1. 다음 프로그램은 ssu_hole.txt 파일을 생성한 후 첫 부분에 buf1 스트링을 쓰고, 오프셋의 위치를 파일의 처음부터 15000 만큼 이동시키고 buf2 스트링을 쓰는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define CREAT_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

char buf1[] = "1234567890";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    char *fname = "ssu_hole.txt";
    int fd;

    if ((fd = creat(fname, CREAT_MODE)) < 0) {
        fprintf(stderr, "creat error for %s\n", fname);
        exit(1);
    }

    if (write(fd, buf1, 12) != 12) {
        fprintf(stderr, "buf1 write error\n");
        exit(1);
    }

    if (lseek(fd, 15000, SEEK_SET) < 0) {
        fprintf(stderr, "lseek error\n");
        exit(1);
    }

    if (write(fd, buf2, 12) != 12) {
        fprintf(stderr, "buf2 write error\n");
        exit(1);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# ls -l ssu_hole.txt
-rw-r--r-- 1 root root 15012 Jan 3 03:53 ssu_hole.txt
root@localhost:/home/oslab# od -c ssu_hole.txt
0000000   1   2   3   4   5   6   7   8   9   0  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0035220  \0  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0035240   I   J  \0  \0
0035244
```

2. 다음은 주어진 파일을 umask()를 통해 마스크 값을 지정하고, 생성된 파일의 접근 권한을 확인한다. umask()를 호출하여 파일 생성 시 파일의 마스크 값을 0으로 만들고 생성한 ssu_file1의 파일 접근 권한을 RW_MODE로 변경한다. umask()로 그

룹 사용자와 다른 사용자의 읽기, 쓰기 권한에 대해 마스크를 설정하여 ssu_file2 파일은 파일의 소유자만 읽기, 쓰기가 가능하게 한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#Include <sys/stat.h>

#define RW_MODE
(S_IRUSR¦S_IWUSR¦S_IRGRP¦S_IWGRP¦S_IROTH¦S_IWOTH)

int main(void)
{
    char *fname1 = "ssu_file1" ;
    char *fname2 = "ssu_file2" ;

    umask(0);

    if (creat(fname1, RW_MODE) < 0) {
        fprintf(stderr, "creat error for %s\n" , fname1);
        exit(1);
    }
    umask(S_IRGRP ¦ S_IWGRP ¦ S_IROTH ¦ S_IWOTH);

    if (creat(fname2, RW_MODE) < 0) {
        fprintf(stderr, "creat error for %s\n" , fname2);
        exit(1);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# umask 002
root@localhost:/home/oslab# umask
0002
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# ls -l ssu_file1 ssu_file2
-rw-rw-rw-   1 root root 0 Jan  8 22:09 ssu_file1
-rw------- 1 root root 0 Jan  8 22:09 ssu_file2
root@localhost:/home/oslab# umask
0002
```

3. 다음 프로그램은 chmod()를 호출하여 인자로 지정한 모드로 파일 접근 권한을 변경한다. 일반 사용자 권한으로 파일 접근 권한을 변경할 수 없기 때문에 에러 메시지가 출력된다. 접근 권한이 변경된 파일은 쉘에서 " ls -l " 명령어를 통해 확인할 수 있다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>

#define MODE_EXEC (S_IXUSR¦S_IXGRP¦S_IXOTH)

int main(int argc, char *argv[])
{
```

```
    struct stat statbuf;
    int i;

    if (argc < 2) {
        fprintf(stderr, "usage: %s <file1><file2> ... <fileN>\n" , argv[0]);
        exit(1);
    }

    for(i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) {
            fprintf(stderr, "%s : stat error\n" , argv[i]);
            continue;
        }

        statbuf.st_mode != MODE_EXEC;
        statbuf.st_mode ^= (S_IXGRP|S_IXOTH);
        if(chmod(argv[i], statbuf.st_mode) < 0)
            fprintf(stderr, "%s : chmod error\n", argv[i]);
        else
            printf("%s : file permission was changed.\n", argv[i]);
    }
    exit(0);
}
```

실행결과 [일반 사용자 권한으로 실행]

```
root@localhost:/home/oslab# mkdir ssu_test_dir
root@localhost:/home/oslab# ./a.out /bin/su /home/oslab/ssu_test_dir
/bin/su : chmod error
/home/oslab/ssu_test_dir : file permission was changed.
```

4. 다음 프로그램은 첫 번째 인자로 입력받은 파일의 심볼릭 링크 파일을 두 번째 인자로 입력받은 파일이름으로 생성하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

int main(int argc, char *argv[])
{
    if(argc != 3){
        fprintf(stderr, "usage: %s <actualname> <symname>\n", argv[0]);
        exit(1);
    }

    if(symlink(argv[1], argv[2]) < 0){
        fprintf(stderr, "symlink error\n");
        exit(1);
    }
    else
        printf("symlink: %s -> %s\n", argv[2], argv[1]);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out /home/oslab/oslab /bin/oslab
symlink: /bin/oslab -> /home/oslab/oslab
root@localhost:/home/oslab# oslab
This is oslab file
```

