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. 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.

```c
#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
```

10. 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.

```c
#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
```

11. 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>
```

```c
#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?
```

12. 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()
```

13. 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
```

14. 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
```

15. 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
```

16. 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
```

17. 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
..
..
```

18. 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.
```

19. 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
```

20. 다음 주어진 ssu_test.txt 파일을 한 줄씩 읽고 그 줄을 출력하며 전체 줄 수를 출력하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open(), read(), lseek(), close()를 각 한 번씩 사용할 것
3. open()함수의 에러 처리를 위해 fprintf()를 사용할 것
4. 출력을 위해 printf()를 두 번 사용할 것
5. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 함

<ssu_test.txt>

```
Linux System Programming!
Unix System Programming!
Linux Mania
Unix Mania
```

20.c

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

#define BUFFER_SIZE 1024
#define WORD_MAX 100

int main(void)
{
    int fd;
    int length = 0, offset = 0, count = 0;
    char *fname = "ssu_test.txt";
    char buf[WORD_MAX][BUFFER_SIZE];
    int i;

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

    while ((length = read(fd, buf[count], BUFFER_SIZE)) > 0 ) {
        buf[count][length] = '\0';
        for ( i = 0 ; i < BUFFER_SIZE ; i++) {
            if (buf[count][i] == '\n') {
                if (i == 0)
                    break;
                offset = offset + i + 1;
                lseek(fd, offset, SEEK_SET);
                count++;
            }
```

```
        }
    }

    close(fd);
    for (i = 0 ; i < count ; i++)
        printf("%s\n", buf[i]);
    printf("line number : %d \n", count);
    exit(0);
}
```

실행결과

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

UNIX System Programming!
Linux Mania
Unix Mania

Linux Mania
Unix Mania

Unix Mania

line number : 4
```

21. 구조체 내용을 파일에 저장한 후, 해당 파일을 읽어 출력을 하는 프로그램을 작성하시오. 단, 아래 조건과 실행 결과를 바탕으로 프로그램을 작성 할 것.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ftest.txt. 파일을 fopen()을 통해 쓰기 모드로 한 번 호출하고 에러 처리를 위하여 fprintf()를 사용할 것
3. 구조체의 내용을 파일에 저장하기 위하여 fwrite()를 한 번만 사용할 것
4. ftest.txt 파일을 fopen()을 통해 읽기 모드로 한 번 호출하고 에러 처리를 위하여 fprintf()를 사용할 것
5. fread(), fclose()를 각각 두 번 사용하고 rewind()를 한 번 사용할 것

21.c

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

typedef struct _person {
    char name[10];
    int age;
    double height;
} Person;

int main(void)
{
    FILE *fp;
    int i, res;
    Person ary[3] = {{"Hong GD", 500, 175.4},
        {"Lee SS", 350,180.0},
```

```
        {"King SJ", 500, 178.6}};
    Person tmp;

    if((fp = fopen("ftest.txt", "wb")) == 0) {
        fprintf(stderr, "fopen error!\n");
        exit(1);
    }
    fwrite(&ary,sizeof(ary),1,fp);

    fclose(fp);

    if ((fp = fopen("ftest.txt", "rb")) == 0) {
        fprintf(stderr , "fopen error!\n");
        exit(1);
    }

    printf("[ First print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

    rewind(fp);
    printf("[ Second print]\n");

    while (!feof(fp)) {
        if ((res = fread(&tmp, sizeof(Person), 1, fp)) != 1)
            break;
        printf("%s %d %.2lf\n", tmp.name, tmp.age, tmp.height);
    }

    fclose(fp);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
[ First print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
[ Second print]
Hong GD 500 175.40
Lee SS 350 180.00
King SJ 500 178.60
root@localhost:/home/oslab#
```

