Soongsil University

School of Computing

Final Exam

Course Title	리눅스 시스템 프로그래밍 설계 및 실습
Instructor	**Jiman Hong**
Date of Exam	**June 16, 2018**
Duration of Exam	**4 hours**
Number of Exam Page (including this cover page)	**32**
Number of Questions	**20+보너스**
Exam Type	**Closed Book**
Additional Materials Allowed	**None**

Student ID (학번)	
Name (이름)	
File Size (학생이 직접 입력)	**(bytes)**

Show Your Work and Reasoning Clearly!
Write legibly!
Write only to the space provide for each question!
Good Luck!

주의 사항

0. 가상화 프로그램인 VMWare에 리눅스 운영체제가 실행된 상태에서
(1) 101호의 경우 id : lsp_exam , passwd : oslab 으로 로그인
(2) 303호의 경우 id : room303, passwd : !lgesw으로 로그인

1. 현재 작업 디렉토리(/home/oslab)에 자기 "학번"을 파일 이름으로 하는 디렉토리를 생성(mkdir 명령어) 할 것
<예. 학생 학번이 20122336 일 경우 %mkdir 20122336>

2. 1번~10번 문제의 경우
(1) 답을 저장할 파일의 확장자 명은 .txt이며 각 문제 번호와 소문제 번호(문제마다 네모 박스에 기입된 번호)를 이용하여 파일을 생성해야 함. 각 문제에 대해 각 4~6개의 파일을 생성하고 답을 작성해야 함. 소문제의 번호가 같은 경우 같은 답이며 하나의 파일만 생성해야 함
(2) 각 문제의 답을 모르거나 답이 틀리더라도 NULL 파일을 생성해야 함. 파일을 생성하지 않을 경우 해당 문제의 배점 점수 만큼 감점
(3) 1~10번 문제에 대해 답을 파일에 작성할 경우 50개의 .txt 파일(1번~10번)이 /home/oslab/학번 디렉토리에 존재해야 함
(4) 네모 박스 안의 들어갈 내용만 써야 함. 예를 들어 세미콜론(문장 마침 기호) 두 번 사용, 괄호 개수 등이 안 맞는 경우는 0점 처리

3. 11번~20번 프로그램 문제의 경우
(1) 답을 저장할 파일의 확장자는 .c 이며 각 문제 번호를 이용하여 파일을 생성해야 함
(2) 각 문제의 답을 모르거나 답이 틀리더라도 NULL 파일을 생성해야 함. 파일을 생성하지 않을 경우 해당 문제의 배점 점수 만큼 감점
(3) 11~20번 문제의 답을 파일에 작성할 경우 10개 .c 파일(11번~20번)이 /home/oslab/학번 디렉토리에 존재해야 함
(4) 주어진 기능을 모두 구현하지 못하고 일부 기능만 구현했을 경우라도 반드시 컴파일이 에러 없이 실행되어야 하며 에러가 발생할 경우 해당 문제는 0점 처리
(5) 컴파일 시 warning이 한 개 발생할 때마다 해당 문제의 점수에서 10% 감점
(6) 개인적으로 디버깅을 위한 코드 등 문제와 상관 없는 코드는 반드시 삭제해야 함. 지우지 않을 경우 해당 문제의 점수에서 10% 감점
(7) 프로그램 구현 문제에서 주어진 조건을 변경하거나, 변수를 추가/변경할 경우 해당 문제는 0점 처리
(8) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리. 단 쓰레드의 tid, pid 등의 변수와 쓰레드 실행 순서 등 허용 가능한 출력 결과는 달라도 됨

4. 보너스 제의 경우
(1) 답을 저장할 파일은 21.c 이며 각 문제 번호를 이용하여 파일을 생성해야 함
(2) 답을 모르거나 답이 틀리더라도 NULL 파일은 생성하지 않아도 됨
(3) 주어진 기능을 모두 구현하지 못하고 일부 기능만 구현했을 경우라도 반드시 컴파일이 에러 없이 진행되어야 하며 에러가 발생할 경우 해당 문제는 0점 처리
(4) 컴파일 시 warning이 한 개 발생할 때마다 해당 문제의 점수에서 10% 감점
(5) 개인적으로 디버깅을 위한 코드 등 문제와 상관 없는 코드는 반드시 삭제해야 함. 지우지 않을 경우 해당 문제의 점수에서 10% 감점
(6) 프로그램 구현 문제에서 주어진 조건을 변경하거나, 변수를 추가/변경할 경우 해당 문제는 0점 처리
(7) 주어진 출력 결과와 하나라도 다를 경우 해당 문제는 0점 처리. 단 쓰레드의 tid, pid 등의 변수와 쓰레드 실행 순서 등 허용 가능한 출력 결과는 달라도 됨

