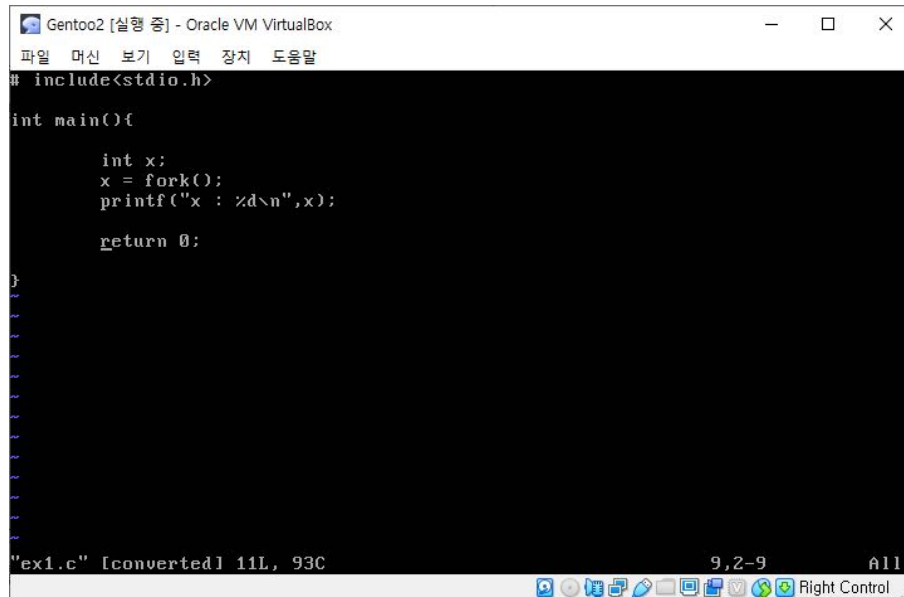


1) Run the program below. What happens? Explain the result.

ex1.c:

```
void main(){
    int x;
    x=fork();
    printf("x:%d\n", x);
}
```



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

int main(){

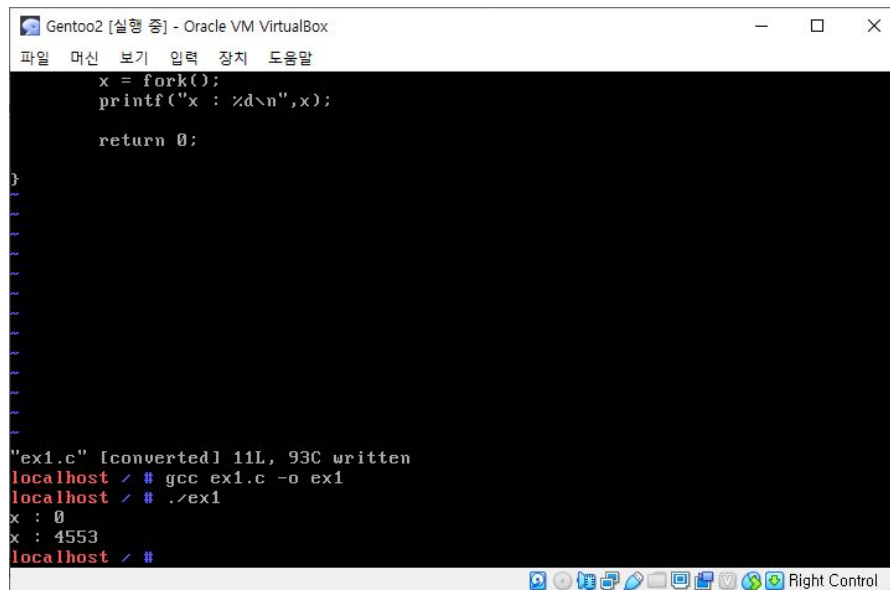
    int x;
    x = fork();
    printf("x : %d\n",x);

    return 0;

}

"ex1.c" [converted] 11L, 93C
9,2-9  All
Right Control
```

<ex1.c 작성>



```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
x = fork();
printf("x : %d\n",x);

return 0;

}

"ex1.c" [converted] 11L, 93C written
localhost / # gcc ex1.c -o ex1
localhost / # ./ex1
x : 0
x : 4553
localhost / #
Right Control
```

<ex1.c 파일 실행>

x: 0, x: 4553이 출력되었다. child process가 0을 출력하였고, parent process가 4553을 출력하였다. 둘 중 무엇이 먼저 실행완료가 될지는 scheduling에 따라 정해진다.

2) Try below and explain the result.

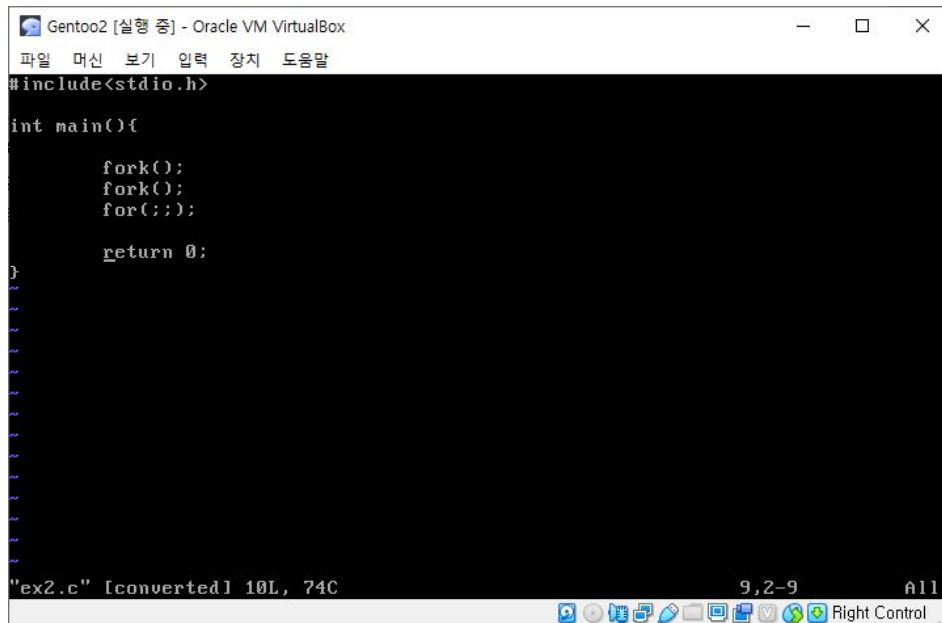
ex1.c:

```
void main(){
    fork();
    fork();
    for(;;);
}
```

```
# gcc -o ex1 ex1.c
```

```
# ./ex1 &
```

```
# ps -ef
```



```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
#include<stdio.h>
int main(){
    fork();
    fork();
    for(;;);
    return 0;
}
"ex2.c" [converted] 10L, 74C
9,2-9 All
Right Control
```

<ex2.c 작성>

