd ./xv6-public
$ ls
```

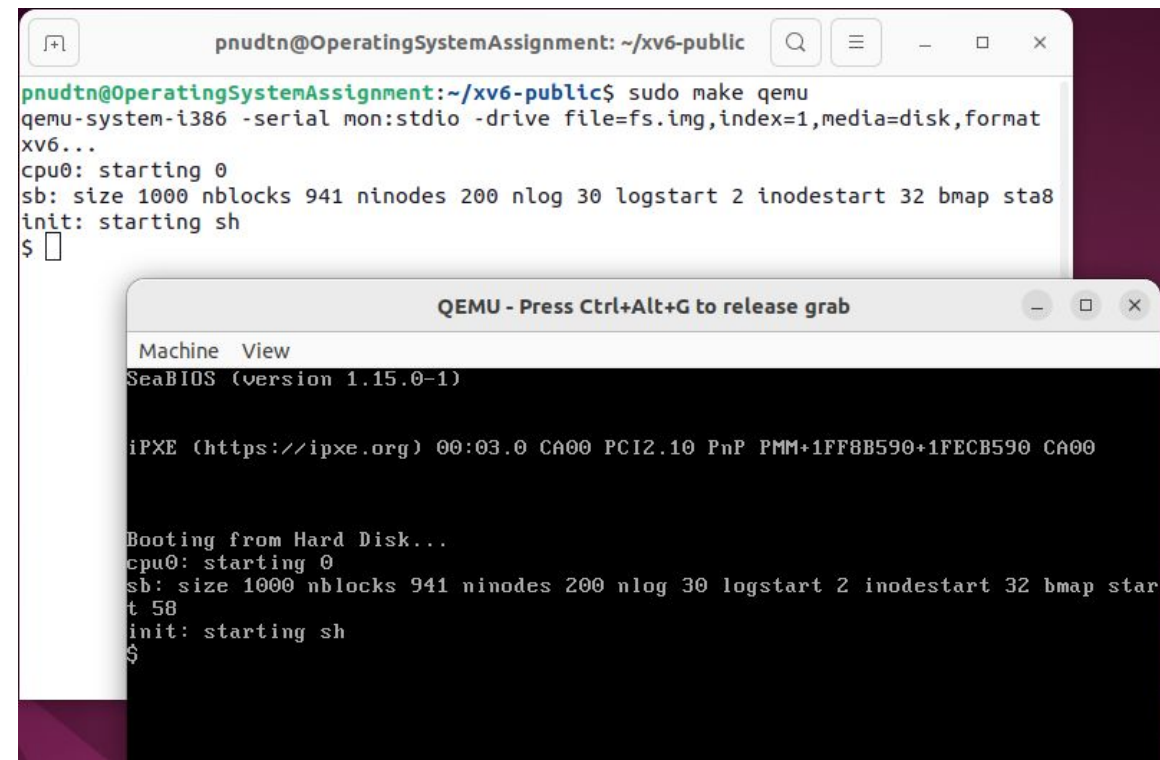
```
pnudtn@OperatingSystemAssignment:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos  xv6-public
pnudtn@OperatingSystemAssignment:~$ cd ./xv6-public/
pnudtn@OperatingSystemAssignment:~/xv6-public$ ls
asm.h          entryother.S  ioapic.c      memlayout.h  pr.pl         spinlock.h    traps.h
bio.c          entry.S       kalloc.c      mkdir.c      README        spinp         TRICKS
bootasm.S     exec.c        kbd.c         mkfs.c       rm.c          stat.h        types.h
bootmain.c    fcntl.h       kbd.h         mmu.h        runoff        stressfs.c    uart.c
buf.h          file.c        kernel.ld     mp.c         runoff1       string.c      ulib.c
BUGS           file.h        kill.c        mp.h         runoff.list   swtch.S       umalloc.c
cat.c          forktest.c    lapic.c       Notes        runoff.spec   syscall.c     user.h
console.c     fs.c          LICENSE       param.h      sh.c          syscall.h     usertests.c
cuth           fs.h          ln.c          picirq.c     show1         sysfile.c     usys.S
date.h         gdbutil       log.c         pipe.c       sign.pl       sysproc.c     vectors.pl
defs.h         grep.c        ls.c          printf.c     sleep1.p     toc.ftr       vm.c
dot-bochsrc    ide.c         main.c        printpcs     sleeplock.c  toc.hdr       wc.c
echo.c         init.c        Makefile      proc.c       sleeplock.h  trapasm.S     x86.h
elf.h          initcode.S    memide.c      proc.h       spinlock.c   trap.c        zombie.c
pnudtn@OperatingSystemAssignment:~/xv6-public$
```

4. xv6 설치 및 빌드

❖ 아래 명령어로 build & run을 한번에 가능

\$ sudo make qemu # 새로운 창에서 실행
또는
\$ sudo make qemu-nox # 현재 창에서 바로 실행

- 이후 쉘에 Ctrl A + X 를 입력하여 xv6 종료 가능
- 참고
 - \$sudo make 입력시 빌드만 진행하여 xv6.img를 생성
 - 아래와 같이 qemu-gdb 등 qemu와 관련된 다양한 make 옵션 존재
 - Makefile 에서 확인 가능

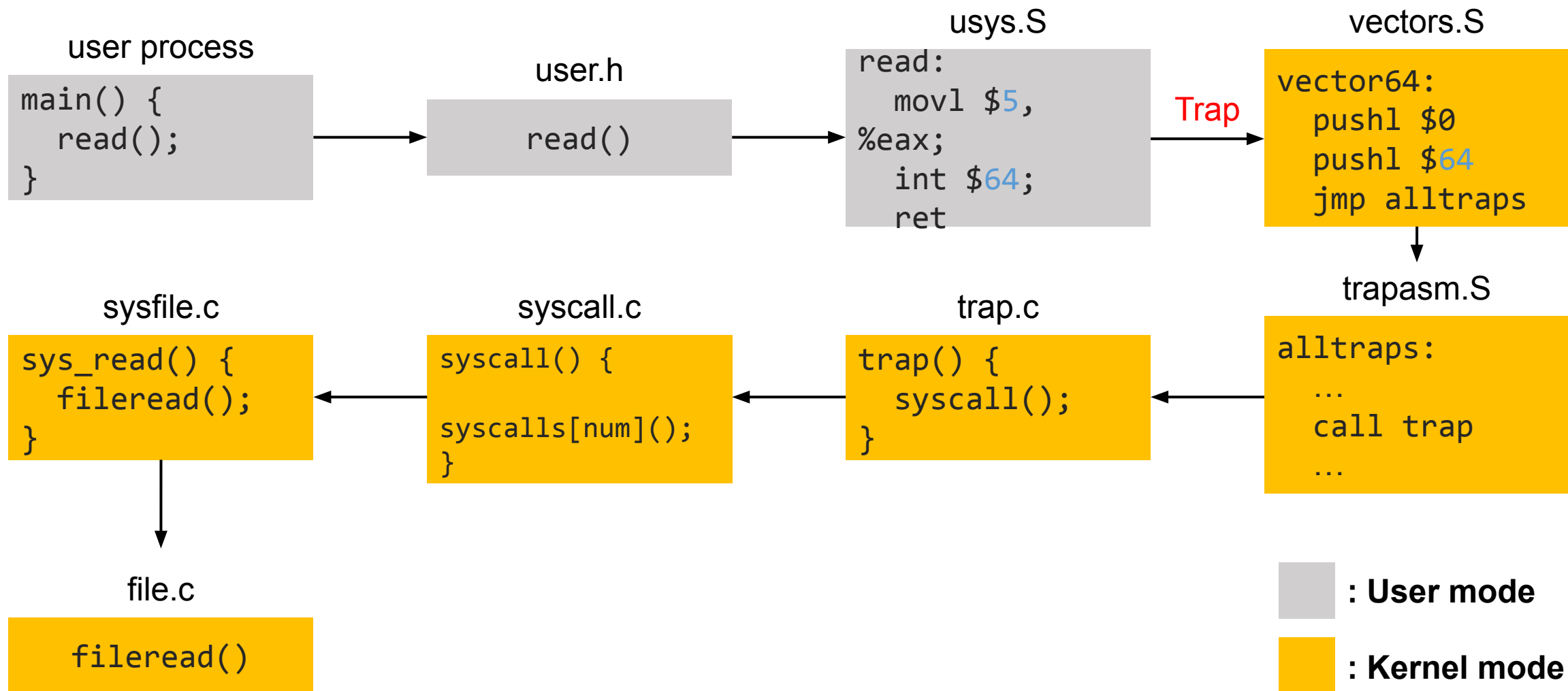


