설계과제 개요 : SoongSil Shell #2

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.

cp (copy) 명령어는 유닉스 / 리눅스 쉘에서 한 파일을 어떤 장소에서 다른 장소로 다른 파일 시스템으로 옮길 때 사용하는 명령어이다. 원본 파일은 그대로 남아있고 새로운 파일이 기존 파일과 같은 이름으로 혹은 다른 이름으로 생기게 된다.

2. 목표

새로운 명령어를 시스템 함수를 사용하여 구현함으로써 쉘의 원리를 이해하고, 유닉스/리눅스 시스템에서 제공하는 여러 파일 속성과 디렉토리를 이용하여 프로그램을 작성함으로써 시스템 프로그래밍 설계 및 응용 능력을 향상시킨다. 또한, 파일 입/출력을 이해함으로써 표준 입출력 라이브러리 / 파일속성과 디렉토리 / 프로세스에 관한 대략적인 정보를 습득하고, 이해할 수 있는 능력을 향상시킨다.

- 아래의 쉘 명령어 구현

명령어	내용
ssu_cp	리눅스 시스템 상에 사용자로부터 원하는 파일이나 디렉토리를 옵션에 따라 지정한 경로에 복사하는 프로그램

3. 팀 구성

개인별 프로젝트

4. 개발환경

가. OS : **Ubuntu 16.04**

나. Tools : vi(m), gcc, gdb

5. 보고서 제출 방법

1) 제출할 파일

- 보고서와 소스파일을 함께 압축하여 제출
  - 보고서 파일 : 워드(hwp 또는 MS-word)로 작성
  - 소스코드 : ssu_cp.c, Makefile이 존재해야 함. 그 외 추가적으로 기능을 구분하여 만든 소스코드들.
  - 압축파일명 : #P2_학번_버전.zip (예. #P2_20180000_v1.0.zip)

2) 제출할 곳

- http://oslab.ssu.ac.kr/main 접속 [2018 LSP] ⇒ [Homework] ⇒ [과제제출]
- 게시물 제목은 파일이름과 동일함
- 압축된 파일을 첨부하여 과제 제출

3) 제출 기한

- 4월 24일(화) 오후 11시 59분 59초까지 과제 게시판으로 제출

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

1) ssu_cp

가) 프로그램 설명

- ssu_cp는 리눅스 시스템 상에 사용자로부터 원하는 파일이나 디렉토리를 옵션에 따라 지정한 경로에 복사하는 프로그램
- ssu_cp의 규칙은 Unix / Linux 의 cp 명령어 규칙을 완전히 따르지 않음
- ssu_cp는 파일/디렉토리의 이름 혹은 경로에 따라 parsing 후 실행
- 파일 / 디렉토리의 이름 길이한계는 <POSIX.1 표준의 파일 시스템의 제한>을 설정하는 매개 변수로 <limits.h> 헤더파일의 PATH_MAX 매크로를 사용하도록 함

나) 프로그램 명세

- **ssu_cp [OPTION] [SOURCE] [TARGET]** -기본 명령어
  - 기본 cp 명령어는 실행 예시 [0-0]과 같이 [SOURCE] 와 [TARGET]을 차례로 명시함
  - 실행 예시 [0-1]과 같이 경로명에 '\' (역슬래시) 가 올 경우 아랫줄에서 이어서 입력을 받음
  - 실행 예시 [0-2]과 같이 [SOURCE] 와 [TARGET]은 경로명으로 입력하여도 같은 동작을 해야함
  - 옵션이 있지 않으면 디렉터리 복사를 할 수 없음
  - 옵션들에 대해서 실행 예시 [0-2]과 같이 입/출력함
  - 옵션은 선택적인 사항임
  - 옵션은 중복되어서 나오면 오류처리 함
  - 옵션은 중복되어서 사용할 수 있되 -s 옵션의 경우 독립적으로 동작을 해야함
  - 복사 대상은 모든 파일형식과 디렉토리임
  - [TARGET]에 해당하는 파일이 이미 있으면 강제로 덮어씀
  - [SOURCE] 파일이 심볼릭 링크인 경우 가리키는 원본파일을 복사함
  - [SOURCE]와 [TARGET] 에 해당하는 파일/디렉터리 가 같으면 오류처리 함

