

※ 다음 물음에 답하시오 [1~19]

1 다음 프로그램은 자식 프로세스에서 사용자 모드로 CPU를 사용한 시간과 시스템 모드에서 CPU를 사용한 시간을 출력한다. 아래 실행결과를 보고 빈칸을 채우시오. [1.5점]

```
1c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/resource.h>
#include <time.h>

double ssu_maketime(struct timeval *time);
void term_stat(int stat);
void ssu_print_child_info(int stat, struct rusage *rusage);

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

    if ( (pid = fork()) == 0 ) {
        char *args[] = {"find", "/", "-maxdepth", "4", "-name", "stdio.h", NULL};
        if ( execv("/usr/bin/find", args) < 0 ) {
            fprintf(stderr, "exec error\n");
            exit(1);
        }
    }
    if ( wait3(&status, 0, &rusage) == pid)
        ssu_print_child_info(status, &rusage);
    else { fprintf(stderr, "wait3 error\n");
           exit(1);
         }
    exit(0);
}

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

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

```

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

```

실행결과

```

root@localhost:/home/oslab# ./ssu_execv_1
/usr/include/stdio.h
find: '/run/user/1000/gvfs': 허가 거부
Termination info follows
normally terminated. exit status = 1
user CPU time : 0.06(sec)
system CPU time : 0.07(sec)

```

2. 다음 프로그램은 자식 프로세스들의 종료 상태를 출력한다. 아래 실행결과를 보고 빈칸을 채우시오. [3점]

2.c

```

#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# ./ssu_wait_1
normal termination, exit status = 7
abnormal termination, signal number = 6 (core file generated)
abnormal termination, signal number = 8 (core file generated)

```

3. 다음 프로그램은 디몬 코딩 규칙에 따라 디몬 프로세스를 생성한다. 아래 실행결과를 보고 빈칸을 채우시오. [2점]

```
3.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 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 ((pid = fork()) < 0) {
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if (pid != 0)
        exit(0);
    pid = getpid();
    printf("process %d running as daemon\n", pid);
    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# ./ssu_daemon

```
parent process : 12153
daemon process initialization
process 12154 running as daemon
```

4. 다음 프로그램은 raise()를 호출하여 자기 자신에게 시그널을 전달한다. 아래 실행결과를 보고 빈칸을 채우시오. [2점]

4.c

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

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

int main(void)
{
    if ( signal(SIGINT, ssu_signal_handler1) == SIG_ERR ) {
        fprintf(stderr, "cannot handle SIGINT\n");
        exit(EXIT_FAILURE);
    }

    if ( signal(SIGUSR1, ssu_signal_handler2) == SIG_ERR ) {
        fprintf(stderr, "cannot handle SIGUSR1\n");
        exit(EXIT_FAILURE);
    }

    raise(SIGINT);
    raise(SIGUSR1);
    printf("main return\n");
    exit(0);
}

void ssu_signal_handler1(int signo) {
    printf("SIGINT 시그널 발생\n");
}

void ssu_signal_handler2(int signo) {
    printf("SIGUSR1 시그널 발생\n");
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_raise
SIGINT 시그널 발생
SIGUSR1 시그널 발생
main return
```

5. 다음 프로그램은 pthread_join()를 호출하여 생성한 스레드가 종료될 때까지 기다린다. 아래 실행결과를 보고 빈칸을 채우시오. [1.5점]

<div>5.c</div> <pre>#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 (pthread_create(&tid1, NULL, ssu_thread, (void *)&thread1) != 0) { fprintf(stderr, "pthread_create error\n"); exit(1); } if (pthread_create(&tid2, NULL, ssu_thread, (void *)&thread2) != 0) { fprintf(stderr, "pthread_create error\n"); exit(1); } pthread_join(tid1, (void *)&status); pthread_join(tid2, (void *)&status); exit(0); } void *ssu_thread(void *arg) { 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; }</pre>	<div>실행결과</div> <pre>root@localhost:/home/oslab# ./ssu_pthread_join_1 1 : 0 2 : 0 1 : 1 2 : 1</pre>
--	---