```
pnudtn@OperatingSystemAssignment: ~/xv6-public
pnudtn@OperatingSystemAssignment:~/xv6-public$ sudo make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
$
```

```
Makefile
228 qemu-memfs: xv6memfs.img
229     $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256
230
231 qemu-nox: fs.img xv6.img
232     $(QEMU) -nographic $(QEMUOPTS)
233
234 .gdbinit: .gdbinit.tmpl
235     sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@
236
237 qemu-gdb: fs.img xv6.img .gdbinit
238     @echo "**** Now run 'gdb'." 1>&2
239     $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUDEBUG)
240
241 qemu-nox-gdb: fs.img xv6.img .gdbinit
242     @echo "**** Now run 'gdb'." 1>&2
243     $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUDEBUG)
```

5. xv6에서의 시스템 콜 처리 과정

❖ Example: read system call



5. xv6에서의 시스템 콜 처리 과정

❖ xv6에서의 시스템 콜 처리 과정

- xv6는 시스템 콜을 trap으로 처리함
- trap0이 발생하면, CPU는 INT 명령어를 실행함
 - INT 명령어
 - 인터럽트를 발생시키는 역할을 하는 어셈블리 명령어
 - “INT x”의 형태를 가지며, 여기서 x는 인터럽트의 종류를 나타내는 파라미터
- xv6에서는 시스템 콜을 처리하기 위해 `usys.S` 파일에서 INT 명령어를 실행함
 - `int $T_SYSCALL;`

```
ASH usys.S
1  #include "syscall.h"
2  #include "traps.h"
3
4  #define SYSCALL(name) \
5      .globl name; \
6      name: \
7          movl $SYS_ ## name, %eax; \
8          int $T_SYSCALL; \
9          ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
```

5. xv6에서의 시스템 콜 처리 과정

- ❖ traps.h 내 각 트랩의 번호가 정의되어 있음

C traps.h > ...

```
25 // These are arbitrarily chosen, but with care not to overlap
26 // processor defined exceptions or interrupt vectors.
27 #define T_SYSCALL      64      // system call
28 #define T_DEFAULT      500     // catchall
```

- ❖ syscall.h 내 각 시스템 콜의 번호가 정의되어 있음

C syscall.h > ...

```
1 // System call numbers
2 #define SYS_fork      1
3 #define SYS_exit      2
4 #define SYS_wait      3
5 #define SYS_pipe      4
6 #define SYS_read      5
7 #define SYS_kill      6
8 #define SYS_exec      7
9 #define SYS_fstat     8
10 #define SYS_chdir     9
11 #define SYS_dup      10
```

ASM usys.S

```
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
```

```
.globl read;
read:
    movl $5, %eax;
    int $64;
    ret
```

5. xv6에서의 시스템 콜 처리 과정

❖ “INT n” 명령어는 아래의 작업을 실행함

- 레지스터 `eip`가 Program Counter(PC), `esp`가 Stack Pointer(SP) 라고 할 때
- Interrupt Descriptor Table(IDT)의 n 번째 엔트리를 fetch
- SP의 값을 다른 레지스터에 임시 저장
- `esp`를 커널 스택의 위치로 전환하여 커널 영역으로 이동
- 인터럽트 종료 시 유저 프로세스로 돌아갈 수 있도록, 유저 프로세스의 PC와 SP를 커널 스택에 저장
- IDT에서 새로운 PC를 `eip`에 저장함
 - 새로운 PC는 커널 트랩 핸들러를 가리킴

```
.globl read;
read:
    movl $5, %eax;
    int $64;
    ret
```


5. xv6에서의 시스템 콜 처리 과정

❖ (참고) Interrupt Descriptor Table(IDT)는 빌드 도중 trap.c 내에서 초기화됨

- IDT는 인터럽트 또는 트랩의 정보가 담겨있는 테이블, vector 테이블을 이용해 초기화
- 시스템 콜에 해당하는 테이블 엔트리는 트랩으로서 접근됨
- Descriptor Privilege Level(DPL)을 User level(=3)로 하여 사용자 프로그램에서도 명시적으로 INT 명령어를 통해 호출 가능하도록 함

C trap.c > ...

