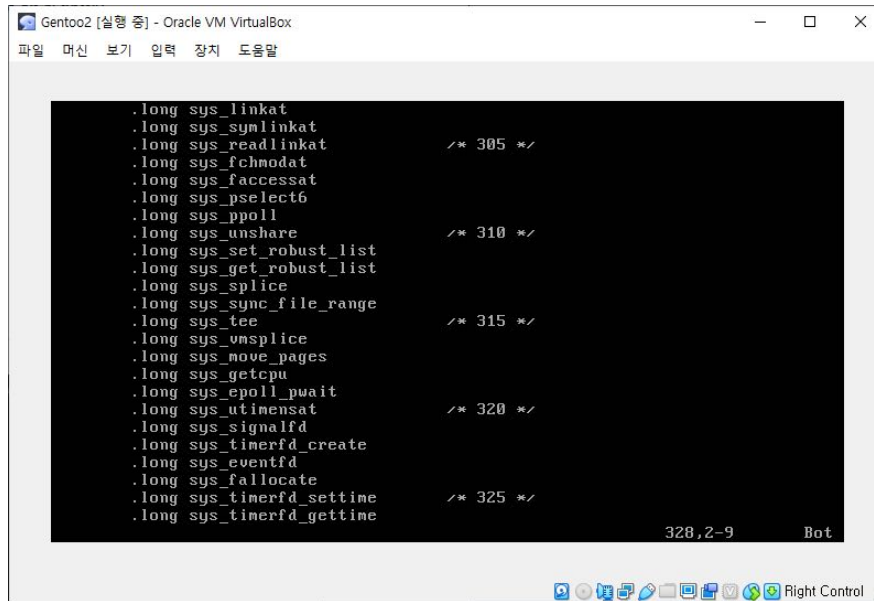
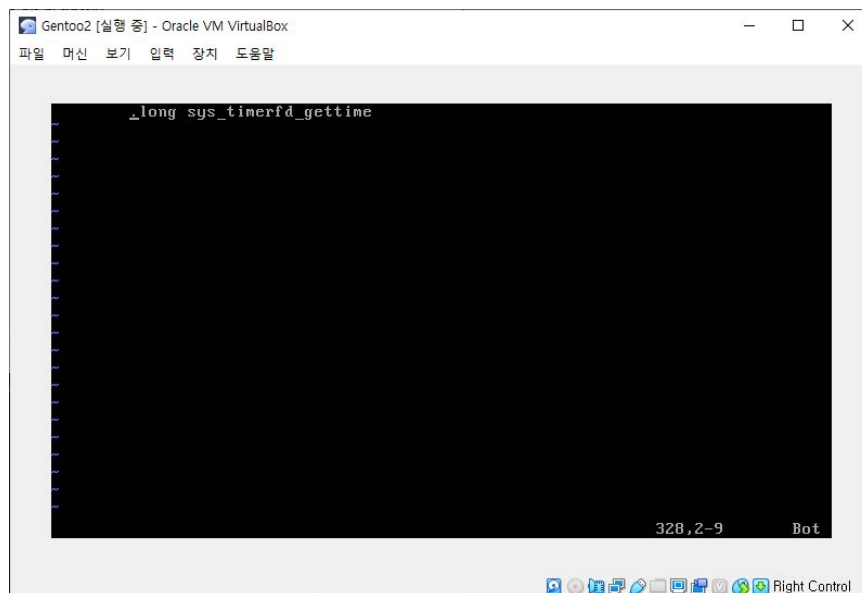


7) sys\_call\_table[] is in arch/x86/kernel/syscall\_table\_32.S. How many system calls does Linux 2.6 support? What are the system call numbers for exit, fork, execve, wait4, read, write, and mkdir? Find system call numbers for sys\_ni\_syscall, which is defined at kernel/sys\_ni.c. What is the role of sys\_ni\_syscall?



```
.long sys_linkat
.long sys_symlinkat
.long sys_readlinkat /* 305 */
.long sys_fchmodat
.long sys_faccessat
.long sys_pselect6
.long sys_ppoll
.long sys_unshare /* 310 */
.long sys_set_robust_list
.long sys_get_robust_list
.long sys_splice
.long sys_sync_file_range
.long sys_tee /* 315 */
.long sys_vmsplice
.long sys_move_pages
.long sys_getcpu
.long sys_epoll_pwait
.long sys_utimensat /* 320 */
.long sys_signalfd
.long sys_timerfd_create
.long sys_eventfd
.long sys_fallocate
.long sys_timerfd_settime /* 325 */
.long sys_timerfd_gettime
```

<System call table>



```
.long sys_timerfd_gettime
```

<sys\_timerfd\_gettime이 system call table의 마지막에 위치해있음. 즉  
327번에 위치해있고, 이로써 리눅스에는 327개의 system call이  
존재한다는 것을 알 수 있음>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

ENTRY(sys_call_table)
    .long sys_restart_syscall      /* 0 - old "setup()" system call, used f
or restarting */
    .long sys_exit
    .long sys_fork
    .long sys_read
    .long sys_write
    .long sys_open                /* 5 */
    .long sys_close
    .long sys_waitpid
    .long sys_creat
    .long sys_link
    .long sys_unlink              /* 10 */
    .long sys_execve
    .long sys_chdir
    .long sys_time
    .long sys_mknod
    .long sys_chmod              /* 15 */
    .long sys_lchown16
    .long sys_ni_syscall          /* old break syscall holder */
    .long sys_stat
    .long sys_lseek
    .long sys_getpid             /* 20 */
    .long sys_mount

