※ 각 문제에서 주어진 프로그램 실행 시 주어진 실행결과가 나올 수 있도록, 빈 칸을 채우시오. [1-6]
<<주의 사항>>
(1) 답을 저장할 파일의 확장자는 .txt이며 각 문제 번호와 소문제 번호(문제마다 네모 박스에 기입된 번호)를 이용하여 파일을 생성해야 함
(2) 총 23개의 소문제가 있음. 각 소문제 당 1점 부여.
(3) 각 문제의 답을 모르거나 답이 틀리더라도 파일은 생성해야 함 (NULL 파일이라도 상관 없음)

1. 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 = [ 1-2.txt ] ;
    pgid = [ 1-1.txt ] ;
    printf("pid: %d, pgid: %d\n", pid, pgid);
    exit(0);
}
```

실행결과

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

2. 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;

    [   2-2.txt   ] ;
    [   2-4.txt   ] ;

    switch( [   2-3.txt   ] )
    {
        case 1:
            printf("SIGINT is included.\n");
            break;
        case 0:
            printf("SIGINT is not included.\n");
            break;
        default:
            printf("failed to cal sigismember()\n");
    }

    switch( [   2-1.txt   ] )
    {
```

```
        case 1:
            printf("SIGSYS is included.\n");
            break;
        case 0:
            printf("SIGSYS is not included.\n");
            break;
        default:
            printf("failed to cal sigismember()\n");
    }

    exit(0);
}
```

실행결과

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

3. 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;

    [ 3-2.txt ] ;
    [ 3-1.txt ] ;

    while(1)
    {
        printf("count: %d\n", count+);
        sleep(1);
        if( [ 3-3.txt ] ) = 0){
            if( [ 3-4.txt ] ){
                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.
```

4. 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 <erno.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( [ 4-5.txt ] ){
        fprintf(stderr, "signal(SIGINT) eror\n");
        exit(1);
    }

    [ 4-2.txt ] ;
    [ 4-3.txt ] ;
    [ 4-1.txt ] ;
    [ 4-4.txt ] ;

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

```
    ssu_print_mask("in critical region: ");

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

    ssu_print_mask("after return from sigsuspend: ");

    if(sigprocmask(SIG_SETMASK, &old_mask, NUL) < 0){
        fprintf(stderr, "SIG_SETMASK() eror\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 = erno;

    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:
```

5. 다음은 쓰레드를 생성한 후 생성한 쓰레드가 종료될 때까지 기다리는 프로그램이다. 아래 실행결과를 보고 빈칸을 채우시오.

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

        5-2.txt        ;

int main(void)
{
    pthread_t tid1, tid2;
    int thread1 = 1;
    int thread2 = 2;
    int status;

    // ssu_thread를 포함하여 답안 작성
    if( 5-1.txt , ssu_thread, 5-6.txt ) != 0){
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    // ssu_thread를 포함하여 답안 작성
    if( 5-5.txt , ssu_thread, 5-7.txt ) != 0){
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    5-3.txt ;
    5-4.txt ;
    exit(0);
}

5-2.txt {
    int thread_index;
    int i;

    thread_index = *(int *)arg);

    for(i = 0; i < 5; i+){
        printf("%d : %d\n", thread_index, i);
        sleep(1);
    }
```

```
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
1 : 0
2 : 0
1 : 1
2 : 1
1 : 2
2 : 2
1 : 3
2 : 3
1 : 4
2 : 4
```

6. 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 = [      6-1.txt      ] ;
    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#
```

7. 다음 프로그램은 부모 프로세스가 자식 프로세스를 생성 후 종료할 때까지 기다리고 출력한다 프로세스는 execlp()를 사용하여 를 실행시키고 프로세스는 를 사용하여 를 실행시킨다 아래 실행결과를 ‘date’ , child2 execlp() ‘who’ 보고 빈칸을 채우시오.

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

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(void)
{
        pid_t child1, child2;
        int pid, status;

        if ((child1 = fork()) == 0)
                [ 7-3.txt ] ;

        if ((child2 = fork()) == 0)
                [ 7-1.txt ] ;

        printf("parent: waiting for children\n");

        while ( [ 7-2.txt ] ) {
                if (child1 == pid)
                        printf("parent: first child: %d\n", (status >> 8));
                else if (child2 == pid)
                        printf("parent: second child: %d\n", (status >> 8));
        }

        printf("parent: all children terminated\n");
        exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
parent: waiting for children
oslab tty7 2017-01-17 10:44 (:0)
parent: second child: 0
Tue Jan 17 11:38:10 KST 2017
parent: first child: 0
parent: all children terminated
```

※ 주어진 프로그램 실행 시 아래 실행결과가 나올 수 있도록, 빈 칸을 채우시오. [1-15, 총 45점]
(주의할 점 : (1) 답을 저장할 파일의 확장자는 .txt이며 각 문제 번호와 소문제 번호(문제마다 네모 박스에 기입된 번호)를 이용하여 파일을 생성해야 함 (2) 각 문제당 동일한 소문제번호가 있을 경우 한 개의 파일만 생성하면 됨. 예를 들어, 1번 문제에 (1)번과 (2)번 소문제가 두 개 있으나 1-1.txt과 1-2.txt 파일은 하나만 생성하면 됨)

1.

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

int main(int argc, char *argv[]) {
    char line[256];
    FILE *fp;
    int uid;

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

    printf("initially uid = %d and euid = %d\n", (1)          , (2)          );
    fp = fopen(argv[1], "r");

    if (fp == NULL) {
        fprintf(stderr, "first open error for %s\n", argv[1]);
        exit(1);
    }
    else {
        printf("first open successful:\n");

        while (fgets(line, 255, fp) != NULL)
            fputs( (3)          );

        fclose(fp);
    }
    setuid( (4)          );
    printf("after setuid(%d):\n uid=%d and euid=%d\n", uid, (1)          , (2)          );
    fp = fopen(argv[1], "r");

    if (fp == NULL) {
        fprintf(stderr, "second open error for %s\n",argv[1]);
        exit(1);
    }
    else {
        printf("second open successful:\n");

        while (fgets(line, 255, fp) != NULL)
            fputs(line, stdout);

        fclose(fp);
```

```
    }
    exit(0);
}
```

실행결과

```
// ssu_setuid_test는 테스트를 위한 임시파일
root@localhost:/home/oslab# ls -l 1 ssu_setuid_test
-rwxrwxr-x 1 user1 user1 7760 Jan 10 04:52 1
-rw------- 1 user1 user1   11 Jan 10 04:54 ssu_setuid_test
root@localhost:/home/oslab# cat ssu_setuid_test
RESTRICTED
root@localhost:/home/oslab#./1 ssu_setuid_test
initially uid = 1001 and euid = 1001
first open successful:
RESTRICTED
after setuid(1001):
 uid=1001 and euid=1001
second open successful:
RESTRICTED
```

2.

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

int main(void)
{
    sigset_t set;

    sigemptyset(&set);
    (1)                    ;

    switch ( (2)                    )
    {
        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 ( (3)                    )
    {
        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# ./2
SIGINT is included.
SIGSYS is not included.
```

3.