```
11 // Interrupt descriptor table (shared by all CPUs).
12 struct gatedesc idt[256];
13 extern uint vectors[]; // in vectors.S: array of 256 entry pointers
14 struct spinlock tickslock;
15 uint ticks;
16
17 void
18 tvinit(void)
19 {
20     int i;
21
22     for(i = 0; i < 256; i++)
23         SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
24     SETGATE(idt[T_SYSCALL], 1, SEG_KCODE<<3, vectors[T_SYSCALL], DPL_USER);
25
26     initlock(&tickslock, "time");
27 }
```

C mmu.h > ...

```
160 // Set up a normal interrupt/trap gate descriptor.
161 // - istrap: 1 for a trap (= exception) gate, 0 for an interrupt gate.
162 // - interrupt gate clears FL_IF, trap gate leaves FL_IF alone
163 // - sel: Code segment selector for interrupt/trap handler
164 // - off: Offset in code segment for interrupt/trap handler
165 // - dpl: Descriptor Privilege Level -
166 //       the privilege level required for software to invoke
167 //       this interrupt/trap gate explicitly using an int instruction.
168 #define SETGATE(gate, istrap, sel, off, d) \
169 { \
170     (gate).off_15_0 = (uint)(off) & 0xffff; \
171     (gate).cs = (sel); \
172     (gate).args = 0; \
173     (gate).rsv1 = 0; \
174     (gate).type = (istrap) ? STS_TG32 : STS_IG32; \
175     (gate).s = 0; \
176     (gate).dpl = (d); \
177     (gate).p = 1; \
178     (gate).off_31_16 = (uint)(off) >> 16; \
179 }
```

5. xv6에서의 시스템 콜 처리 과정

❖ (참고) vector 테이블은 vectors.S
내에 정의되어 있음

- 시스템 콜에 대한 벡터인 vector64에서
alltraps로 jump
- alltraps에서 트랩에 필요한 트랩 프레임을 쌓고
trap을 호출

ASM vectors.S

```
1278 # vector table
1279 .data
1280 .globl vectors
1281 vectors:
1282     .long vector0
1283     .long vector1
1284     .long vector2
1285     .long vector3
1286     .long vector4
1287     .long vector5
1288     .long vector6
1289     .long vector7
```

ASM vectors.S

```
317 .globl vector64
318 vector64:
319     pushl $0
320     pushl $64
321     jmp alltraps
```

ASM trapasm.S

```
3 # vectors.S sends all traps here.
4 .globl alltraps
5 alltraps:
6 # Build trap frame.
7     pushl %ds
8     pushl %es
9     pushl %fs
10    pushl %gs
11    pushal
12
13 # Set up data segments.
14    movw $(SEG_KDATA<<3), %ax
15    movw %ax, %ds
16    movw %ax, %es
17
18 # Call trap(tf), where tf=%esp
19    pushl %esp
20    call trap
21    addl $4, %esp
```

5. xv6에서의 시스템 콜 처리 과정

❖ (참고) 트랩 프레임에 대한 상세한 내용은 코멘터리 북 42p-44p 참고

- [xv6 – DRAFT as of September 4, 2018 \(mit.edu\)](#)

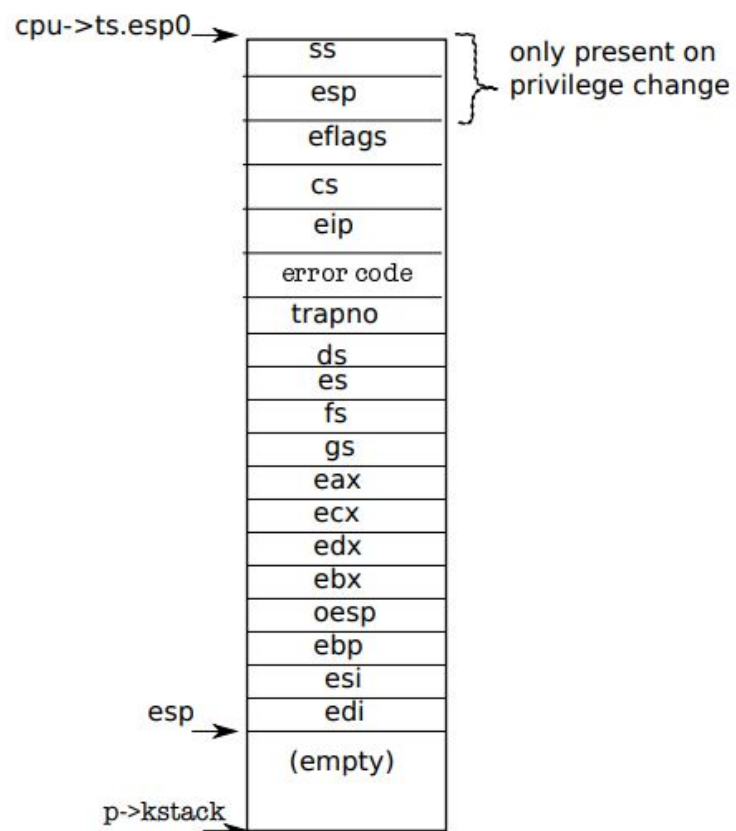


Figure 3-2. The trapframe on the kernel stack

5. xv6에서의 시스템 콜 처리 과정

❖ trap.c 내 trap 함수가 정의되어 있음

- trapframe 구조체의 트랩 번호를 확인하여 T_SYSCALL(=64)이면 syscall 함수를 호출

C trap.c > ...

