설계과제 개요 : SoongSil Backup Program

Linux System Programming by Jiman Hong (jiman@ssu.ac.kr),
Sping 2018, School of CSE, Soongsil University

1. 개요

쉘(Shell)은 유닉스/리눅스 컴퓨팅 환경에서 기본적이고 중요한 부분으로, 명령 줄(Command line, cmd line)이라고도 부르며 사용자와 커널 사이의 인터페이스를 담당한다. 사용자의 명령을 해석하여 커널기능을 이용할 수 있게 해주고 자체적으로 프로그래밍 기능을 제공하여 반복적으로 수행해야 하는 명령어를 처리할 수 있다. 또, 각 사용자들의 환경을 저장하여 사용자에 맞는 환경을 제공한다.
운영체제에서는 프로세스 및 사용자 사이의 소통을 위한 몇 가지 도구를 제공하고 있다. 몇 가지 도구들 중의 하나인 시그널은 비동기적인 사건의 발생을 통지하기 위한 용도와, 사건을 동기화시키기 위한 용도로 사용될 수 있다. 예를 들어, SIGALRM은 사건을 동기화시키기 위해서, SIGKILL, SIGSEGV는 비동기적인 사건을 통지하기 위해서 사용한다.
디몬 프로세스는 멀티태스킹 운영체제에서 사용자가 직접제어하지 않고, 백그라운드 상에서 여러 작업을 수행하는 프로그램을 뜻한다. 시스템 로그를 남기는 syslogd처럼 보통 디몬 프로세스는 끝이 'd'로 끝난다. 디몬은 대개 프로세스 트리에서 init 바로 아래에 위치한다. 디몬이 되는 방법은 일반적으로 자식 프로세스를 생성하고 자신의 수행을 마치면서 init이 고아가 된 자식 프로세스를 자기 밑으로 데려가도록 하는 방식이다.

2. 목표

새로운 명령어를 시스템 함수를 사용하여 구현함으로써 쉘의 원리를 이해하고, 유닉스/리눅스 시스템에서 제공하는 여러 시스템 자료구조를 이용하여 프로그램을 작성함으로써 시스템 프로그래밍 설계 및 응용 능력을 향상시킨다.

- 아래의 쉘 명령어 구현

명령어	내용
ssu_backup	리눅스 시스템 상에서 사용자가 백업을 원하는 파일이나 디렉토리를 옵션에 따라 백업하고 백업된 파일을 다시 복구하는 프로그램

3. 팀 구성

개인별 프로젝트

4. 개발환경

가. OS : **Ubuntu 16.04**
나. Tools : vi(m), gcc, gdb

5. 보고서 제출 방법

가. 제출할 파일
  가) 보고서와 소스파일을 함께 압축하여 제출
    (1) 보고서 파일 : 워드(hwp 또는 MS-Word)로 작성
    (2) 소스코드 : ssu_backup.c, Makefile이 존재해야 함. 그 외 추가적으로 구현한 소스코드
    (3) 압축파일명 : #P3_학번_버전.zip (예. #P3_20180000_v1.0.zip)
나. 제출할 곳

가) http://oslab.ssu.ac.kr/main 접속 [2018 LSP] ⇒ [Homework] ⇒ [과제제출]