5. 다음 프로그램은 getcwd()를 호출하여 chdir()로 변경한 현재 작업 디렉토리를 출력한다. 작업 디렉토리를 변경 후 getcwd()를 통해 작업디렉토리가 변경된 것을 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define PATH_MAX 1024

int main(void)
{
    char *pathname;

    if(chdir("/home/oslab") < 0){
        fprintf(stderr, "chdir error\n");
        exit(1);
    }

    pathname = malloc(PATH_MAX);

    if(getcwd(pathname, PATH_MAX) == NULL){
        fprintf(stderr, "getcwd error\n");
        exit(1);
    }

    printf("current directory = %s\n", pathname);
    exit(0);
}
```

실행결과

```
root@localhost:~/ssu_osidr$ ./a.out
current directory = /home/oslab
```

6. 다음 프로그램은 내용 첫 줄은 버퍼를 설정하여 출력할 내용을 한꺼번에 출력하고, 두 번째 줄은 설정되어 있던 버퍼를 해제하여 출력할 내용을 즉시 출력하도록 하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define BUFFER_SIZE 1024

int main(void)
{
    char buf[BUFFER_SIZE];

    setbuf(stdout, buf);
    printf("Hello, ");
    sleep(1);
    printf("OSLAB!!");
```

```
        sleep(1);
        printf("\n");
        sleep(1);

        setbuf(stdout, NULL);
        printf("How");
        sleep(1);
        printf(" are");
        sleep(1);
        printf(" you?");
        sleep(1);
        printf("\n");
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
Hello, OSLAB!!
How are you?
```

7. 다음 프로그램은 자식 프로세스들의 종료 상태를 출력하는 프로그램이다. 정상적으로 종료되었을 때에는 exit()의 인자값을, abort()를 호출하거나 0으로 나누기 연산을 하면 시그널을 받아 비정상적으로 종료한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

void ssu_echo_exit(int status);

int main(void)
{
        pid_t pid;
        int status;

        if((pid = fork()) < 0){
                fprintf(stderr, "fork error\n");
                exit(1);
        }
        else if(pid == 0)
                exit(7);
        if(wait(&status) != pid){
                fprintf(stderr, "wait error\n");
                exit(1);
        }

        ssu_echo_exit(status);

        if((pid = fork()) < 0){
                fprintf(stderr, "fork error\n");
                exit(1);
        }
        else if(pid == 0)
                abort();
```

```
    if(wait(&status) != pid){
        fprintf(stderr, "wait error\n");
        exit(1);
    }

    ssu_echo_exit(status);

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if(pid == 0)
        status /= 0;
    if(wait(&status) != pid){
        fprintf(stderr, "wait error\n");
        exit(1);
    }

    ssu_echo_exit(status);
    exit(0);
}

void ssu_echo_exit(int status){
    if(WIFEXITED(status))
        printf("normal termination, exit status = %d\n", WEXITSTATUS(status));
    else if(WIFSIGNALED(status))
        printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),
#ifdef WCOREDUMP
                WCOREDUMP(status) ? " (core file generated)" : "");
#else
                "");
#endif
    else if(WIFSTOPPED(status))
        printf("child stopped, signal number = %d\n", WSTOPSIG(status));
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
normal termination, exit status = 7
abnormal termination, signal number = 6 (core file generated)
abnormal termination, signal number = 8 (core file generated)
```

8. 아래 보기를 이용하여 gcc의 컴파일 과정과 각 과정에서 사용되는 명령을 완성하시오. (숫자만 작성)

1. 링킹, 2. 목적코드 생성, 3. 전처리, 4. 컴파일	5. as, 6. cc1, 7. cc1 -E, 8. collect2

과정	명령		과정	명령		과정	명령		과정	명령
3	7	→	2	6	→	4	5	→	1	8

9. 다음은 주어진 프로그램을 gcc로 컴파일 후 gdb로 디버깅 하는 과정을 보여준다. 아래 실행결과와 같은 결과를 얻을 수 있도록 명령을 작성하시오.

<bug.c>

```
#include <stdio.h>
#define NUM 5

int score[NUM];

int sum(int cnt){
        int i;
        int sum;

        for(i = 0; i < cnt ; i++){
                sum += score[i];
        }
        return sum;
}

int main()
{
        int i = 0;
        int cnt = 0;

        printf("input scores. input -1 to finish.\n");

        for(i = 0;i < NUM;i++)  {
                printf("score #%d : ", cnt+1);
                scanf("%d", score[cnt]);
                if(score[cnt] == -1)
                        break;
                cnt++;
        }

        printf("%d scores read.\n", cnt);
        printf("--- result ---\n");
        printf("sum : %d avg : %d\n", sum(cnt), sum(cnt)/cnt);

        return 0;
}
```

실행결과

```
:~/source/gdb$ gcc -g bug.c or gcc -ggdb bug.c
:~/source/gdb$ ./a.out
input scores. input -1 to finish.
score #1 : 90
score #2 : 80
score #3 : 70
score #4 : 60
score #5 : 50
5 scores read.
--- result ---
sum : -1217277252 avg : -243455520
:~/source/gdb$ gdb a.out -q
Reading symbols from a.out...done.
(gdb) b(reak) sum or b(reak) 10
Breakpoint 1 at 0x8048491: file bug.c, line 10.
(gdb) i(nfo) b or i(nfo) break or i(nfo) breakpoint or i(nfo) breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048491 in sum at bug.c:10
(gdb) r(un)
Starting program: /home/oslab/source/gdb/a.out
input scores. input -1 to finish.
score #1 : 90
score #2 : 80
score #3 : 70
score #4 : 60
score #5 : 50
5 scores read.
--- result ---
```