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

(1) ssu_signal_handler( (2) );
(3) ;

int main(void)
{
    ssu_func = (4) ;

    while (1) {
        printf("process running...\n");
        sleep(1);
    }

    exit(0);
}

(1) ssu_signal_handler( (2) ) {
        printf("SIGINT 시그널 발생.\n");
        printf("SIGINT를 SIG_DFL로 재설정 함.\n");
        (5) ;
}
```

실행결과

```
root@localhost:/home/oslab# ./3
process running...
process running...
process running...
^CSIGINT 시그널 발생.
SIGINT를 SIG_DFL로 재설정 함.
process running...
process running...
^C
root@localhost:/home/oslab#
```

4.

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

void ssu_signal_handler(int signo);

(1)  jump_buffer;

int main(void)
{
    signal(SIGINT, ssu_signal_handler);

// your code starts here!
    while (1) {
        if ( (2)  ) {
         printf("Hit Ctrl-c at anytime ... \n");
         pause();
        }
    }

    exit(0);
}
void ssu_signal_handler(int signo) {
        char character;

        signal(signo, SIG_IGN);
        printf("Did you hit Ctrl-c?\n" "Do you really want to quit? [y/n]");
        character = getchar();

        if (character =='y' || character == 'Y')
            exit(0);
        else {
            signal(SIGINT, ssu_signal_handler);
            longjmp (jump_buffer, 1);
        }
}
```

실행결과

```
root@localhost:/home/oslab# ./4
Hit Ctrl-C at anytime ...
^CDid you hit Ctrl-C?
Do you really want to quit? [y/n] y
root@localhost:/home/oslab# ./4
Hit Ctrl-C at anytime ...
Did you hit Ctrl-C?
Do you really want to quit? [y/n] n
^C^C^C^C^C^C
```

5.

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

int main(void)
{
    (1)  old_set;
    (1)  sig_set;

    sigemptyset(&sig_set);
    sigaddset(&sig_set, SIGINT);
    sigprocmask( (2)  );
    (3)  ;
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./5
// 5번 프로그램은 기존 시그널 집합에 있는
// 시그널이 발생할 때까지 대기하는 프로그램이다.
^C
root@localhost:/home/oslab#
```

6.

```
#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;
    // 인자 전달은 없다고 가정
    if ( (1)  )    {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    pid = getpid();
    tid = (2)  ;
    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 = (2)                    ;
    printf("New Thread: pid %u tid %u \n",
        (unsigned int)pid, (unsigned int)tid);
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./6
Main Thread: pid 3222 tid 3075864320
New Thread: pid 3222 tid 3075861312
```

7.

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

void *ssu_thread(void *arg);

int main(void)
{
    pthread_t tid1, tid2;
    int thread1 = 1;
    int thread2 = 2;
    int status;
    if ( (1)                                                    ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    if ( (2)                                                    )  {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    pthread_join( (3)                    );
    pthread_join( (4)                    );
    exit(0);
}

void *ssu_thread(void *arg) {
    int thread_index;
    int i;
    (5)                          ;
    for (i = 0; i < 5; i++) {
        printf("%d : %d\n", thread_index, i);
```

```
        sleep(1);
    }
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./7
1 : 0
2 : 0
1 : 1
2 : 1
1 : 2
(생략)
```

8.

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

void *ssu_loop1(void *arg);
void *ssu_loop2(void *arg);

pthread_mutex_t mutex =  (1)          ;
int shared_value;

int main(void)
{
    pthread_t tid1, tid2;
    int status;

    shared_value = 0;
    if (pthread_create(&tid1, NULL, ssu_loop1, NULL) != 0 ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    sleep(1);
    if (pthread_create(&tid2, NULL, ssu_loop2, NULL) != 0 ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }
    if ( (2)                                                ) {
        fprintf(stderr, "pthread_join error\n");
        exit(1);
    }
    if ( (3)                                                ) {
        fprintf(stderr, "pthread_join error\n");
        exit(1);
    }
    status =  (4)                    ;
    printf("code  =  %d\n", status);
```

```
    printf("programming is end\n");
    exit(0);
}

void *ssu_loop1(void *arg) {
    int i;
    for (i = 0; i < 10; i++) {
        (5)                              ;
        printf("loop1 : %d\n", shared_value);
        shared_value++;
         if (i == 10)
            return NULL;

        (6)                              ;
        sleep(1);
    }
    return NULL;
}

void *ssu_loop2(void *arg) {
    int i;

    for (i = 0; i < 10; i++) {
        (5)                              ;
        printf("loop2 : %d\n", shared_value);
        shared_value++;
        (6)                              ;
        sleep(2);
    }
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./8
loop1 : 0
loop1 : 1
loop2 : 2
loop1 : 3
loop1 : 4
loop2 : 5
loop1 : 6
loop2 : 7
loop1 : 8
loop1 : 9
loop2 : 10
loop1 : 11
loop1 : 12
loop2 : 13
loop1 : 14
loop2 : 15
loop2 : 16
```

```
loop2 : 17
loop2 : 18
loop2 : 19
code  =  0
programming is end
```

9.

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

int ssu_daemon_init(void);

int main(void)
{
        pid_t pid;

        pid = getpid();
        printf("parent process : %d\n", pid);
        printf("daemon process initialization\n");

        if (ssu_daemon_init() < 0) {
        fprintf(stderr, "ssu_daemon_init failed\n");
        exit(1);
    }

    exit(0);
}

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

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

    pid = getpid();
    printf("process %d running as daemon\n", pid);
    (2)                    ;
    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 = (3)                    ;
    dup(0); // 10
    dup(0);
    return 0;
}
```

실행결과

```
root@localhost:/home/oslab# ./9
parent process : 12153
daemon process initialization
process 12154 running as daemon
```

10.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

#define BUFFER_SIZE 1024

struct ssu_msgbuf  {
    char msg_text[BUFFER_SIZE];
    long msg_type;
};

int main(void)
{
    struct ssu_msgbuf buf;
    key_t key;
    int msg_queueid;

    if ((key = ftok("ssu_dummy.c", 'B')) == -1) {
        fprintf(stderr, "ftok error\n");
        exit(1);
    }

    if ((   (1)                                    ) == -1) {
        fprintf(stderr, "msgget error\n");
        exit(1);
    }
    printf("Enter lines of text, ^D to quit: \n");
    buf.msg_type = 1;
```