나) 게시물 제목은 파일이름과 동일함

다) 압축된 파일을 첨부하여 과제 제출

다. 제출 기한

가) 5월 29일(화) 오후 11시 59분 59초

6. 보고서 양식

보고서는 다음과 같은 양식으로 작성

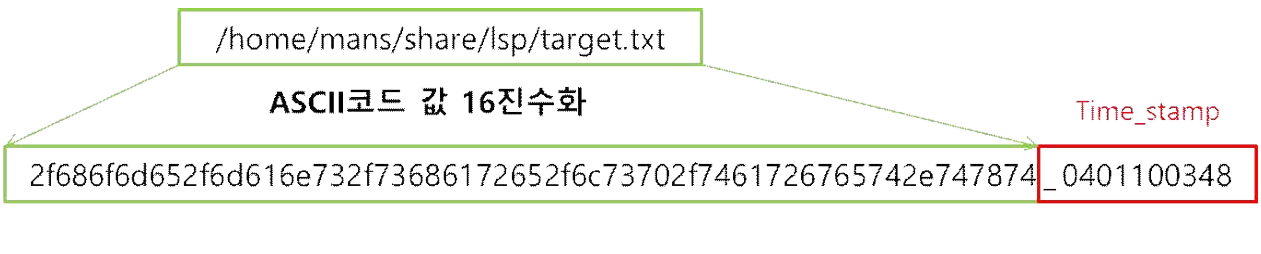
1. 과제 개요
2. 설계
3. 구현
   - 각 함수별 기능
4. 테스트 및 결과
   - 테스트 프로그램의 실행 결과를 분석
5. 소스코드(주석 포함)

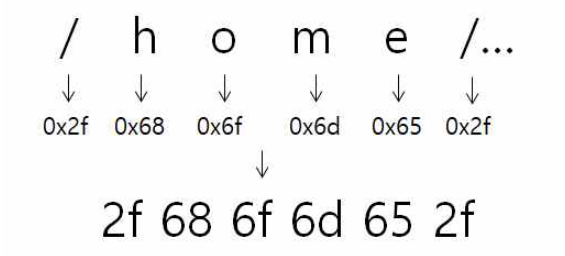
7. 설계 및 구현

가. ssu_backup 프로그램 수행 과정

1) ssu_backup은 실행시 백업대상이 될 파일이름과 백업주기를 함께 인자로 받음
2) ssu_backup은 기본적으로 **디몬 프로세스로 수행**되며, 수행되는 결과를 로그로 남김
3) 디몬 프로세스를 생성 할 때, 교재의 p.396-399를 참조해 디몬 프로세스 생성 규칙을 따를 것.
4) **정상적인 수행에 관한 로그는 별도의 로그파일**에 작성하며, 정상적인 수행이 불가능한(디몬 프로세스가 종료되어야 하는) **에러처리에 관한 로그는 syslog를 사용하여 작성**함
5) ssu_backup 프로그램이 존재하는 디렉토리에 백업용 디렉토리를 생성, 백업주기마다 백업대상을 백업용 디렉토리에 백업함
6) 백업파일 이름은 기존 백업 대상 파일의 절대경로를 16진수화하여, "_백업시간"(MMDDHHMMSS)을 붙인다.
   가) 백업대상 경로에는 **한글이 포함되면 안됨**(ASCII로 표현할 수 있는 알파벳 경로 사용)
   나) 백업파일이름이 길어 파일명 길이제한(default 255)을 넘을 경우 에러처리

예시 <백업대상 절대경로 16진수화>





실행 예시 <백업된 파일 이름>

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt 5
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 22994 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ cd backup/
mans@mans-virtual-machine:~/share/lsp/backup$ ls
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0401100343
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0401100348
mans@mans-virtual-machine:~/share/lsp/backup$
```

7) ssu_backup 프로그램은 **동시에 여러개를 수행할 수 없음**

8) ssu_backup 실행 시 **디몬 프로세스를 생성하기 전**에 다른 ssu_backup 프로세스가 실행중인지 확인한 후, 이미 실행중인 ssu_backup이 있다면 **사용자 정의 시그널인 SIGUSR1**을 기존 실행중이던 프로세스에 전달

9) SIGUSR1을 받은 ssu_backup 디몬 프로세스는 SIGUSR1에 관한 처리로 로그에 종료시점을 남기고 종료함

10) SIGUSR1으로 인한 종료는 정상적인 수행으로 간주, 별도의 로그파일에 **종료된 디몬 프로세스의 pid를 포함한 로그를 작성**함

11) 백업수행시마다 별도의 로그파일에 백업시간, 백업대상 파일명, 파일크기, mtime을 기록

12) 백업중 백업대상 파일에 수정이 일어났을 경우 **수정되었음을 알리는 로그(modified)**를 남겨야 하며, 삭제되었을 경우 **삭제에 관한 로그메시지(deleted)**를 남기고 디몬 프로세스를 종료함

실행 예시 <중복된 실행 시 기존 프로세스 종료, 로그에 pid 작성, 새로운 백업 디몬프로세스 실행>

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt 5
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 22183 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt 5
Send signal to ssu_backup process<22183>
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 22185 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ cat backup_log.txt
[0401 18:11:18] target.txt [size:12/mtime:0401 18:02:33]
[0401 18:11:23] target.txt [size:12/mtime:0401 18:02:33]
[0401 18:11:24] ssu_backup<pid:22183> exit
[0401 18:11:24] target.txt [size:12/mtime:0401 18:02:33]
mans@mans-virtual-machine:~/share/lsp$
```