```
Breakpoint 1, sum (cnt=5) at bug.c:10
10                  for(i = 0; i < cnt ; i++){
(gdb)
```

10. 다음 프로그램은 dup2()를 호출하여 표준 출력 1번 파일 디스크립터를 4번으로 복사하고 4번 파일 디스크립터를 인자로 하여 write()를 호출하면 표준 출력에 쓰는 것과 같은 효과를 보인다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFFER_SIZE 1024

int main(void)
{
    char buf[BUFFER_SIZE];
    char *fname = "ssu_test.txt";
    int fd;
    int length;

    if ((fd = open(fname, O_RDONLY, 0644)) < 0) {
        fprintf(stderr, "open error for %s\n", fname);
        exit(1);
    }

    if (dup2(1, 4) != 4) {
        fprintf(stderr, "dup2 call failed\n");
        exit(1);
    }

    while (1) {
        length = read(fd, buf, BUFFER_SIZE);

        if (length <= 0)
            break;

        write(4, buf, length);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_dup2_2
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
```

11. 다음 프로그램은 rename()을 호출하여 파일의 이름을 ssu_test1.txt에서 ssu_test2.txt로 변경 하여 두 번째 open()이 실패하는 것을 보여준다. 아래 실행결과를 보고 빈칸을 채우시오.

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

```
#include <fcntl.h>

int main(int argc, char *argv[])
{
    int fd;

    if (argc != 3) {
        fprintf(stderr, "usage: %s <oldname> <newname>\n", argv[0]);
        exit(1);
    }

    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        fprintf(stderr, "first open error for %s\n", argv[1]);
        exit(1);
    }
    else
        close(fd);

    if (rename(argv[1], argv[2]) < 0) {
        fprintf(stderr, "rename error\n");
        exit(1);
    }

    if ((fd = open(argv[1], O_RDONLY)) < 0)
        printf("second open error for %s\n", argv[1]);
    else {
        fprintf(stderr, "it's very strange!\n");
        exit(1);
    }

    if ((fd = open(argv[2], O_RDONLY)) < 0) {
        fprintf(stderr, "third open error for %s\n", argv[2]);
        exit(1);
    }

    printf("Everything is good!\n");
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vi ssu_test1.txt
root@localhost:/home/oslab# ./ssu_rename ssu_test1.txt ssu_test2.txt
second open error for ssu_test1.txt
Everything is good!
root@localhost:/home/oslab# ls ssu_test2.txt
ssu_test2.txt
```

12. 다음 프로그램은 setvbuf()를 사용하여 버퍼를 조작하는 것을 보여준다. setvbuf()를 테스트하기 앞서 tty 명령어를 통해 터미널의 번호를 확인한다. tty 명령어는 현재 표준입력에 접속된 터미널 장치 파일 이름을 출력하는 명령어로 현재 장치 파일 이름을 알아낸 후, 해당 장치의 버퍼를 조작한다. 아래 프로그램을 실행하기 위해 현재 버퍼인 /dev/pts/19를 오픈하고, setvbuf()를 설정하는 ssu_setbuf()를 호출한다. ssu_setbuf()의 첫 번째 인자인 파일에 대해 버퍼를 설정하고, 해당 파일의 파일 디스크립터를 얻는다. 얻은 파일 디스크립터를 통해 그에 맞는 모드와 크기를 설정한 후 setvbuf()를 호출하여 해당 파일의 버퍼를 설정한다.

"Hello, UNIX!!" 출력은 버퍼가 설정된 후 실행되기 때문에 버퍼에 넣은 후 한 번에 출력하고, "HOW ARE YOU?" 출력은

버퍼가 NULL로 설정된 후 실행되기 때문에 fprintf()를 호출할 때마다 버퍼에 넣지 않고 바로 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

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

#define BUFFER_SIZE 1024

void ssu_setbuf(FILE *fp, char *buf);

int main(void)
{
    char buf[BUFFER_SIZE];
    char *fname = "/dev/pts/19";
    FILE *fp;

    if ((fp = fopen(fname, "w")) == NULL) {
        fprintf(stderr, "fopen error for %s", fname);
        exit(1);
    }

    ssu_setbuf(fp, buf);
    fprintf(fp, "Hello, ");
    sleep(1);
    fprintf(fp, "UNIX!!");
    sleep(1);
    fprintf(fp, "\n");
    sleep(1);
    ssu_setbuf(fp, NULL);
    fprintf(fp, "HOW");
    sleep(1);
    fprintf(fp, " ARE");
    sleep(1);
    fprintf(fp, " YOU?");
    sleep(1);
    fprintf(fp, "\n");
    sleep(1);
    exit(0);
}

void ssu_setbuf(FILE *fp, char *buf) {
    size_t size;
    int fd;
    int mode;

    fd = fileno(fp);

    if (isatty(fd))
        mode = _IOLBF;
    else
        mode = _IOFBF;

    if (buf == NULL) {
```

```
        mode = _IONBF;
        size = 0;
    }
    else
        size = BUFFER_SIZE;

    setvbuf(fp, buf, mode, size);
}
```

실행결과

```
oslab@localhost:~$ tty
/dev/pts/19
oslab@localhost:~$ ./ssu_setvbuf
Hello, UNIX!!
HOW ARE YOU?
```

13. (1) 다음 프로그램은 gcc로 컴파일 시에 어떤 옵션을 주면 아래 보이는 컴파일 결과를 확인할 수 있는가? 단, 정확한 옵션만 파일에 쓰시오. 옵션이 필요 없는 경우에는 빈(NULL) 파일로 생성하시오. -Wconversion w_ex9.c

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
        double x = -3.14;
        double y = abs(x);   /* fabs(x)*/
        printf ("x = %g |x| = %g\n", x, y);
        return 0;
}
```