```
1 : 2
2 : 2
1 : 3
2 : 3
1 : 4
2 : 4
```

6. 다음 프로그램은 `sigaction()`을 호출하여 `SIGUSR1` 시그널에 관한 액션을 변경해서 확인하고 `sigprocmask()`를 호출하여 `sigusr1`을 블록한 다음, `sigusr1` 시그널을 발생시켜서 블록되었는지 확인한다. 아래 실행결과를 보고 빈칸을 채우시오. [2점]

6.c

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

void ssu_signal_handler(int signo) {
    printf("ssu_signal_handler control\n");
}

int main(void) {
    struct sigaction sig_act;
    sigset_t sig_set;
    sigemptyset(&sig_act.sa_mask);
    sig_act.sa_flags = 0;
    sig_act.sa_handler = ssu_signal_handler;
    sigaction(SIGUSR1, &sig_act, NULL);
    printf("before first kill\n");
    kill(getpid(), SIGUSR1);
    sigemptyset(&sig_set);
    sigaddset(&sig_set, SIGUSR1);
    sigprocmask(SIG_SETMASK, &sig_set, NULL);
    printf("before second kill\n");
    kill(getpid(), SIGUSR1);
    printf("after second kill\n");
    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./ssu_sigaction_1
before first kill()
ssu_signal_handler control
before second kill()
after second kill()
```

7. 다음 프로그램은 ssu_dummy.c 파일과 프로젝트 값 B로 key를 얻고, 그 key에 해당하는 메시지 큐 ID를 얻고 나서, 얻은 메시지 큐에 메시지를 보낸다. 아래 실행결과를 보고 빈칸을 채우시오. [2점]

7.c
<pre>#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 ((msg_queueid = msgget(key, 0644 IPC_CREAT)) == -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 (msgctl(msg_queueid, IPC_RMID, NULL) == -1) { fprintf(stderr, "msgctl error"); exit(1); } exit(0); }</pre>
실행 결과
<pre>root@localhost:/home/oslab# touch ssu_dummy.c root@localhost:/home/oslab# ./ssu_msgqueue_1 Enter lines of text, ^D to quit:</pre>

Hi OSLAB!
Nice to meet you!

8. 다음 프로그램 실행 시 아래 실행 결과가 나올 수 있도록 프로그램을 완성하시오. [5점]

8.c

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

void ssu_signal(int signo);

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");
    }
}

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

실행결과

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

9. setjmp0, longjmp0 실행 후 값을 확인하는 프로그램이다. 실행결과를 보고 프로그램을 완성하시오 [5점]

9.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 ex5-2.c -o
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 = 80, loc_volatile = 81, loc_register = 12
ret_val : 1
root@localhost:/home/oslab# gcc ex5-2.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
```

10. fcntl()을 사용하여 Nonblock 플래그를 설정하는 프로그램을 작성하시오. [6점]

< 조 건 >

1. 주어진 변수는 그대로 사용할 것
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 errno를 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);
}
```

10.c

```
#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
..
..

```

11. 프로그램 실행 시 아래와 같은 실행 결과가 나올 수 있도록 프로그램을 완성하시오. [5점]

test.txt

```

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

```

11.c

```

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

int main(void)
{
    struct flock lock;
    char *fname = "test.txt";
    int fd;
    pid_t pid;
    char c;

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

    lock.l_type = F_WRLCK;

    if((pid = fork0) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0){
        sleep(1);
        int i = 0;
        while(read(fd, &c, 1) > 0){
            lock.l_whence = SEEK_SET;
            lock.l_start = i++;

```

```

        lock.l_len = 1;
        if(fcntl(fd, F_SETLKW, &lock) == -1){
            fprintf(stderr, "fcntl error\n");
            exit(1);
        }
        fprintf(stderr, "%c", c);
    }
    exit(1);
}else{
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 1;
    if(fcntl(fd, F_SETLKW, &lock) == -1){
        fprintf(stderr, "fcntl error\n");
        exit(1);
    }
    while(1);
}
}

```

실행결과

```

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

```

12. 다음 첫 번째 인자로 받은 파일에 입력받은 파일에서 수정을 원하는 레코드에 write락을 설정하고 수정하는 프로그램을 작성하시오. [5점]