```
35 //PAGEBREAK: 41
36 void
37 trap(struct trapframe *tf)
38 {
39     if(tf->trapno == T_SYSCALL){
40         if(myproc()->killed)
41             exit();
42         myproc()->tf = tf;
43         syscall();
44         if(myproc()->killed)
45             exit();
46         return;
47     }
```

C x86.h > ...

```
150 struct trapframe {
151     // registers as pushed by pusha
152     uint edi;
153     uint esi;
154     uint ebp;
155     uint oesp;      // useless & ignored
156     uint ebx;
157     uint edx;
158     uint ecx;
159     uint eax;
160
161     // rest of trap frame
162     ushort gs;
163     ushort padding1;
164     ushort fs;
165     ushort padding2;
166     ushort es;
167     ushort padding3;
168     ushort ds;
169     ushort padding4;
170     uint trapno;
```


5. xv6에서의 시스템 콜 처리 과정

❖ syscall.c 내 syscall 함수가 정의되어 있음

- 매핑된 함수 포인터를 이용해 각 시스템 콜 함수를 호출
- sys_read, sys_write 등 파일 관련 시스템 콜은 sysfile.c에 정의되어 있음
- sys_fork, sys_exit 등 프로세스 관련 시스템 콜은 sysproc.c에 정의되어 있음

C syscall.c > ...

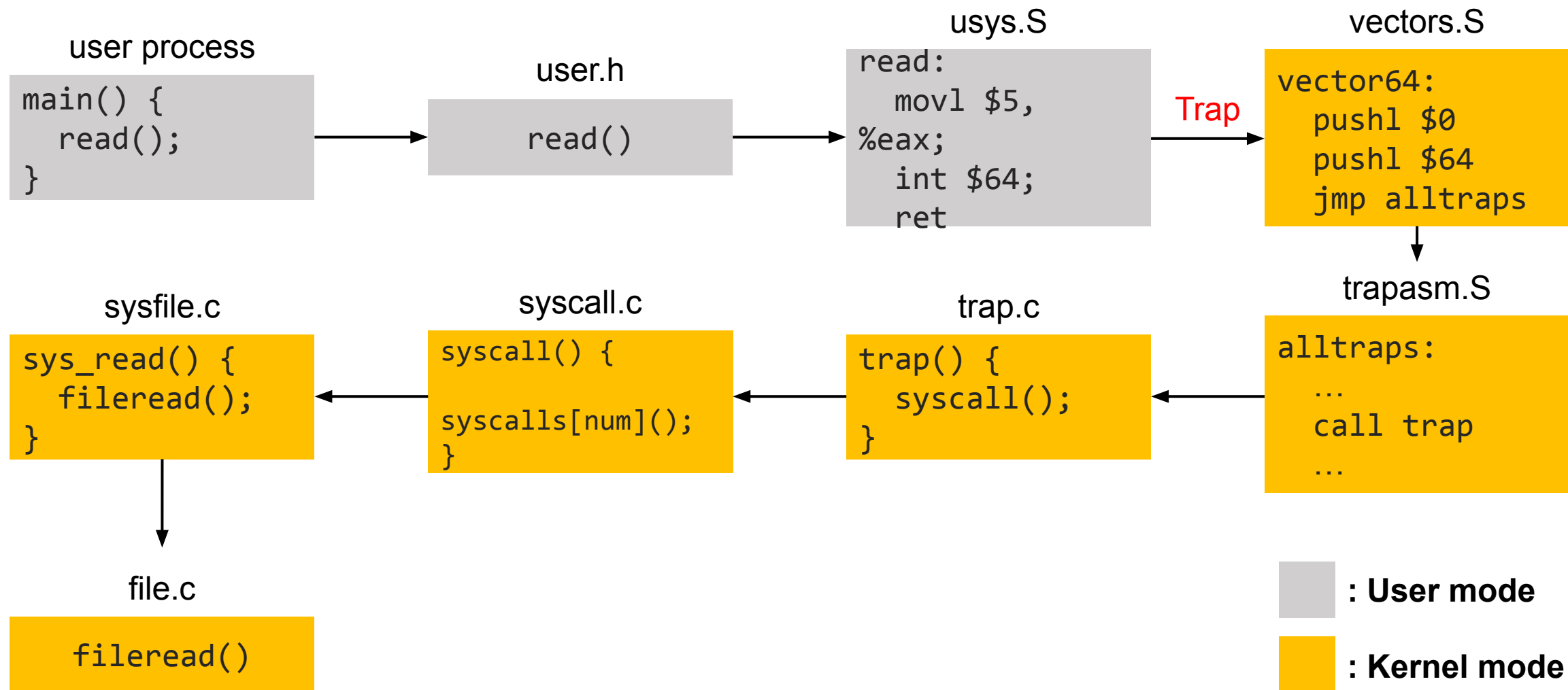
```
131 void
132 syscall(void)
133 {
134     int num;
135     struct proc *curproc = myproc();
136
137     num = curproc->tf->eax;
138     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
139         curproc->tf->eax = syscalls[num]();
140     } else {
141         cprintf("%d %s: unknown sys call %d\n",
142             curproc->pid, curproc->name, num);
143         curproc->tf->eax = -1;
144     }
145 }
```

C syscall.c > ...