컴파일 결과

```
w_ex9.c: In function ‘main’:
w_ex9.c:7:17: warning: conversion to ‘int’ from ‘double’ may alter its value [-Wfloat-conversion]
  double y = abs(x);   /* fabs(x)*/
                 ^

:~/source/gcc/warning$
```

13. (2) 다음은 프로그램을 gcc로 컴파일 후 gdb로 디버깅 하는 과정을 보여주고 있다. 빈칸의 gdb의 적절한 명령어를 쓰시오.

```
#include <stdio.h>
#define NUM 5

int score[NUM];

int sum(int cnt){
        int i;
        int sum;

        for(i = 0; i < cnt ; i++){
                sum += score[i];
        }
        return sum;
}

int main()
{
        int i = 0;
        int cnt = 0;

        printf("input scores. input -1 to finish.\n");
```

```
        for(i = 0;i < NUM;i++)  {
                printf("score #%d : ", cnt+1);
                scanf("%d", score[cnt]);
                if(score[cnt] == -1)
                        break;
                cnt++;
        }

        printf("%d scores read.\n", cnt);
        printf("--- result ---\n");
        printf("sum : %d avg : %d\n", sum(cnt), sum(cnt)/cnt);

        return 0;
}
```

컴파일 및 gdb 실행 결과

```
:~/source/gdb$ gcc -Wall -W -g bug.c
bug.c: In function ‘main’:
bug.c:25:9: warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’ [-Wformat=]
   scanf("%d", score[cnt]);
         ^
:~/source/gdb$
:~/source/gdb$ ./a.out
input scores. input -1 to finish.
score #1 : 102
세그멘테이션 오류 (core dumped)
:~/source/gdb$
:~/source/gdb$ gdb a.out -q
Reading symbols from a.out...done.
(gdb) r
Starting program: /home/oslab/source/gdb/a.out
input scores. input -1 to finish.
score #1 : 102

Program received signal SIGSEGV, Segmentation fault.
_IO_vfscanf_internal (s=0xb7fbb5a0 <_IO_2_1_stdin_>, format=0x804865f "%d", argptr=0xbffff604 "", errp=0x0)
    at vfscanf.c:1902
1902    vfscanf.c: 그런 파일이나 디렉터리가 없습니다.
(gdb) bt
#0  _IO_vfscanf_internal (s=0xb7fbb5a0 <_IO_2_1_stdin_>, format=0x804865f "%d", argptr=0xbffff604 "",
    errp=0x0) at vfscanf.c:1902
#1  0xb7e6513e in __isoc99_scanf (format=0x804865f "%d") at isoc99_scanf.c:37
#2  0x08048520 in main () at bug.c:25
(gdb) q
A debugging session is active.

        Inferior 1 [process 3812] will be killed.

Quit anyway? (y or n) y
```

14. 다음 프로그램은 putenv()를 통해 환경변수를 등록하고 출력을 통해 등록된 환경 변수를 확인한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>

void ssu_printenv(char *label, char ***envpp);

extern char **environ ;

int main(int argc, char *argv[],    char *envp[]    )
{
```

```
    ssu_printenv("Initially", &envp);
    putenv("TZ=PST8PDT")  ;
    ssu_printenv("After changing TZ", &envp);
    putenv("WARNING=Don't use envp after putenv()") ;
    ssu_printenv("After setting a new variable", &envp);
    printf("value of WARNING is %s\n",   getenv("WARNING")  );
    exit(0);
}

void ssu_printenv(char *label, char ***envpp) {
    char **ptr;

    printf("---- %s ---\n", label);
    printf("envp is at %8o and contains %8o\n", envpp, *envpp);
    printf("environ is at %8o and contains %8o\n", &environ, environ);
    printf("My environment variable are:\n");

    for (ptr = environ; *ptr; ptr++)
        printf("(%8o) = %8o -> %s\n", ptr, *ptr, *ptr);

    printf("(%8o) = %8o\n", ptr, *ptr);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
---- Initially ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 27754643474
My environment variable are:
(27754643474) = 27754644205 -> SHELL=/bin/bash
(27754643500) = 27754644225 -> TERM=xterm-256color
(27754643504) = 27754644251 -> USER=root

(중략)

(27754643624) = 27754647723 -> _=./a.out
(27754643630) = 27754647742 -> OLDPWD=/home
(27754643634) =          0
---- After changing TZ ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
My environment variable are:
(1004432020) = 27754644205 -> SHELL=/bin/bash
(1004432024) = 27754644225 -> TERM=xterm-256color
(1004432030) = 27754644251 -> USER=root

(중략)

(1004432154) = 27754647742 -> OLDPWD=/home
(1004432160) = 1001103372 -> TZ=PST8PDT
(1004432164) =          0
---- After setting a new variable ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
My environment variable are:
```

```
(1004432020) = 27754644205 -> SHELL=/bin/bash
(1004432024) = 27754644225 -> TERM=xterm-256color
(1004432030) = 27754644251 -> USER=root