실행 예시 [0-0] 예시

```
minijw@localhost:~/ssu_cp$ ./ssu_cp a1.txt a2.txt
target : a2.txt
src : a1.txt
```

실행 예시 [0-1] 예시

```
minijw@localhost:~/ssu_cp$ ./ssu_cp a1.txt ~/ssu_cp\
> /test/test.txt
target : /home/minijw/ssu_cp/test/test.txt
src : a1.txt
```

실행 예시 [0-2] 예시

```
minijw@localhost:~/ssu_cp/test$ ./ssu_cp a1.txt ~/ssu_cp/ssuaa1.txt
target : /home/minijw/ssu_cp/ssuaa1.txt
src : a1.txt
```

실행 예시 [0-3] 옵션 예시

```
minijw@localhost:~/ssu_cp$ ./ssu_cp -p a1.txt ssua1.txt
target : ssua1.txt
src : a1.txt
**********file info**********
파일 이름 : a1.txt
데이터의 마지막 읽은 시간 : 2018.03.30 17:54:43
데이터의 마지막 수정 시간 : 2018.03.30 17:47:23
데이터의 마지막 변경 시간 : 2018.03.30 17:47:23
OWNER : minijw
GROUP : minijw
file size : 0
```

- **오류처리** -오류
  - 모든 오류는 오류의 원인을 출력하고 해당 명령어의 사용법을 출력함
  - 예를 들어 [SOURCE] 파일이 없을 경우 "No Such file or directory" 를 출력
  - 다른 오류 문구를 써 넣어도 상관없으나 해당하는 오류의 원인을 정확하게 명명해야함. 아닐 경우 감점 요인이 될 수 있음
  - 오류 출력의 형식의 경우 특정 양식을 두진 않지만 실행 예시 [0-4]처럼 반드시 오류의 원인과 명령어의 사용법을 제시하여야 함

실행 예시 [0-4] 에러 예시

```
minijw@localhost:~/ssu_cp$ ./ssu_cp abc bbb
target : bbb
src : abc
ssu_cp: abc: No such file or directory
ssu_cp error
usage : in case of file
 cp [-i/n][-l][-p]        [source][target]
 or cp [-s][source][target]
 in case of directory cp [-i/n][-l][-p][-r][-d][N]
 minijw@localhost:~/ssu_cp$
```

- **[OPTION] : 실행할 프로그램의 옵션** -옵션 입력구현
  - 서로 다른 옵션은 중복 사용할 수 있음
  - 같은 옵션이 두 번이상 나온 경우 오류 처리를 함
  - 옵션의 사용 순서는 상관없음
  - 옵션은 대문자/소문자를 구분하지 않음
  - 실행 예시[1-0]에서 's option is on' 처럼 해당하는 옵션이 적용되었는지 출력하여야 함
  - 파일의 경우, [i/n][-l][-p] [SOURCE][TARGET] 형태로 동작함
  - 디렉터리 복사의 경우 [-i/n][-l][-p][-r][-d][N] [SOURCE][TARGET] 형태로 동작하며 -d 옵션의 경우 반드시 숫자 [N]이 바로 뒤에 와야함
  - [N]은 1~10까지의 숫자가 와야함
  - -s 옵션의 경우 [-s][SOURCE][TARGET] 형식으로 단독적으로 쓰이는 옵션

- **-s 옵션** -s옵션
  - 심볼릭 링크를 생성하여 복사를 수행함.
  - [SOURCE] 가 디렉토리의 경우 생략함.
  - 실행 예시[1-0]은 -s 옵션의 예시를 보여줌
  - ssua1.txt라는 a1.txt을 가리키는 링크파일을 확인할 수 있음
  - 실행 예시[1-1]처럼 심볼릭 링크를 이 옵션을 사용하여 복사할 경우 해당하는 심볼릭 링크의 링크파일이 생성됨

실행 예시[1-0] -s 옵션 예시

```
minijw@localhost:~/ssu_cp/test$ ./ssu_cp -s a1.txt ssua1.txt
 s option is on
target : ssua1.txt
src : a1.txt
minijw@localhost:~/ssu_cp/test$ ls -al
합계 56
drwxrwxr-x 2 minijw minijw  4096  3월 30 21:10 .
drwxrwxr-x 3 minijw minijw  4096  3월 30 21:09 ..
-rw-rw-r-- 1 minijw minijw     0  3월 30 21:10 a1.txt
-rw-rw-r-- 1 minijw minijw     7  3월 25 17:40 a2.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 30 21:10 ssu_cp
lrwxrwxrwx 1 minijw minijw     6  3월 30 21:10 ssua1.txt -> a1.txt
```