2,1-8 Top
Right Control
```

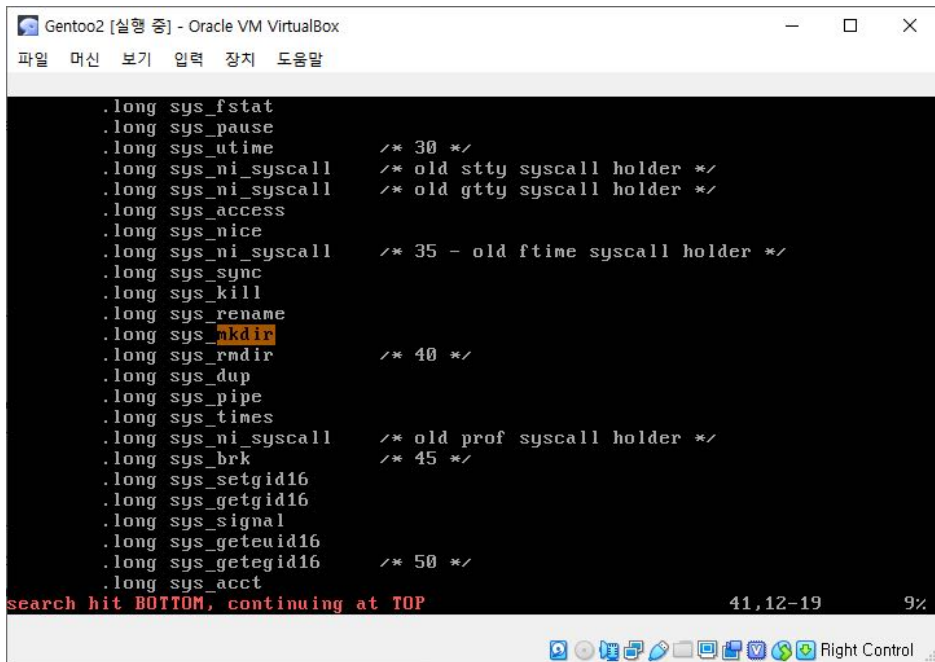
<exit(1), fork(2), read(3), write(4) 위치>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

    .long sys_syslog
    .long sys_setitimer
    .long sys_getitimer          /* 105 */
    .long sys_newstat
    .long sys_newlstat
    .long sys_newfstat
    .long sys_uname
    .long sys_iopl                /* 110 */
    .long sys_vhangup
    .long sys_ni_syscall          /* old "idle" system call */
    .long sys_vm86old
    .long sys_wait4
    .long sys_swapoff            /* 115 */
    .long sys_sysinfo
    .long sys_ipc
    .long sys_fsync
    .long sys_sigreturn
    .long sys_clone              /* 120 */
    .long sys_setdomainname
    .long sys_newuname
    .long sys_modify_ldt
    .long sys_adjtimex
    .long sys_mprotect           /* 125 */
    .long sys_sigprocmask

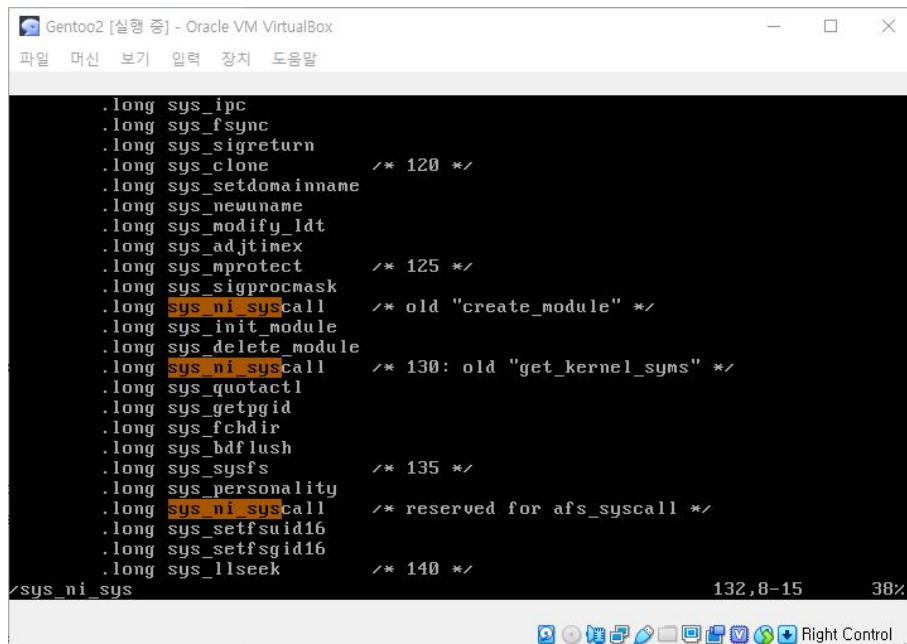
116,12-19 34%
Right Control
```