5. 시험을 완료하면
(1) /home/oslab/ 디렉토리로 이동하여 학번을 이용하여 자신이 생성한 디렉토리의 크기(du -b 명령어)를 확인하고 시험지 첫장에 File Size라고 써 있는 칸에 디렉토리의 크기(byte)를 쓸 것
<예. 학생 학번이 20122336 일 경우 %du -b 20122336 >
(2) tar -cvf 학번(디렉토리).tar 학번(디렉토리)로(ex %tar -cvf 20120000.tar 20120000) tar 파일을 생성 후 shalsum 학번.tar(ex %shalsum 20120000.tar)로 hash value를 확인 후 학생이 별도 보관(사진 또는 복사)
(3) 감독관에게 USB(2개 복사) 복사를 요청
(4) USB에 복사된 파일의 크기와 (1)에서 확인한 파일 크기와 (2) 해쉬 값을 확인
(5) 사용한 PC에 디렉토리와 파일은 지우지 말고 그대로 둘 것
(6) 자신의 USB에 복사할 경우 감독관에게 허락받아야 함

6. 가산점 문제(17, 18, 19번)라고 명시된 문제는 (기본 배점 기준 추가 가산점 있음 : 완전히 구현한 학생 수(A)와 완전히 구현하지 못한 학생 수(B, 일부 기능 구현, warning 모두 포함)를 비교하여 (1) A < B/10 이면 완전히 구현한 학생들에게 기본 배점 추가 부여, (2) A < B/9 이면 완전히 구현한 학생들에게 (기본 배점-2) 추가 부여, (3) A < B/8 이면 완전히 구현한 학생들에게 (기본 배점-4) 추가 부여

7. 다음 사항을 어길 경우 F학점 처리
(1) 감독관이 허락하기 전에 USB 등을 마운트 시킬 경우 (자신이 작성한 답을 복사하기를 원할 경우 시험지를 제출하고 감독관의 감독 하에 USB 등에 복사)
(2) 유무선 네트워크 액세스 디바이스(동글, 에그 등)를 이용할 경우 부정행위로 간주
(3)^-Alt (컨트롤-알트) 키 등을 눌러 리눅스 콘솔 외 윈도우에 접근할 경우 부정행위로 간주
(4) 콘솔 터미널은 최대 2개까지만 열 수 있으며 추가 터미널을 생성을 시도할 경우 부정행위로 간주
(5) 1-10번 문제의 답을 시험지에 쓰는 경우 부정행위로 간주(옆 학생이 볼 수 있기 때문에 절대 네모박스 문제는 시험지에 답을 쓰지 말 것)
(6) 기타 부정 행위

※ 각 문제에서 주어진 프로그램 실행 시 주어진 실행결과가 나올 수 있도록, 빈 칸을 채우시오. [1-10, 네모 박스 당 0.5점, 총 25점]

1. 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);

(1)                          ;

int main(int argc, char *argv[],   (2)                    )
{
    ssu_printenv("Initially", &envp);
    (3)                          ;
    ssu_printenv("After changing TZ", &envp);
    (4)                                                  ;
    ssu_printenv("After setting a new variable", &envp);
    printf("value of WARNING is %s\n",   (5)                    );
    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()
```

2. 다음 프로그램은 부모 프로세스가 자식 프로세스를 생성 후 종료할 때까지 기다리고 출력한다. child1 프로세스는 execlp()를 사용하여 'date'를 실행시키고, child2 프로세스는 execlp()를 사용하여 'who'를 실행시킨다. 아래 실행결과를 보고 빈칸을 채우시오.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
(1)

int main(void)
{
    pid_t child1, child2;
    int pid, status;

    if ((child1 = fork()) == 0)
        (2)                                    ;

    if ((child2 = fork()) == 0)
        (3)                                    ;

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

    while (  (4)                               ) {
        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
```