실행 예시[1-1] -s 옵션 예시

```
minijw@localhost:~/ssu_cp$ ls -al a5.txt
lrwxrwxrwx 1 minijw minijw 6  3월 28 17:17 a5.txt -> a1.txt
minijw@localhost:~/ssu_cp$ ./ssu_cp -s a5.txt a6.txt
 s option is on
target : a6.txt
src : a5.txt
minijw@localhost:~/ssu_cp$ ls -al a6.txt
lrwxrwxrwx 1 minijw minijw 6  3월 31 00:51 a6.txt -> a5.txt
```

- **-i 옵션** -i 옵션
  - 옵션이 없을 경우 기존의 파일이 있을 경우 강제로 덮어씀
  - SOURCE 파일의 이름이나 경로에 같은 이름의 기존파일을 덮어써야 할 경우, 덮어쓸 것인가 사용자에게 확인함
  - 어떤 파일을 덮어쓰기 할지 명시해야함
  - ex. ~$ 기존 [TARGET] 파일이 있습니다. 덮어쓸까요? (y/n) -> y
  - y 의 경우 덮어쓰기를 진행하고 그 외의 알파벳의 경우 진행하지 않음.

실행 예시[2] -i 옵션 예시

```
minijw@localhost:~/ssu_cp/test$ ls
a1.txt  a2.txt  ssu_cp  ssua1.txt
minijw@localhost:~/ssu_cp/test$ ./ssu_cp -i a1.txt a2.txt
 i option is on
target : a2.txt
src : a1.txt
a1.txt
overwrite a2.txt (y/n)? y
```

- -l 옵션 -l 옵션
  - [SOURCE] 파일의 소유주, 그룹 , 권한 , 시간정보를 보존하여 [TARGET]에 복사
  - 실행 예시 [3]에서 임의의 파일 권한을 변경한 후 -l 옵션을 사용하여 소유주, 그룹, 권한 시간정보를 같이 복사함을 확인

실행 예시 [3] -l 옵션

```
minijw@localhost:~/ssu_cp/test$ chmod ugo+rwx a1.txt
minijw@localhost:~/ssu_cp/test$ ls -al
합계 52
drwxrwxr-x 2 minijw minijw  4096  3월 30 21:10 .
drwxrwxr-x 3 minijw minijw  4096  3월 30 21:09 ..
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a1.txt
-rw-rw-r-- 1 minijw minijw     0  3월 30 23:38 a2.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 30 21:10 ssu_cp
lrwxrwxrwx 1 minijw minijw     6  3월 30 21:10 ssua1.txt -> a1.txt
minijw@localhost:~/ssu_cp/test$ ./ssu_cp -l a1.txt a3.txt
 l option is on
target : a3.txt
src : a1.txt
minijw@localhost:~/ssu_cp/test$ ls -al
합계 52
drwxrwxr-x 2 minijw minijw  4096  3월 30 23:46 .
drwxrwxr-x 3 minijw minijw  4096  3월 30 21:09 ..
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a1.txt
-rw-rw-r-- 1 minijw minijw     0  3월 30 23:38 a2.txt
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a3.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 30 21:10 ssu_cp
lrwxrwxrwx 1 minijw minijw     6  3월 30 21:10 ssua1.txt -> a1.txt
minijw@localhost:~/ssu_cp/test$ ./ssu_cp a1.txt a4.txt
target : a4.txt
src : a1.txt
 minijw@localhost:~/ssu_cp/test$ ls -al
합계 52
drwxrwxr-x 2 minijw minijw  4096  3월 30 23:46 .
drwxrwxr-x 3 minijw minijw  4096  3월 30 21:09 ..
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a1.txt
-rw-rw-r-- 1 minijw minijw     0  3월 30 23:38 a2.txt
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a3.txt
-rwxrwxr-x 1 minijw minijw     0  3월 30 23:46 a4.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 30 21:10 ssu_cp
lrwxrwxrwx 1 minijw minijw     6  3월 30 21:10 ssua1.txt -> a1.txt
```