실행 예시 <수정되지 않은 파일의 백업 로그, 수정된 파일의 백업 로그, 삭제된 파일의 백업 로그>

```
[0401 17:33:21] target.txt [size:14/mtime:0401 17:30:53]
[0401 17:33:26] target.txt is modified [size:30/mtime:0401 17:33:24]
[0401 17:33:31] target.txt is deleted
mans@mans-virtual-machine:~/share/lsp$
```

나. 프로그램 명세

1) ssu_backup <FILENAME> [PERIOD] [OPTION]

2) <FILENAME>

가) 백업 대상이 될 파일의 이름, 절대경로와 상대경로 모두 입력 가능해야함

나) 존재하지 않는 파일을 대상으로 할 경우 에러처리

다) 옵션이 주어지지 않았을 때는 일반파일만 가능, 일반파일이 아닌 입력에 대한 에러처리

3) <PERIOD>

가) 백업 수행 주기, 초단위이며 3초부터 10초까지로 범위를 제한

나) 범위를 벗어난(3초 미만 혹은 10초 초과) 주기를 입력받았을 시 에러처리

다) -c 옵션, -r옵션에서는 입력받지 않음

4) [OPTION]

가) -m : 백업 대상이 수정된 경우에만 백업을 수행함

(1) -m 옵션이 없는 기본 수행에서는 백업 대상의 수정여부와 상관 없이 백업을 수행

(2) -m 옵션을 적용했을 시에는 주기마다 백업 대상의 수정여부를 확인하여, **수정된 경우에만 백업**을 수행

실행 예시 <-m 옵션> (파일이 수정되었을 때만 백업이 되는것을 백업시간을 통해 확인)

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt 5 -m
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 23786 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ vi target.txt
mans@mans-virtual-machine:~/share/lsp$ vi target.txt
mans@mans-virtual-machine:~/share/lsp$ vi target.txt
mans@mans-virtual-machine:~/share/lsp$ cat backup_log.txt
[0403 14:35:03] target.txt is modified [size:14/mtime:0403 14:35:01]
[0403 14:35:08] target.txt is modified [size:5/mtime:0403 14:35:06]
[0403 14:36:03] target.txt is modified [size:26/mtime:0403 14:36:00]
mans@mans-virtual-machine:~/share/lsp$
```

나) -n [N] : 가장 최근 N개의 백업파일만 남기도록 백업을 수행함

(1) -n 옵션이 없는 기본 수행에서는 계속해서 백업파일을 생성함

(2) -n 옵션을 적용했을 시에는 **백업파일의 개수가 N개가 넘지 않도록 가장 오래된 백업파일들을 삭제**하고 가장 최근 N개의 백업파일만을 남겨놓아야 함

(3) -n 옵션 뒤에 개수를 입력하지 않거나 자연수가 아닌값이 입력됐을 시 에러처리

(4) 백업 수행이 일어나 오래된 백업파일이 삭제될 때, **삭제메시지와 함께 삭제된 백업파일의 원본파일명, 백업파일 크기, 백업된 시간**을 로그에 기록(백업대상파일 삭제에 관한 로그메시지와는 다름)

(5) -m 옵션과 함께 사용할 수 있어야 함

실행 예시 <-n 옵션> (5초주기, 3개의 백업파일을 남기게 수행, 중간에 파일을 수정한 후 로그파일 확인)

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt 5 -n 3
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 23821 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ vi target.txt
mans@mans-virtual-machine:~/share/lsp$ cat backup_log.txt
[0403 14:41:36] target.txt [size:14/mtime:0403 14:40:00]
[0403 14:41:41] target.txt is modified [size:25/mtime:0403 14:41:38]
[0403 14:41:46] target.txt [size:25/mtime:0403 14:41:38]
[0403 14:41:51] Delete backup [target.txt, size:14, btime:0403 14:41:36]
[0403 14:41:51] target.txt [size:25/mtime:0403 14:41:38]
[0403 14:41:56] Delete backup [target.txt, size:25, btime:0403 14:41:41]
[0403 14:41:56] target.txt [size:25/mtime:0403 14:41:38]
mans@mans-virtual-machine:~/share/lsp$
```