```

Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
root      182      2  0 05:17 ?      00:00:00 [kswapd0]
root      225      2  0 05:17 ?      00:00:00 [aio/0]
root      907      2  0 05:17 ?      00:00:00 [scsi_eh_0]
root      924      2  0 05:17 ?      00:00:00 [khpsbpkt]
root      965      2  0 05:17 ?      00:00:00 [kpsmoused]
root      969      2  0 05:17 ?      00:00:00 [kstripped]
root      972      2  0 05:17 ?      00:00:00 [kondemand/0]
root      985      2  0 05:17 ?      00:00:00 [rpciod/0]
root     1073      2  0 05:17 ?      00:00:00 [kjournald]
root     1170      1  0 05:17 ?      00:00:00 /sbin/udevd --daemon
root     4312      1  0 05:17 ?      00:00:00 /usr/sbin/syslog-ng
root     4427      1  0 05:17 ?      00:00:00 /usr/sbin/cron
root     4491      1  0 05:17 tty1     00:00:00 /bin/login --
root     4493      1  0 05:17 tty2     00:00:00 /sbin/agetty 38400 tty2 linux
root     4495      1  0 05:17 tty3     00:00:00 /sbin/agetty 38400 tty3 linux
root     4497      1  0 05:17 tty4     00:00:00 /sbin/agetty 38400 tty4 linux
root     4499      1  0 05:17 tty5     00:00:00 /sbin/agetty 38400 tty5 linux
root     4501      1  0 05:17 tty6     00:00:00 /sbin/agetty 38400 tty6 linux
root     4514      1  0 05:18 tty1     00:00:00 -bash
root     4586     4514 24 05:44 tty1     00:00:43 ./ex2
root     4587     4586 24 05:44 tty1     00:00:43 ./ex2
root     4588     4587 24 05:44 tty1     00:00:43 ./ex2
root     4589     4586 24 05:44 tty1     00:00:43 ./ex2
root     4596     4514  0 05:47 tty1     00:00:00 ps -ef
localhost ~ #

```

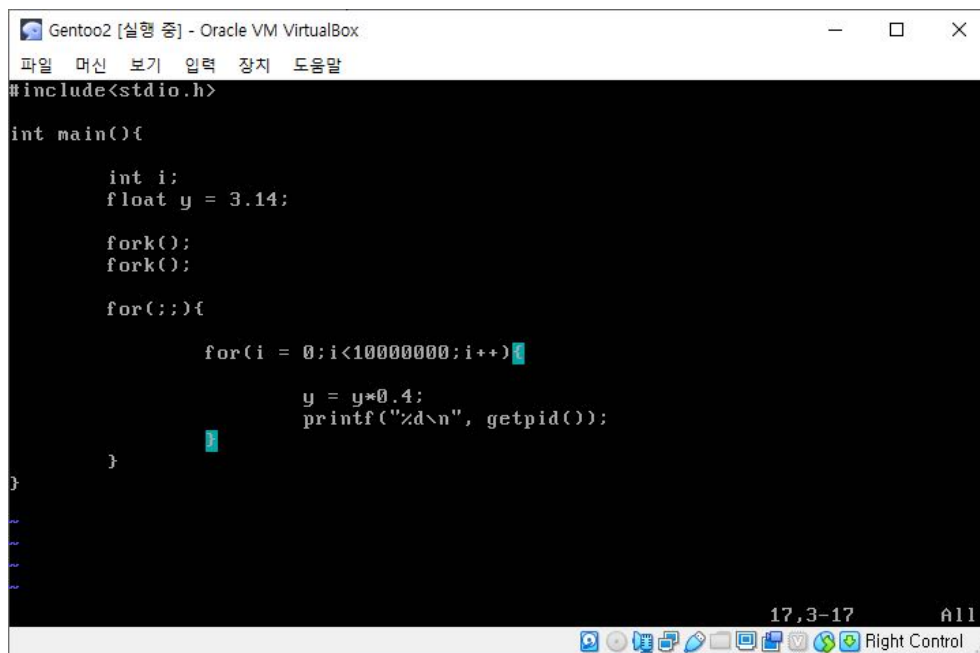
<ps -ef 결과>

pid가 4586인 프로세스가 ex2에 대한 원래의 프로세스이다. pid가 4587인 프로세스는 첫 번째 fork로 인해 생성된 프로세스(pid가 4586인 프로세스를 복사한 것)이다. pid가 4589인 프로세스는 두 번째 fork로 인해 생성된 프로세스(pid가 4586인 프로세스를 복사한 것)이다. pid가 4588인 프로세스는 pid가 4587인 프로세스를 복제한 프로세스이다.

3) Run following code. What happens? Explain the result.

ex1.c:

```
void main(){
    int i; float y=3.14;
    fork();
    fork();
    for(;;){
        for(i=0;i<10000000;i++) y=y*0.4;
        printf("%d\n", getpid());
    }
}
```



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

int main(){

    int i;
    float y = 3.14;

    fork();
    fork();

    for(;;){

        for(i = 0;i<10000000;i++){

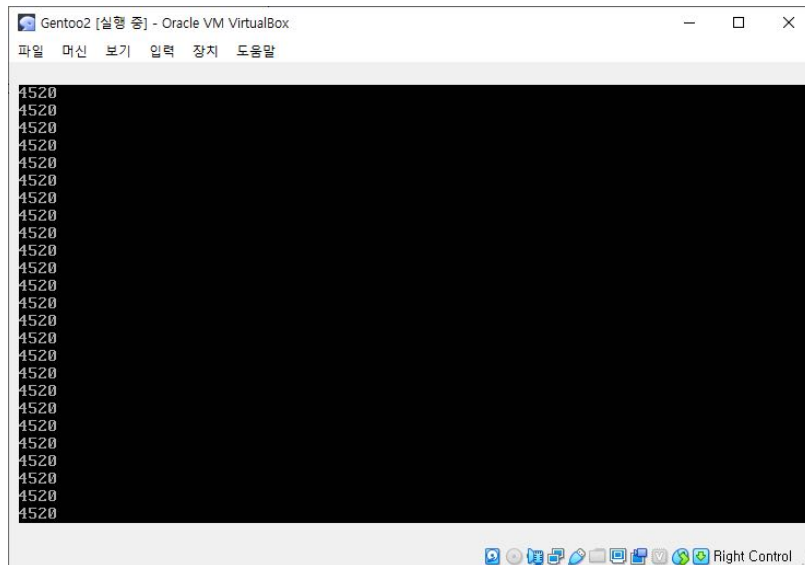
            y = y*0.4;
            printf("%d\n", getpid());

        }

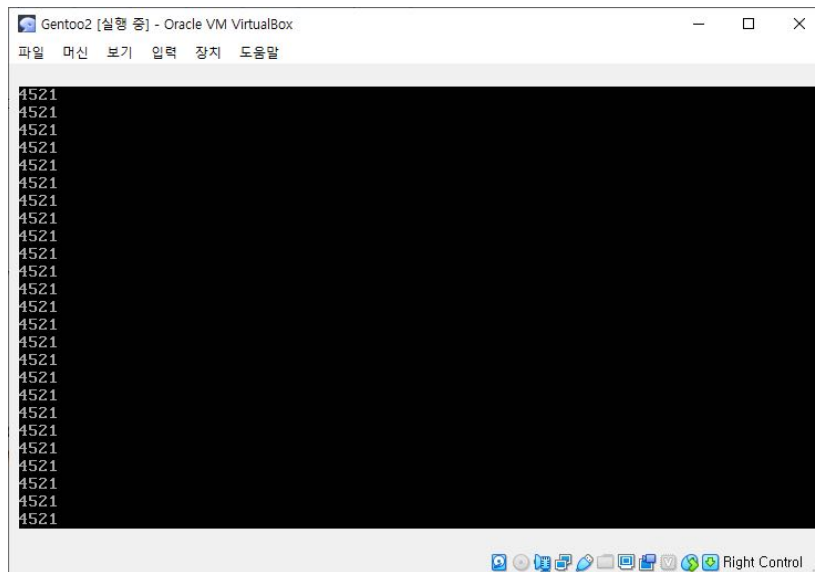
    }

}
```