(중략)

(1004432160) = 1001103372 -> TZ=PST8PDT
(1004432164) = 1001103430 -> WARNING=Don't use envp after putenv()
(1004432170) =          0
value of WARNING is Don't use envp after putenv()
```

15. 다음 프로그램은 times()를 사용하여 system() 실행의 클록시간, 사용자 CPU 시간, 시스템 CPU 시간을 측정하여 보여준다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/times.h>
#include <sys/wait.h>

void ssu_do_cmd(char *cmd);
void ssu_print_times(clock_t real, struct tms *tms_start, struct tms *tms_end);
void ssu_echo_exit(int status);

int main(int argc, char *argv[])
{
    int i;

    setbuf(stdout, NULL);

    for (i = 1; i < argc; i++)
        ssu_do_cmd(argv[i]);

    exit(0);
}

void ssu_do_cmd(char *cmd) {
    struct tms tms_start, tms_end;
    clock_t start, end;
    int status;

    printf("\ncommand: %s\n", cmd);
    if (  (start = times(&tms_start)) == -1 ) {
        fprintf(stderr, "times error\n");
        exit(1);
    }

    if ((status = system(cmd)) < 0) {
        fprintf(stderr, "system error\n");
        exit(1);
    }

    if (  (end = times(&tms_end)) == -1 ) {
        fprintf(stderr, "times error\n");
```

```
            exit(1);
        }

        ssu_print_times(end-start, &tms_start, &tms_end);
        ssu_echo_exit(status);
}

void ssu_print_times(clock_t real, struct tms *tms_start, struct tms *tms_end) {
        static long clocktick = 0;

        if (clocktick == 0)
            if (  (clocktick = sysconf(_SC_CLK_TCK)) < 0  ) {
                fprintf(stderr, "sysconf error\n");
                exit(1);
            }

        printf("  real:  %7.2f\n", real / (double) clocktick);
        printf("  user:  %7.2f\n",
          (tms_end->tms_utime - tms_start->tms_utime) / (double) clocktick);
        printf("  sys:   %7.2f\n",
          (tms_end->tms_stime - tms_start->tms_stime) / (double) clocktick);
        printf("  child user:  %7.2f\n",
          (tms_end->tms_cutime - tms_start->tms_cutime) / (double) clocktick);
        printf("  child sys:   %7.2f\n",
          (  tms_end->tms_cstime - tms_start->tms_cstime  ) / (double) clocktick);
}

void ssu_echo_exit(int status) {
        if (WIFEXITED(status))
            printf("normal termination, exit status = %d\n",
                    WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
            printf("abnormal termination, signal number = %d%s\n",
                    WTERMSIG(status),
#ifdef  WCOREDUMP
                    WCOREDUMP(status) ? " (core file generated)" : "");
#else
                    "");
#endif
        else if (WIFSTOPPED(status))
            printf("child stopped, signal number = %d\n",
                    WSTOPSIG(status));
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out "sleep 5" date

command: sleep 5
  real:      5.00
  user:      0.00
  sys:       0.00
  child user:      0.00
  child sys:       0.00
normal termination, exit status = 0
```

```
command: date
Tue Jan 10 22:17:37 PST 2017
  real:       0.00
  user:       0.00
  sys:        0.00
  child user:       0.00
  child sys:        0.00
normal termination, exit status = 0
```

16. 다음 프로그램은 쓰레드를 생성 후 프로세스 ID와 쓰레드 ID를 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>

void *ssu_thread(void *arg);

int main(void)
{
    pthread_t tid;
    pid_t pid;

    if (  pthread_create(&tid, NULL, ssu_thread, NULL) != 0 ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    pid = getpid() ;
    tid = pthread_self() ;
    printf("Main Thread: pid %u tid %u \n",
            (unsigned int)pid, (unsigned int)tid);
    sleep(1);
    exit(0);
}

void *ssu_thread(void *arg) {
    pthread_t tid;
    pid_t pid;

    pid = getpid();
    tid = pthread_self() ;
    printf("New Thread: pid %d tid %u \n", (int)pid, (unsigned int)tid);
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# gcc -o ssu_pthread_create_1 ssu_pthread_create_1.c -lpthread
root@localhost:/home/oslab# ./ssu_pthread_create_1
Main Thread: pid 3222 tid 3075864320
New Thread: pid 3222 tid 3075861312
```

17. 다음 프로그램은 메인 쓰레드가 pthread_join()을 호출하여 생성된 쓰레드가 종료될 때까지 기다린다. 쓰레드를 생성할

때 ssu_thread1을 먼저 생성하고 쓰레드 아이디는 tid1에 저장한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *ssu_thread1(void *arg);
void *ssu_thread2(void *arg);

int main(void)
{
    pthread_t tid1, tid2;

    if ( pthread_create(&tid1, NULL, ssu_thread1, NULL) != 0 ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    if (  pthread_create(&tid2, NULL, ssu_thread2, NULL) != 0  ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    printf("thread1의 리턴을 기다림\n");
    pthread_join(tid1, NULL)  ;
    exit(0);
}

void *ssu_thread1(void *arg) {
    int i;

    for (i = 5; i != 0; i--) {
        printf("thread1: %d\n", i);
        sleep(1);
    }

    printf("thread1 complete\n");
    return NULL;
}

void *ssu_thread2(void *arg) {
    int i;

    for (i = 8; i != 0; i--) {
        printf("thread2: %d\n", i);
        sleep(1);
    }

    printf("thread2 complete\n");
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
```

```
thread1의 리턴을 기다림
thread2: 8
thread1: 5
thread2: 7
thread1: 4
thread2: 6
thread1: 3
thread2: 5
thread1: 2
thread2: 4
thread1: 1
thread1 complete
thread2: 3
```