실행 예시 <-n 옵션> (3개의 백업파일을 유지하기 위해 가장 최근 파일만 남기고 오래된 파일은 삭제됨)

```
mans@mans-virtual-machine:~/share/lsp/backup$ ls
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144136
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144141
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144146
mans@mans-virtual-machine:~/share/lsp/backup$ ls
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144146
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144151
2f686f6d652f6d616e732f73686172652f6c73702f7461726765742e747874_0403144156
mans@mans-virtual-machine:~/share/lsp/backup$
```

다) -d : 디렉토리 안에 있는 모든 파일들에 대해 백업을 수행함

(1) 인자로 받은 <FILENAME>이 디렉토리가 아닐 경우 에러처리

(2) 백업대상 디렉토리에 있는 파일뿐만 아니라 **서브디렉토리에 있는 파일까지 모두 백업**

(3) -d 옵션으로 인해 백업이 수행 되어야 할 시, **각 파일에 대한 백업수행마다 스레드(pthread 라이브러리)를 생성해서 수행**

(4) 여러개의 스레드에서 백업이 수행되지만, 같은 로그파일에 로그를 작성하기 때문에 **로그파일에 대한 동기화가 이루어져야함**

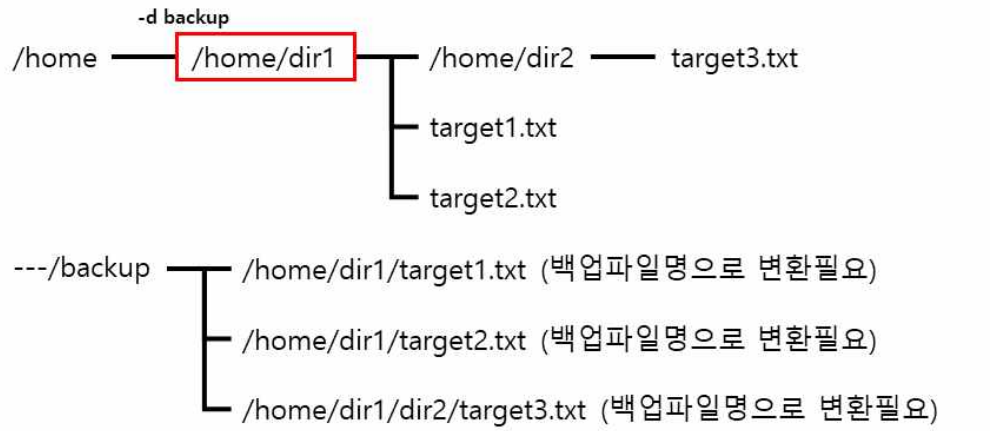
(5) -m 옵션, -n 옵션과 함께 사용할 수 있어야 하며, 이 경우, 각각의 파일마다 수정여부를 체크하고, 각각의 파일마다 -n 옵션으로 설정한 개수를 유지함