○ -n 옵션 -n 옵션

- 기존 TARGET 파일이 해당 경로에 존재한다면 덮어쓰기를 수행하지 않음
- -i 옵션의 'y' 이외의 알파벳을 입력하였을 때 와 같은 옵션임

○ -p 옵션 -p 옵션

- 각 [SOURCE]파일 혹은 [SOURCE]디렉터리의 다음 내용들을 출력하고 cp를 수행함.
  - ✓ 1. 파일 이름
  - ✓ 2. 데이터의 마지막 읽은 시간
  - ✓ 3. 데이터의 마지막 수정 시간
  - ✓ 4. 데이터의 마지막 변경 시간
  - ✓ 5. OWNER
  - ✓ 6. GROUP
  - ✓ 7. file size

실행 예시[4] -p 옵션

```
minijw@localhost:~/ssu_cp/test$ ./ssu_cp -p a1.txt a5.txt
target : a5.txt
src : a1.txt
****************file info**********************
파일 이름 : a1.txt
데이터의 마지막 읽은 시간 : 2018.03.30 23:46:29
데이터의 마지막 수정 시간 : 2018.03.30 21:10:51
데이터의 마지막 변경 시간 : 2018.03.30 23:45:23
OWNER : minijw
GROUP : minijw
file size : 0
```

○ -r 옵션 -r 옵션

- [SOURCE] 디렉터리의 하위 디렉터리와 파일들을 가지고 있을 때 모두 동일하게 복사함
- 실행 예시[5] 에서처럼 하위 디렉터리와 해당하는 파일들을 모두 복사하여야 함
- 리눅스 cp 명령어의 -r 옵션과 유사
- [TARGET] 에 해당하는 디렉터리가 있을 경우 그곳에 복사함
- [TARGET] 에 해당하는 디렉터리가 없을 경우 디렉터리를 만들어서 복사함
- -d 옵션과 동시에 사용할 수 없음

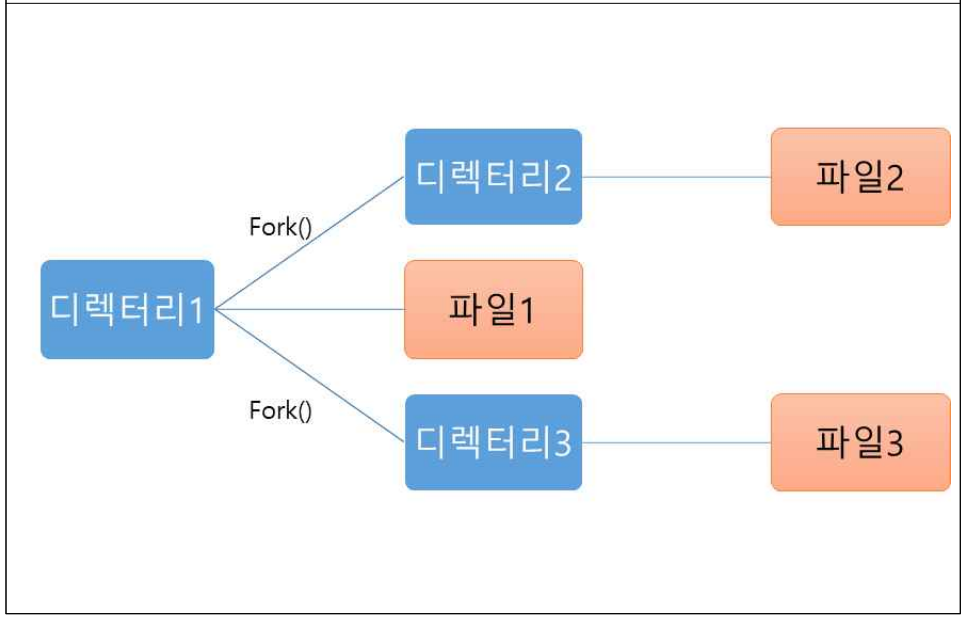
실행 예시[5] -r 옵션

```
minijw@localhost:~/ssu_cp$ ls -al test
합계 60
drwxrwxr-x 4 minijw minijw  4096  3월 31 00:35 .
drwxrwxr-x 3 minijw minijw  4096  3월 31 00:35 ..
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a1.txt
-rw-rw-r-- 1 minijw minijw     0  3월 30 23:38 a2.txt
-rwxrwxrwx 1 minijw minijw     0  3월 30 21:10 a3.txt
-rwxrwxr-x 1 minijw minijw     0  3월 30 23:46 a4.txt
-rwxrwxr-x 1 minijw minijw     0  3월 31 00:09 a5.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 30 21:10 ssu_cp
drwxrwxr-x 2 minijw minijw  4096  3월 31 00:33 test1
drwxrwxr-x 2 minijw minijw  4096  3월 31 00:33 test2
minijw@localhost:~/ssu_cp$ ./ssu_cp -r test test1
 r option is on
target : test1
src : test
minijw@localhost:~/ssu_cp$ ls -al test1
합계 60
drwxrwxr-x 4 minijw minijw  4096  3월 31 00:35 .
drwxrwxr-x 4 minijw minijw  4096  3월 31 00:35 ..
-rwxrwxr-x 1 minijw minijw     0  3월 31 00:35 a1.txt
-rw-rw-r-- 1 minijw minijw     0  3월 31 00:35 a2.txt
-rwxrwxr-x 1 minijw minijw     0  3월 31 00:35 a3.txt
-rwxrwxr-x 1 minijw minijw     0  3월 31 00:35 a4.txt
-rwxrwxr-x 1 minijw minijw     0  3월 31 00:35 a5.txt
-rwxrwxr-x 1 minijw minijw 44904  3월 31 00:35 ssu_cp
drwxrwxr-x 2 minijw minijw  4096  3월 31 00:35 test1
drwxrwxr-x 2 minijw minijw  4096  3월 31 00:35 test2
```