22. 아래 프로그램은 파일 디스크립터를 이용하여 "ssu_test.txt"를 읽고 읽은 내용을 표준출력 및 파일 포인터를 사용하여 "ssu_test_new.txt"에 쓴다. 아래 조건과 실행 결과를 바탕으로 프로그램을 작성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. open() <-> fopen(), close() <-> fclose()로 변경할 것
3. read() <-> fread(), write() <-> fwrite()로 변경할 것
4. lseek() <-> fseek()으로 변경할 것
5. 파일을 읽고 쓰는 순서는 변경 후에도 동일해야 함
6. 코드가 변경되어도 실행결과는 동일해야 함
7. ssu_test_new.txt는 없을 경우 생성되어야 하고, 이미 존재할 경우 기존 내용은 제거되어야 함
8. ssu_test_new.txt의 권한은 0644로 할 것

<ssu_test.txt>

```
Linux System Programming!
Unix System Programming!
```

22.c

```
#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";
        char *new_fname = "ssu_test_new.txt";
        int fd;
        FILE *fp;

        fd = open(fname, O_RDONLY);
        fp = fopen(new_fname, "w");
        if(fd < 0 || fp == NULL){
                fprintf(stderr, "open error for %s\n", fname);
                exit(1);
        }

        read(fd, buf, 25);
        buf[25] = 0;
        printf("first printf : %s\n", buf);
        lseek(fd, 1, SEEK_CUR);
        read(fd, buf+25+1, 24);
        buf[25+1+24] = 0;
        printf("second printf : %s\n", buf+25+1);
        close(fd);
        fwrite(buf, 25, 1, fp);
        fwrite(buf+25, 24, 1, fp);
        fclose(fp);
        exit(0);

        //위 코드를 아래 코드로 변경시 정답
        fp = fopen(fname, "r");
        fd = open(new_fname, O_WRONLY|O_CREAT|O_TRUNC, 0644);
```

```
        if(fd < 0 || fp == NULL){
                fprintf(stderr, "open error for %s\n", fname);
                exit(1);
        }
        fread(buf, 25, 1, fp);
        buf[25] = 0;
        printf("first printf : %s\n", buf);
        fseek(fp, 1, SEEK_CUR);
        fread(buf+25+1, 24, 1, fp);
        buf[25+1+24] = 0;
        printf("second printf : %s\n", buf+25+1);
        fclose(fp);
        write(fd, buf, 25);
        write(fd, buf+25, 24);
        close(fd);
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
first printf : Linux System Programming!
second printf : Unix System Programming!
```

ssu_test_new.txt

```
Linux System Programming!Unix System Programming!
```

23. 다음은 in.txt에서 특정 문자열을 찾아 해당 문자열을 삭제하고 삭제된 문자열자리에 hole을 생성해 out.txt로 출력하는 프로그램이다. 아래 조건과 실행 결과를 바탕으로 다음 프로그램을 작성하시오.

< 조 건 >

1. 다음 프로그램은 특정 문자열을 "bcd"로 함
2. 다음의 순서에 따라 구현할 것
   가. in.txt 파일을 오픈, 파일의 사이즈를 size변수에 저장
   나. in.txt 파일의 내용을 읽어 buf 변수에 저장 (in.txt 파일의 크기는 100을 넘지 않음)
   다. in.txt 파일을 닫음
   라. out.txt 파일을 오픈
   마. buf 변수에 저장된 문자열을 out.txt에 출력. 단, out.txt에 대한 출력은 제공된 ssu_write() 함수만 사용할 것 (이외의 출력함수는 허용하지 않음)
   바. 단, pattern과 일치하는 문자열은 원래 문자열을 출력하지 않고 pattern의 길이만큼의 hole을 생성
   사. out.txt 파일을 닫음
3. ssu_write() 함수는 수정을 금지함
4. 프로그램이 정상적으로 수행되었을 시 in.txt파일과 out.txt파일의 크기는 일치할 것

in.txt 예시

```
aaaaabcdbbbbbcdcccbcddbcde
```

23.c

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

int ssu_write(int fd, char *buf);