(6) 디렉토리에 관한 수정/삭제에 대해선 로그메시지를 남기지 않아도 됨

(7) -d 옵션의 경우 백업대상으로 주어진 디렉토리만 존재한다면, 디렉토리 내에 일반파일이 존재하지 않아도 디몬프로세스를 종료하지 않음

(8) 백업 디렉토리 내의 새로운 파일 생성도 생성된 파일에 대한 "수정"으로 간주함

예시 <-d 옵션 수행시 백업 대상 파일>



실행 예시 <-d 옵션>

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup targetdir 5 -d
Daemon process initialization.
mans@mans-virtual-machine:~/share/lsp$ process 8436 running as ssu_backup daemon.

mans@mans-virtual-machine:~/share/lsp$ cd targetdir/
mans@mans-virtual-machine:~/share/lsp/targetdir$ ls
subdir  target1.txt  target2.txt
mans@mans-virtual-machine:~/share/lsp/targetdir$ cd subdir/
mans@mans-virtual-machine:~/share/lsp/targetdir/subdir$ ls
target3.txt
mans@mans-virtual-machine:~/share/lsp/targetdir/subdir$
```

```
mans@mans-virtual-machine:~/share/lsp/backup$ ls
2f686f6d652f6d616e732f73686172652f6c73702f7461726765746469722f7375626469722f74617267657433 2e747874_0403150101
2f686f6d652f6d616e732f73686172652f6c73702f746172676574312e747874_0403150101
2f686f6d652f6d616e732f73686172652f6c73702f746172676574322e747874_0403150101
mans@mans-virtual-machine:~/share/lsp/backup$
```

라) -c : 인자로 들어온 파일이 백업되어있을 경우, 가장 최근 백업된 파일과 원본 파일 내용을 비교해서 출력함

(1) -c 옵션은 다른 옵션과 함께 사용할 수 없음

(2) -c 옵션에 대한 수행은 디몬프로세스를 생성하지 않으며, 이미 실행중인 ssu_backup 디몬프로세스를 종료시킨 후에 수행함

(3) 인자로 들어온 파일에 대한 백업파일이 존재하지 않는경우 터미널에 백업이 존재하지 않는다는 메시지를 출력함

(4) 백업파일이 존재하는 경우 가장 최근에 백업된 파일의 이름을 “원본명_백업시간”으로 출력한 후 diff 명령어를 사용하여 백업 파일과 현재 파일을 비교. man diff 참조

(5) 자식 프로세스를 생성하여 diff 명령어를 수행하도록 구현

실행 예시 <-c 옵션> (실행중인 디몬프로세스에 시그널을 보내고, diff를 사용해 백업파일과 현재파일 비교)

```
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt -c
Send signal to ssu_backup process<8539>
<Compare with backup file[target.txt_0405131011]>
2c2
<LSP P#3
---
>LSP P#3 Clear!!
mans@mans-virtual-machine:~/share/lsp$
```

마) -r : 인자로 들어온 파일이 백업되어있을 경우, 백업된 파일의 백업시간과 파일크기를 리스트로 출력한 후 사용자가 복구할 백업본을 선택할 수 있게함

(1) -r 옵션은 다른 옵션과 함께 사용할 수 없음

(2) -r 옵션에 대한 수행은 디몬프로세스를 생성하지 않으며, 이미 실행중인 ssu_backup 디몬프로세스를 종료시킨 후에 수행함

(3) 인자로 들어온 파일에 대한 백업파일이 존재하지 않는경우 터미널에 백업이 존재하지 않는다는 메시지를 출력함

(4) 백업파일이 존재하는 경우 백업파일의 원본명, 백업시간, 파일크기를 리스트로 출력함.

(5) 리스트는 사용자가 선택할 수 있게 넘버링이 되어 있어야하며, 복구할 파일을 선택하지 않을 경우를 위한 선택지도 필요함

(6) 사용자가 복구할 파일을 선택하면 원본파일을 백업파일로 교체한 후 복구된 파일의 내용을 출력함

실행 예시 <-r 옵션>

