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)	**21**
Number of Questions	**14**
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 3~6개의 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.
(3) 1~10번 문제에 대해 답을 파일에 작성할 경우 45개의 .txt 파일(1번~10번)이 /home/oslab/학번 디렉토리에 존재해야 함

3. 11번~14번 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.
(3) 11~14번 문제의 답을 파일에 작성할 경우 4개 .c 파일(11번~14번)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 -0.1 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. 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 sha1sum 학번.tar(ex %sha1sum 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. 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 1.2점, 총 54점]

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);

extern char **environ ;

int main(int argc, char *argv[],    char *envp[]    )
{
    ssu_printenv("Initially", &envp);
    putenv("TZ=PST8PDT")  ;
    ssu_printenv("After changing TZ", &envp);
    putenv("WARNING=Don't use envp after putenv()") ;
    ssu_printenv("After setting a new variable", &envp);
    printf("value of WARNING is %s\n",    getenv("WARNING")   );
    exit(0);
}

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

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

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

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

실행결과

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

(중략)

(27754643624) = 27754647723 -> _=./a.out
(27754643630) = 27754647742 -> OLDPWD=/home
(27754643634) =        0
---- After changing TZ ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
```

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

(중략)

(1004432154) = 27754647742 -> OLDPWD=/home
(1004432160) = 1001103372 -> TZ=PST8PDT
(1004432164) =        0
---- After setting a new variable ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
My environment variable are:
(1004432020) = 27754644205 -> SHELL=/bin/bash
(1004432024) = 27754644225 -> TERM=xterm-256color
(1004432030) = 27754644251 -> USER=root

(중략)

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

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>
#include <sys/wait.h>

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

    if ((child1 = fork()) == 0)
        execlp("date", "date", (char *)0) ;

    if ((child2 = fork()) == 0)
        execlp("who", "who", (char *)0) ;

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

    while (  (pid = wait(&status)) != -1 ) {
        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) {
    struct tms tms_start, tms_end;
    clock_t start, end;
    int status;

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

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

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

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

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

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

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

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

실행결과

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

command: sleep 5
  real:      5.00
  user:      0.00
  sys:       0.00
  child user:      0.00
  child sys:       0.00
```

```
normal termination, exit status = 0

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

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>
#include <pthread.h>
#include <string.h>

void *ssu_thread(void *arg);

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

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

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

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

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

실행결과

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

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>
#include <pthread.h>

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

int main(void)
{
    pthread_t tid1, tid2;

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

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

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

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

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

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

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

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

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

실행결과

```
root@localhost:/home/oslab# ./a.out
thread1의 리턴을 기다림
thread2: 8
thread1: 5
thread2: 7
thread1: 4
thread2: 6
thread1: 3
thread2: 5
thread1: 2
thread2: 4
thread1: 1
thread1 complete
thread2: 3
```

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

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER ;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER ;

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

int glo_val = 0;

int main(void)
{
    pthread_t tid1, tid2;

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

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

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

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

        pthread_mutex_unlock(&lock);

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

실행결과

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

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(  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
..
..
```

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

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

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

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

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

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

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

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

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

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

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

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

    exit(0);
}
```

실행결과

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

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>
```

```c
#include <signal.h>

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

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

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

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

    while (1);

    exit(0);
}
```

실행결과

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

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

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

double ssu_maketime(struct timeval *time);

void term_stat(int stat);

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

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

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

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

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

    exit(0);
}

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

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

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

실행결과

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

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

※ 주어진 조건에 맞게 프로그램을 완성하시오. [11-14, 총 46점]

11. 다음 프로그램은 SIGUSR1 시그널을 BLOCK 후 해당 시그널을 보냈을 때 pending되어 있는 시그널을 확인한다. 아래 조건과 실행결과를 보고 프로그램을 완성하시오. [10점]

< 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>
#include <signal.h>

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

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

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

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

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

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

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

```
}
```

실행결과

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

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. [10점]

< 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>
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond2 = PTHREAD_COND_INITIALIZER;

int length;
int buf[100];

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

    for(i = 1; i <= 300; i++){
        pthread_mutex_lock(&mutex);
        buf[length++] = i;

        if(i % 100 == 0)
            pthread_cond_signal(&cond2);
        if(length == 100)
            pthread_cond_wait(&cond1, &mutex);

        pthread_mutex_unlock(&mutex);
    }
}

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

    for(i = 1; i <= 300; i++){
        pthread_mutex_lock(&mutex);

        if(i % 100 == 0)
            pthread_cond_signal(&cond1);
```

```
        if(length == 0)
            pthread_cond_wait(&cond2, &mutex);

        sum += buf[--length];
        pthread_mutex_unlock(&mutex);
    }

    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);
    pthread_join(producer_tid, NULL);
    pthread_join(consumer_tid, NULL);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond1);
    pthread_cond_destroy(&cond2);

    exit(0);
}
```

실행결과

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

13. 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. [20점]

< 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)
{
    struct sigaction sig_act;

    sigemptyset(&sig_act.sa_mask);
    sig_act.sa_flags = 0;
```

```
    sig_act.sa_handler = ssu_signal_handler;
    sigaction(SIGALRM,&sig_act,NULL);

    alarm(1);

    while(1);

    exit(0);
}

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

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

실행 결과

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

14. 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. [6점]

< 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>
#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 14.c
```

```
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
```