int main()
```

```
{
        char buf[128];
        char pattern[4] = "bcd";
        char *pos1=buf, *pos2=buf;
        char *fname_in = "in.txt";
        char *fname_out = "out.txt";
        int size;
        int fd1, fd2; //fd1 is input file, fd2 is output file
        int i=0;
        if((fd1 = open(fname_in, O_RDWR)) < 0){
                fprintf(stderr, "in.txt open() error\n");
                exit(1);
        }

        if((size = lseek(fd1, 0, SEEK_END)) < 0){
                fprintf(stderr, "lseek() error\n");
                exit(1);
        }

        if(lseek(fd1, 0, SEEK_SET) < 0){
                fprintf(stderr, "lseek() error\n");
                exit(1);
        }

        if((fd2 = open(fname_out, O_RDWR ¦ O_CREAT, 0664)) < 0){
                fprintf(stderr, "out.txt open() error\n");
                exit(1);
        }

        if(read(fd1, buf, size) == -1){
                fprintf(stderr, "read() error\n");
                exit(1);
        }

        buf[size] = '\0';
        close(fd1);

        while((pos2 = strstr(pos1, pattern)) != NULL){
                *pos2 = '\0';
                ssu_write(fd2, pos1);
                lseek(fd2, strlen(pattern), SEEK_CUR);

                pos1 = pos2 + strlen(pattern);
        }
        ssu_write(fd2, pos1);

        close(fd1);
        close(fd2);
        return 0;
}
int ssu_write(int fd, char *buf)
{
        return write(fd, buf, strlen(buf));
}
```

실행결과 - vi out.txt 예시

aaaaa^@^@^@bbbb^@^@^@ccc^@^@^@dd^@^@^@e

실행결과

```
root@localhost:/home/oslab# ./a.out
root@localhost:/home/oslab# od -c in.txt
0000000   a   a   a   a   a   b   c   d   b   b   b   b   b   c   d   c
0000020   c   c   b   c   d   d   d   b   c   d   e  \n
0000034
root@localhost:/home/oslab# od -c out.txt
0000000   a   a   a   a   a  \0  \0  \0   b   b   b   b  \0  \0  \0   c
0000020   c   c  \0  \0  \0   d   d  \0  \0  \0   e  \n
0000034
```

24. 다음 주어진 ssu_blank.txt 파일을 읽어, 공백으로 단어를 구분하여, 각 단어를 줄 단위로 ssu_line.txt에 저장하고 단어의 개수를 출력하는 프로그램을 작성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. open() 2번, read()와 setbuf()를 각 1번 씩 사용할 것
4. dup2()는 3번만 사용 할 것
5. open(), creat(), read()에 대한 에러 처리를 위해 fprintf() 3번만 사용할 것
6. write()와 fwrite()를 사용하지 않고 dup2()를 사용하여 텍스트 사용할 것
7. 문자의 개수 출력결과가 나와야 할 것
8. 입력 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

<ssu_blank.txt>

berry grape raisin apple watermelon grapefruit pomelo

24.c

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

void main()
{
    char *fname = "ssu_line.txt";
    char *frname= "ssu_blank.txt";
    int fd_w, fd_r;          //쓰기 파일 디스크립터, 읽을 파일 디스크립터
    size_t length            //read()의 리턴값
    int wordcnt = 1;        //읽은 파일의 문자의 개수
    char buf[50];            //read()에서 읽는 버퍼
    int i = 0;                //for(),while() 문에 조건문 사용

    if((fd_w =open(fname,O_WRONLY|O_CREAT)) < 0){
        fprintf(stderr,"creat error for %s \n", fname);
        exit(1);
    }

    if((fd_r= open(frname,O_RDONLY |O_CREAT)) < 0){
        fprintf(stderr,"open error for %s \n",frname);
```

```
        exit(1);
    }

    setbuf(stdout,NULL);
    if ((read(fd_r,buf,50)) < 0){
        fprintf(stderr,"read range over");
        exit(1);
    }

    close(fd_r);
    dup2(1,fd_r);
    dup2(fd_w,1);
    close(fd_w);

    while(1)
    {
        printf("%c",buf[i]);
        if(buf[i]==' '){
            printf("\n");
            wordcnt++;
        }
        if(buf[i]=='\0')
            break;
        i++;
    }

    dup2(fd_r,1);
    printf("wordcount = %d \n",wordcnt);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
wordcount = 7
root@localhost:/home/oslab# cat ssu_line.txt
berry
grape
raisin
apple
watermelon
grapefruit
pomelo
```