3. 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) {
    (1)                  tms_start, tms_end;
    clock_t start, end;
    int status;

    printf("\ncommand: %s\n", cmd);
    if (    (2)                                    ) {
        fprintf(stderr, "times error\n");
        exit(1);
    }

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

```
    if ( (3)                                    ) {
        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 ( (4)                                    ) {
            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",
      ( (5)                                     ) / (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
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
(1)
#include <string.h>

void *ssu_thread(void *arg);

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

    if (    (2)                                                    ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    pid = getpid() ;
    (3)                                  ;
    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();
    (3)                                  ;
    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 (4)
root@localhost:/home/oslab# ./ssu_pthread_create_1
Main Thread: pid 3222 tid 3075864320
New Thread: pid 3222 tid 3075861312
```

5. 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>
(1)

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

int main(void)
{
    pthread_t tid1, tid2;

    if ( (2) ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    if ( (3) ) {
        fprintf(stderr, "pthread_create error\n");
        exit(1);
    }

    printf("thread1의 리턴을 기다림\n");
    (4) ;
    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
```

6. 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

(1)                                                    ;
(2)                                                    ;

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);
        (3)                                            ;
        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 ( (4)                                    )
          (5)                                       ;
        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
```

7. 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( (1)                                    );

    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( (1)                                    );
    exit(0);
}

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

    if ( (2)                                    ) {
        fprintf(stderr, "fcntl F_GETFL failed");
        exit(1);
    }

    (3)                          ;

    if ( (4)                                    ) {
        fprintf(stderr, "fcntl F_SETFL failed");
        exit(1);
    }
}

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

    if ( (2)                                    ) {
        fprintf(stderr, "fcntl F_GETFL failed");
        exit(1);
    }

    (5)                          ;

    if ( (4)                                    ) {
        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
..
..
```

8. 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 (  (1)                                                  ) {
        fprintf(stderr, "open error for %s\n", filename);
        exit(1);
    }

    if (  (2)                                                  ) {
        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 (  (3)                                  )
```

```
        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 ( (4)                                  )
        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 ( (3)                                  )
        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 ( (4)                                  )
        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 : (5)
fd2 : (6)
```

9. 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>
(1)

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

    (3) ;
    (4) ;
    (5) ;

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

    printf("Ctrl-C에 대한 블록을 해제\n");
    (6) ;
    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
```

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

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

double ssu_maketime(struct timeval *time);

void term_stat(int stat);
```

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

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

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

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

    if (    (2)                                                 )
        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 (   (3)                          )
        printf("normally terminated. exit status = %d\n", WEXITSTATUS(stat));
    else if (   (4)                          )
        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(  (5)                          ));
    printf("system CPU time : %.2f(sec)\n", ssu_maketime(  (6)                          ));
}
```

실행결과

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

※ 주어진 조건에 맞게 프로그램을 완성하시오. [11-20, 총75점]

11. 다음 프로그램은 SIGUSR1 시그널을 BLOCK 후 해당 시그널을 보냈을 때 pending되어 있는 시그널을 확인한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [5점]

< 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

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

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

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

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

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

    }
    else{

    }
}
```

실행결과

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

12. 다음 프로그램은 두 개의 쓰레드를 생성하여 producer 쓰레드는 buf에 값을 넣는 작업을 하는 것을, consumer 쓰레드는

buf에 있는 값을 사용하여 총 합을 구한 후 출력하는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [5점]

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. 뮤텍스 관련 변수 mutex와 cond1 및 cond2 는 메크로를 사용하여 초기화 할 것
3. 프로그램의 실행이 끝난 후 mutex와 cond1 및 cond2 변수를 해제할 것
4. producer 쓰레드와 consumer 쓰레드에서 pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_signal(), pthread_cond_wait()을 각각 한 번씩 사용할 것

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

int length;
int buf[100];