```
    while (fgets(buf.msg_text, sizeof(buf.msg_text), stdin) != NULL) {
        int length = strlen(buf.msg_text);

        if (buf.msg_text[length-1] == '\n')
            buf.msg_text[length-1] = '\0';
        if (msgsnd(msg_queueid, &buf, length+1, 0) == -1)
            fprintf(stderr, "msgsnd error");
    }

    if ( (2)                                    == -1) {
        fprintf(stderr, "msgctl error");
        exit(1);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# touch ssu_dummy.c
root@localhost:/home/oslab# ./10
Enter lines of text, ^D to quit:
Hi OSLAB!
Nice to meet you!
```

11.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

#define SHARED_MEMORY_SIZE 1024

int main(int argc, char *argv[])
{
    key_t key;
    char *data;
    int shared_memory_id;

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

    if ((key = ftok("ssu_shmdemo.c", 'R')) == -1) {
        fprintf(stderr, "ftok error\n");
        exit(1);
    }

    if ( (1)                                                    == -1 ) {
```

```
        fprintf(stderr, "shmget error\n");
        exit(1);
    }

    if ((data = shmat(shared_memory_id, (void *)0, 0)) == (char *)(-1)) {
        fprintf(stderr, "shmat error\n");
        exit(1);
    }

    if (argc == 2) {
        printf("writing to segment: \"%s\" \n", argv[1]);
        strncpy(data, argv[1], SHARED_MEMORY_SIZE);
    }
    else
        printf("segment contains: \"%s\" \n", data);

    if ( (2)                    == -1) {
        fprintf(stderr, "shmdt error\n");
        exit(1);
    }
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# touch ssu_shmdemo.c
root@localhost:/home/oslab# ./11 Hello\ World
writing to segment: "Hello World"
root@localhost:/home/oslab# ./11
segment contains: "Hello World"
```

12.

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

#define NAMESIZE 50
#define MAXTRIES 5

struct employee{
        char name[NAMESIZE];
        int salary;
        int pid;
};

int main(int argc, char *argv[])
{
        struct flock lock;
        struct employee record;
        int fd, sum=0, try=0;
```

```
        sleep(10);

        if((fd = open(argv[1], O_RDONLY)) == -1) {
                perror(argv[1]);
                exit(1);
        }

        lock.l_type = F_RDLCK;
        (1)                     ;
        (2)                     ;
        lock.l_len = 0L;

        while(fcntl(fd, F_SETLK, &lock) == -1) {
                if(errno == EACCES) {
                        if(try++ < MAXTRIES) {
                                sleep(1);
                                continue;
                        }
                        printf("%s busy -- try later\n", argv[1]);
                        exit(2);
                }
                perror(argv[1]);
                exit(3);
        }
        sum = 0;
        while(read(fd, (char *)&record, sizeof(record)) > 0) {
                printf("Employee: %s, Salary: %d\n", record.name, record.salary);
                sum += record.salary;
        }
        printf("\nTotal salary: %d\n", sum);

        lock.l_type = F_UNLCK;
        fcntl(fd, F_SETLK, &lock);
        close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ./12 employeefile
Employee: HongKilDong, Salary: 1500000
Employee: LeeSoonShin, Salary: 1900000
Employee: SongSeonHoon, Salary: 450000
Employee: BaekMaKang, Salary: 230000
Employee: KimJongHoon, Salary: 1500000

Total salary: 5580000
```

13.

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

```
#include <errno.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
        struct flock lock;
        int fd;
        char command[100];

        if((fd = open(argv[1], O_RDWR)) == -1) {
                perror(argv[1]);
                exit(1);
        }
        lock.l_type = F_WRLCK;
        lock.l_whence = 0;
        lock.l_start = 0l;
        lock.l_len = 0l;

        if( (1)                          == -1) {
                if (errno == EACCES) {
                        printf("%s busy -- try later\n", argv[1]);
                        exit(2);
                }
                perror(argv[1]);
                exit(3);
        }
        sprintf(command, "vim %s\n", argv[1]);
        (2)                        ;
        lock.l_type = F_UNLCK;
        fcntl(fd, F_SETLK, &lock);
        close(fd);

        return 0;
}
```

실행결과

```
root@localhost:/home/oslab# ./13 ssu_test.txt
[vim 실행]
```

14.

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

#define NAMESIZE 50

struct employee{
        char name[NAMESIZE];
        int salary;
        int pid;
```

```
};

int main(int argc, char *argv[])
{
        struct flock lock;
        struct employee record;
        int fd, recnum, pid;
        long position;

        if((fd = open(argv[1], O_RDWR)) == -1) {
                perror(argv[1]);
                exit(1);
        }

        pid = getpid();
        for(;;){
                printf("\nEnter record number: ");
                scanf("%d", &recnum);
                if(recnum < 0)
                        break;
                position = recnum * sizeof(record);
                lock.l_type = F_WRLCK;
                lock.l_whence = 0;
                lock.l_start = position;
                lock.l_len = sizeof(record);
                if(fcntl(fd, F_SETLKW, &lock) == -1) {
                        perror(argv[1]);
                        exit(2);
                }
                lseek(fd, position, 0);
                if(read(fd, (char*)&record, sizeof(record)) == 0){
                        printf("record %d not found\n", recnum);
                        (1)                    ;
                        (2)                    ;
                        continue;
                }
                printf("Employee: %s, salary: %d\n", record.name, record.salary);
                record.pid = pid;
                printf("Enter new salary: ");
                scanf("%d", &record.salary);
                lseek(fd, position, 0);
                write(fd, (char*)&record, sizeof(record));

                lock.l_type = F_UNLCK;
                fcntl(fd, F_SETLK, &lock);
        }
        close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ./14 employeefile
```

```
Enter record number: 2
Employee: SongSeonHoon, salary: 450000
Enter new salary: 450000

Enter record number: 3
Employee: BaekMaKang, salary: 230000
Enter new salary: 500000

Enter record number: -1
root@localhost:/home/oslab#
```

15.

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

#define NAMESIZE 50

struct employee{
        char name[NAMESIZE];
        int salary;
        int pid;
};

int main(int argc, char *argv[])
{
        struct flock lock;
        struct employee record;
        int fd, recnum, pid;
        long position;
        char ans[5];

        if((fd = open(argv[1], O_RDWR)) == -1) {
                perror(argv[1]);
                exit(1);
        }

        pid = getpid();
        for(;;){
                printf("\nEnter record number: ");
                scanf("%d", &recnum);
                if(recnum < 0)
                        break;

                position = recnum * sizeof(record);
                lock.l_type = F_RDLCK;
                (1)                    ;
                (2)                    ;
```

```
                lock.l_len = sizeof(record);

                if(fcntl(fd, F_SETLKW, &lock) == -1) {
                        perror(argv[1]);
                        exit(2);
                }
                lseek(fd, position, 0);
                if(read(fd, (char*)&record, sizeof(record)) == 0){
                        printf("record %d not found\n", recnum);
                        lock.l_type = F_UNLCK;
                        fcntl(fd, F_SETLK, &lock);
                        continue;
                }
                printf("Employee: %s, salary: %d\n", record.name, record.salary);
                printf("Do you want to update salary (y or n)? ");
                scanf("%s", ans);

                if(ans[0] != 'y') {
                        lock.l_type = F_UNLCK;
                        fcntl(fd, F_SETLK, &lock);
                        continue;
                }
                lock.l_type = F_WRLCK;
                if(fcntl(fd, F_SETLKW, &lock) == -1) {
                        perror(argv[1]);
                        exit(3);
                }
                record.pid = pid;
                printf("Enter new salary: ");
                scanf("%d", &record.salary);

                lseek(fd, position, 0);
                write(fd, (char*)&record, sizeof(record));

                lock.l_type = F_UNLCK;
                fcntl(fd, F_SETLK, &lock);
        }
        close(fd);
}
```

실행결과