25. 다음 프로그램은 SIGUSR1 시그널을 BLOCK 후 해당 시그널을 보냈을 때 pending되어 있는 시그널을 확인한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. sigemptyset(), sigaddset(), sigprocmask()를 각각 한 번씩 사용할 것
3. 자식 프로세스와 부모 프로세스 각각 sigpending()을 한 번씩 사용하고 sigismember()로 pending된 시그널을 검사할 것

25.c

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

```
#include <unistd.h>
#include <signal.h>

void ssu_signal(int signo){
	printf("SIGUSR1 catched!!\n");
}

int main(void)
{
	pid_t pid;
	sigset_t sigset;
	sigset_t pending_sigset;

	sigemptyset(&sigset);
	sigaddset(&sigset, SIGUSR1);
	sigprocmask(SIG_BLOCK,  &sigset, NULL);

	signal(SIGUSR1, ssu_signal);
	kill(getpid(), SIGUSR1);

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

		if(sigismember(&pending_sigset, SIGUSR1))
			printf("child : SIGUSR1 pending\n");
	}
	else{
		sigpending(&pending_sigset);

		if(sigismember(&pending_sigset, SIGUSR1))
			printf("parent : SIGUSR1 pending\n");
	}
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
parent : SIGUSR1 pending
```

26. 다음 프로그램은 /var/log/system.log 파일에 자신의 pid를 로그 메시지로 남기는 디몬 프로세스를 생성한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 디몬 프로그램 작성 규칙에 따라 작성할 것
3. openlog()의 옵션은 없고, facility는 LOG_LPR을 사용할 것
4. 로그는 에러 상태로 출력하고 출력을 한 다음에는 닫을 것
5. getdtablesize()를 사용하여 모든 파일 디스크립터를 닫을 것
6. 로그를 출력한 후 5초간 정지 후 프로그램을 종료할 것

26.c

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

int ssu_daemon_init(void);

int main(void)
{
	printf("daemon process initialization\n");

	if(ssu_daemon_init() < 0){
		fprintf(stderr, "ssu_daemon_init failed\n");
		exit(1);
	}
	openlog("ex8", 0, LOG_LPR);
	syslog(LOG_ERR, "My pid is %d", getpid());
	sleep(5);
	closelog();

	exit(0);
}

int ssu_daemon_init(void) {
	int fd, maxfd;
	pid_t pid;

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

	setsid();
	signal(SIGTTIN, SIG_IGN);
	signal(SIGTTOU, SIG_IGN);
	signal(SIGTSTP, SIG_IGN);

	maxfd = getdtablesize();
	for(fd = 0; fd < maxfd; fd++)
		close(fd);

	umask(0);
	chdir("/");

	fd = open("/dev/null", O_RDWR);
	dup(0);
	dup(0);