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

    for(i = 1; i <= 300; i++){

    }
}

void *ssu_thread_consumer(void *arg){
    int i;
    int sum = 0;

    for(i = 1; i <= 300; i++){

    }

    printf("%d\n", sum);
}

int main(void){
    pthread_t producer_tid, consumer_tid;

    pthread_create(&producer_tid, NULL, ssu_thread_producer, NULL);
    pthread_create(&consumer_tid, NULL, ssu_thread_consumer, NULL);

    exit(0);
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
45150
```

13. 다음 프로그램은 /var/log/system.log 파일에 자신의 pid를 로그 메시지로 남기는 디몬 프로세스를 생성한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [5점]

< 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

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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);
    }

    exit(0);
}

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

}
```

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)
```

14. 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. [5점]

< 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

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

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

void ssu_signal_handler(int signo);

int count = 0;

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

    while(1);

    exit(0);
}

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

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

실행 결과

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

15. 다음 프로그램은 kill() 호출을 통해서 main()함수의 인자로 들어온 프로세스 ID에 시그널을 보내는 것을 보여준다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [5점]

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ./a.out [-signal] <pid>의 형태로 입력을 받으며 signal의 위치는 변경될 수 없고 '-' 로 시작해야 함
3. 프로그램 실행 시 지정된 시그널이 없는 경우 SIGTERM을 기본으로 사용함
4. 프로그램 실행 시 시그널은 하나만 지정 가능하고 프로세스 ID는 여러 개 지정 가능함
5. 시그널이 정수로 입력된 경우 모든 시그널은 사용 가능하고 문자로 입력될 경우 SIGKILL, SIGINT, SIGUSR1만 사용 가능함
6. 시그널의 입력은 대수문자의 구분이 없음

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

#define pid_max 20

int main(int argc, char *argv[])
{
    int pid[pid_max];
    int pid_count = 0;
    int sig_pst = 0;        //signal position in argv[]
    int n = 0;
```

```
    if (argc <1) {
        fprintf(stderr, "usage: %s <-signal> <pid> ...\n", argv[0]);
        exit(1);
    }

    if(strstr(argv[1],"-")!=NULL){

    }
    else{

    }

    exit(0);
}
```

실행 결과1

```
root@localhost:/home/oslab# ./ssu_loop &
[1] 32212
root@localhost:/home/oslab# ./a.out 32212
[1]+  Terminated                ./ssu_loop
```

실행 결과2

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop & ./ssu_loop &
[2] 32220
[3] 32221
[4] 32222
root@localhost:/home/oslab# ./a.out -sigusr1 32220 32221 32222
[4]+  User defined signal 1    ./ssu_loop
[2]-  User defined signal 1    ./ssu_loop
[3]+  User defined signal 1    ./ssu_loop
```

실행 결과3

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop &
[2] 32229
[3] 32230
root@localhost:/home/oslab# ./a.out -10 32229 32230
[3]+  User defined signal 1    ./ssu_loop
[2]+  User defined signal 1    ./ssu_loop
```

실행 결과4

```
root@localhost:/home/oslab# ./ssu_loop & ./ssu_loop &
[1] 32293
[2] 32294
root@localhost:/home/oslab# ./a.out 32293 -sigint 32294
[1]-  Terminated                ./ssu_loop
[2]+  Terminated                ./ssu_loop
```

실행 결과5

```
root@localhost:/home/oslab# ./a.out 32293 -sigint 32294 -sigkill
<-signal> only can use 1 time
```

16. 다음 프로그램은 두 개의 쓰레드를 생성하여 permutation 결과를 출력하는 것을 보여준다. main()에서 ssu_thread1()과 ssu_thread2()를 각각 호출하여 pthread_cond_signal()과 pthread_cond_wait()을 통해 번갈아가며 permutation 순열을 출력한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [10점]

< 조 건 >

1. 각 함수가 정의된 해더파일을 정확히 쓸 것
2. mutex와 cond는 메크로를 사용하여 초기화를 할 것
3. 프로그램의 실행이 끝난 후 mutex와 cond의 해제를 할 것
4. scanf()를 한 번 사용하여 입력을 받으며 10P4와 같은 형태로 입력을 받을 것
5. 잘못된 입력이 있을 경우 다시 입력을 받을 것
6. 생성된 쓰레드는 각각 pthread_cond_wait(), pthread_cond_signal()을 두 번씩 사용하고 pthread_mutex_lock(), pthread_mutex_unlock()을 한 번씩 사용할 것
7. result는 결과를 저장하고 buf는 sprintf()를 사용하여 수식을 저장할 것