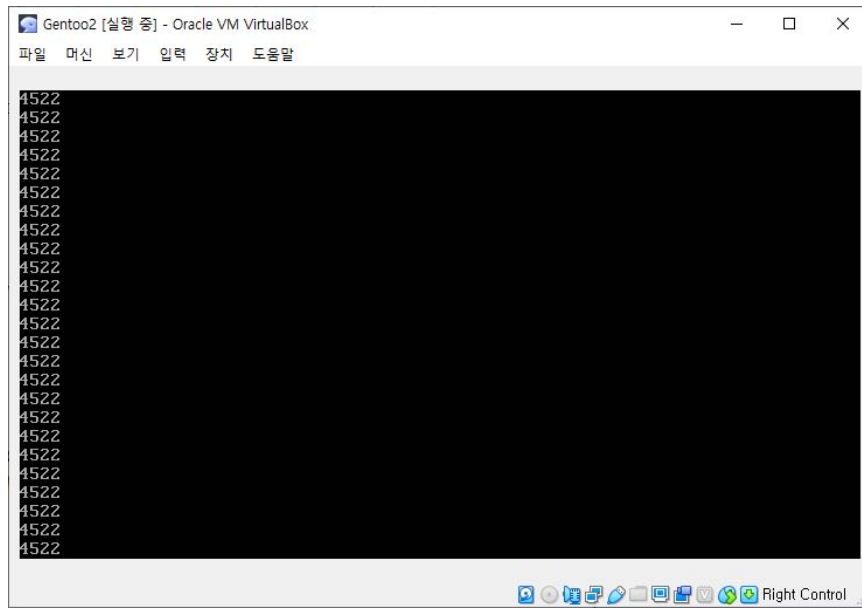
<ex3.c 작성>



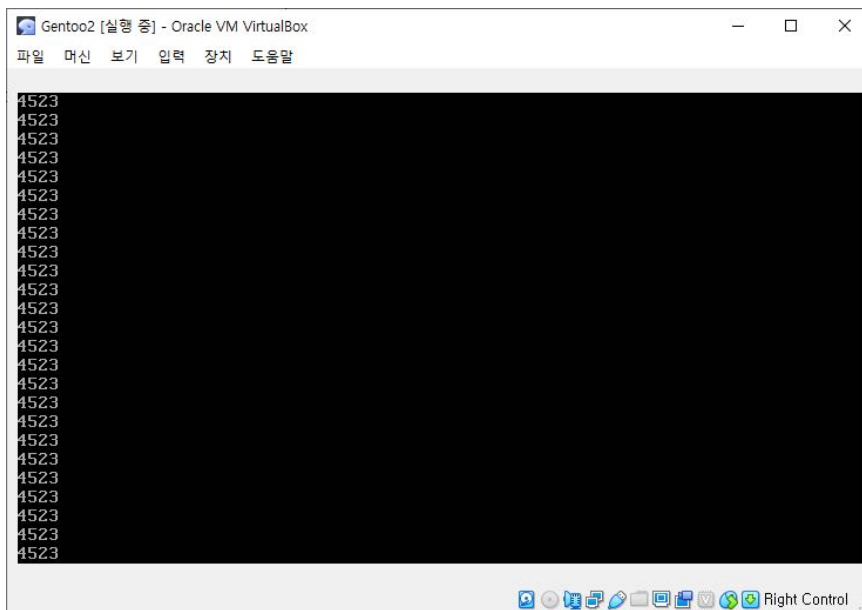
<pid가 4520인 프로세스>



<pid가 4521인 프로세스>



<pid가 4522인 프로세스>

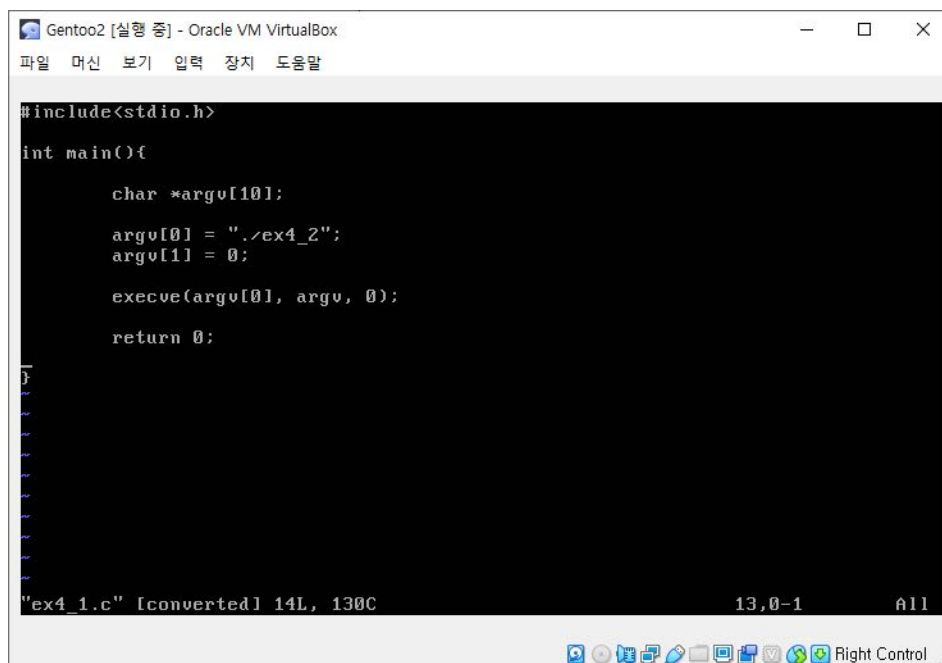


<pid가 4523인 프로세스>

fork()함수가 두 개 호출되었기 때문에, pid가 4520, 4521, 4522, 4523인 프로세스 4개가 실행된다. 프로세스가 실행되는 순서는 cpu의 scheduling에 따라 결정된다.