```
root@localhost:/home/oslab# ./15 employeefile

Enter record number: 2
Employee: SongSeonHoon, salary: 450000
Do you want to update salary (y or n)? y
Enter new salary: 230000

Enter record number: 3
Employee: BaekMaKang, salary: 500000
Do you want to update salary (y or n)? y
```

```
Enter new salary: 250000

Enter record number: 3
Employee: BaekMaKang, salary: 250000
Do you want to update salary (y or n)? 400000

Enter record number: -1
root@localhost:/home/oslab#
```

※ 주어진 조건에 맞게 프로그램을 완성하시오. [7-17, 총7점]
<주의 사항>
(1) 각 문제의 답을 저장할 파일의 확장자는 문제번호 .c 임
(2) 각 문제의 답을 모르거나 답이 틀리더라도 NULL 파일을 생성해야 함
(3) 주어진 기능을 모두 구현하지 못하고 특정 부분만 수행되는 프로그램을 작성했더라도 컴파일 시에 에러가 발생할 경우 문제 파일 이름만 생성하고 NULL 파일(문제번호.c)로 만들어야 함. 즉, 일부 기능만 구현했을 경우라도 반드시 컴파일이 에러 없이 진행되어야 하며 에러가 발생할 경우 해당 문제는 0점 처리
(4) 컴파일 시 warning이 한 개마다 해당 문제의 점수에서 10% 감점
(5) 프로그램 구현 문제에서 주어진 조건을 변경하거나, 변수를 추가/변경할 경우 해당 문제는 1개 추가/변경 시 해당 문제의 점수에서 30% 감점
(6) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리. 단, 표준출력이 단순하게 다를 경우 허용

7. 두 개의 쓰레드의 실행 순서를 확인하는 프로그램을 작성하시오 아래의 조건과 실행결과를 보고 프로그램을 완성하시오. (5점)

< 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

7.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_cond_t cond =

int glo_val = 0;

int main(void)
{
    pthread_t tid1, tid2;

    printf("final value: %d\n", glo_val);
    exit(0);
}
ssu_thread1( )
{
    while(1){
        if(glo_val >= VALUE_DONE)
            return NULL;
```

```
    }
}
ssu_thread2()
{
    while(1){


        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
```

8. 디몬(daemon) 프로세스가 5초마다 로그 메시지를 남기게 하고 그 동작을 확인하는 프로그램을 작성하시오. <참고 4> (3점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 로그 메시지를 남기기 위해 openlog(), syslog(), closelog(), slep()을 각각 1번씩 사용할 것
4. 디몬 프로세스 생성을 위해 fork()를 1번 사용할 것
5. getpid(), setsid()를 각각 1번씩 사용하고 표준출력으로 디몬프로세스의 pid를 출력할 것
6. signal()을 3번 사용하여 작업제어와 연관된 시그널을 무시하도록 할 것
7. close()를 1번 사용하여 열려있는 모든 파일 디스크립터를 닫을 것
8. umask()를 1번 사용하여 디몬이 생성할 파일의 접근 허가 모드를 모두 허용하도록 할 것
9. chdir()을 1번 사용하여 현재 디렉토리를 루트 디렉토리로 설정할 것
10. open()을 1번 사용하여 “/dev/nul” 파일을 읽기, 쓰기 모드로 열 것
11. dup()을 2번 사용할 것
12. ssu_daemon_init(), fork()의 에러 처리를 위해 fprintf()를 2번 사용할 것

8.c

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

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


}

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


}
```

실행결과[Ubuntu]

```
root@localhost:/home/oslab# ./a.out
daemon proces initialization
proces 1279 runing as daemon
root@localhost:/home/oslab# ps -ejfc ¦ grep a.out
root 1279 1 1279 1279 TS 19 21:46 ? 0:0:0 ./a.out
root 1281 12038 1280 1207 TS 19 21:46 pts/19 0:0:0 grep --color=auto a.out
root@localhost:/home/oslab# tail -1 /var/log/syslog
Jan 13 21:46:36 oslab-ZBOX-ID91 lpd[1279]: open failed lpd Bad file descriptor
```

9. fcntl()을 사용하여 Nonblock 플래그를 설정하는 프로그램을 작성하시오. (7점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. read()를 1번 사용하여 표준입력에서 읽을 것
4. fprintf()를 1번 사용하여 stder에 read()로 읽은 byte 수를 출력할 것
5. set_flags(), clr_flags()을 각각 1번씩 사용하여 표준출력에 nonblock 플래그를 설정 및 해제할 것
6. write()를 1번 사용하여 표준출력에 read()로 읽은 내용을 쓰고 쓰여진 byte만큼 ptr 포인터를 이동하고 전체 읽은 byte수를 감소시킬 것
7. fprintf()를 1번 사용하여 stder에 기록된 byte 수 및 erno를 출력할 것
8. nonblock 플래그 설정 및 해제를 위해 fcntl()을 set_flags(), clr_flags()에서 각각 2번씩 사용할 것
9. 비트연산자를 사용하여 nonblock 플래그를 설정 및 해제할 것
10. fcntl()의 에러처리를 위해 fprintf()를 4번 사용할 것

ssu_test1.txt(다음 프로그램으로 생성)

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
int main(void)
{
int fd;
int i;
fd = open("./su_test1.txt", O_RDWR ¦ O_CREAT ¦ O_TRUNC, 064);

for(i = 0; i < 5000; i+)
```

```
write(fd, "i", 1);

close(fd);

exit(0);
}
```

9.c

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

char buf[500000];

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

    ptr = buf;
    while(ntowrite > 0){
        errno = 0;

        if(nwrite > 0){
        }
    }


    exit(0);
}
set_flags(int fd, int flags)
{
    int val;


}
clr_flags(int fd, int flags)
{
    int val;


}
```

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
..
..
```

10. 다음 첫 번째 인자로 받은 파일에 employe 구조체를 저장하는 프로그램이다. 파일디스크립트 fcntl()를 활용하여 플래그를 수정해서 출력의 결과가 EOF에 이어지게 하는 프로그램을 작성하시오. (7점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것.
4. 인자로 받은 파일을 읽기 쓰기 모드로 열기 위해 open()을 1번 사용할 것
5. fcntl()을 2번, getpid() 1번만 사용할 것
6. 비트연산자를 활용하여 플래그를 추가할 것
7. write()를 1번 사용할 것
8. fcntl(), write()에 대한 에러 처리를 위해 fprintf()를 3번 사용할 것
9. 인자로 받은 파일의 끝에 이어지게 작성할 것

10.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<eror.h>
#include<fcntl.h>
#define NAMESIZE 50
#define DUMMY 0

struct employe{
      char name[NAMESIZE];
      int salary;
      int pid;
};

int main(int argc, char *argv[])
{
      struct flock lock;
      struct employe record;
      int fd, flags, length,pid;

      if( )
      {
            fprintf(stder, "Usage :%s file \n",argv[0]);
            exit(1);
      }
      if( )
```