```
mans@mans-virtual-machine:~/share/lsp$ vi target.txt
mans@mans-virtual-machine:~/share/lsp$ cat target.txt
lsp P#3 -r before
It is original file.
mans@mans-virtual-machine:~/share/lsp$ ./ssu_backup target.txt -r
Send signal to ssu_backup process<8672>
<target.txt backup list>
0. exit
1. target.txt_0418204017 [size:14]
2. target.txt_0418204022 [size:5]
3. target.txt_0418204027 [size:20]
4. target.txt_0418204032 [size:20]
input : 4
Recovery backup file...
[target.txt]
lsp P#3 -r recovery
mans@mans-virtual-machine:~/share/lsp$ cat target.txt
lsp P#3 -r recovery
mans@mans-virtual-machine:~/share/lsp$
```

다. 세부 기능 및 기능별 요구 조건

1) 입력 요구조건

가) ssu_backup 프로그램은 **동시에 여러개를 수행할 수 없음**

나) ssu_backup 실행 시 디몬 프로세스를 생성하기 전에 기존 실행중이던 프로세스에 시그널을 전달해 종료해야함

다) 옵션에 따라 인자로 받는 <FILENAME>과 [PERIOD] 조건이 달라질 수 있어야함

라) -r, -c 옵션은 PERIOD가 입력되면 안됨

마) -d 옵션은 <FILENAME>에 반드시 디렉토리를 입력해야함

바) 중복으로 사용할 수 있는 옵션에 대해 처리해야함

사) -r, -c 옵션은 디몬 프로세스를 생성하지 않음

2) 출력 요구조건

가) **system() 함수의 사용을 금지함**

나) 정상적인 수행에 관해서는 별도로 하나의 로그파일을 생성해 로그를 작성함

다) 디몬 프로세스가 종료되어야 하는 에러에 대해서는 syslog를 사용하여 로그를 작성함

라) -r, -c 옵션 수행은 터미널에 직접 출력함

과제 구현에 필요한 함수(필수 사용 X)

1. fork(), getpid()

- 리눅스 시스템에서 새로운 프로세스를 생성할 때 사용하는 함수
- getpid()를 통해서 현재 실행 중인 프로세스의 PID를 얻을 수 있음

```
#include <unistd.h>
pid_t fork(void);
                                리턴값: 자식 프로세스의 경우 0, 부모 프로세스의 경우 자식의 프로세스 ID,
                                오류 시 -1을 리턴하고 errno가 설정됨
```

```
#include <unistd.h>
#include <sys/types.h>
pid_t getpid(void);
                                리턴값 : 호출한 프로세스의 프로세스 ID
```

fork_getpid_example.c

```
#include..<stdio.h>
#include..<stdlib.h>
#include..<unistd.h>
int..main(void)
{
    int pid;
    pid = fork();
    if (pid > 0) {
        printf("Parent : %d -> fork() -> : %d\n", getpid(), pid);
        sleep(1);
    }
    else if (pid == 0)
        printf("Child : %d\n", getpid());
    else if (pid == -1) {
        perror("fork error : ");
        exit(0);
    }
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./fork_getpid_example
Parent : 10220 -> fork() -> 10221
Child : 10221
```

2. syslog()

- 디몬 프로세스는 제어 터미널이 없기 때문에 문제가 발생했을 경우 syslog를 사용해 에러를 기록함