4) Try below and explain the result.

```
ex1.c:
void main(){
char *argv[10];
    argv[0]="./ex2";
    argv[1]=0;
    execve(argv[0], argv, 0);
}
ex2.c
void main(){
    printf("korea\n");
}
#gcc -o ex1 ex1.c
#gcc -o ex2 ex2.c
#./ex1
```

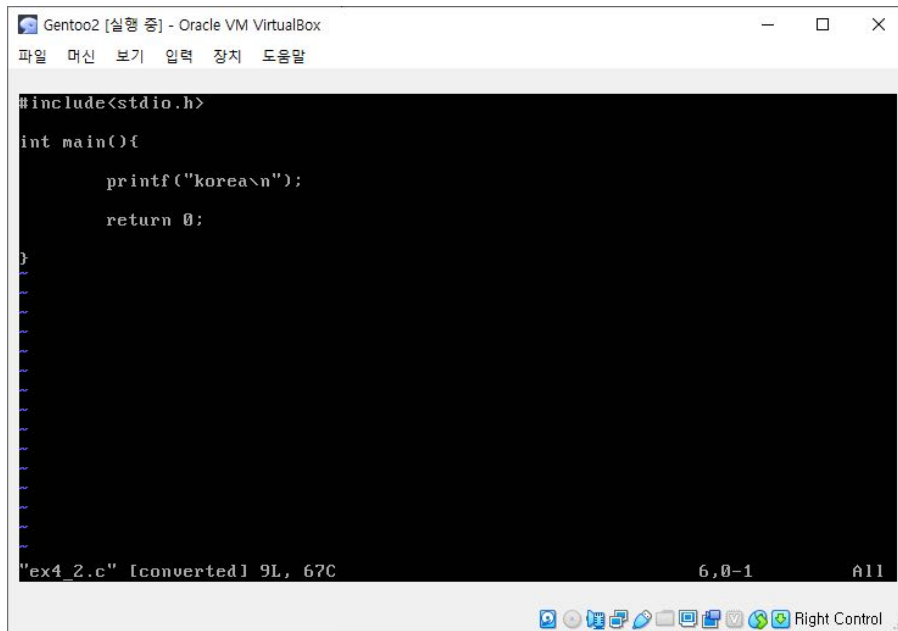


```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  마신  보기  입력  장치  도움말

#include<stdio.h>
int main(){
    char *argv[10];
    argv[0] = "./ex4_2";
    argv[1] = 0;
    execve(argv[0], argv, 0);
    return 0;
}

"ex4_1.c" [converted] 14L, 130C 13,0-1 All
Right Control
```

<ex4_1.c 작성>



```
#include<stdio.h>

int main(){

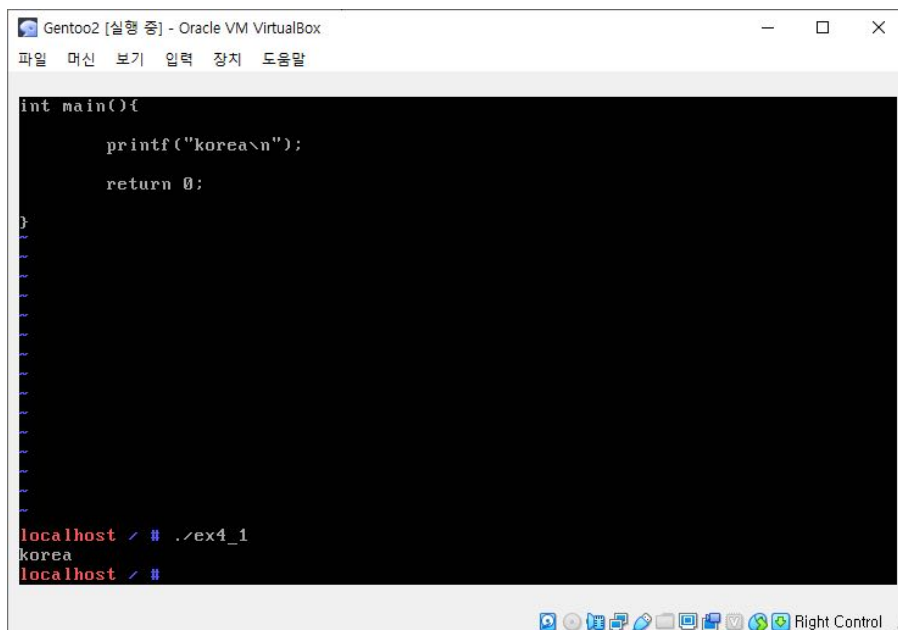
    printf("korea\n");

    return 0;

}
```

"ex4_2.c" [converted] 9L, 67C 6,0-1 All

<ex4_2.c 작성>



```
int main(){

    printf("korea\n");

    return 0;

}
```

```
localhost / # ./ex4_1
korea
localhost / #
```

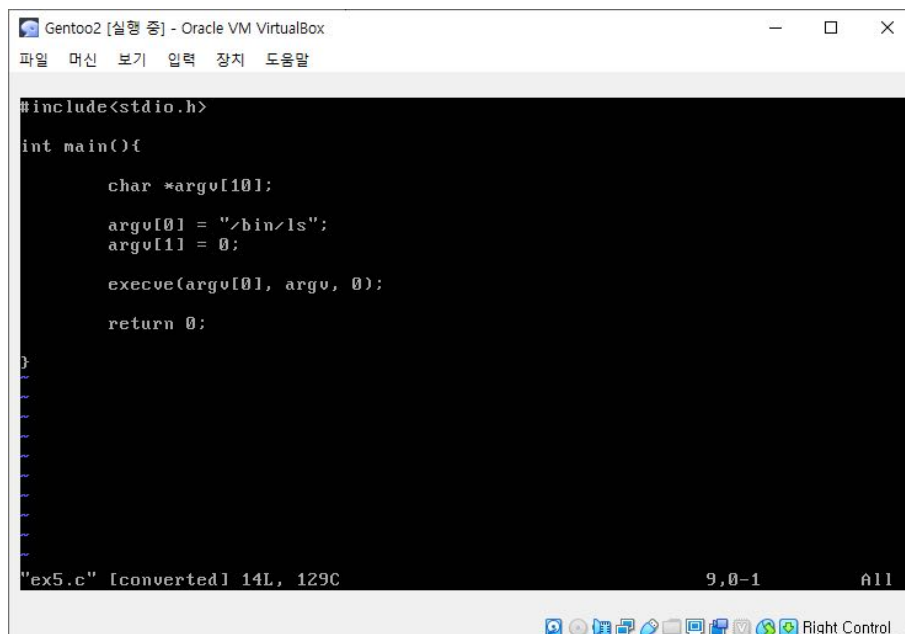
<ex4_1.c 실행>

execve()함수는 현재 프로세스를 다른 프로세스로 바꿀 때 사용하는 함수이다. 즉, execve()함수를 실행하는 순간 현재의 프로세스를 종료시킨다. 그 다음 execve()에 인자로 입력한 파일의 경로로 이동하여, 해당 프로세스를 실행시킨다. ex4_1에서 execve("./ex4_2")함수를 호출했기 때문에, ex4_2가 실행된 것이다.