```
    {
        fprintf(stder, "Open eror :%s file \n",argv[1]);
        exit(1);
    }

    while(1)
    {

    }
    close(fd);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# touch ssu_employeefile
root@localhost:/home/oslab# ./a.out ssu_employeefile
Enter employee name : BaekMaKang
Enter employee salary :3000000
Enter employee name : HanRaSan
Enter employee salary :13000000
Enter employee name : BakDooSan
Enter employee salary :1900000
Enter employee name : .
```

11. 다음 첫 번째 인자로 받은 파일에 입력받은 파일에서 수정을 원하는 레코드에 write락을 설정하고 수정하는 프로그램을 작성하시오. (7점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 10번 문제의 su_employefile을 인자로 사용할 것
4. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것
5. fcntl()을 3번 사용할 것
6. lsek()을 2번 사용할 것
7. read(), write()를 각각 1번 사용할 것
8. lock을 설정하기 위한 fcntl()의 예외 처리를 위해 peror()을 1번 사용할 것
9. 0이하의 record number 입력 받을 시 프로그램을 종료할 것
10. 데이터가 존재하지 않는 record number 입력 받을 시 lock을 해제하고 다음 입력을 받을 것
11. 기록을 시작할 때 기록할 부분을 lock하고 기록이 완료되면 unlock할 것
12. 수정된 레코드는 pid를 수정하여 저장 할 것
13. lock이 설정된 상태라면 락이 해제 될 때 까지 기다리게 할 것

11.c

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

#define NAMESIZE 50

struct employe{
    char name[NAMESIZE];
    int salary;
    int pid;
```

```
};

int main(int argc, char *argv[])
{
    struct flock lock;
    struct employe record;
    int fd,recnum,pid;
    long position;

    if(fd = open(argv[1],O_RDWR) =-1)
    {
        perror(argv[1]);
        exit(1);
    }
    pid =getpid();
    while(1)
    {

    }
    close(fd);
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out ssu_employeefile
Enter record number :2
employee:HanRaSan, salary: 13000000
Enter new salary : 450000
Enter record number :1
employee:BakDooSan, salary: 1900000
Enter new salary : 500000
```

12. 다음 첫 번째 인자로 받은 파일의 내용을 두 번째 인자로 받은 파일의 이름으로 mmap()와 memcpy()를 이용하여 복사하는 프로그램을 작성하시오. (3점)

< 조 건 >

1. 주어진 변수를 그대로 사용할 것, 변수의 추가나 삭제 시 감점
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. 본 프로그램의 실행파일은 2개의 인자만 받도록 할 것.
4. 인자로 받은 파일을 열기 위해 첫 번째 인자는 읽기모드로, 두 번째 인자는 읽기쓰기모드로 open()을 2번 사용할 것
5. fstat()을 1번, lsek()을 1번, write()를 1번 사용할 것
6. mmap()을 2번, memcpy()를 1번 사용할 것
7. open(), fstat(), lsek(), write(), mmap()에 대한 에러처리를 위해 printf()를 7번 사용할 것
8. 텍스트 파일이 변경되어도 정상적으로 작동되어야 할 것

ssu_test.txt

```
Linux System Programming!
```

12.c

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

```
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>

#define FILE_MODE (S_IRUSR ¦ S_IWUSR ¦ S_IRGRP ¦ S_IROTH)

int main(int argc, char *argv[])
{
    int fdin, fdout;
    void *src, *dst;
    struct stat statbuf;

    if ( ) {
        printf("usage: %s <fromfile> <tofile>", argv[0]);
        exit(1);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# vim ssu_test.txt
root@localhost:/home/oslab# ./a.out ssu_test.txt ssu_test_1.txt
root@localhost:/home/oslab# ls -al ssu_test.txt ssu_test_1.txt
-rw-r--r-- 1 root root 26 6월 9 10:48 ssu_test.txt
-rw-r--r-- 1 root root 26 6월 9 10:48 ssu_test_1.txt
```

13. 다음 pthread_cond_signal()을 호출하여 피보나치 수열을 출력하는 프로그램을 작성하시오. (5점)

< 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. main() 함수 내에서는 pthread_create() 2번, pthread_cond_signal()을 1번, pthread_join()을 2번 사용할 것
5. ssu_thread1() 함수 내에서 pthread_mutex_lock(), pthread_mutex_unlock()을 각각 1번, pthread_cond_wait(), pthread_cond_signal()을 각각 2번 사용할 것
6. ssu_thread2() 함수 내에서 pthread_mutex_lock(), pthread_mutex_unlock()을 각각 1번, pthread_cond_wait(), pthread_cond_signal()을 각각 2번 사용할 것
7. ssu_thread1()와 ssu_thread2()를 실행하는 Thread1, Thread2가 번갈아 가면서 출력되어야 함.

13.c

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

pthread_mutex_t mutex1 =
pthread_mutex_t mutex2 =
pthread_cond_t cond1 =
pthread_cond_t cond2 =

int count = 0;
int input = 0;
int t1 = 0, t2 = 0;
```

```
int main(void)
{
    pthread_t tid1, tid2;
    int status;

    if ( ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    if ( ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    while (1) {

    }

    printf("complete \n");
    exit(0);
}

ssu_thread1( ) {
    while (1) {
        if (input < 2) {

        }

        if (input == count) {

        }

        if (count == 0) {

        }
        else if (count % 2 == 0) {

        }


    }

    return NULL;
}

ssu_thread2( ) {
    while (1) {

        if (input < 2){

        }
```

```
        if (input == count) {

        }

        if (count == 1) {

        }
        else if (count % 2 == 1) {

        }

    }
    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
2개 이상의 개수 입력 : 20
Thread 1 : 0
Thread 2 : 1
Thread 1 : 1
Thread 2 : 2
Thread 1 : 3
Thread 2 : 5
Thread 1 : 8
Thread 2 : 13
Thread 1 : 21
Thread 2 : 34
Thread 1 : 55
Thread 2 : 89
Thread 1 : 144
Thread 2 : 233
Thread 1 : 377
Thread 2 : 610
Thread 1 : 987
Thread 2 : 1597
Thread 1 : 2584
Thread 2 : 4181
complete
```

※ 주어진 조건에 맞게 프로그램을 완성하시오. [16-20, 각 5점, 총 25점]
(주의할 점 : gcc로 옵션 없이 컴파일 시 (1) 에러 발생 시 0점 처리 (2) 주어진 제약조건을 변경할 경우 0점 처리 (3) 출력 결과가 하나라도 다를 경우 0점 처리 (쓰레드 스케줄링에 따라 다른 출력 결과가 나오는 경우는 제외) (4) warning 한 개당 2.5점 감점)

16. 다음은 SIGCHLD 시그널에 함수를 등록하고 해당 시그널을 기다리는(suspend) 프로그램이다.

< 조 건 >

1. SIGCHLD시그널을 받으면 ssu_signal_handler()가 호출 (sigaction() 이용)
2. 자식 프로세스를 생성하여 자식은 3초 뒤 종료
3. 부모 프로세스는 SIGCHLD이외의 시그널을 무시하며 SIGCHLD 시그널을 기다림 (sigsuspend() 이용)

16.c

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

void ssu_signal_handler(int signo);

int main(void)
{
        struct sigaction sig_act;
        sigset_t blk_set;
        pid_t pid;

        // your code starts here!
}

void ssu_signal_handler(int signo) {
        printf("in ssu_signal_handler() function\n");
}
```

실행결과

```
root@localhost:/home/oslab# ./16
before fork
after fork in parent, suspend...
after fork in child, sleep...
in ssu_signal_handler() function
after suspend
```

17. 다음은 생성한 두 사용자 쓰레드 중 특정 쓰레드를 메인 쓰레드에서 식별하는 프로그램이다.