18. 다음 프로그램은 pthread_cond_signal()을 이용하여 두 쓰레드의 실행 순서를 지정한다. mutex와 cond의 초기화는 메크로를 사용해야 하며, mutex의 초기화를 먼저 실행한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define VALUE_DONE  10
#define VALUE_STOP1  3
#define VALUE_STOP2  6

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER ;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER ;

void *ssu_thread1(void *arg);
void *ssu_thread2(void *arg);

int glo_val = 0;

int main(void)
{
    pthread_t tid1, tid2;

    pthread_create(&tid1, NULL, &ssu_thread1, NULL);
    pthread_create(&tid2, NULL, &ssu_thread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("final value: %d\n", glo_val);
    exit(0);
}

void *ssu_thread1(void *arg) {
    while(1) {
        pthread_mutex_lock(&lock);
        thread_cond_wait(&cond, &lock) ;
        glo_val++;
        printf("global value ssu_thread1: %d\n", glo_val);
        pthread_mutex_unlock(&lock);

        if (glo_val >= VALUE_DONE)
```

```
                return NULL;
        }
}

void *ssu_thread2(void *arg) {
        while(1) {
                pthread_mutex_lock(&lock);
                if ( glo_val < VALUE_STOP1 || glo_val > VALUE_STOP2 )
                        pthread_cond_signal(&cond) ;
                else {
                        glo_val++;
                        printf("global value ssu_thread2: %d\n", glo_val);
                }

                pthread_mutex_unlock(&lock);

                if (glo_val >= VALUE_DONE)
                        return NULL;
        }
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
global value ssu_thread1: 1
global value ssu_thread1: 2
global value ssu_thread1: 3
global value ssu_thread2: 4
global value ssu_thread2: 5
global value ssu_thread2: 6
global value ssu_thread2: 7
global value ssu_thread1: 8
global value ssu_thread1: 9
global value ssu_thread1: 10
final value: 10
```

19. 다음 프로그램은 fcntl()을 사용하여 nonblocking을 설정하는 것을 보여준다. fcntl()의 플래그는 메크로를 사용해야 한다. 아래 실행결과를 보고 빈칸을 채우시오.

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

void set_flags(int fd, int flags);
void clr_flags(int fd, int flags);

char    buf[500000];

int main(void)
{
        int     ntowrite, nwrite;
        char    *ptr;

        ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
```

```
    fprintf(stderr, "reading %d bytes\n", ntowrite);

    set_flags(   STDOUT_FILENO, O_NONBLOCK   );

    ptr = buf;
    while (ntowrite > 0) {
        errno = 0;
        nwrite = write(STDOUT_FILENO, ptr, ntowrite);
        fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

        if (nwrite > 0) {
            ptr += nwrite;
            ntowrite -= nwrite;
        }
    }
    clr_flags( STDOUT_FILENO, O_NONBLOCK );
    exit(0);
}

void set_flags(int fd, int flags) // 파일 상태 플래그를 설정함
{
    int     val;

    if ( (val = fcntl(fd, F_GETFL, 0)) < 0 ) {
        fprintf(stderr, "fcntl F_GETFL failed");
        exit(1);
    }

    val |= flags ;

    if ( fcntl(fd, F_SETFL, val) < 0 ) {
        fprintf(stderr, "fcntl F_SETFL failed");
        exit(1);
    }
}

void clr_flags(int fd, int flags) // 파일 상태 플래그를 해제함
{
    int     val;

    if ( (val = fcntl(fd, F_GETFL, 0)) < 0 ) {
        fprintf(stderr, "fcntl F_GETFL failed");
        exit(1);
    }

    val &= ~flags ;

    if ( fcntl(fd, F_SETFL, val) < 0 ) {
        fprintf(stderr, "fcntl F_SETFL failed");
        exit(1);
    }
}
```

실행결과

```
root@localhost:/home/oslab# ls -l ssu_test1.txt
```

```
-rw-r--r-- 1 root root 500000 Jun  8 04:11 ssu_test1.txt
root@localhost:/home/oslab# ./a.out <ssu_test1.txt >ssu_test2.txt
reading 500000 bytes
nwrite = 500000, errno = 0
root@localhost:/home/oslab# ls -l ssu_test2.txt
-rw-r--r-- 1 root root 500000 Jun  8 04:12 ssu_test2.txt
root@localhost:/home/oslab# ./a.out <ssu_test1.txt 2> ssu_test3.txt

[ssu_test3.txt]
reading 500000 bytes
nwrite = 8192, errno = 0
nwrite = -1, errno = 11
nwrite = -1, errno = 11
..
..
nwrite = -1, errno = 11
nwrite = 3840, errno = 0
nwrite = -1, errno = 11
..
..
```