※ 'int execve(const char *path, char *const argv[], char *const envp[]);'가 execve()함수의 선언부분이다. argv와 envp는 포인터 배열이고, 해당 배열의 마지막에는 NULL 포인터가 저장되어야한다.

5) Run following code and explain the result.

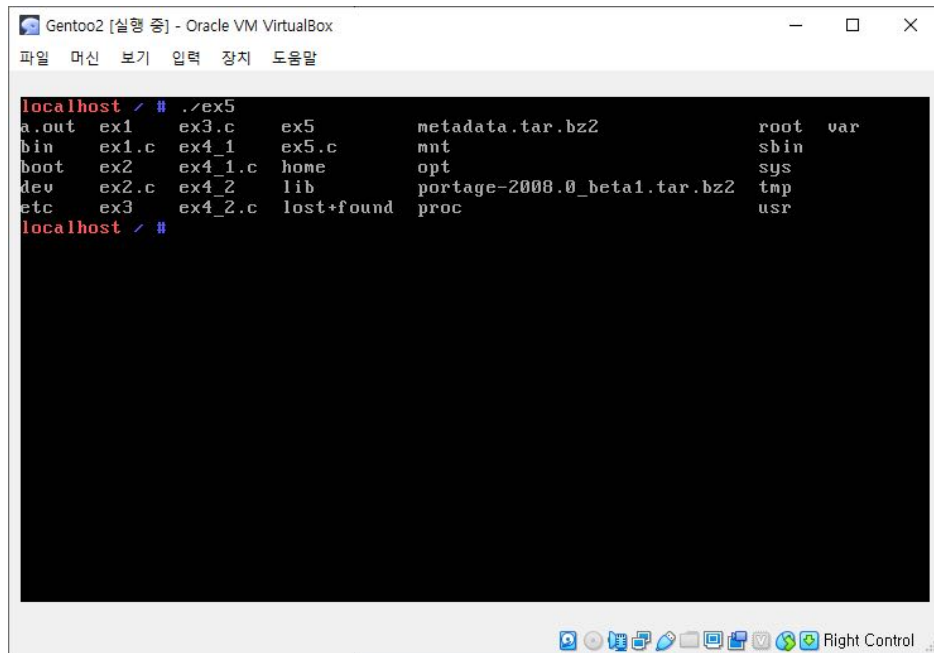
```
void main(){
char *argv[10];
    argv[0]="/bin/ls";
    argv[1]=0;
    execve(argv[0], argv, 0);
}
```

A screenshot of a virtual machine window titled "Gentoo2 [실행 중] - Oracle VM VirtualBox". The window shows a terminal with a C program. The code is:

```
#include<stdio.h>
int main(){
    char *argv[10];
    argv[0] = "/bin/ls";
    argv[1] = 0;
    execve(argv[0], argv, 0);
    return 0;
}
```

 The terminal output at the bottom shows the file path "ex5.c" [converted] 14L, 129C, the line number 9,0-1, and the column number 111. The window has a menu bar with "파일", "머신", "보기", "입력", "장치", and "도움말". The bottom status bar shows various icons and the text "Right Control".

<ex5.c 작성>



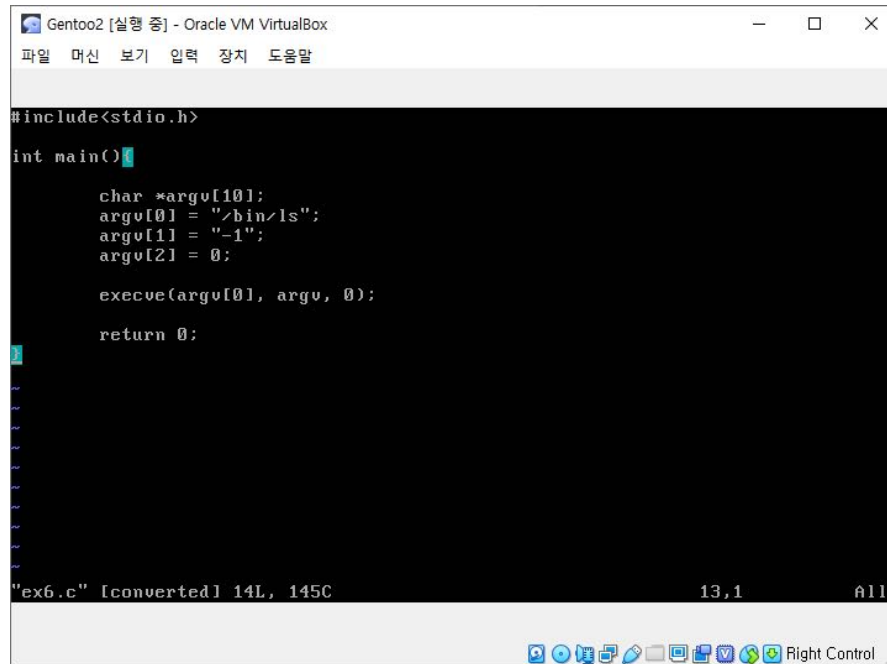
```
localhost # ./ex5
a.out ex1 ex3.c ex5 metadata.tar.bz2 root var
bin ex1.c ex4_1 ex5.c mnt sbin
boot ex2 ex4_1.c home opt sys
dev ex2.c ex4_2 lib portage-2008.0_beta1.tar.bz2 tmp
etc ex3 ex4_2.c lost+found proc usr
localhost #
```

<ex5.c 실행>

해당 결과물이 출력되는 이유는 10.4의 설명과 동일하다.