< 조 건 >

1. 쓰레드를 두 개 생성하고, 각각의 tid를 loc_tid 배열에 저장
2. 두 번째 쓰레드는 자신의 tid를 전역변수에 저장하는 루틴을 수행
3. main() 함수는 쓰레드 생성 직후 5초간 sleep
4. main() 함수에서 생성시 저장했던 tid 2개와 전역변수의 tid를 함수를 통해 비교하여, 두 번째로 생성된 쓰레드 tid가 loc_tid배열의 두 번째 인덱스에 저장되었다는 것을 확인하고 출력

17.c

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

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

pthread_t glo_tid;

int main(void)
{
        pthread_t loc_tid[2];
        int i;

        // your code starts here!

        sleep(5);

        for (i = 0; i < 2; i++) {
                // your code starts here!
        }
        exit(0);
}

void *ssu_thread1(void *arg) {
        printf("in ssu_thread1\n");
        return NULL;
}

void *ssu_thread2(void *arg) {
        printf("in ssu_thread2\n");

        // your code starts here!

        return NULL;
}
```

실행결과 (실행시마다 다름)

```
root@localhost:/home/oslab# ./17
in ssu_thread1
in ssu_thread2
second thread assigns it's tid to global tid
```

18. 다음은 쓰레드 두 개를 생성하고, 공유변수를 출력하는 프로그램이다.

< 조 건 >

1. 쓰레드 두개생성
2. 두 쓰레드 생성 시 인자로 1, 2를 전달
3. 전달받은 번호로 몇 번째 쓰레드에서 출력하는지 구분
4. 전역변수 glo_val을 선언하고 값을 0으로 초기화
5. 각 쓰레드는 mutex계열 함수를 사용하여 락을 걸고, glo_val값을 증가시킴
6. glo_val값이 10이 되면 종료
7. 쓰레드 함수는 총 1개만 생성

18.c

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

void *ssu_thread(void *arg);

pthread_mutex_t lock;
int glo_val;

int main(void) {
        pthread_t tid1, tid2;

        // your code starts here!
}

void *ssu_thread(void *arg) {
        int num;
        while (/* your code */) {
                // your code starts here!
        }

}
```

실행결과

```
root@localhost:/home/oslab# ./18
global value ssu_thread1: 1
global value ssu_thread1: 2
global value ssu_thread1: 3
global value ssu_thread1: 4
global value ssu_thread1: 5
global value ssu_thread1: 6
global value ssu_thread1: 7
global value ssu_thread1: 8
global value ssu_thread1: 9
global value ssu_thread2: 10
final value : 10
```

19. 다음은 자식프로세스와 pipe를 통해 통신하는 프로그램이다.

< 조 건 >

1. 자식 프로세스 생성
2. 자식 프로세스와 부모 프로세스의 통신을 위한 파이프 생성
3. 자식 프로세스는 부모 프로세스에게 파이프를 통해 “OSLAB“문자열 전달
4. 부모 프로세스는 자식 프로세스로부터 전달받은 문자열을 읽고 출력

19.c

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

#define BUFFER_SIZE 1024

int main(void) {
        char buf[BUFFER_SIZE];
        int pid;
        int pipe_fd[2];

         // your code starts here!
}
```

실행결과

```
root@localhost:/home/oslab# ./19
 CHILD: writing to the pipe
PARENT: reading from pipe
 CHILD: exiting
PARENT: read “OSLAB“
```

20. 다음은 proc 파일시스템을 참조하여 현재 실행중인 프로세스 자신이 open한 파일의 파일디스크립터 번호와 symbolic link 파일들의 경로를 출력하는 프로그램이다.

< 조 건 >

1. 주어진 변수와 함수만을 이용해서 프로그램 구현
2. 주어진 헤더파일은 추가 및 제거가 불가능
3. 에러 처리 문자열은 함수에 따라서 적절하게 출력

변수 설명
char *arg_container[ASCII_MAX] [ARG_MAX] // [옵션의 ASCII 값] [최대 인자 개수]
각 옵션에 해당하는 PID 인자들을 문자열의 형태로 저장하는 배열
2. int arg_ptr[ASCII_MAX]
각 옵션에 해당하는 PID 인자의 개수를 저장하는 배열

int ssu_proc_pid_fd(void);
- 입력 인자 : 없음
리턴값 : 정상 종료시 0, ssu_proc_pid_fd_derived() 함수 호출 오류시 -1 리턴
수행 과정
① 실행하는 프로세스 자신의 PID를 주어진 변수 설명을 참조하여 적절히 추가
② arg_container배열을 적절히 참조하여 NULL이면 중도에 반복문을 종료
③ NULL이 아니라면 fd_path배열에 proc 파일시스템에 있는 해당 PID의 fd 디렉터리의 경로를 만듬
④ fd_path가 실제로 존재하는지를 확인하고, 읽기 권한이 있는지를 에러 처리 포함하여 확인
⑤ 두 조건이 만족되면 ssu_proc_pid_fd_derived()를 호출. 호출한 함수가 정상 종료되지 않았다면 -1을 리턴
⑥ ②~⑤의 과정을 최대로 받을 수 있는 인자 개수만큼 반복
⑦ 위 과정이 모두 정상적으로 수행되었다면 ssu_proc_pid_fd()도 정상종료(return 0)

int ssu_proc_pid_fd_derived (char *fd_path);
입력 인자 : fd_path = proc 파일시스템의 프로세스 디렉터리 내부의 fd디렉터리를 가리키는 문자열
리턴값 : 정상 종료시 0, 디렉터리 open 실패시 -1 리턴
수행 과정
① 디렉터리 함수를 통하여 인자로 받은 fd_path 디렉터리를 open함. 적절한 에러처리 필요
② 적절한 디렉터리 함수를 통하여 open한 디렉터리 내부를 탐색
③ 현재 디렉터리(.), 부모 디렉터리(..)를 제외한 모든 디렉터리에 대해 적절한 함수를 호출하여 symbolic link가 가리키고 있는 경로를 얻어서 출력결과와 동일한 형태로 출력(띄어쓰기 주의!)
④ 모든 디렉터리에 대해 수행한 다음, 해당 디렉터리를 닫고 정상종료(return 0)

20.c

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

#define SHORT_BUF 1024
#define MAX_BUF 4096
#define ASCII_MAX 256
#define ARG_MAX 16

char *arg_container[ASCII_MAX][ARG_MAX];
int arg_ptr[ASCII_MAX];
```

```
int ssu_proc_pid_fd(void);
int ssu_proc_pid_fd_derived(char *);

int main(void)
{
        ssu_proc_pid_fd();
        exit(0);
}

int ssu_proc_pid_fd(void) {
        char fd_path[MAX_BUF] = {0};          // /proc/[PID]/fd의 경로를 저장하는 배열
        char pid_buf[SHORT_BUF] = {0};        // PID를 문자열으로 저장하는 배열
        char opt = 'f';                       // 옵션을 나타내는 문자
        int j = 0;

        // your code starts here!
}

int ssu_proc_pid_fd_derived (char *fd_path) {
        char fd_res[SHORT_BUF] = {0};         // symbolic link를 readlink()로 읽은 결과를 저장하는 배열
        char fd_tmp[SHORT_BUF] = {0};         // 문자열 조작에 임시적으로 사용할 수 있는 배열
        DIR *dir_ptr = NULL;
        struct dirent *dir_entry = NULL;

        // your code starts here!
}
```