20. 다음 프로그램은 파일 open() 후 자식 프로세스 생성 시 자식 프로세스에게 물려주는 플래그를 확인하는 것을 보여준다. open()된 파일은 읽기, 쓰기 모드이며 fcntl()의 플래그는 메크로를 사용해야 한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main(void)
{
    char *filename = "ssu_test.txt";
    int fd1, fd2;
    int flag;

    if ( (fd1 = open(filename, O_RDWR | O_APPEND, 0644)) < 0 ) {
        fprintf(stderr, "open error for %s\n", filename);
        exit(1);
    }

    if ( fcntl(fd1, F_SETFD, FD_CLOEXEC) == -1 ) {
        fprintf(stderr, "fcntl F_SETFD error\n");
        exit(1);
    }

    if (  (flag = fcntl(fd1, F_GETFL, 0)) == -1  ) {
        fprintf(stderr, "fcntl F_GETFL error\n");
        exit(1);
    }

    if ( flag & O_APPEND )
        printf("fd1 : O_APPEND flag is set.\n");
```

```
    else
        printf("fd1 : O_APPEND flag is NOT set.\n");

    if ((flag = fcntl(fd1, F_GETFD, 0)) == -1) {
        fprintf(stderr, "fcntl F_GETFD error\n");
        exit(1);
    }

    if (  flag & FD_CLOEXEC  )
        printf("fd1 : FD_CLOEXEC flag is set.\n");
    else
        printf("fd1 : FD_CLOEXEC flag is NOT set.\n");

    if ((fd2 = fcntl(fd1, F_DUPFD, 0)) == -1) {
        fprintf(stderr, "fcntl F_DUPFD error\n");
        exit(1);
    }

    if ((flag = fcntl(fd2, F_GETFL, 0)) == -1) {
        fprintf(stderr, "fcntl F_GETFL error\n");
        exit(1);
    }

    if ( flag & O_APPEND )
        printf("fd2 : O_APPEND flag is set.\n");
    else
        printf("fd2 : O_APPEND flag is NOT set.\n");

    if ((flag = fcntl(fd2, F_GETFD, 0)) == -1) {
        fprintf(stderr, "fcntl F_GETFD error\n");
        exit(1);
    }

    if ( flag & FD_CLOEXEC )
        printf("fd2 : FD_CLOEXEC flag is set.\n");
    else
        printf("fd2 : FD_CLOEXEC flag is NOT set.\n");

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
fd1 : O_APPEND flag is set.
fd1 : FD_CLOEXEC flag is set.
fd2 : O_APPEND flag is set.
fd2 : FD_CLOEXEC flag is NOT set.
```

21. 다음 프로그램은 시그널 집합을 만들어서 그 집합에 시그널을 추가한 다음, sigprocmask() 호출을 통해서 시그널을 블록시켰다가 다시 블록을 해제하는 것을 보여준다. 단, 프로그램이 실행되는 동안 지정된 시그널 외에는 마스크에 추가되거나 빠지면 안 된다. 아래 실행결과를 보고 빈칸을 채우시오.

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

```
#include <signal.h>

int main(void)
{
    sigset_t    sig_set;
    int count;

    sigemptyset(&sig_set);
    sigaddset(&sig_set, SIGINT);
    sigprocmask(SIG_BLOCK, &sig_set, NULL);

    for (count = 3 ; 0 < count ; count--) {
        printf("count %d\n", count);
        sleep(1);
    }

    printf("Ctrl-C에 대한 블록을 해제\n");
    sigprocmask(SIG_UNBLOCK, &sig_set, NULL);
    printf("count중 Ctrl-C입력하면 이 문장은 출력 되지 않음.\n");

    while (1);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
count 3
count 2
count 1
Ctrl-C에 대한 블록을 해제
count중 Ctrl-C입력하면 이 문장은 출력 되지 않음.
^C
root@localhost:/home/oslab# ./a.out
count 3
^Ccount 2
^Ccount 1
Ctrl-C에 대한 블록을 해제
```

22. 다음 프로그램은 자식 프로세스에서 사용자 모드로 CPU를 사용한 시간과 시스템 모드에서 CPU를 사용한 시간을 출력한다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/resource.h>
#include <sys/wait.h>

double ssu_maketime(struct timeval *time);

void term_stat(int stat);

void ssu_print_child_info(int stat, struct rusage *rusage);
```

```
int main(void)
{
    struct rusage rusage;
    pid_t pid;
    int status;

    if ((pid = fork()) == 0) {
        char *args[] = {"find", "/", "-maxdepth", "4", "-name", "stdio.h", NULL};

        if (  execv("/usr/bin/find", args) < 0  ) {
            fprintf(stderr, "exec error\n");
            exit(1);
        }
    }

    if (  wait3(&status, 0, &rusage) == pid )
        ssu_print_child_info(status, &rusage);
    else {
        fprintf(stderr, "wait3 error\n");
        exit(1);
    }

    exit(0);
}

double ssu_maketime(struct timeval *time) {
    return ((double)time -> tv_sec + (double)time -> tv_usec/1000000.0);
}

void term_stat(int stat) {
    if (  WIFEXITED(stat) )
        printf("normally terminated. exit status = %d\n", WEXITSTATUS(stat));
    else if ( WIFSIGNALED(stat) )
        printf("abnormal termination by signal %d. %s\n", WTERMSIG(stat),
#ifdef WCOREDUMP
                WCOREDUMP(stat)?"core dumped":"no core"
#else
                NULL
#endif
                );
    else if (WIFSTOPPED(stat))
        printf("stopped by signal %d\n", WSTOPSIG(stat));
}

void ssu_print_child_info(int stat, struct rusage *rusage) {
    printf("Termination info follows\n");
    term_stat(stat);
    printf("user CPU time : %.2f(sec)\n", ssu_maketime( &rusage->ru_utime ));
    printf("system CPU time : %.2f(sec)\n", ssu_maketime( &rusage->ru_stime ));
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
/usr/include/stdio.h
find: '/run/user/1000/gvfs': 허가 거부
```

```
Termination info follows
normally terminated. exit status = 1
user CPU time : 0.06(sec)
system CPU time : 0.07(sec)
```