<wait4(114) 위치>



```
.long sys_fstat
.long sys_pause
.long sys_utime /* 30 */
.long sys_ni_syscall /* old stty syscall holder */
.long sys_ni_syscall /* old gtty syscall holder */
.long sys_access
.long sys_nice
.long sys_ni_syscall /* 35 - old ftime syscall holder */
.long sys_sync
.long sys_kill
.long sys_rename
.long sys_mkdir
.long sys_rmdir /* 40 */
.long sys_dup
.long sys_pipe
.long sys_times
.long sys_ni_syscall /* old prof syscall holder */
.long sys_brk /* 45 */
.long sys_setgid16
.long sys_getgid16
.long sys_signal
.long sys_geteuid16
.long sys_getegid16 /* 50 */
.long sys_acct
search hit BOTTOM, continuing at TOP 41,12-19 9%
```

<mkdir(39) 위치>



```
.long sys_ipc
.long sys_fsync
.long sys_sigreturn
.long sys_clone /* 120 */
.long sys_setdomainname
.long sys_newuname
.long sys_modify_ldt
.long sys_adjtimex
.long sys_mprotect /* 125 */
.long sys_sigprocmask
.long sys_ni_syscall /* old "create_module" */
.long sys_init_module
.long sys_delete_module
.long sys_ni_syscall /* 130: old "get_kernel_syms" */
.long sys_quotactl
.long sys_getpgid
.long sys_fchdir
.long sys_bdflush
.long sys_sysfs /* 135 */
.long sys_personality
.long sys_ni_syscall /* reserved for afs_syscall */
.long sys_setfsuid16
.long sys_setfsgid16
.long sys_llseek /* 140 */
/sys_ni_sys 132,8-15 38%
```

<sys\_ni\_syscall에는 아무런 system call 함수가 들어있지 않음 즉, 빈칸임,  
새로 선언한 system call 함수를 이 곳에다 넣으면 됨>

8) Change the kernel such that it prints "length 17 string found" for each printf(s) when the length of s is 17. Run a program that contains a printf() statement to see the effect. printf(s) calls write(1, s, strlen(s)) system call which in turn runs

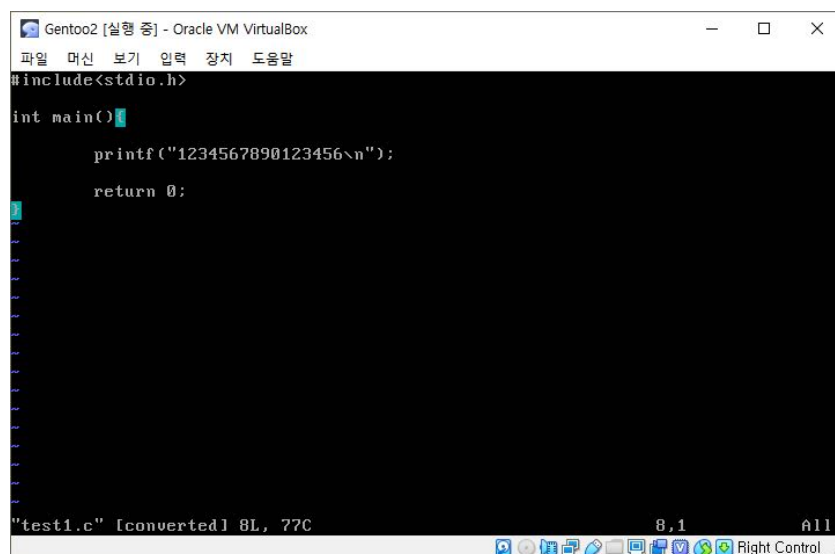
```
mov eax, 4 ; eax<--4. 4 is system call number for "write"
int 128
```

INT 128 will make the cpu stop running current process and jump to the location written in IDT[128]. IDT[128] contains the address of system\_call (located in arch/x86/kernel/entry\_32.S). Finally, system\_call will execute

```
call *sys_call_table(%eax,4)
```

which eventually calls sys\_write() since eax=4 for write() system call (the target function location is sys\_call\_table+eax\*4).

\* Sometimes the compiler generate "sysenter" instead of "int 128". In this case the cpu jumps to ia32\_sysenter\_target (also in entry\_32.S) instead of system\_call.

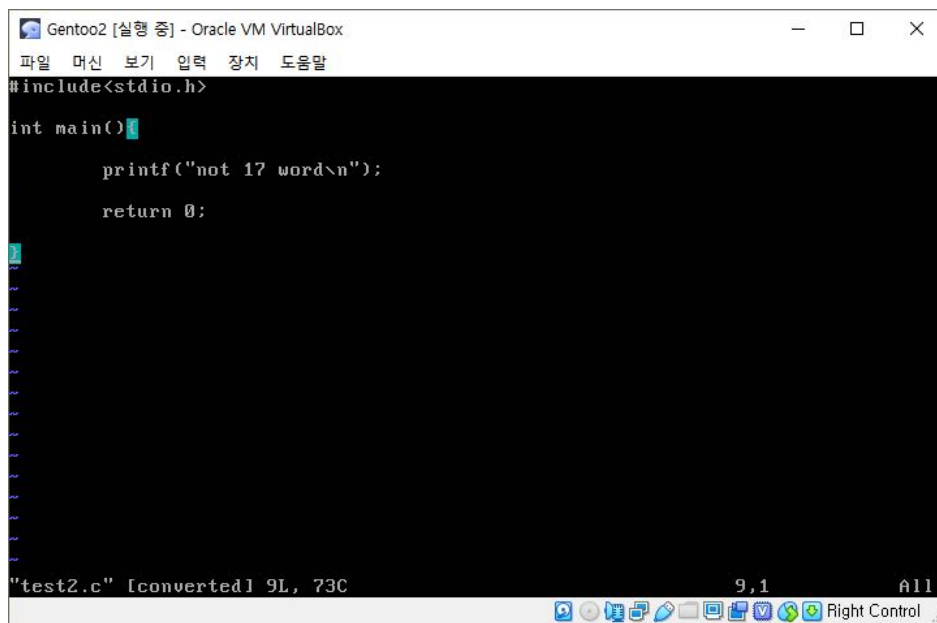


```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
#include<stdio.h>

int main()
{
    printf("1234567890123456\n");
    return 0;
}

"test1.c" [converted] 8L, 77C      8.1      A11
Right Control
```