< 조 건 >

1. 주어진 변수는 그대로 사용할 것
2. 각 함수가 정의된 헤더파일을 정확히 쓸 것
3. ssu_employefile을 인자로 사용할 것
4. 본 프로그램의 실행파일은 1개의 인자만 받도록 할 것
5. fcntl()을 3번 사용할 것
6. lseek()을 2번 사용할 것
7. read(), write()를 각각 1번 사용할 것
8. lock을 설정하기 위한 fcntl()의 예외 처리를 위해 fprintf()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

ssu_employee.c

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

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

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

    if(argc < 2)
    {
        fprintf(stderr, "Usage :%s file \n", argv[0]);
        exit(1);
    }
    if((fd = open (argv[1], O_RDWR)) < 0)
    {
        fprintf(stderr, "Open error :%s file \n", argv[1]);
        exit(1);
    }
    if((flags = fcntl(fd, F_GETFL, DUMMY)) == -1)
    {
        fprintf(stderr, "fcntl F_GETFL Error \n");
        exit(1);
    }
```



```

}
flags |= O_APPEND;
if(fcntl(fd,F_SETFL,flags)==-1){
    fprintf(stderr,"fcntl F_SETFL error \n");
    exit(1);
}
pid=getpid();
while(1)
{
    printf("Enter employee name : ");
    scanf("%s",record.name);
    if(record.name[0]=='.')
        break;

    printf("Enter employee salary :");
    scanf("%d",&record.salary);
    record.pid=pid;
    length=sizeof(record);
    if(write(fd,(char *)&record,length)!=length)
    {
        fprintf(stderr,"record write error \n");
        exit(1);
    }
}
close(fd);
exit(0);
}

```

```

root@localhost:/home/oslab# touch ssu_employeeefile
root@localhost:/home/oslab# ./a.out ssu_employeeefile
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 : .

```

12.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)
{
    printf("\n Enter 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);
        lock.l_type=F_UNLCK;
        fcntl(fd,F_SETLK,&lock);
        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);
exit(0);
}

```

실행 결과

```

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

```

Enter new salary : 500000

13. 다음 프로그램은 /var/log/system.log 파일에 자신의 pid를 로그 메시지로 남기는 디몬 프로세스를 생성하는 프로그램이다. 아래 실행 결과가 나올 수 있도록, 프로그램을 완성하시오. [5점]

〈 조 건 〉

1. 디몬 프로세스를 생성하기 위한 7가지 규칙을 모두 사용할 것

13.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 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);
    }
    while(1){
        openlog("a.out", 0, LOG_LPR);
        syslog(LOG_ERR, "My pid is %d", getpid());
        closelog();
        sleep(5);
    }
    exit(0);
}

int ssu_daemon_init(void) {
    pid_t pid;
    int fd, maxfd;
    if((pid = fork0) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if(pid != 0) - 2
        exit(0);
    pid = getpid();
    setsid(); - 3
    signal(SIGTTIN, SIG_IGN);
    signal(SIGTTOU, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);
    maxfd = getdtablesize();
    for(fd = 0; fd < maxfd; fd++) - 5
```

```
close(fd);
```

```
umask(0); - 1
```

```
chdir("/"); - 4
```

```
fd = open("/dev/null", O_RDWR); - 6
```

```
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 a.out: My pid is 3409
```

(PID는 바뀔 수 있음)

14. 다음은 팬딩 중인 시그널 집합을 찾고 SIGINT 시그널이 포함되어 있는지 검사하는 프로그램이다. 아래 실행결과를 보고 프로그램을 작성하시오. [5점]

< 조 건 >

1. 주어진 변수는 그대로 사용할 것
2. sleep()을 통해 매초 팬딩 중인 시그널 집합을 확인할 것

14.c

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

int main(void)
{
    sigset_t pendingset;
    sigset_t sig_set;
    int count = 0;

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

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

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
count: 0
count: 1
count: 2
count: 3
^Zcount: 4
^Zcount: 5
count: 6
count: 7
count: 8
^CSIGINT가 블록되어 대기 중. 무한 루프를 종료.
```