```
#include <syslog.h>
void openlog(const char *ident, int option, int facility);
void syslog(int priority, const char *format, ...);
```

ssu_syslog.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("lpd", LOG_PID, LOG_LPR);
        syslog(LOG_ERR, "open failed lpd %m");
        closelog();
        sleep(5);
    }
    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_syslog
daemon process initialization
process 12279 running as daemon
root@localhost:/home/oslab# ps -ejfc | grep ssu_syslog
root 12279 1 12279 12279 TS 19 21:46 ?00:00:00 ./ssu_syslog
root 12281 12038 12280 12007 TS 19 21:46 pts/19 00:00:00 grep --color=auto ssu_syslog
root@localhost:/home/oslab# tail -1 /var/log/syslog
Jan 13 21:46:36 oslab-ZBOX-ID91 lpd[12279]: open failed lpd Bad file descriptor
```

3. signal()

- 리눅스 시스템에서 시그널 처리를 설정
- 1. 기존의 방법을 따를지, 2. 시그널을 무시할지, 3. 지정된 함수를 실행하게 할 것인지를 등록
- 지정된 함수는 반드시 sighandler_t의 형태를 가진 함수이어야 함

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

인자	signum	시그널 번호를 가짐
	handler	SIG_DFL(디폴트 시그널 핸들러), SIG_IGN(무시) 혹은 지정된 함수를 가짐
반환값	sighandler_t	호출 이전의 signum에 대한 시그널 핸들러
	SIG_ERR	에러 발생시 반환하며, errno가 설정됨

signal_example.c

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

void (*old_handler)(int);

void sigint_handler(int signo) {
    printf("sigint caught!\n");
    signal(SIGINT, old_handler);
}

int main(void)
{
    old_handler = signal(SIGINT, sigint_handler);
    while(1) {
        printf("Hi\n");
        sleep(1);
    }

    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./signal_example
Hi
Hi^C
sigint caught!
Hi
Hi
^C
oslab@localhost:~$
```

4. sigaction()

- signal()보다 향상된 기능의 시그널 처리를 결정하는 함수
- sigaction()은 struct sigaction 구조체 값을 사용하므로 다양한 지정이 가능

```
struct sigaction {
    void (*sa_handler)(int);     // 시그널을 처리하기 위한 핸들러. SIG_DFL, SIG_IGN 또는 핸들러 함수
    void (*sa_sigaction)(int, siginfo_t *, void *); // 밑의 sa_flags가 SA_SIGINFO일때
                                                    // sa_handler대신에 동작하는 핸들러
    sigset_t sa_mask;              // 시그널을 처리하는 동안 블록할 시그널 집합의 마스크
    int sa_flags;                  // 아래 표 설명 참조
    void (*sa_restorer)(void);     // 사용 하지 않음
}
```

sa_flag 옵션	의미
SA_NOCLDSTOP	signum이 SIGCHILD일 경우, 자식 프로세스가 멈추었을 때, 부모 프로세스에 SIGCHILD가 전달되지 않음
SA_ONESHOT	시그널을 받으면 설정된 행도을 취하고 시스템 기본 설정인 SIG_DFL로 재설정됨
SA_RESTART	시그널 처리에 의해 방해 받은 시스템 호출은 시그널 처리가 끝나면 재시작
SA_NOMASK	시그널을 처리하는 동안에 전달되는 시그널은 블록되지 않음
SA_SIGINFO	이 옵션이 사용되면 sa_handler대신에 sa_sigaction이 동작되며, sa_handler보다 더 다양한 인수를 받을 수 있음. sa_sigaction이 받는 인수에는 시그널 번호, 시그널이 만들어진 이유, 시그널을 받는 프로세스의 정보가 있음

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

인자	signum	시그널 번호
	*act	새롭게 설정할 행동
	*oldact	함수 호출 전까지의 주어진 시그널 번호에 대한 행동 정보 반환
반환값	성공 시 0, 실패 시 -1이 반환 됨	

sigaction_example.c

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

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(int signo) {
    printf("sigint caught!\n");
    sigaction(SIGINT, &act_old, NULL);
}

int main(void)
{
    act_new.sa_handler = sigint_handler;
    sigemptyset(&act_new.sa_mask);
```

```
    sigaction(SIGINT, &act_new, &act_old);
    while(1) {
        printf("Hi\n");
        sleep(1);
    }
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./sigaction_example
Hi
Hi^C
sigint caught!
Hi
Hi
^C
oslab@localhost:~$
```