<test1.c 파일 생성>

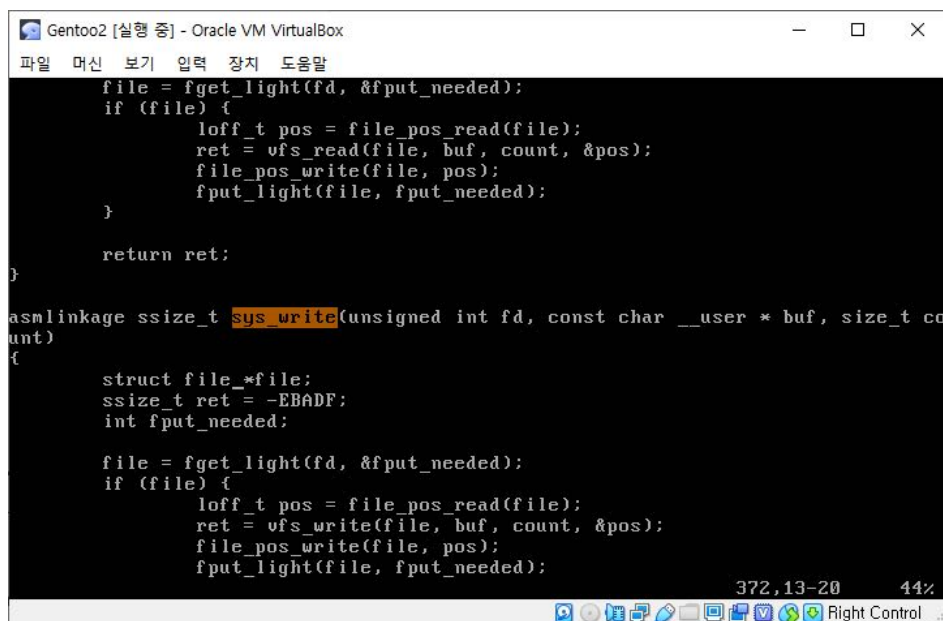


```
#include<stdio.h>

int main()
{
    printf("not 17 word\n");
    return 0;
}
```

"test2.c" [converted] 9L, 73C

<test2.c 파일 생성>



```
file = fget_light(fd, &fput_needed);
if (file) {
    loff_t pos = file_pos_read(file);
    ret = vfs_read(file, buf, count, &pos);
    file_pos_write(file, pos);
    fput_light(file, fput_needed);
}

return ret;
}

asmlinkage ssize_t sys_write(unsigned int fd, const char __user * buf, size_t count)
{
    struct file *file;
    ssize_t ret = -EBADF;
    int fput_needed;

    file = fget_light(fd, &fput_needed);
    if (file) {
        loff_t pos = file_pos_read(file);
        ret = vfs_write(file, buf, count, &pos);
        file_pos_write(file, pos);
        fput_light(file, fput_needed);
    }
}
```

372,13-20 44%

<fs/read\_write.c>에 선언되어 있는 sys\_write()함수>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

    loff_t pos = file_pos_read(file);
    ret = vfs_read(file, buf, count, &pos);
    file_pos_write(file, pos);
    fput_light(file, fput_needed);
}

return ret;
}

asmlinkage ssize_t sys_write(unsigned int fd, const char __user * buf, size_t count)
{
    struct file *file;
    ssize_t ret = -EBADF;
    int fput_needed;

    if(count == 17){
        printk("length 17 string found\n");

        file = fget_light(fd, &fput_needed);
"read_write.c" [converted] 835L, 18121C written
localhost fs # cd ..
localhost linux-2.6.25.10 # _
```

<sys\_write()함수 변경>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

localhost linux-2.6.25.10 # ./test1
length 17 string found
1234567890123456
localhost linux-2.6.25.10 # ./test2
not 17 word
localhost linux-2.6.25.10 #
```