6) Run following code and explain the result.

```
void main(){  
char *argv[10];  
    argv[0]="/bin/ls";  
    argv[1]="-l";  
    argv[2]=0;  
    execve(argv[0], argv, 0);  
}
```

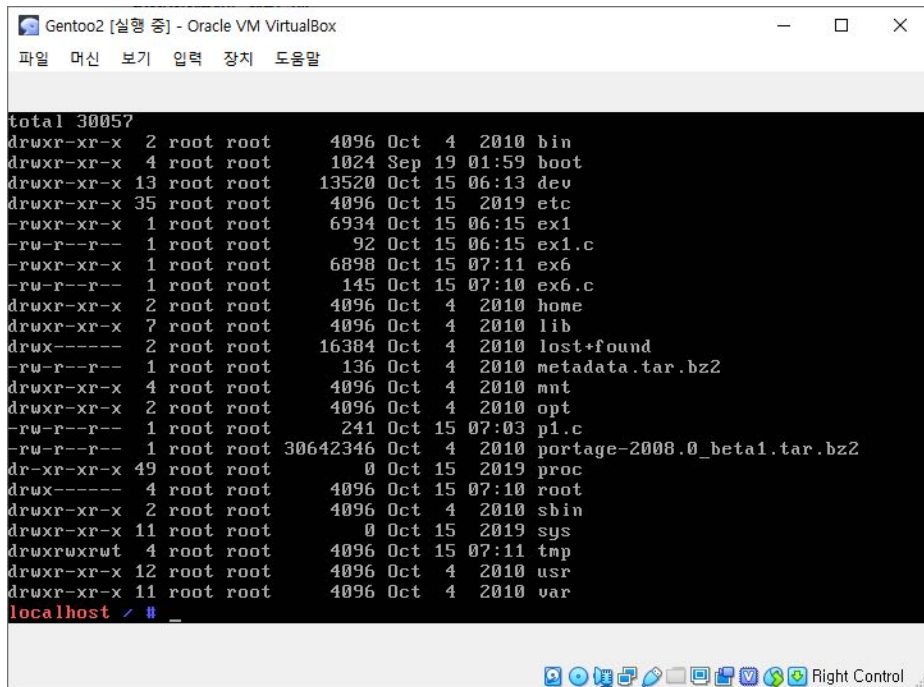


The screenshot shows a terminal window titled "Gentoo2 [실행 중] - Oracle VM VirtualBox". The terminal displays the following C code being executed:

```
#include<stdio.h>  
  
int main()  
{  
    char *argv[10];  
    argv[0] = "/bin/ls";  
    argv[1] = "-l";  
    argv[2] = 0;  
  
    execve(argv[0], argv, 0);  
  
    return 0;  
}
```

The terminal output shows the execution of the program, which results in a list of files and directories in the current directory, displayed in a long format due to the "-l" argument. The status bar at the bottom of the terminal window indicates "ex6.c" [converted] 14L, 145C, 13,1, and All.

<ex6.c 작성>



```
total 30057
drwxr-xr-x  2 root root    4096 Oct  4 2010 bin
drwxr-xr-x  4 root root   1024 Sep 19 01:59 boot
drwxr-xr-x 13 root root  13520 Oct 15 06:13 dev
drwxr-xr-x 35 root root    4096 Oct 15 2019 etc
-rwxr-xr-x  1 root root   6934 Oct 15 06:15 ex1
-rw-r--r--  1 root root     92 Oct 15 06:15 ex1.c
-rwxr-xr-x  1 root root   6898 Oct 15 07:11 ex6
-rw-r--r--  1 root root    145 Oct 15 07:10 ex6.c
drwxr-xr-x  2 root root    4096 Oct  4 2010 home
drwxr-xr-x  7 root root    4096 Oct  4 2010 lib
drwx----- 2 root root   16384 Oct  4 2010 lost+found
-rw-r--r--  1 root root    136 Oct  4 2010 metadata.tar.bz2
drwxr-xr-x  4 root root    4096 Oct  4 2010 mnt
drwxr-xr-x  2 root root    4096 Oct  4 2010 opt
-rw-r--r--  1 root root    241 Oct 15 07:03 p1.c
-rw-r--r--  1 root root 30642346 Oct  4 2010 portage-2008.0_beta1.tar.bz2
dr-xr-xr-x 49 root root     0 Oct 15 2019 proc
drwx----- 4 root root    4096 Oct 15 07:10 root
drwxr-xr-x  2 root root    4096 Oct  4 2010/sbin
drwxr-xr-x 11 root root     0 Oct 15 2019 sys
drwxrwxrwt  4 root root    4096 Oct 15 07:11 tmp
drwxr-xr-x 12 root root    4096 Oct  4 2010 usr
drwxr-xr-x 11 root root    4096 Oct  4 2010 var
localhost #
```

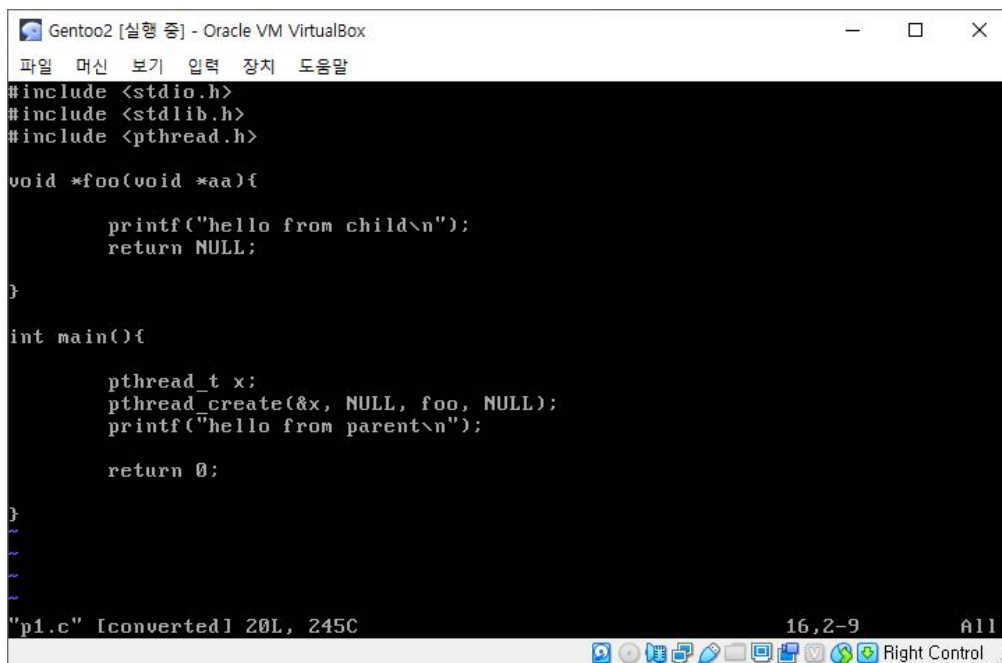
<ex6.c 실행>

argv의 모든 원소들을 실행시킨다. 'bin/ls'를 '-l' 옵션을 적용하여 실행시킨다. '-l'의 옵션은 해당 디렉토리 내 하위 디렉토리와 파일의 구체적인 정보까지 출력한다.

7) Run following code and explain the result.