```
107 static int (*syscalls[])(void) = {
108     [SYS_fork]    sys_fork,
109     [SYS_exit]    sys_exit,
110     [SYS_wait]    sys_wait,
111     [SYS_pipe]    sys_pipe,
112     [SYS_read]    sys_read,
113     [SYS_kill]    sys_kill,
114     [SYS_exec]    sys_exec,
115     [SYS_fstat]   sys_fstat,
116     [SYS_chdir]   sys_chdir,
117     [SYS_dup]     sys_dup,
118     [SYS_getpid]  sys_getpid,
119     [SYS_sbrk]    sys_sbrk,
120     [SYS_sleep]   sys_sleep,
121     [SYS_uptime]  sys_uptime,
122     [SYS_open]    sys_open,
123     [SYS_write]   sys_write,
124     [SYS_mknod]   sys_mknod,
125     [SYS_unlink]  sys_unlink,
126     [SYS_link]    sys_link,
127     [SYS_mkdir]   sys_mkdir,
128     [SYS_close]   sys_close,
129 };
```

5. xv6에서의 시스템 콜 처리 과정

❖ Example: read system call



5. xv6에서의 시스템 콜 처리 과정

❖ 시스템 콜의 구현 코드로 바로 jump 하지 않고 여러 과정을 거치는 이유

- 사용자 프로그램의 코드보다 OS를 더 높은 권한에서 실행해야 함
- 사용자 프로그램이 올바르게 커널 코드를 호출했음을 신뢰할 수 없음
- Ch02. OS Services – System Calls 참고

6. xv6 assignment.1

- ❖ 새로운 시스템 콜 구현
- ❖ 시스템 콜 테스트를 위한 유저 프로그램 구현

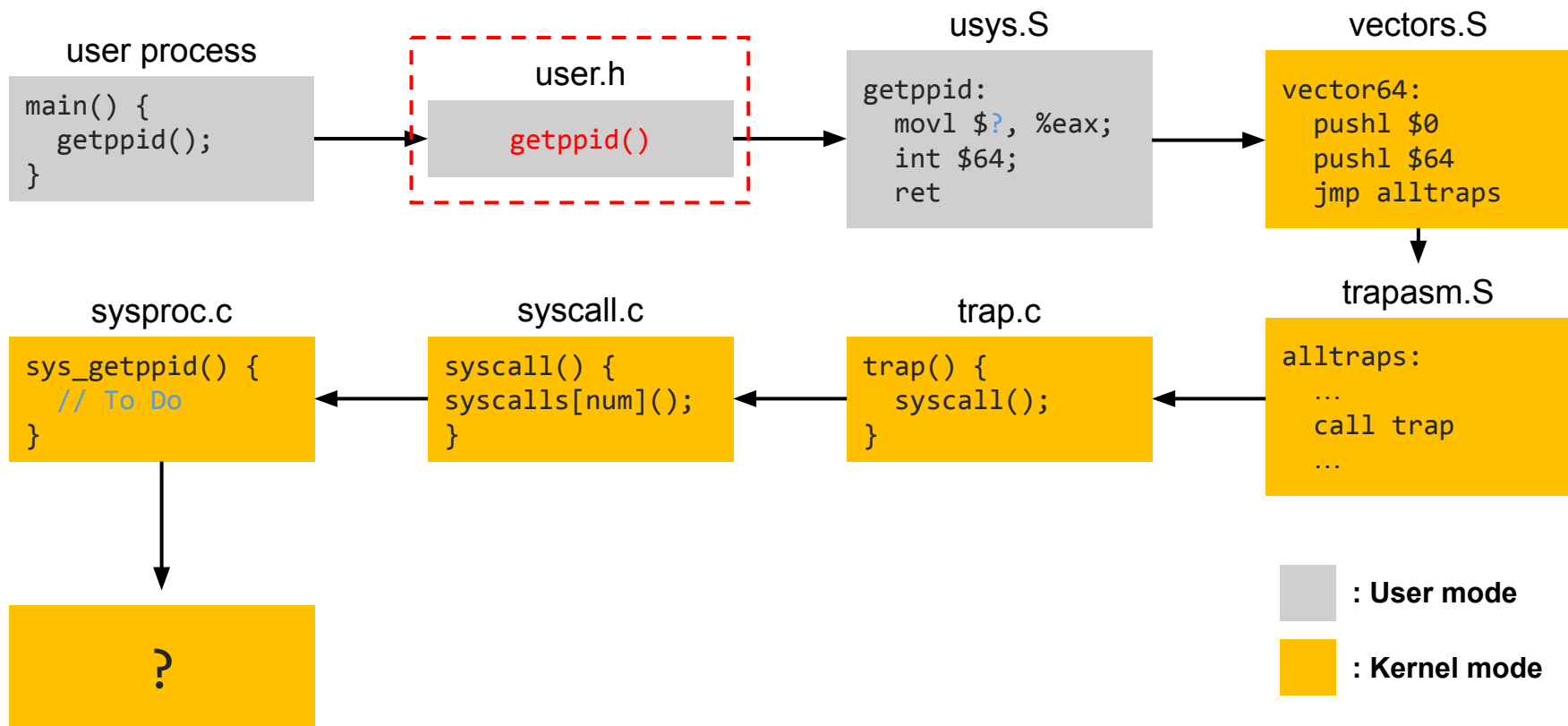
6-1. 새로운 시스템 콜 구현

❖ 부모 프로세스의 PID를 반환하는 시스템 콜 getppid를 구현

- `sysproc.c` 파일에서 현재 프로세스의 PID를 반환하는 `getpid` 시스템 콜을 참고
- `proc.h` 파일에서 프로세스의 상태를 저장하는 `proc` 구조체를 참고

6-1. 새로운 시스템 콜 구현 (API)

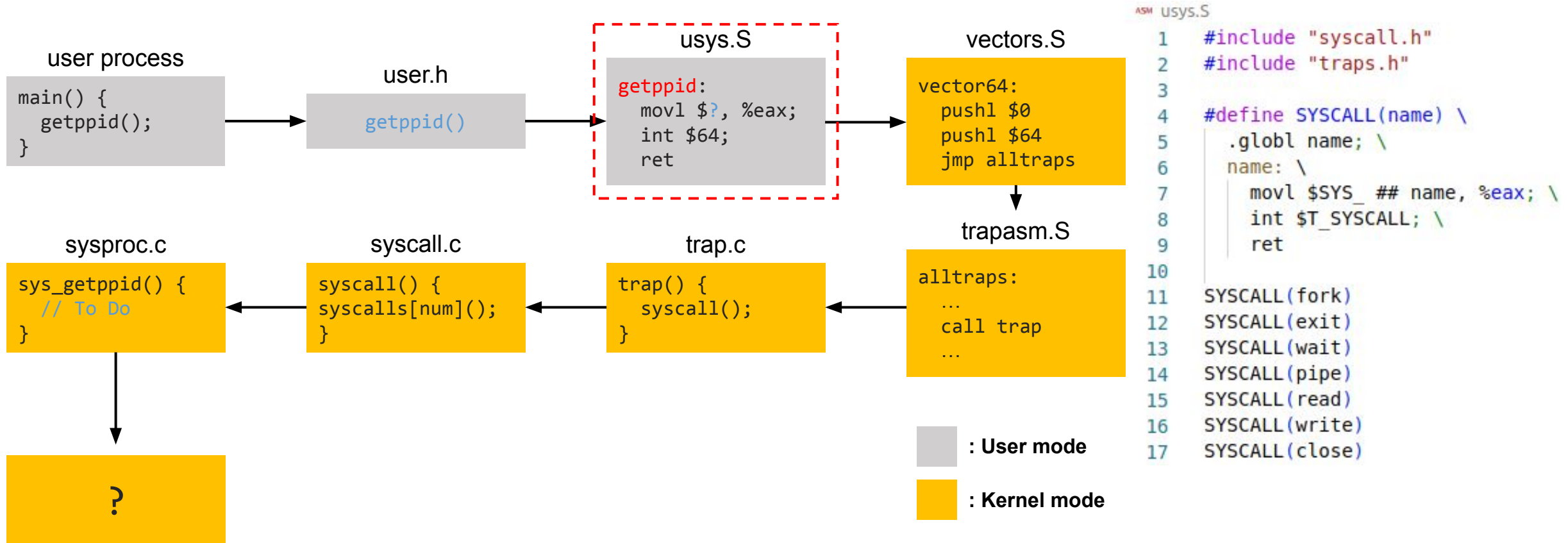
- ❖ user.h – 유저 프로세스에서 getppid 시스템 콜을 호출할 수 있도록 헤더 파일에 선언