15. 다음 프로그램은 UTC 기준 1970년 1월 1일 00:00:00이후 현재까지 흐른 시간을 출력하는 프로그램이다. 아래 실행결과를 보고 소스코드를 완성하시오. [15점]

< 조 건 >

1. clock_gettime()을 사용할 것(man page 활용)

15.c

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

#define SECS_IN_DAY (24 * 60 * 60)

static void displayClock(clockid_t clock, char *name);

int main(int argc, char *argv[])
{
    displayClock(CLOCK_REALTIME, "CLOCK_REALTIME");
    exit(0);
}

static void displayClock(clockid_t clock, char *name)
{
    struct timespec ts;
    if (clock_gettime(clock, &ts) == -1) {
        perror("clock_gettime");
        exit(EXIT_FAILURE);
    }
    printf("%-15s: %10ld.%03ld (" , name, (long) ts.tv_sec, ts.tv_nsec / 1000000);
    long days = ts.tv_sec / SECS_IN_DAY;
    if (days > 0)
        printf("%ld days + ", days);
    printf("%2ldh %2ldm %2lds", (ts.tv_sec % SECS_IN_DAY) / 3600, (ts.tv_sec % 3600) / 60, ts.tv_sec % 60);
    printf(")\n");
    if (clock_getres(clock, &ts) == -1) {
        perror("clock_getres");
        exit(1);
    }
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
CLOCK_REALTIME : 1591671943.248 (18422 days + 3h 5m 43 s)
```

16. 다음 프로그램은 인자로 입력받은 명령어를 실행시킨다. 명령어 실행 시 명령어의 표준에러는 출력하지 않는다. 정상수행 되었을 경우 ssu_log파일에 로그를 출력하고, 비정상 수행된 경우 로그를 출력하지 않는다. [10점]

< 조 건 >

1. 주어진 변수는 그대로 사용할 것
2. 로그의 형태는 “[요일 월 일 시:분:초 년] 명령어” 의 형태로 작성
3. 인자로 입력받은 명령어를 수행하기 위해 system()을 사용할 것

16.c

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

#define BUFFER_SIZE 1024

int main(int argc, char* argv[]){
    char buf[BUFFER_SIZE];
    time_t now;
    char *filename = "ssu_log";
    int ret;
    FILE *fp;

    fp = fopen(filename,"a+");
    strcpy(buf,argv[1]);
    strcat(buf," 2> /dev/null");
    ret = system(buf);
    if(WIFEXITED(ret) && !(WEXITSTATUS(ret) == 127 || WEXITSTATUS(ret) == -1)){
        time(&now);
        strcpy(buf,ctime(&now));
        buf[strlen(buf)-1] = 0;
        fprintf(fp,"%s] %s\n",buf,argv[1]);
    }

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out "echo ssu_final"
ssu_final
root@localhost:/home/oslab# ./a.out "echo ssu_final"
root@localhost:/home/oslab# cat ssu_log
[Tue Jun  9 11:22:02 2020] echo ssu_final
```

17. sigismember에서 어떤 시그널이 대기중인지 확인하는 프로그램이다. 단. 초기값이 공집합인 sigset_t set 집합에 SIGQUIT, SIGINT, SIGTSTP 시그널을 추가하고 이를 블록시킨 후 3초 뒤에 pending된 시그널을 sigset_t pending_sigset에 설정한다. 아래 실행결과를 보고 프로그램을 완성하시오. [10점]