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

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

pthread_mutex_t mutex1
pthread_mutex_t mutex2
pthread_cond_t cond1
pthread_cond_t cond2

int count = 0;
int input1 = 0, input2 = 0;
int result;
char buf[BUFSIZ];

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

    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);
    }

    while(1){
        //입력, 양수P양수의 형태만 가능

    }

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

```
}

void *ssu_thread1(void *arg){
    while(1){

    }

    return NULL;
}

void *ssu_thread2(void *arg){
    while(1){

    }

    return NULL;
}
```

실행결과

```
root@localhost:/home/oslab# ./a.out
12p8
12P22
12P8
Thread 1 : 12=12
Thread 2 : 12 x 11=132
Thread 1 : 12 x 11 x 10=1320
Thread 2 : 12 x 11 x 10 x 9=11880
Thread 1 : 12 x 11 x 10 x 9 x 8=95040
Thread 2 : 12 x 11 x 10 x 9 x 8 x 7=665280
Thread 1 : 12 x 11 x 10 x 9 x 8 x 7 x 6=3991680
Thread 2 : 12 x 11 x 10 x 9 x 8 x 7 x 6 x 5=19958400
complete
```

17. 다음 프로그램은 파일과 파일의 접근 권한 모드를 메인함수의 인자로 입력받아 입력된 파일의 모드를 변경한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [5점, 가산점 문제]

< 조 건 >

1. 각 함수가 정의된 헤더파일을 정확히 쓸 것
2. ./a.out MODE FILE의 형태로 입력을 받고 MODE는 8진수임
3. ./a.out -b MODE FILE의 형태로 입력을 받을 경우 MODE는 2진수임
4. MODE는 한 개만 지정 가능하고 FILE은 여러 개 지정 가능함
5. getopt()를 한 번 사용하여 옵션 처리를 할 것
6. MODE가 잘 못 입력된 경우 에러 처리를 할 것

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

int main(int argc, char *argv[]){
    mode_t mode=0;
    int digit;
    int flag_b=0;
    int i, j;
    char mode_arg[30] = {0, };
```

```
    if(argc < 3){
        fprintf(stderr, "usage : ssu_chmod MODE FILE...\n");
        exit(1);
    }

    for(i = strlen(mode_arg)-1, j=0; i >= 0; i--, j++){
        digit = mode_arg[i] - '0';
    }

    for(i = flag_b==0?2:3; i < argc; i++)
        if(chmod(argv[i], mode) < 0)
            fprintf(stderr, "chmod() error\n");

    return 0;
}
```

실행결과

```
root@localhost:/home/oslab# ls -al a.txt b.txt
---------- 1 kym kym 0  6월 14 11:12 a.txt
---------- 1 kym kym 0  6월 14 11:45 b.txt
root@localhost:/home/oslab# ./a.out 4744 a.txt b.txt
root@localhost:/home/oslab# ls -al a.txt b.txt
-rwsr--r-- 1 kym kym 0  6월 14 11:12 a.txt
-rwsr--r-- 1 kym kym 0  6월 14 11:45 b.txt
```

실형결과

```
root@localhost:/home/oslab# ls -al a.txt
---------- 1 kym kym 0  6월 14 11:12 a.txt
root@localhost:/home/oslab# ./a.out -b 0100111110110 a.txt
root@localhost:/home/oslab# ls -al a.txt
-rwsrw-rw- 1 kym kym 0  6월 14 11:12 a.txt
```

실행결과