5. pthread_create()

- 새로운 스레드를 생성하는 라이브러리 함수
- pthread_create()함수가 성공해 새 스레드가 생성되면, 새로운 스레드의 ID가 tidp가 가리키는 주소에 저장됨
- pthread_attr_t인자는 여러 스레드 특성들을 설정하는데 쓰임, NULL을 지정하면 기본 특성을 가짐
- 세번째 인자로 주어진 start_rtn 함수포인터가 가리키는 함수에서 새로운 스레드가 시작됨
- 새로운 스레드가 시작되는 함수에 인자를 전달하고 싶다면 전달하고 싶은 정보를 담은 구조체 주소를 네번째 인자인 arg를 통해 전달할 수 있음

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr,
                   void *(*start_rtn)(void *), void *restrict arg);
                                                    리턴값: 성공 시 0, 실패 시 오류 번호
```

ssu_pthread_create.c

```
#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
```

8. 설계 구성 요소

가. 목표 설정

1) 주어진 요구 조건을 이해하고 명확한 설계 목표를 설정한다.

2) 설계의 목표 및 요구조건을 문서화한다.

나. 분석

1) 목표 설정에서 명시한 요구 조건을 분석하고 해결을 위한 기본 전략을 수립한다.

2) 문제 해결을 위한 배경 지식을 이론 강의에서 듣고 개별적으로 학생들이 다양한 경로를 통해 자료를 찾아 분석한다.

다. 합성 (구조 설계)

1) 분석 결과를 토대로 적절한 구조를 도출한다.

2) 도출된 구조를 토대로 모듈을 작성한다.

라. 제작 (구현)

1) 각 모듈의 입출력을 명세한다.

2) 구조와 모듈을 C 언어로 구현하고 이를 컴파일하여 실행 파일을 만든다.

마. 시험 및 평가(성능 평가)

1) 모듈의 입출력 명세에 따라, 다양한 입력 데이타를 작성하고 모듈을 테스트한다.

2) 작성된 실행 파일이 안정적으로 수행되는지 다양하게 테스트하고 이를 평가한다.

바. 결과 도출

1) 안정적으로 수행되는 최종 결과(Output)를 캡쳐하여 최종 결과물은 보고서에 반영한다.

9. 평가 준거(방법)

가. 평가 도구 1)에 대한 평가 준거

1) 소스 코드 분석 및 새로운 모듈의 설계가 제대로 이루어졌는가?

가) 설계 요구 사항을 제대로 분석하였는가?

나) 설계의 제약 조건을 제대로 반영하였는가?

다) 설계 방법이 적절한가?

2) 문서화

가) 소스 코드에 주석을 제대로 달았는가?

나) 설계 보고서가 잘 조직화되고 잘 쓰여졌는가?

나. 평가 도구 2)에 대한 평가 준거

1) 요구조건에 따라 올바르게 수행되는가?

10. 기타

가. 보고서 제출 마감은 제출일 자정까지

나. 지연 제출 시 감점

1) 1일 지연 시 마다 30% 감점

2) 3일 지연 후부터는 미제출 처리

다. 압축 오류, 파일 누락

1) 50% 감점 처리 (추후 확인)

라. copy 발견 시

1) F 처리

11. 점수 배점

가. 백업을 수행하는 디몬 프로세스 생성 20
나. 시그널을 통한 기존 디몬 프로세스 종료 15
다. 별도의 로그파일과 syslog를 통한 로그 작성 15
라. -d 옵션 구현 15
마. -r 옵션 구현 15
바. 백업대상 절대경로 16진수 아스키값 변환 7
사. 실행 시 입력 받은 인자 처리 7
아. -m 옵션 구현 2
자. -n 옵션 구현 2
차. -c 옵션 구현 2

※ 필수적으로 구현해야 할 기능 : 가, 나, 다, 라, 마, 바, 사