<test1.c와 test2.c를 컴파일 한 결과>

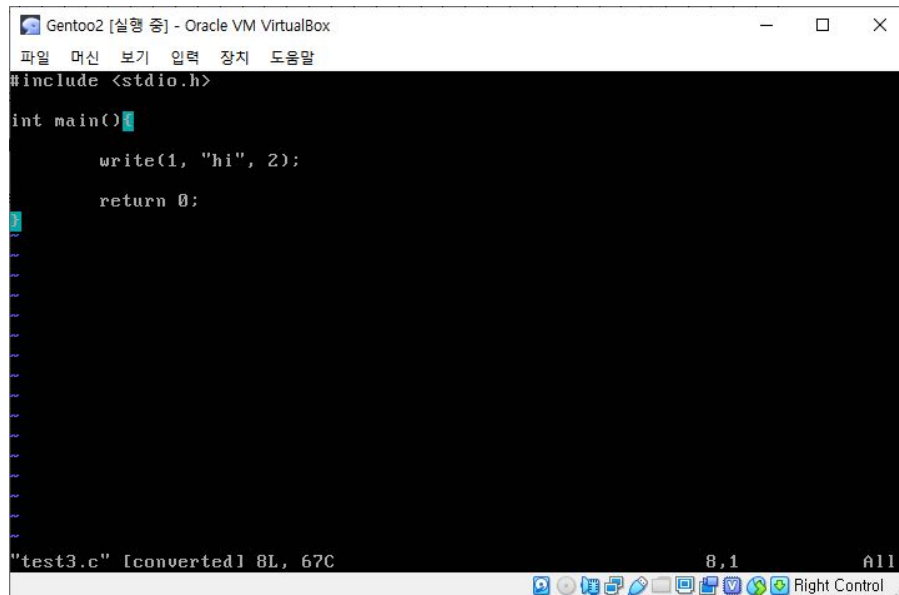
9) You can call a system call indirectly with "syscall()".

```
write(1, "hi", 2);
```

can be written as

```
syscall(4, 1, "hi", 2); // 4 is the system call number for "write" system call
```

Write a program that prints "hello" in the screen using syscall.

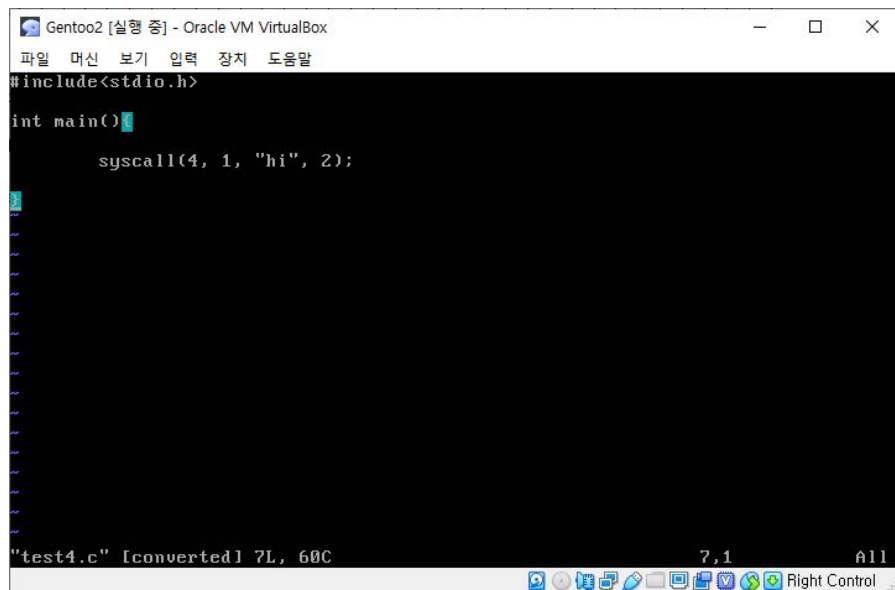


```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
#include <stdio.h>

int main()
{
    write(1, "hi", 2);
    return 0;
}

"test3.c" [converted] 8L, 67C  8,1  All
Right Control
```

<test3.c 파일>



```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
#include<stdio.h>

int main()
{
    syscall(4, 1, "hi", 2);
}

"test4.c" [converted] 7L, 60C  7,1  All
Right Control
```

<test4.c 파일>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

"test4.c" [New File]                                0,0-1      A111
length 17 string found
-- INSERT --                                          0,1        A111
length 17 string found
"test4.c" [New] 7L, 60C written
localhost ~ # ls
bootinglog  linux-2.6.25.10      temp      test4.c
irq         linux-2.6.25.10.tar.gz  test3.c
localhost ~ # gcc te
temp/      test3.c  test4.c
localhost ~ # gcc test3.c -o test3
localhost ~ # gcc test4.c -o test4
localhost ~ # ./test3
hilocalhost ~ # ./test4
hilocalhost ~ # _
```

<test3.c와 test4.c를 컴파일 한 결과>



10) Create a new system call, `my_sys_call` with system call number 17 (system call number 17 is one that is not being used currently). Define `my_sys_call()` just before `sys_write()` in `read_write.c`. Write a program that uses this system call:

```
void main(){
    syscall(17); // calls a system call with syscall number 17
}
```

When the above program runs, the kernel should display

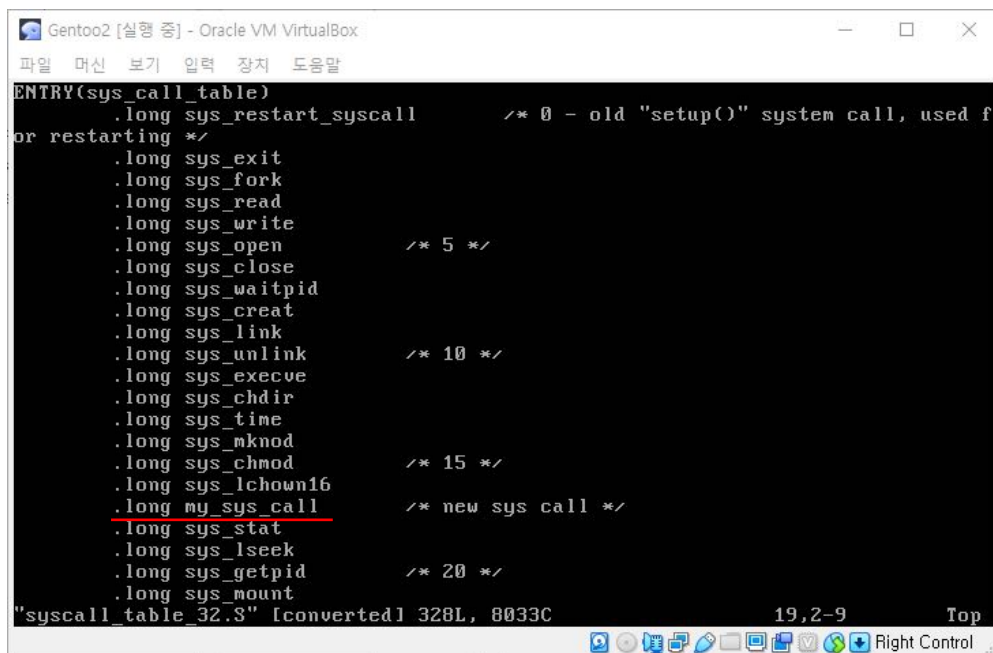
hello from `my_sys_call`

To define a new system call with syscall number x

- insert the new system call name in `arch/x86/kernel/syscall_table_32.S` at index x
- define the function in appropriate file (such as "read\_write.c")
- recompile and reboot

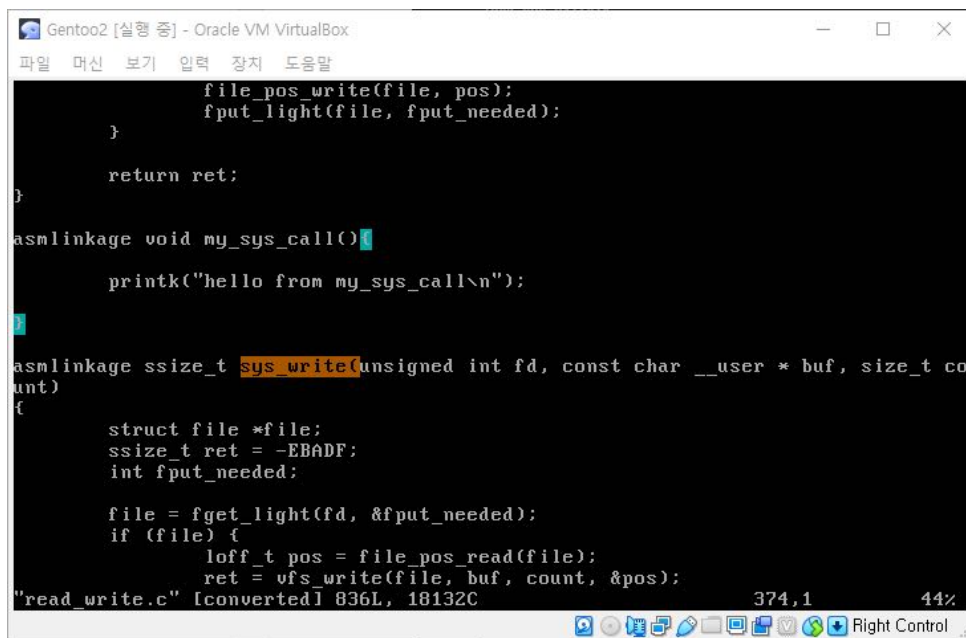
To use this system call in a user program

```
- void main(){
    syscall(x);
}
```



```
ENTRY(sys_call_table)
.long sys_restart_syscall    /* 0 - old "setup()" system call, used f
or restarting */
.long sys_exit
.long sys_fork
.long sys_read
.long sys_write
.long sys_open              /* 5 */
.long sys_close
.long sys_waitpid
.long sys_creat
.long sys_link
.long sys_unlink           /* 10 */
.long sys_execve
.long sys_chdir
.long sys_time
.long sys_mknod
.long sys_chmod            /* 15 */
.long sys_lchown16
.long my_sys_call           /* new sys call */
.long sys_stat
.long sys_lseek
.long sys_getpid           /* 20 */
.long sys_mount
"syscall_table_32.S" [converted] 32BL, 8033C 19,2-9 Top
```

<system call table의 17번 위치에 'my\_sys\_call' 추가>



```
        file_pos_write(file, pos);
        fput_light(file, fput_needed);
    }