```
root@localhost:/home/oslab# ./a.out 944 a.txt
MODE error : 944
```

18. 다음 프로그램은 디렉터리를 복사한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [15점, 가산점 문제]

< 조 건 >

1. 다음 프로그램은 일반 파일은 복사하지 않고 디렉터리 파일만 복사하는 프로그램임
2. 명령어는 ‘./a.out [-d][N] [SOURCE][TARGET]’ 의 형식을 따름
3. [SOURCE]는 복사대상 디렉터리, [TARGET]은 복사된 디렉터리
4. [-d] 옵션은 필수 옵션임
5. [N] 옵션은 1~9까지 숫자를 입력할 수 있으며, 입력 된 N개만큼 자식 프로세스(fork())를 생성함을 의미하고 N개의 자식 프로세스들이 하위 디렉터리를 복사함. 단
   (1) 하위 디렉터리의 개수 < N이면 하위 디렉터리 개수만큼의 자식 프로세스가 디렉터리를 복사하고 (N-하위 디렉터리 개수)의 자식프로세스는 생성하지 않음
   (2) 하위 디렉터리의 개수 > N이면 첫 번째 자식 프로세스는 첫 번째 하위 디렉터리를 복사, 두 번재 자식프로세스는 두 번째 하위 디렉터리를 복사 하는 등 N개의 자식프로세스들이 순서대로 하나의 하위 디렉터리를 복사하고 부모 프로세스가 나머지 하위 디렉터리 복사
   (3) 하위 디렉터리의 개수 = N이면 N개의 자식 프로세스가 N개의 하위 디렉터리를 순서대로 복사
6. 디렉터리 복사를 완료한 후에는 [SOURCE] 디렉터리 이름과 [TARGET] 디렉터리 이름을 표준출력
7. 생성된 자식 프로세스들은 각자 복사한 디렉터리의 이름과 pid를 표준출력
8. [TARGET] 디렉터리가 이미 존재하는 경우 덮어씀
9. [SOURCE]와 [TARGET] 이 동일한 이름의 디렉터리인 경우 오류 처리
10. scandir() 과 같은 메모리 동적할당 관련 함수들을 사용할 경우 반드시 메모리 회수를 해야 함
11. 주어진 변수 외에 추가적인 변수 사용, 구조체 변경, 불필요한 헤더파일 삽입은 허용하지 않음

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

#define BUFFER_SIZE 1024
#define ERROR_MSG "ssu_cp error\nusage : cp [-d][N] (1 <= N < 10)\n"
#define isdigit(x) (x>='0'&&x<='9')?1:0 //숫자 판단 매크로

void check_flag(int argc, char *argv[]);
static int filter(const struct dirent *dirent); //현재 디렉터리(.)와 상위디렉터리(..)를 생략하는 함수
void copy_dir(char *source, char *target);

int flag_d ,N;

int main(int argc, char *argv[])
{
        char opt;
        char source[PATH_MAX+1];
        char target[PATH_MAX+1];
        struct stat source_info, target_info;
        while(                          ) //옵션 인자처리
        {
                switch(opt)
                {
```

```
                        case '?' :
                                break;
                }
        }

if (flag_d) {



                copy_dir(source, target);
        }
        exit(0);
}

void copy_dir(char *source, char *target) {
        struct stat source_info;
        struct stat target_info;
        struct dirent **namelist;
        char source_name[PATH_MAX];
        pid_t check_pid;
        int status;
        int source_length;
        int target_length;
        int count;
        int i;

        if (stat(source, &source_info) < 0) {

        }
        printf( "src : %s , dst : %s \n",source, target);

        //디렉터리 복사 수행
}
static int filter(const struct dirent *dirent) {

}
```

실행결과

```
root@localhost:/home/oslab# ./a.out -d3 test2 test3
target : test3
source : test2
src : test2 , dst : test3
Directory test2/test3 copied by Process id : 32466
src : test2/test3 , dst : test3/test3
Directory test2/test1 copied by Process id : 32465
src : test2/test1 , dst : test3/test1
Directory test2/test1/aa copied by Process id : 32467
src : test2/test1/aa , dst : test3/test1/aa

root@localhost:/home/oslab# tree -fa test2 test3
test2
├── test2/test1
│   └── test2/test1/aa
└── test2/test3
```

```
test3
├── test3/test1
│   └── test3/test1/aa
└── test3/test3
6 directories, 0 files
```