```
C user.h > ...  
4  // system calls  
5  int fork(void);  
6  int exit(void) __attribute__((noreturn));  
7  int wait(void);  
8  int pipe(int*);  
9  int write(int, const void*, int);  
10 int read(int, void*, int);  
11 int close(int);  
12 int kill(int);  
13 int exec(char*, char**);  
14 int open(const char*, int);  
15 int mknod(const char*, short, short);  
16 int unlink(const char*);  
17 int fstat(int fd, struct stat*);  
18 int link(const char*, const char*);  
19 int mkdir(const char*);  
20 int chdir(const char*);  
21 int dup(int);  
22 int getpid(void);  
23 char* sbrk(int);  
24 int sleep(int);  
25 int uptime(void);
```

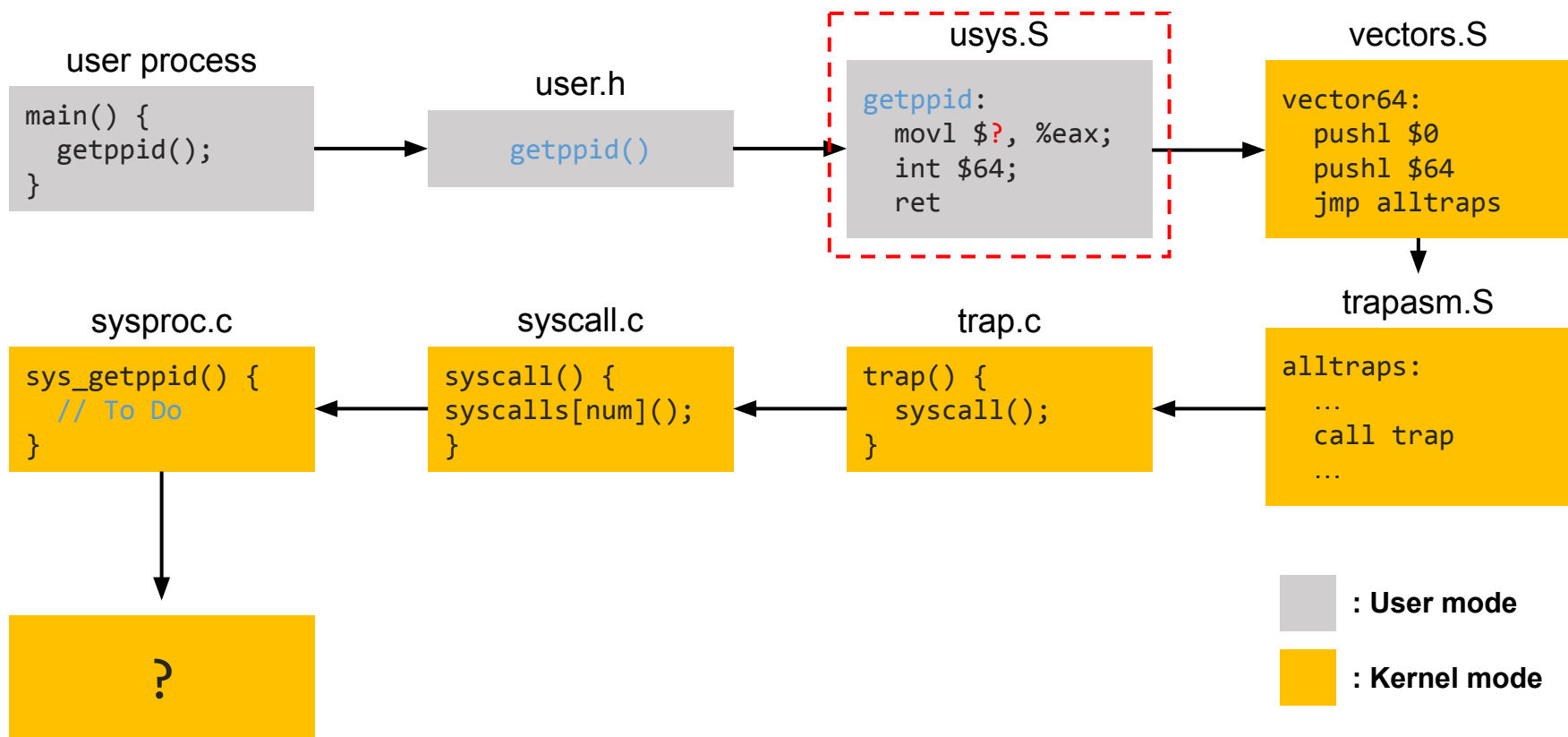
6-1. 새로운 시스템 콜 구현 (API)

- ❖ `usys.S` – `#define`으로 정의된 매크로 인자에 `getppid`를 기입하여 어셈블리 코드 생성



6-1. 새로운 시스템 콜 구현 (API)

❖ syscall.h – getpid 시스템 콜에 대한 번호 정의

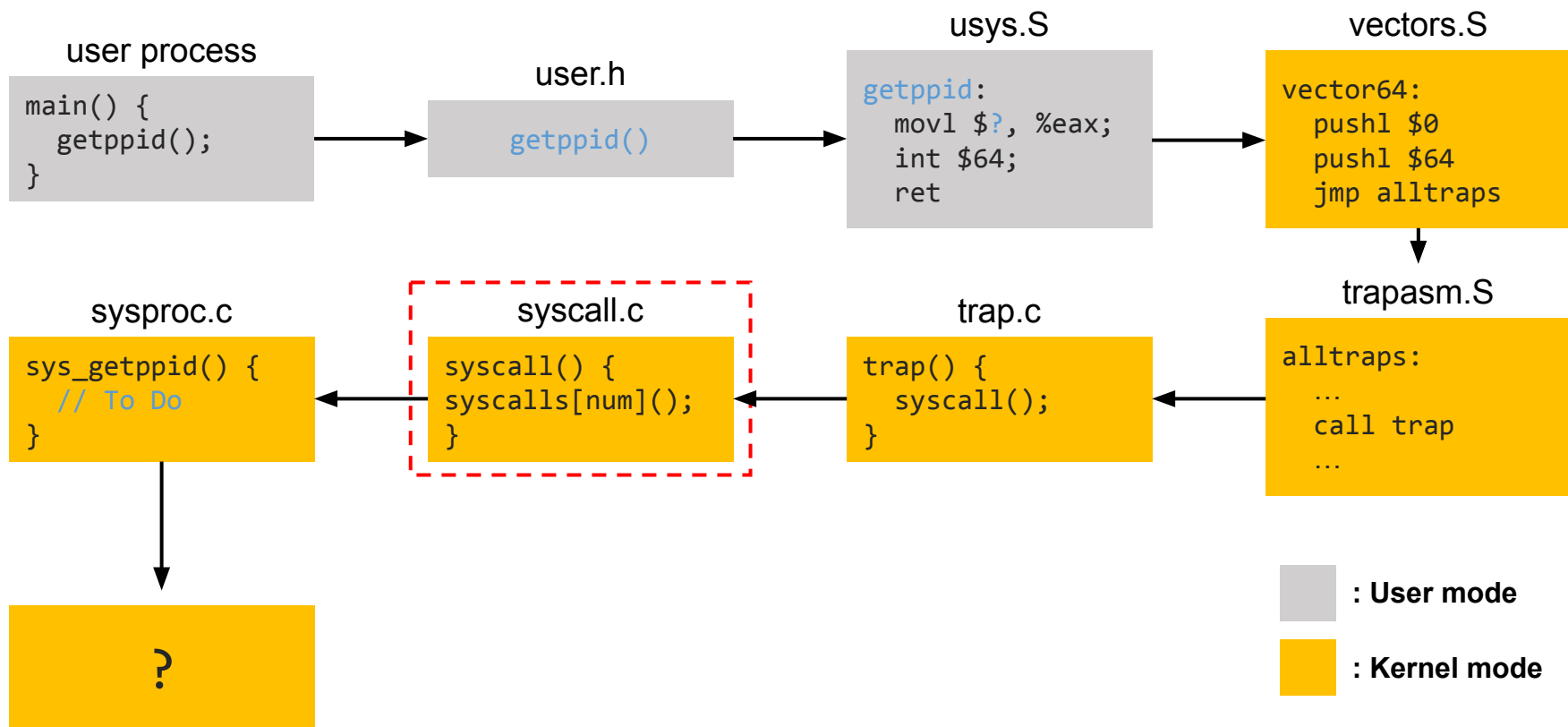


C syscall.h > ...