    return ret;
}

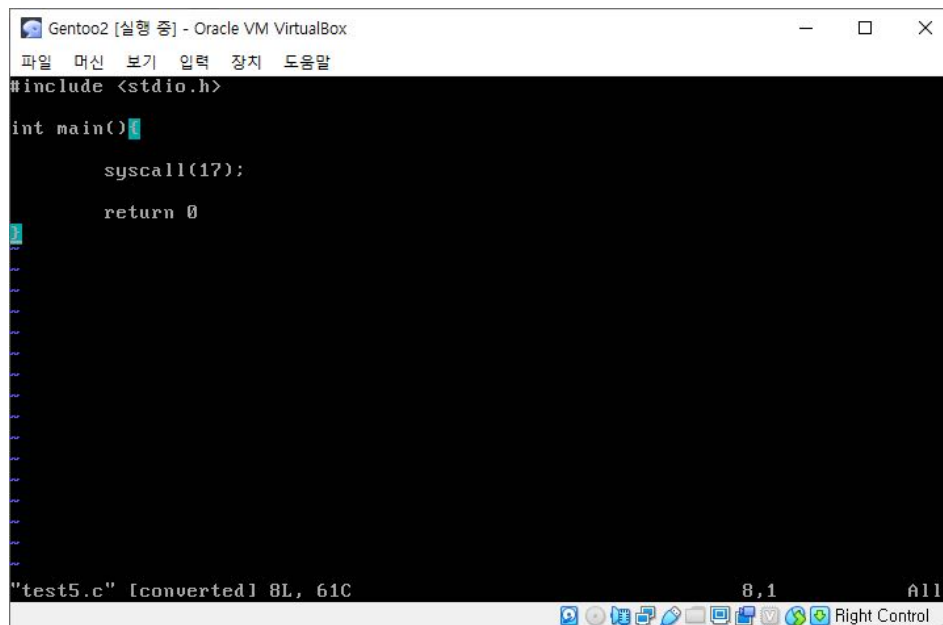
asm(
void my_sys_call()
{
    printk("hello from my_sys_call\n");
}

ssize_t sys_write(unsigned int fd, const char __user * buf, size_t count)
{
    struct file *file;
    ssize_t ret = -EBADF;
    int fput_needed;

    file = fget_light(fd, &fput_needed);
    if (file) {
        loff_t pos = file_pos_read(file);
        ret = vfs_write(file, buf, count, &pos);
    }
}

"read_write.c" [converted] 836L, 18132C 374,1 44%
```

<read\_write.c 내에 'my\_sys\_call()'함수 추가>



```
#include <stdio.h>

int main()
{
    syscall(17);

    return 0;
}

"test5.c" [converted] 8L, 61C 8,1 A11
```

<test5.c 파일 생성>

```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
localhost fs # echo 8 > /proc/sys/kernel/printk
localhost fs # cd ..
localhost linux-2.6.25.10 # echo 8 > /proc/sys/kernel/printk
localhost linux-2.6.25.10 # cd ..
localhost ~ # echo 8 > /proc/sys/kernel/printk
localhost ~ # ls
copyDmesgLog.txt  linux-2.6.25.10.tar.gz  test5      x
linux-2.6.25.10   temp                          test5.c    y
localhost ~ # ./test5
hello from my_sys_call
localhost ~ #
```

<test5.c 컴파일 한 결과>

11) Modify the kernel such that it displays the system call number for all system calls. Run a simple program that displays "hello" in the screen and find out what system calls have been called.

```

.long sys_unlink          /* 10 */
.long sys_execve
.long sys_chdir
.long sys_time
.long sys_mknod
.long sys_chmod          /* 15 */
.long sys_lchown16
.long print_all_sys_calls /* new sys call */
.long sys_stat
.long sys_lseek
.long sys_getpid        /* 20 */
.long sys_mount
.long sys_oldumount
.long sys_setuid16
.long sys_getuid16
.long sys_stime          /* 25 */
.long sys_ptrace
.long sys_alarm
.long sys_fstat
.long sys_pause
.long sys_utime          /* 30 */
.long sys_ni_syscall     /* old */
.long sys_ni_syscall     /* old gtty syscall holder */
.long sys_access

```

-- INSERT --

19,13-20 3%

<syscall\_table의 17번에 'print\_all\_sys\_calls'을 추가하였음>

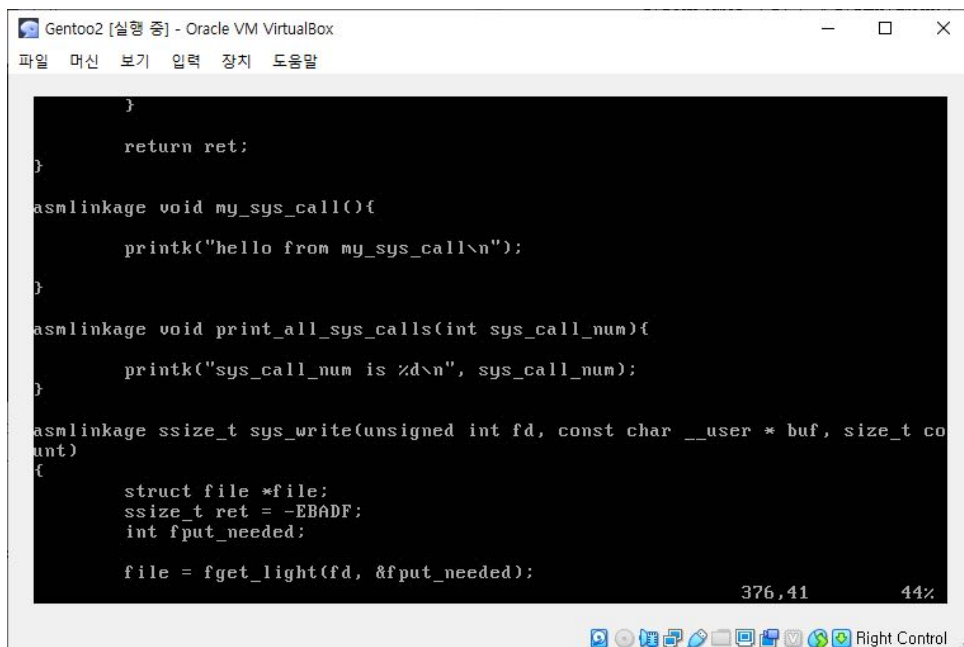
```