실행결과 // PID, 터미널 번호는 시스템 환경마다 다를 수 있음

```
root@localhost:/home/oslab# ./20                    // 1. 일반 실행
File Descriptor number: 0, Opened File: /dev/pts/9
File Descriptor number: 1, Opened File: /dev/pts/9
File Descriptor number: 2, Opened File: /dev/pts/9
File Descriptor number: 3, Opened File: /proc/29472/fd
root@localhost:/home/oslab# ./20 &                  // 2. background으로 실행
[1] 29473
root@localhost:/home/oslab# File Descriptor number: 0, Opened File: /dev/pts/9
File Descriptor number: 1, Opened File: /dev/pts/9
File Descriptor number: 2, Opened File: /dev/pts/9
File Descriptor number: 3, Opened File: /proc/29473/fd

[1]+  완료                    ./20
root@localhost:/home/oslab#
```

※ 주어진 제약조건에 맞게 프로그램을 완성하시오. [21-23, 각 10점, 총 30점]
(주의할 점 : gcc로 옵션 없이 컴파일시 (1) 에러 발생시 0점 처리 (2) 주어진 제약조건을 변경할 경우 0점 처리 (3) 출력 결과가 하나라도 다를 경우 0점 처리 (쓰레드 스케줄링에 따라 다른 출력 결과가 나오는 경우는 제외) (4) warning 한 개당 2.5점 감점)

21. 다음은 다중 쓰레드를 사용하여 test.txt파일을 읽는 프로그램이다.

< 조 건 >

1. 쓰레드 개수(n)를 입력받음 (쓰레드 개수는 정수이며, 읽는 파일크기의 약수임)
2. 입력받은 개수만큼 쓰레드를 생성
3. 각 쓰레드는 test.txt파일을 1/n만큼 읽음
4. 각 쓰레드에서 읽은 내용은 전역포인터 변수인 buf변수에 겹치지 않게 저장
5. 소스 코드상에 배열 형태의 변수는 사용불가, 포인터를 사용한 malloc을 사용
6. buf포인터 동적 할당 시, buf의 크기는 파일크기를 기반으로 할당해야함
7. 사용한 동적 할당 변수들은 반드시 해제

test.txt 예시

```
123456789
```

21.c

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

char * buf;
int readsize;

void *read_thread(void *arg) {
        int num;
        int fd;

        // your code starts here!
}

int main(void) {
        struct stat stat_buf;
        int         status;
        int thread_num;
        pthread_t *tid;

        // your code starts here!
}
```

실행결과

```
root@localhost:/home/oslab# ./21
Number of thread : 9
123456789
```

22. 다음은 사용자 쓰레드를 사용하여 1부터 100까지 합을 구하는 프로그램이다.

< 조 건 >

1. 총 10개의 쓰레드를 사용(main() 함수 포함)
2. 각 쓰레드는 연속된 10개의 숫자를 더한 후 종료
3. 생성되는 쓰레드는 더하기 시작하는 숫자를 인자로 전달 받음
4. 전역변수를 활용하여 각 쓰레드에서 구한 합을 총합에 더함
5. 가장 마지막으로 덧셈을 끝내 종료되는 쓰레드에서 총합을 화면에 출력
6. 경쟁 조건에 있는 전연변수 사용에는 mutex를 사용하여 동기화 처리

int main(void)
 - 쓰레드에서 덧셈을 시작할 숫자를 인자로 넘겨주며 9개의 쓰레드 생성
 - 화면에 "start : 1, main thread" 출력
 - 1부터 10까지 더함

void *ssu_thread(void *arg)
 - 쓰레드에서 실행되는 함수
 - 인자로 받은 수를 화면에 "start : 수" 형식으로 출력
 - 인자로 받은 수부터 연속된 10개의 숫자를 더함

void aad_sum(int)
 - 총합을 구하는 함수
 - 인자로 받은 수를 전역변수 sum에 더함

void finish_thread(void)
 - 쓰레드를 종료하는 함수
 - 전변수로 finish를 사용하여 현재 종료되는 쓰레드가 마지막 쓰레드인지 확인
 - 마지막 쓰레드인 경우 총합을 "total : 수" 형식으로 화면에 출력

22.c

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

void *ssu_thread(void *arg);
void add_sum(int loc_sum);
void finish_thread(void);

pthread_mutex_t mutex;
int finish;
int total_sum;

int main(void){
        pthread_t tid;
        int loc_sum = 0;
        int i;

        // your code starts here!
}

void *ssu_thread(void *arg){
        int loc_sum = 0;
        int start;
```

```
        int i;

        // your code starts here!
}

void add_sum(int loc_sum){
        // your code starts here!
}

void finish_thread(void){
        // your code starts here!
}
```

실행결과

```
root@localhost:/home/oslab# ./22
start : 21
start : 41
start : 31
start : 11
start : 51
start : 61
start : 71
start : 1, main thread
start : 91
start : 81
total : 5050
```

23. 다음은 부모 프로세스에서 입력 받은 메시지를 2개의 자식 프로세스를 통해 화면에 출력하는 프로그램이다.