```
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
```

6-1. 새로운 시스템 콜 구현 (System Call Table)

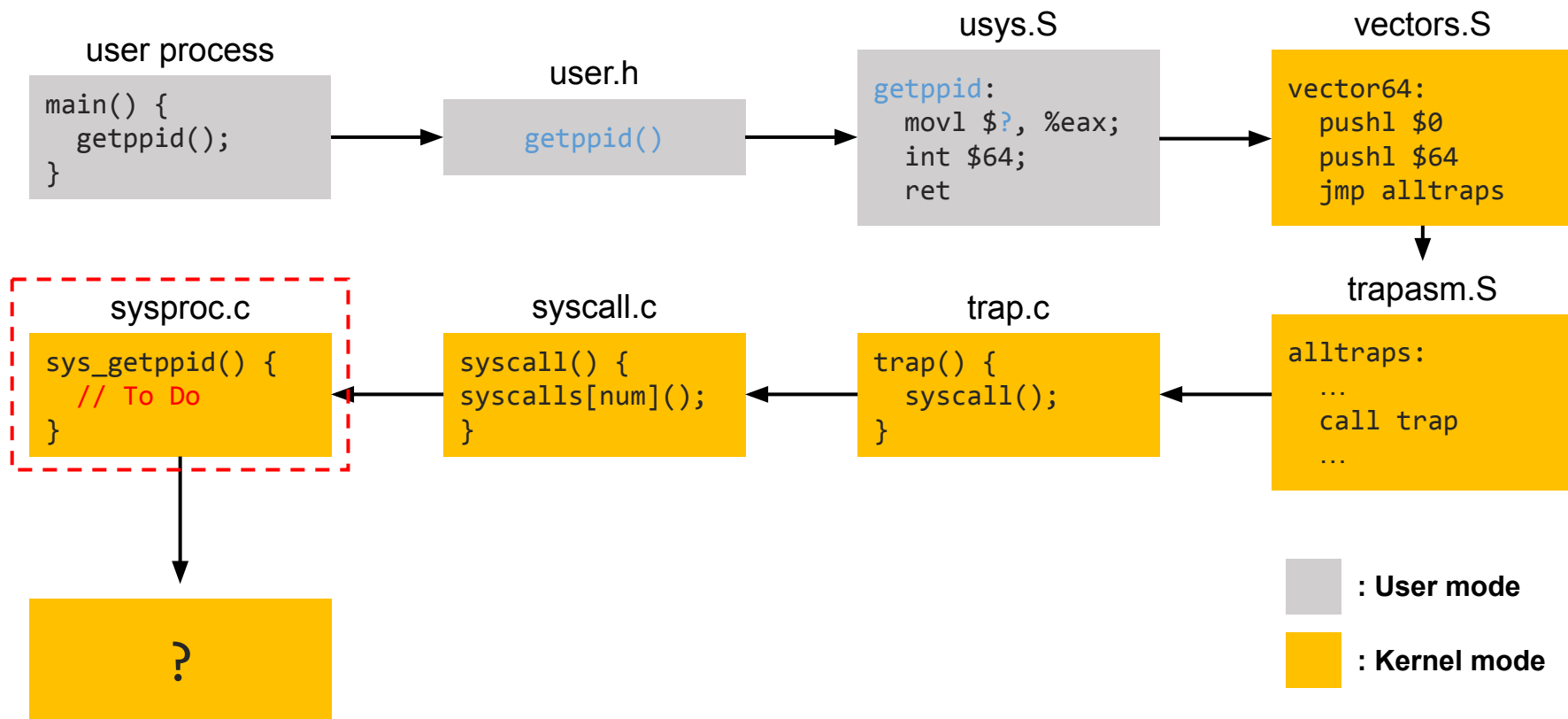
- ❖ syscall.c - 함수 포인터 배열에 getppid 시스템 콜에 대한 포인터를 추가