GET_THREAD_INFO(%ebp)
# system call tracing in operation / emu
/* Note, _TIF_SECCOMP is bit number 8, and so it needs testw and not testb */
testw $( _TIF_SYSCALL_EMU | _TIF_SYSCALL_TRACE | _TIF_SECCOMP | _TIF_SYSCALL_AUDIT ), TI_flags(%ebp)
jnz syscall_trace_entry
cmpl $(nr_syscalls), %eax
jae syscall_badsys
syscall_call:
    pushl %eax
    call print_all_sys_calls
    popl %eax

    call *sys_call_table(,%eax,4)
    movl %eax,PT_EAX(%esp) # store the return value
syscall_exit:
    LOCKDEP_SYS_EXIT
    DISABLE_INTERRUPTS(CLBR_ANY) # make sure we don't miss an interrupt
                                # setting need_resched or sigpending
                                # between sampling and the iret
"entry_32.S" [converted] 1118L, 26313C written
localhost kernel # _

```

<entry\_32.S 파일 내 'syscall\_call'부분을 수정함>



```
    }
    return ret;
}

asm linkage void my_sys_call(){
    printk("hello from my_sys_call\n");
}

asm linkage void print_all_sys_calls(int sys_call_num){
    printk("sys_call_num is %d\n", sys_call_num);
}

asm linkage ssize_t sys_write(unsigned int fd, const char __user * buf, size_t count)
{
    struct file *file;
    ssize_t ret = -EBADF;
    int fput_needed;

    file = fget_light(fd, &fput_needed);
```

<read\_write.c 파일 내 'print\_all\_sys\_calls()'라는 system call 함수를 선언함  
해당 함수는 특정 system call 함수가 실행 될 때마다, 해당 system call number를  
출력해주는 역할을 수행함>

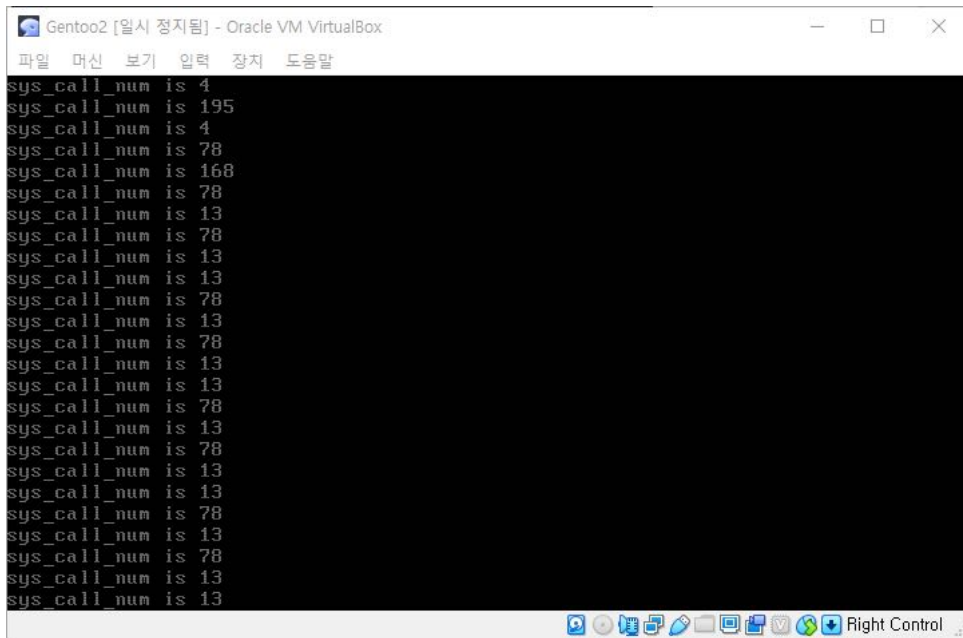
push, call, pop은 '어셈블리어'에 속하는 명령어들이다.

'push'는 stack 영역에 4byte 공간을 할당하고 해당 값을 할당된 공간에 저장한다. 보통, 뒤에 call되는 함수의 매개변수에 인자값을 전달하기 위해서 사용된다.

'call'은 접근 할 함수로 이동하여 실행한다.

'pop'은 이전 'push'로 저장되어진 4byte 공간을 제거한다.

이를 통해, system call이 발생할 때마다 'print\_all\_sys\_calls()'의 system call 함수가 호출되고, '%eax'가 해당 함수의 인자로 사용되어진다는 것을 알 수 있다.



```
sys_call_num is 4
sys_call_num is 195
sys_call_num is 4
sys_call_num is 78
sys_call_num is 168
sys_call_num is 78
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 78
sys_call_num is 13
sys_call_num is 13
```

<커널을 재컴파일하고 리부팅을 한 후, 호출되어지는 모든 system call 함수의 번호를 화면에 출력하기위해 'echo 8 > /proc/sys/kernel/printk'코드를 입력하였음, 이후 위 사진처럼 특정 system call 함수의 번호가 끊임없이 출력되는 오류가 발생함>

위 오류가 발생하였지만, 해당 오류를 통해 printf("hello")가 실행되었을 때 이를 통해 호출되어지는 모든 system call 함수들의 number가 화면에 출력될 것임을 예측할 수 있다.