19. 다음 프로그램은 이 프로그램은 인자로 받은 디렉토리 내의 모든 일반파일에 대한 생성, 수정, 삭제에 대해 모니터링한다. (설계과제 3번 ssu_backup의 답안과 동일한 구조로 구현했지만 파일의 백업을 제외하고 모니터링만을 수행) 모든 일반파일에 대해 각 파일마다 하나의 쓰레드를 통해 모니터링을 수행하며, 각 쓰레드는 파일의 변화에 대해 표준출력으로 출력하기 때문에 동기화를 해야 한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [15점, 가산점 문제]

< 조 건 >

1. 프로그램의 인자로 디렉토리명을 받음. 단, 절대경로와 상대경로 디렉터리를 모두 처리해야 함
2. 명령어는 './a.out [DIRECTORY]' 형식을 따름
3. 인자가 디렉토리가 아니거나 존재하지 않는 파일(일반 파일 포함)일 경우 stderr으로 에러 메시지 출력
4. 처음 프로그램을 실행했을 때 모니터링 디렉토리에 있는 파일에 대해서는 별도의 메시지를 출력하지 않음
5. 프로그램이 실행된 후, 모니터링 디렉토리와 하위 디렉토리에 있는 모든 일반파일에 대한 생성, 수정, 삭제 행위를 모니터링하며, 관련 행위의 메시지를 표준출력으로 출력
6. 일반파일이 생성될 때 마다 하나의 모니터링 쓰레드가 만들어지고 그 파일에 대해 모니터링을 수행
7. 모니터링 쓰레드는 모니터링 대상이 되는 파일이 삭제될 때까지 종료하지 않음
8. 모니터링 대상이 되는 파일이 수정되어 st_mtime이 변화되었을 때, 수정 메시지를 표준출력으로 출력
9. 동시에 여러 개의 파일에 대해 모니터링 쓰레드가 실행되어야 하기 때문에 생성, 수정, 삭제 행위 관련 메시지 표준출력 시 반드시 mutex를 사용하여 동기화
10. 생성, 수정, 삭제 행위 메시지는 다음의 형식을 반드시 준수하여야 함, 아래와 같이 출력 메시지는 경로가 아닌 파일명을 사용해야 함
- 생성 시 : [MMDD HH:MM:SS] FILE is created [size:--/mtime:MMDD HH:MM:SS]
- 수정 시 : [MMDD HH:MM:SS] FILE is modified [size:--/mtime:MMDD HH:MM:SS]
- 삭제 시 : [MMDD HH:MM:SS] FILE is deleted
11. 주어진 함수 구조, 변수 및 구조체는 변경 가능

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

#define MAX_DIR 64
#define  true 1
#define false 0
typedef int bool;

char pathname[PATH_MAX];
char dirname[PATH_MAX];
pthread_mutex_t mutex;

typedef struct thread_struct
{
        struct thread_struct *next;
        pthread_t tid;
        char *data;
} ssu_thread;

ssu_thread *HEAD = NULL, *TAIL = NULL;
```

```
char *dir_list[MAX_DIR];
int dir_cnt = 0;
bool isnewfile = false;
int startfile_fin = 0;

void directory_monitor(char *link);
ssu_thread *make_thread(char *data);
void thread_function(void *arg);
void monitor_function(char *name);
void print_log(char *name, int size, time_t mt, int index);
int check_list(char *data);
void delete_list(char *name);

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

        if (argc != 2)
        {
                fprintf(stderr, "usage : ./<<<MW2>>> <directory>\n");
                exit(1);
        }

        time_t old = 0, new = 0;
        time_t dir_time_now[MAX_DIR] = {0};
        time_t dir_time_new;
        while (1)
        {
                directory_monitor();
        }
}

void directory_monitor(char *link)
{
        struct dirent **namelist;
        int cont_cnt ;

        for (int i = 0; i < cont_cnt; i++)
        {
                struct stat src;

                if (S_ISDIR(src.st_mode))
                {
                        directory_monitor();
                }
                else
                {
                        make_thread();
                }
        }

        for (int i = 0; i < cont_cnt; i++)
                free(namelist[i]);
        free(namelist);
        return;
}
```