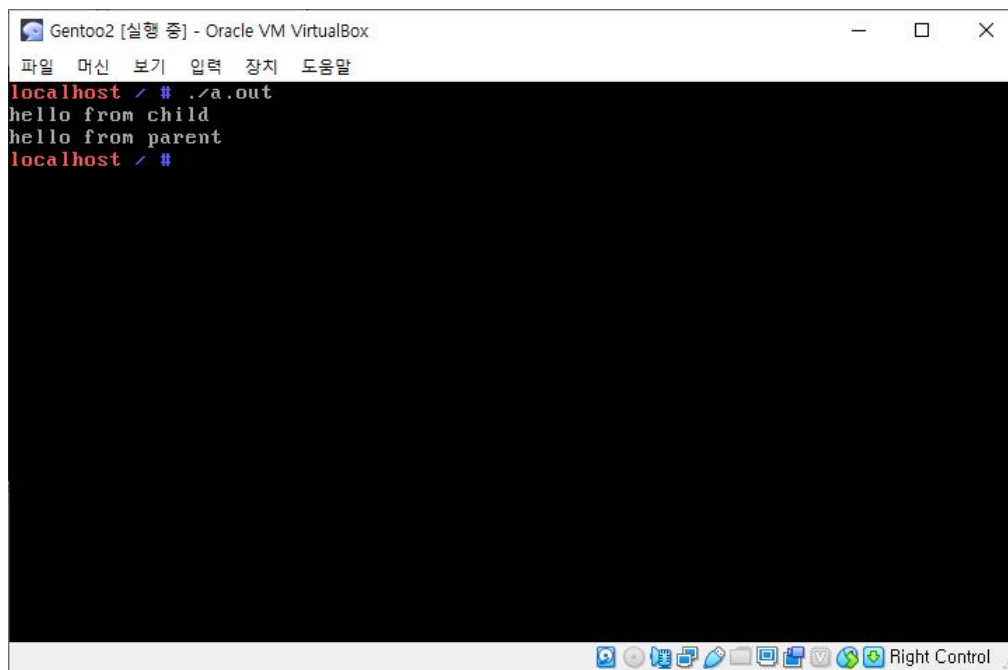
p1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void * foo(void * aa){
    printf("hello from child\n");
    return NULL;
}
void main(){
    pthread_t x;
    pthread_create(&x, NULL, foo, NULL); // make a child which starts at foo
    printf("hello from parent\n");
```



```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *foo(void *aa){
    printf("hello from child\n");
    return NULL;
}
int main(){
    pthread_t x;
    pthread_create(&x, NULL, foo, NULL);
    printf("hello from parent\n");
    return 0;
}
"p1.c" [converted] 20L, 245C 16,2-9 All
Right Control
```

<p1.c작성>

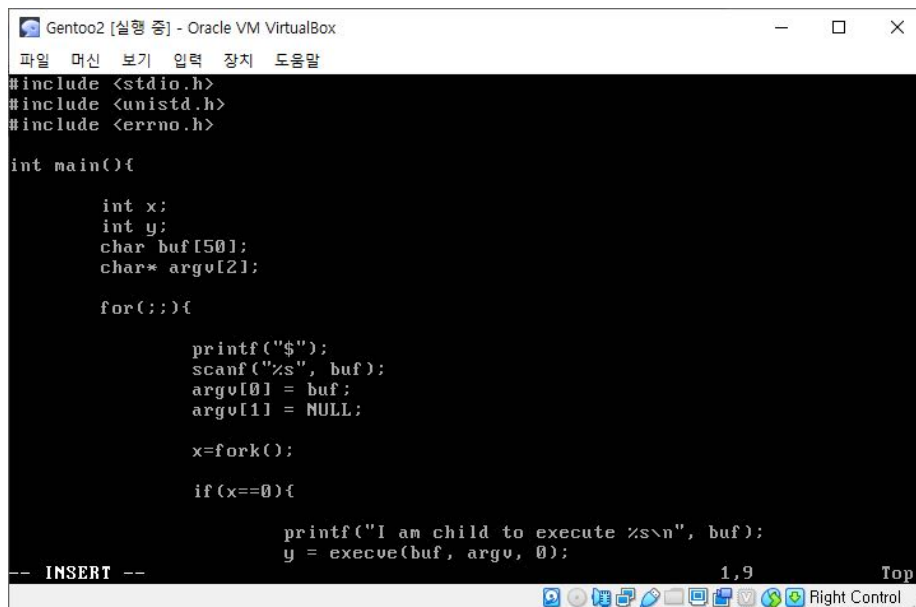


```
localhost / # ./a.out
hello from child
hello from parent
localhost / #
```

<p1.c 실행>

pthread_create()함수는 해당 프로세스의 body는 공유하고, 해당 process descriptor만 복제하는 함수이다. pthread_create()함수의 세 번째 매개변수에는 해당 함수가 호출된 이후에 수행할 명령 위치가 인자로 들어간다. 해당 함수 호출 이후에 foo 함수가 실행되어 "hello from child"가 먼저 출력되고, 이 다음 "hello from parent"가 출력된다.

- 1) Try the shell code in section 7. Try Linux command such as `/bin/ls`, `/bin/date`, etc.



```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

int main(){

    int x;
    int y;
    char buf[50];
    char* argv[2];

    for(;;){

        printf("$");
        scanf("%s", buf);
        argv[0] = buf;
        argv[1] = NULL;

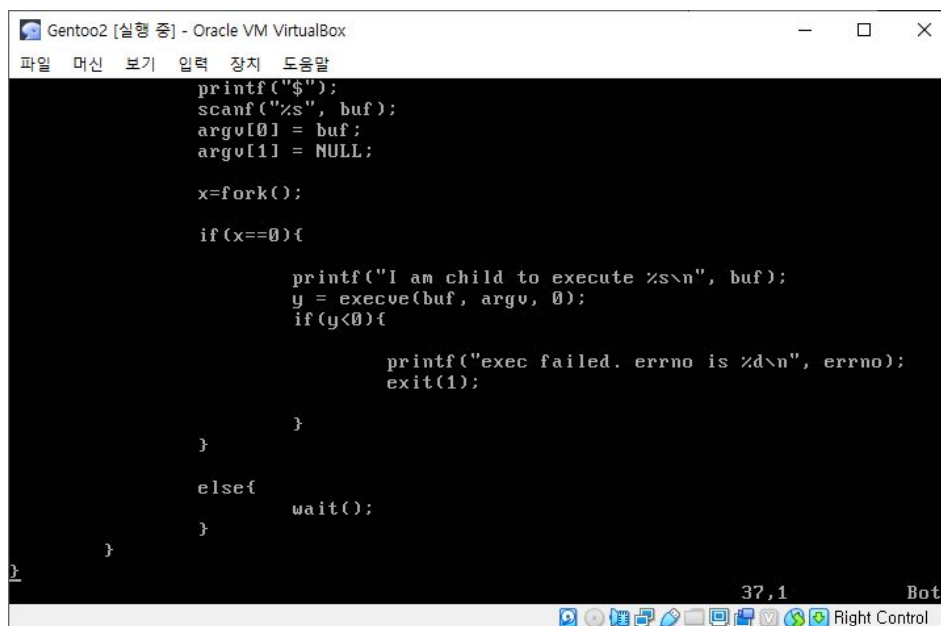
        x=fork();

        if(x==0){

            printf("I am child to execute %s\n", buf);
            y = execve(buf, argv, 0);

-- INSERT --
```

<shell code작성 앞부분(파일명 : ex7.c)>



```
        printf("I am child to execute %s\n", buf);
        y = execve(buf, argv, 0);
        if(y<0){

            printf("exec failed. errno is %d\n", errno);
            exit(1);