	return 0;
```

```
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
daemon process initialization
root@localhost:/home/oslab# ps -e ¦ grep a.out
3409     ?      00:00:00 a.out
root@localhost:/home/oslab# tail -1 /var/log/syslog
Jan 17 18:32:59 oslab ex8: My pid is 3409
(PID는 바뀔 수 있음)
```

27. 다음 프로그램은 alarm() 호출을 통해서 시간을 설정하고 지정된 시간이 지나면 SIGALRM에 대한 시그널 핸들러를 통해서 문자열과 값을 출력하는 것을 보여준다. 아래 조건과 실행 결과를 바탕으로 프로그램을 작성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. signal()을 sigaction()으로 변경하여 ssu_signal_handler()를 등록할 것
3. sigaction() 사용을 위한 추가 변수는 사용 가능함
4. 실행 결과는 코드 변경 후에도 동일해야 함
5. alarm()을 main() ssu_signal_handler()에서 각각 1번씩 사용하여 1초마다 ssu_signal_handler()가 실행되게 할 것

27.c

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

void ssu_signal_handler(int signo);

int count = 0;

int main(void)
{
	struct sigaction sig_act;

	sigemptyset(&sig_act.sa_mask);
	sig_act.sa_flags = 0;
	sig_act.sa_handler = ssu_signal_handler;
	sigaction(SIGALRM,&sig_act,NULL);

	alarm(1);

	while(1);

	exit(0);
}

void ssu_signal_handler(int signo) {
	printf("alarm %d\n", count++);
	alarm(1);

	if(count > 3)
		exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./a.out
alarm 0
alarm 1
alarm 2
alarm 3
```

28. 다음 프로그램은 setjmp()와 longjmp()를 호출하여 함수 경계를 넘나드는 분기를 수행할 때 변수의 타입에 따른 값을 확인하는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. 변수의 값을 출력하기 위해 printf()를 두 번 사용하고 ret_val의 출력을 위해 printf()를 한 번 사용할 것
3. setjmp()로 분기를 위해 longjmp()를 한 번 사용할 것
4. ssu_func()를 재귀적으로 호출할 것

28.c

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

void ssu_func(int loc_var, int loc_volatile, int loc_register);

int count = 0;
static jmp_buf glob_buffer;

int main(void)
{
    register int loc_register;
    volatile int loc_volatile;
    int loc_var;
    int ret_val;

    loc_var = 10;  loc_volatile = 11; loc_register = 12;

    if ((ret_val = setjmp(glob_buffer)) != 0) {
        printf("after longjmp, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);

        printf("ret_val : %d\n", ret_val);
        exit(0);
    }

    loc_var = 80; loc_volatile = 81; loc_register = 82;
    ssu_func(loc_var, loc_volatile, loc_register);
    exit(0);
}

void ssu_func(int loc_var, int loc_volatile, int loc_register) {
    if (count == 3)
        longjmp(glob_buffer, 1);
    count++;
    printf("ssu_func, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);

    ssu_func(loc_var + 1, loc_volatile + 1, loc_register + 1);
```

```
    printf("ssu_func, loc_var = %d, loc_volatile = %d, loc_register = %d\n", loc_var, loc_volatile, loc_register);
}
```

실행결과

```
root@localhost:/home/oslab# gcc 20.c -O2
root@localhost:/home/oslab# ./a.out
ssu_func, loc_var = 80, loc_volatile = 81, loc_register = 82
ssu_func, loc_var = 81, loc_volatile = 82, loc_register = 83
ssu_func, loc_var = 82, loc_volatile = 83, loc_register = 84
after longjmp, loc_var = 10, loc_volatile = 81, loc_register = 12
ret_val : 1
```

29. 두 개의 쓰레드의 실행 순서를 확인하는 프로그램을 작성하시오 아래의 조건과 실행결과를 보고 프로그램을 완성하시오.

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. mutex와 cond의 초기화는 매크로를 사용할 것
4. pthread_create() 2번 사용하여 2개의 쓰레드를 생성할 것
5. pthread_join()을 2번 사용하여 생성된 쓰레드가 종료될 때까지 기다리게 할 것
6. ssu_thread1()은 시그널이 오기 전까지 블록 상태가 되기 위해 pthread_cond_wait()를 1번 사용할 것
7. ssu_thread2()은 cond로 시그널을 보내기 위해 pthread_cond_signal()을 1번 사용할 것
8. pthread_mutex_lock() 2번, pthread_mutex_unlock() 2번 사용하여 공유변수 glo_val을 번갈아 사용할 것
9. glo_val이 VALUE_STOP1보다 작거나 VALUE_STOP2보다 클 때 cond로 시그널을 보낼 것

29.c

```
#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);
                pthread_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
```