< 조 건 >

1. sleep()을 사용할 것

17.c

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

int main()
{
    sigset_t pending_sigset;
    sigset_t set;

    sigemptyset(&set);
    sigemptyset(&pending_sigset);

    sigaddset(&set,SIGQUIT);
    sigaddset(&set,SIGINT);
    sigaddset(&set,SIGTSTP);

    sigprocmask(SIG_BLOCK,&set,NULL);

    printf("Kill SIGQUIT, SIGINT, and SIGTSTP signals to this process.\n");

    sleep(3);
    if(sigpending(&pending_sigset)!=0){
        printf("\n\nBlocked SIGNALS\n");
        if(sigismember(&pending_sigset,SIGQUIT))
            printf("SIGQUIT\n");
        if(sigismember(&pending_sigset,SIGINT))
            printf("SIGINT\n");
        if(sigismember(&pending_sigset,SIGTSTP))
            printf("SIGTSTP\n");
    }

    sigprocmask(SIG_UNBLOCK,&set,NULL);

    printf("If you kill SIGQUIT or SIGINT or SIGTSTP Signals, this message will not be shown\n");

    exit(0);
}
```

실행결과

```
% a.out
Kill SIGQUIT, SIGINT, and SIGTSTP signals to this process.
^Z^C
```

Blocked Signals

SIGINT SIGTSTP

18. 다음 프로그램은 인자로 입력받은 디렉토리의 일반파일을 tar명령어를 통해 테이핑한다. 아래 실행결과를 보고 프로그램을 완성하시오. 단. tar 명령어 수행으로 생성된 tar파일은 현재 작업 디렉토리에 위치해야 하며, tar파일 내부에 디렉토리가 포함되지 않아야 한다. [10점]

18.c
<pre> #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <dirent.h> #include <sys/stat.h> #include <string.h> #define BUFFER_SIZE 1024 int main(int argc, char* argv[]){ struct dirent *dirp; DIR *dp; struct stat statbuf; char cmd[BUFFER_SIZE]; char tar_path[BUFFER_SIZE]; getcwd(tar_path,BUFFER_SIZE); strcat(tar_path,"/tmp.tar"); strcpy(cmd,"tar -rf tmp.tar"); dp = opendir(argv[1]); chdir(argv[1]); while((dirp = readdir(dp)) != NULL){ stat(dirp->d_name,&statbuf); if(S_ISREG(statbuf.st_mode)){ strcat(cmd," "); strcat(cmd,dirp->d_name); } } system(cmd); link("tmp.tar",tar_path); unlink("tmp.tar"); exit(0); } </pre>
<p>실행결과</p> <pre> root@localhost:/home/oslab# ls -al ssu_dir 합계 20 drwxr-xr-x 3 room-205 room-205 4096 6월 9 11:36 . drwxr-xr-x 21 room-205 room-205 4096 6월 9 11:35 .. -rw-r--r-- 1 room-205 room-205 392 6월 8 19:36 a.c -rw-r--r-- 1 room-205 room-205 362 6월 8 19:32 b.c brw-r--r-- 1 root root 3, 10 6월 8 21:21 block crw-r--r-- 1 root root 3, 10 6월 8 21:21 character prw-r--r-- 1 root root 0 6월 8 21:21 fifo drwxr-xr-x 2 room-205 room-205 4096 6월 8 21:21 lsp root@localhost:/home/oslab# ./a.out ssu_dir root@localhost:/home/oslab# tar -tvf tmp.tar -rw-r--r-- room-205/room-205 392 2020-06-08 19:36 a.c -rw-r--r-- room-205/room-205 362 2020-06-08 19:32 b.c </pre>

19. 다음은 SIGCHLD 시그널에 함수를 등록하고 해당 시그널을 기다리는(suspend) 프로그램이다. 아래 실행결과를 보고 프로그램을 완성하시오. [5점]

< 조 건 >

1. sigaction()을 사용할 것
2. 자식 프로세스를 생성하고 자식프로세스는 3초 뒤 종료할 것
3. 부모 프로세스는 SIGCHLD이외의 시그널을 무시하며 SIGCHLD 시그널을 기다리게 할 것 (sigsuspend())

19.c

```
void ssu_signal_handler(int signo);

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

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

    sigfillset(&blk_set);
    sigdelset(&blk_set, SIGCHLD);

    printf("before fork\n");
    if((pid = fork0) < 0)
    {
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if(pid == 0)
    {
        printf("after fork in child, sleep...\n");
        sleep(3);

        exit(0);
    }
    else
    {
        printf("after fork in parent, suspend...\n");

        sigsuspend(&blk_set);
        printf("after suspend\n");
    }

    exit(0);
}

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

```
}
```

실행결과

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