        }

    }

    else{
        wait();
    }

}
```

<shell code작성 뒷부분(파일명 : ex7.c)>

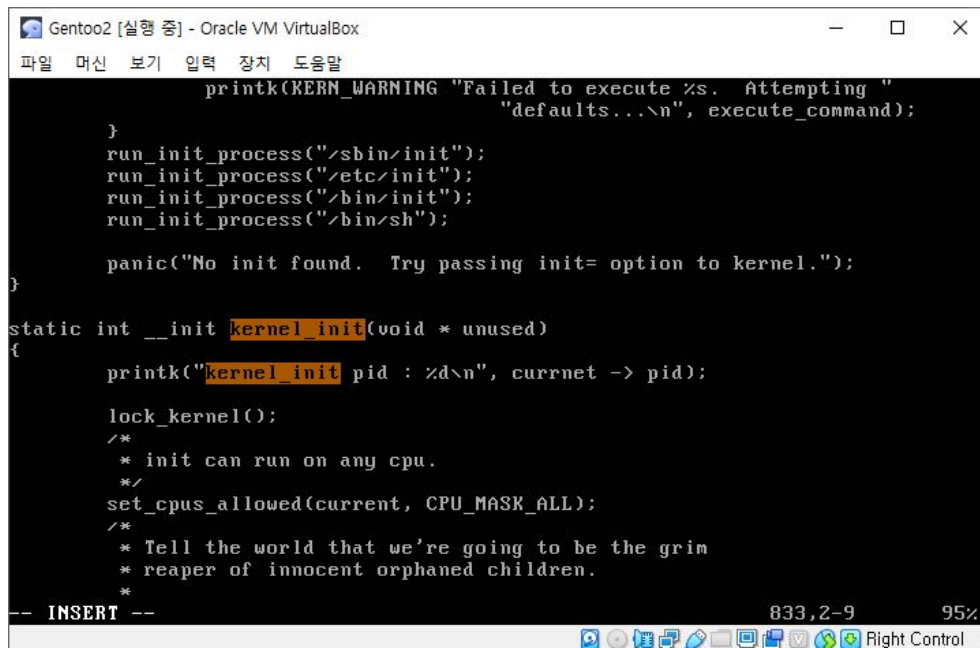
```
Gentoo2 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

"ex7.c" [converted] 37L, 417C written
localhost / # gcc ex7.c -o ex7
ex7.c: In function 'main':
ex7.c:28: warning: incompatible implicit declaration of built-in function 'exit'
localhost / # ls
a.out  ex1.c  ex4_1.c  ex7.c      opt      sys
bin    ex2    ex4_2    home      p1.c     tmp
boot  ex2.c  ex4_2.c  lib       portage-2008.0_beta1.tar.bz2  usr
dev    ex3    ex5      lost+found  proc     var
etc    ex3.c  ex5.c    metadata.tar.bz2  root
ex1    ex4_1  ex7      mnt       sbin
localhost / # ./ex7
$/bin/ls
I am child to execute /bin/ls
a.out  ex1.c  ex4_1.c  ex7.c      opt      sys
bin    ex2    ex4_2    home      p1.c     tmp
boot  ex2.c  ex4_2.c  lib       portage-2008.0_beta1.tar.bz2  usr
dev    ex3    ex5      lost+found  proc     var
etc    ex3.c  ex5.c    metadata.tar.bz2  root
ex1    ex4_1  ex7      mnt       sbin
$/bin/date
I am child to execute /bin/date
Tue Oct 15 08:21:19 KST 2019
$
```

<ex7.c 실행>

fork()를 실행하여 현재 프로세스(parent process)에 대한 child process를 복제한다. child process에선 execve()함수의 인자로 입력받은 것(buf)을 실행한다. 이런 행위가 무한 반복된다.

2) Print the pid of the current process (current->pid) inside rest_init() and kernel_init(). The pid printed inside rest_init() will be 0, but the pid inside kernel_init() is 1. 0 is the pid of the kernel itself. Why do we have pid=1 inside kernel_init()?



```
printk(KERN_WARNING "Failed to execute %s. Attempting "
        "defaults...\n", execute_command);
}
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");

panic("No init found. Try passing init= option to kernel.");
}

static int __init kernel_init(void * unused)
{
    printk("kernel_init pid : %d\n", current->pid);

    lock_kernel();
    /*
     * init can run on any cpu.
     */
    set_cpus_allowed(current, CPU_MASK_ALL);
    /*
     * Tell the world that we're going to be the grim
     * reaper of innocent orphaned children.
     */
    -- INSERT --
833,2-9 95%
```

<kernel_init()함수에 pid출력문 추가>

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

/*
 * We need to finalize in a non-__init function or else race conditions
 * between the root thread and the init thread may cause start_kernel to
 * be reaped by free_initmem before the root thread has proceeded to
 * cpu_idle.
 *
 * gcc-3.4 accidentally inlines this function, so use noinline.
 */

static void noinline __init_refok rest_init(void)
{
    __releases(kernel_lock)

    printk("rest_init pid : %d\n", current->pid);

    int pid;

    kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
    numa_default_policy();
    pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
    kthreadd_task = find_task_by_pid(pid);
    unlock_kernel();
}

436,10-17  48%
[Icons] Right Control
```

<rest_init()함수에 pid출력문 추가>

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

Checking 'hlt' instruction... OK.
SMP alternatives: switching to UP code
Freeing SMP alternatives: 17k freed
ACPI: Core revision 20070126
Parsing all Control Methods:
Table [DSDT](id 0001) - 253 Objects with 28 Devices 80 Methods 4 Regions
Parsing all Control Methods:
Table [SSDT](id 0002) - 0 Objects with 0 Devices 0 Methods 0 Regions
    tbxface-0598 [00] tb_load_namespace      : ACPI Tables successfully acquired
ACPI: setting ELCR to 0200 (from 0e00)
evxfevnt-0091 [00] enable                  : Transition to ACPI mode successful
rest_init pid : 0
kernel_init pid : 1
CPU0: AMD Ryzen 7 2700X Eight-Core Processor      stepping 02
SMP motherboard not detected.
Brought up 1 CPUs
net_namespace: 244 bytes
NET: Registered protocol family 16
No dock devices found.
ACPI: bus type pci registered
PCI: PCI BIOS revision 2.10 entry at 0xfda26, last bus=0
PCI: Using configuration type 1
Setting up standard PCI resources
evgpeblk-0956 [00] ev_create_gpe_block      : GPE 00 to 07 [_GPE] 1 regs on int 0x9
107,11  33%
[Icons] Right Control
```

<dmesg 결과>

rest_init()함수는 kernel_thread() 함수를 호출해서 별도의 스레드를 만든 다음 kernel_init 함수를 수행하게 한다. 즉, process를 복제했기 때문에, 'pid : 1'이 나타난 것이다.