```
ssu_thread *make_thread(char *data){
        ssu_thread *temp

        if(pthread_create() != 0){
                fprintf(stderr, "pthread_crate error\n");
        }
        return temp;
}

void thread_function(void *arg){
        char *data = (char *)arg;
        monitor_function(data);
}

void monitor_function(char *name){
        struct stat src_sc;
        time_t old = 0, new = 0;
        int i = 0;
        int check = 0;

        while (true)
        {
                if ()
                {
                        print_log(name, src_sc.st_size, new, 1);
                }
                else if()
                {
                        print_log(name, src_sc.st_size, new, 2);
                }
                old = new;
        }
}

void print_log(char *name, int size, time_t mt, int index){
        time_t timer;
        struct tm *t ;
        char temp[14];
        struct tm *tt;
        int k;

        switch (index)
        {
        case 1:
                printf("message\n");
                break;
        case 2:
                printf("message\n");
                break;
        case 3:
                printf("message\n");
                break;
        }
```

```
}


int check_list(char *data)
{
        ssu_thread *cur;
        for (cur = HEAD; cur != NULL; cur = cur->next)
        {
                if (  )
                        return 0;
        }
        return 1;
}

void delete_list(char *name){
        ssu_thread *cur;
        for (cur = HEAD; cur != NULL; cur = cur->next)
        {
                if (  )
                {
                }
        }
        return;
}
```

실행결과 // 테스트를 위해 컨트롤 터미널 2개가 필요, 하나의 터미널은 프로그램 실행, 하나는 파일 수정

```
root@localhost:/home/oslab# ls
1.txt  2.txt  dir2
root@localhost:/home/oslab# ls dir2
3.txt
root@localhost:/home/oslab# vi 4.txt              -> (1) dir1/4.txt 생성
root@localhost:/home/oslab# ls
1.txt  2.txt  4.txt  dir2
root@localhost:/home/oslab# vi 1.txt              -> (2) dir1/1.txt 수정
root@localhost:/home/oslab# rm ./dir2/3.txt       -> (3) dir2/3.txt 삭제
root@localhost:/home/oslab1# vi ./dir2/4.txt      -> (4) dir2/4.txt 생성
root@localhost:/home/oslab# rm *                  -> (5) dir1/1.txt,2.txt,4.txt 동시삭제
```

실행결과 // 파일수정시 vi에디터를 사용할 경우 .swp 파일에 대한 메시지가 출력될 수 있음(고려하지 않아도 됨)

```
root@localhost:/home/oslab# ./a.out dir1
[0612 01:40:32] 4.txt is created [size:11/mtime:0612 01:40:32]    -> (1)
[0612 01:40:35] 1.txt is modified [size:23/mtime:0612 01:40:36]   -> (2)
[0612 01:40:37] 3.txt is deleted                                  -> (3)
[0612 01:40:40] 4.txt is created [size:8/mtime:0612 01:40:40]     -> (4)
[0612 01:40:42] 1.txt is deleted                                  -> (5) *순서가 바뀔 수 있음
[0612 01:40:42] 2.txt is deleted                                  -> (5) *순서가 바뀔 수 있음
[0612 01:40:42] 4.txt is deleted                                  -> (5) *순서가 바뀔 수 있음
```

20. 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. [5점]

< 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

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

    }

    exit(0);
}

void ssu_func(int loc_var, int loc_volatile, int loc_register) {
    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
```

※ 보너스 문제

다음 프로그램은 사용자 쓰레드를 사용하여 1부터 100까지 합을 구한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [10점]

< 조 건 >

1. 총 10개의 쓰레드를 사용(main() 함수 포함)
2. 각 쓰레드는 연속된 10개의 숫자를 더한 후 종료해야 함
3. 생성되는 쓰레드는 덧셈을 시작하는 숫자를 인자로 전달 받음
4. 전역변수(total_sum)에 각 쓰레드에서 구한 합을 더함
5. 가장 마지막으로 덧셈을 끝내는 쓰레드가 total_sum을 표준출력으로 출력
6. total_sum은 mutex를 사용하여 동기화 처리
7. 주어진 변수 외에 추가적인 변수 사용, 구조체 변경, 불필요한 헤더파일 삽입은 허용하지 않음

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

```
#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;
}

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

void add_sum(int loc_sum){
```

```
}

void finish_thread(void){
}
```

실행결과

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