```
C syscall.c > ...  
107 static int (*syscalls[])(void) = {  
108 [SYS_fork] sys_fork,  
109 [SYS_exit] sys_exit,  
110 [SYS_wait] sys_wait,  
111 [SYS_pipe] sys_pipe,  
112 [SYS_read] sys_read,  
113 [SYS_kill] sys_kill,  
114 [SYS_exec] sys_exec,  
115 [SYS_fstat] sys_fstat,  
116 [SYS_chdir] sys_chdir,  
117 [SYS_dup] sys_dup,  
118 [SYS_getpid] sys_getpid,  
119 [SYS_sbrk] sys_sbrk,  
120 [SYS_sleep] sys_sleep,  
121 [SYS_uptime] sys_uptime,  
122 [SYS_open] sys_open,  
123 [SYS_write] sys_write,  
124 [SYS_mknod] sys_mknod,  
125 [SYS_unlink] sys_unlink,  
126 [SYS_link] sys_link,  
127 [SYS_mkdir] sys_mkdir,  
128 [SYS_close] sys_close,  
129 };
```

6-1. 새로운 시스템 콜 구현 (Handler)

❖ sysproc.c: getppid 시스템 콜의 실제 동작을 소스 코드로 구현



```
C sysproc.c > ...
10  int
11  sys_fork(void)
12  {
13      return fork();
14  }
15
16  int
17  sys_exit(void)
18  {
19      exit();
20      return 0; // not reached
21  }
22
23  int
24  sys_wait(void)
25  {
26      return wait();
27  }
28
29  int
30  sys_kill(void)
31  {
32      int pid;
33
34      if(argint(0, &pid) < 0)
35          return -1;
36      return kill(pid);
37  }
38
39  int
40  sys_getpid(void)
41  {
42      return myproc()->pid;
43  }
```

6-2. 시스템 콜 테스트를 위한 유저 프로그램 구현

❖ 오른쪽과 같이 hw1.c을 작성

- 본인의 학번과 이름을 출력하도록 수정
- fork를 활용하여 getppid의 동작을 확인할 수 있음

```
C hw1.c > ...
1  // hw1: user program for testing getppid system call
2
3  #include "types.h"
4  #include "stat.h"
5  #include "user.h"
6
7  int
8  main(void)
9  {
10     int pid;
11
12     printf(1, "sid: 202312345\n");
13     printf(1, "name: Gil-dong Hong\n");
14     printf(1, "current pid: %d\n", getpid());
15
16     pid = fork();
17
18     if (pid == 0) {
19         printf(1, "child: pid = %d, ppid = %d\n", getpid(), getppid());
20     }
21     else {
22         wait();
23         printf(1, "parent: pid = %d, ppid = %d\n", getpid(), getppid());
24     }
25
26     exit();
27 }
```


6-2. 시스템 콜 테스트를 위한 유저 프로그램 구현

❖ Makefile 수정

- 빌드 시 작성한 프로그램(hw1.c)도 함께 빌드하도록 오른쪽 그림과 같이 184번째 라인을 추가
- 이후 xv6를 구동하여 아래와 같이 hw1을 실행할 수 있음

```
$ hw1
sid: 202312345
name: Gil-dong Hong
current pid: 3
child: pid = 4, ppid = 3
parent: pid = 3, ppid = 2
$ █
```

M 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\
```


6-2. 시스템 콜 테스트를 위한 유저 프로그램 구현

❖ 제출

- 아래의 명령어를 통해 xv6-public 디렉토리를 압축
- 반드시 **clean**을 하고, 확장자는 **zip**으로 할 것

```
~/xv6-public$ sudo make clean
```

```
~/xv6-public$ cd ..
```

```
~$ sudo zip -r 학번-hw1.zip ./xv6-public
```

- hw1 프로그램 실행 결과를 캡처한 스크린샷과 압축 폴더(학번-hw1.zip)를 PLATO에 함께 업로드
 - 스크린샷 예시