< 조 건 >

1. 부모와 자식 프로세스간의 메시지를 전달하기 위해 공유메모리를 생성
2. key와 id값을 생성한 후 전역변수 data에 공유메모리 생성(key 생성을 위한 인자들은 임의로 설정)
3. 공유 메모리의 마지막 1byte는 부모와 자식 프로세스 간의 동기화를 위해 사용
4. 부모 프로세스에서 입력 받은 메시지의 크기는 127byte를 넘지 않는 것으로 가정

부모 프로세스
1. 자식 프로세스 2개 생성
2. “parent : ” 출력 후 메시지를 입력받음
3. 입력 받은 메시지를 공유메모리에 저장 후 자식 프로세스들에게 SIGUSR1을 전달
4. 입력 받은 메시지가 “exit”일 경우 SIGKILL을 자식 프로세스들에게 전달 후 종료
5. 두 개의 자식 프로세스에서 부모 프로세스가 전달한 메시지를 출력하기 까지 공유메모리의 마지막 1byte를 계속 검사하면서 대기
※두 개의 자식 프로세스는 공유메모리 마지막 1byte에 메시지를 출력했다는 표시를 함 (표시 및 검사 방법은 자유)
6. [2. “parent : ” 출력 후 메시지를 입력받음] 반복

자식 프로세스
1. SIGUSR1 시그널에 대한 핸들러를 등록 한 후 sleep(1)을 호출하는 무한루프
2. 부모 프로세스로부터 받은 메시지를 화면에 출력 후 공유 메모리 data의 가장 마지막 1byte에 출력 했다는 표시

void ssu_signal_handler1(int signo)
- 첫 번째 자식 프로세스에서 SIGUSR1 시그널에 대한 핸들러로 사용하는 함수
- “child1 : 부모로부터 받은 메시지”를 출력

void ssu_signal_handler2(int signo)
 - 두 번째 자식 프로세스에서 SIGUSR1 시그널에 대한 핸들러로 사용하는 함수
 - “child2 : 부모로부터 받은 메시지”를 출력

23.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_MEM_SIZ 128
#define BUF_SIZ 128

void ssu_signal_handler1(int signo);
void ssu_signal_handler2(int signo);

char *data;

int main(void)
{
        key_t key;
        pid_t pid[2];
        char buf[BUF_SIZ];
        int shm_id;
        int n;
        // your code starts here!
}

void ssu_signal_handler1(int signo){
        // your code starts here!
}

void ssu_signal_handler2(int signo){
        // your code starts here!
}
```

실행결과

```
root@localhost:/home/oslab# ./23
parent : hello world
child1 : hello world
child2 : hello world
parent : lsp final exam
child1 : lsp final exam
child2 : lsp final exam
parent : su go
child2 : su go
child1 : su go
parent : exit
```

※ (보너스 문제) 주어진 제약조건에 맞게 프로그램을 완성하시오. [15점]
(주의할 점 : gcc로 옵션 없이 컴파일시 (1) 에러 발생시 0점 처리 (2) 주어진 제약조건을 변경할 경우 0점 처리 (3) 출력 결과가 하나라도 다를 경우 0점 처리 (쓰레드 스케줄링에 따라 다른 출력 결과가 나오는 경우는 제외) (4) warning 한 개당 2.5점 감점)

다음은 명령행 인자로 문자열 하나를 입력받고, 파이프를 사용하여 터미널 정보를 얻어 와서 open한 다음, 26개의 자식 프로세스를 생성하여 경쟁 조건이 없이 그대로 출력하고, 부모 프로세스가 종료된 다음에 자식 프로세스 모두를 SIGUSR1 시그널 전송을 통해 종료하는 프로그램이다.

< 조 건 >

1. 실행 결과는 반드시 터미널로 출력되어야 하며, printf() 함수 사용 불가능함
2. 어떠한 경우에도 임시파일(FIFO, 텍스트 파일 포함) 생성을 허용하지 않음
3. system() 함수 사용 불가능
4. 출력에 있어서 경쟁 조건이 발생하면 안됨

변수 설명
alpha_proc[NUM_ALPHA]
자식 프로세스의 PID를 저장할 배열
2. pnc_pipe1[ ] [ ], pnc_pipe2[ ] [ ]
부모-자식 간 사용할 파이프

int main(void);
명령행 인자로 문자열을 하나 받음(argv[1])
- 기존의 표준출력을 닫고도 터미널의 정보를 얻어내서 터미널에 출력
새롭게 터미널을 사용할 수 있게 되면 init_str 문자열 출력
main() 함수가 끝나고 자식 프로세스의 종료를 위하여 atexit()로 함수 등록
- 부모 프로세스에서 자식 프로세스를 26개 생성하며, 각 프로세스마다 부모-자식간에 사용할 파이프를 2개씩 가지게 구현(입력용, 출력용)
- 각각의 자식 프로세스는 알파벳 소문자만을 출력하는 프로세스이며(a ~ z), 부모 프로세스에서는 공백 문자(' ')만 출력
자식 프로세스의 종료는 SIGUSR1 시그널 핸들러를 통해서 구현
- 부모 프로세스에서 문자열을 순회, 파이프로 자식 프로세스와 통신하면서 명령행 인자로 받은 문자열을 출력
문자열 출력이 끝나면 정상 종료

void ssu_atexit(void);
main() 함수가 끝나고 수행되고, 자식 프로세스를 종료

void ssu_sigusr1_handler(int signo);
핸들러가 호출 되었을 때, 자식 프로세스는 종료

ssu_final_bonus.c

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

```
#define BUF_SIZE 2048
#define STDIN 0
#define STDOUT 1
#define NUM_ALPHA 26

int alpha_proc[NUM_ALPHA];                // 26개의 프로세스를 저장할 배열
int pnc_pipe1[NUM_ALPHA][2];              // 부모-자식간에 사용할 파이프1
int pnc_pipe2[NUM_ALPHA][2];              // 부모-자식간에 사용할 파이프2
char *init_str = "Linux System Programming Bonus!\n";       // 터미널을 open하고 출력할 문자열
char *end_str = "End of Bonus!\n";        // 자식 프로세스를 모두 종료시키고 마지막에 출력할 문자열
int term_fd;                              // 터미널에 관련된 파일디스크립터

void ssu_atexit(void);
void ssu_sigusr1_handler(int signo);

int main(int argc, char *argv[])
{
        int termpipe_fd[2], term_fd, i;          // 터미널의 정보를 알아내서 pipe를 이용하여 부모에게 전송
        char cur_term[BUF_SIZE] = {0};           // 터미널의 정보를 저장할 문자열
        char c, tmp;
        pipe(termpipe_fd);
        close(STDOUT);

        // 에러처리에는 stderr 사용 가능하나, 일반 문자열을 stderr으로 출력하면 안됨
        // your code starts here!

        exit(0);
}

void ssu_atexit(void) {
        // your code starts here!
}

void ssu_sigusr1_handler(int signo) {
        // your code starts here!
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_final_bonus "see you on operating systems"
Linux System Programming Bonus!
see you on operating systems
End of Bonus!
```

<참고> 코드 작성에 필요한 함수

memset : s가 가르키는 메모리 영역의 처음 n바이트를 상수 바이트 c로 채우는 함수

```
#include <string.h>
void *memset(void *s, int c, size_t n);
                                        리턴값: void *s에 대한 포인터, 실패시 NULL
```

gettablesize : 기술자(descriptor) 테이블 크기를 알아내는 함수

```
#include <unistd.h>
int getdtablesize(void);
                                        리턴값: 프로세스가 열 수 있는 파일의 최대 갯수를 리턴
```

strtok : 문자열을 문자로 자르는 함수

```
#include <string.h>
char *strtok(char *restrict s1, const char *restrict s2);
                        리턴값: 잘라내기한 문자열의 첫 번째 포인터를 반환하며, 문자열이 없다면 NULL을 리턴
```