23. 다음은 자식 프로세스를 생성하고 변수값을 수정하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

int main(void)
{
        pid_t pgid;
        pid_t pid;

        pid = getpid();
        pgid = getpgrp();
        printf("pid: %d, pgid: %d\n", pid, pgid);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
pid: 29894, pgid: 29894
```

24. 다음은 시그널 집합에 특정 시그널이 포함되어 있는지 검사하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
int main(void)
{
        sigset_t set;

        sigemptyset(&set);
        sigaddset(&set, SIGINT);

        switch(sigismember(&set, SIGINT))
        {
                case 1:
                        printf("SIGINT is included.\n");
                        break;
                case 0:
                        printf("SIGINT is not included.\n");
                        break;

                default:
                        printf("failed to call sigismember()\n");
        }

        switch(sigismember(&set, SIGSYS))
        {
                case 1:
                        printf("SIGSYS is included.\n");
```

```
                    break;
                case 0:
                    printf("SIGSYS is not included.\n");
                    break;
                default:
                    printf("failed to call sigismember()\n");
        }
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
SIGINT is included.
SIGSYS is not included.
```

25. 다음은 팬딩 중인 시그널 집합을 찾고 SIGINT 시그널이 포함되어 있는지 검사하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
int main(void)
{
        sigset_t pendingset;
        sigset_t sig_set;
        int count = 0;

        sigfillset(&sig_set);
        sigprocmask(SIG_SETMASK, &sig_set, NULL);

        while(1)
        {
                printf("count: %d\n", count++);
                sleep(1);
                if(sigpending(&pendingset) == 0){
                        if(sigismember(&pendingset, SIGINT)){
                                printf("SIGINT가 블록되어 대기 중. 무한 루프를 종료.\n");
                                break;
                        }
                }
        }
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
count: 0
count: 1
count: 2
count: 3
^Zcount: 4
^Zcount: 5
count: 6
count: 7
count: 8
```

^CSIGINT가 블록되어 대기 중. 무한 루프를 종료.

26. 다음은 기존의 시그널 집합을 가지고 시그널이 발생할 때까지 잠시 팬딩하다가 SIGINT 시그널이 발생하면 그것의 시그널 핸들러가 실행되는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>

static void ssu_func(int signo);
void ssu_print_mask(const char *str);

int main(void)
{
        sigset_t new_mask, old_mask, wait_mask;

        ssu_print_mask("program start: ");

        if(signal(SIGINT, ssu_func) == SIG_ERR){
                fprintf(stderr, "signal(SIGINT) error\n");
                exit(1);
        }

        sigemptyset(&wait_mask);
        sigaddset(&wait_mask, SIGUSR1);
        sigemptyset(&new_mask);
        sigaddset(&new_mask, SIGINT);

        if(sigprocmask(SIG_BLOCK, &new_mask, &old_mask) < 0){
                fprintf(stderr, "SIG_BLOCK() error\n");
                exit(1);
        }

        ssu_print_mask("in critical region: ");

        if(sigsuspend(&wait_mask) != -1){
                fprintf(stderr, "sigsuspend() error\n");
                exit(1);
        }

        ssu_print_mask("after return from sigsuspend: ");

        if(sigprocmask(SIG_SETMASK, &old_mask, NULL) < 0){
                fprintf(stderr, "SIG_SETMASK() error\n");
                exit(1);
        }

        ssu_print_mask("program exit: ");
        exit(0);
}

void ssu_print_mask(const char *str){
        sigset_t sig_set;
```

```
        int err_num;

        err_num = errno;

        if(sigprocmask(0, NULL, &sig_set) < 0){
                fprintf(stderr, "sigprocmask() error\n");
                exit(1);
        }

        printf("%s", str);

        if(sigismember(&sig_set, SIGINT))
                printf("SIGINT ");
        if(sigismember(&sig_set, SIGQUIT))
                printf("SIGQUIT ");
        if(sigismember(&sig_set, SIGUSR1))
                printf("SIGUSR1 ");
        if(sigismember(&sig_set, SIGALRM))
                printf("SIGALRM ");
        printf("\n");
        errno = err_num;
}

static void ssu_func(int signo){
        ssu_print_mask("\nin ssu_func: ");
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
program start:
in critical region: SIGINT
^C
in ssu_func: SIGINT SIGUSR1
after return from sigsuspend: SIGINT
program exit:
```

27. 다음은 fcntl()을 호출하여 파일 디스크립터를 복사하는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>

int main(void)
{
        int testfd;
        int fd;

        fd = open("test.txt", O_CREAT);

        testfd = fcntl(fd, F_DUPFD, 5);
        printf("testfd :%d\n", testfd);
        testfd = fcntl(fd, F_DUPFD, 5);
        printf("testfd :%d\n", testfd);
```

```
        getchar();
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
testfd :5
testfd :6
root@localhost:/home/oslab#
```