○ **-d [N] 옵션 -d 옵션**

- 현재 [SOURCE] 디렉터리를 [TARGET]으로 복사함
- 빠르게 복사하기 위하여 N 개의 프로세스로 나눠서 동시에 디렉터리를 복사함
- N은 1~10 의 정수임
- 그림[6] 에서처럼 하위 디렉토리가 2개인 경우, '-d 2' 의 옵션명령어를 치면 fork()를 두 번 호출하여 자식프로세스를 2개 생성한 후 각 자식 프로세스는 파일2, 파일3을 복사함.
- 부모프로세스는 파일1을 복사함.
- 만약 N 보다 하위 디렉터리 개수가 많을 경우 부모 디렉터리가 복사를 수행함
- 디렉터리 2에 만약 파일이 2개 이상있다면 자식프로세스1이 파일 두 개를 복사함
- 만약 하위 디렉터리 개수보다 N이 많을 경우 남는 디렉터리 개수는 아무것도 수행하지 않고 종료함
- 각 자식프로세스는 각 디렉터리를 복사하는 단위임
- 각 프로세스는 복사한 디렉터리와 해당 프로세스의 pid를 출력해야함
- 링크파일 복사는 고려하지 않음
- (참고) 해당 옵션은 실질적인 cp 동작의 속도를 획기적으로 줄여주는 것과 관련이 없음

그림[6] -d 옵션 실행 예시 도시화



다) 세부 기능 및 기능별 요구 조건

○ 입력 요구조건

- 옵션에 따라 받는 [SOURCE][TARGET]조건이 달라질 수 있어야 함
- 예를 들어 , r 이나 d 옵션은 디렉터리를 복사할 때 유효하므로 [SOURCE]는 반드시 디렉터리가 와야 함
- 파일 / 디렉토리의 이름 길이한계는 <POSIX.1 표준의 파일 시스템의 제한>을 설정하는 매개 변수로 <limits.h> 헤더파일의 PATH_MAX 매크로를 사용하도록 함
- N 의 크기는 1~10여야 함

- 중복적으로 옵션을 사용할 수 있음
- 예를 들어 , './ssu_cp -ip source target ' 인 경우 source 의 정보를 출력하는 p 옵션을 수행하면서 target은 I옵션을 따라 복사가 수행되어야 함

○ 출력 요구조건

- system() 함수의 사용을 일체 금지함
- 각 오류가 발생할 경우 반드시 오류에 대한 원인을 명시하여야 함
- 오류 발생시 해당 명령어 사용법을 반드시 출력해야함
- 각 옵션에 대해 명시한 출력 조건을 따라야 함
- [SOURCE] 와 [TARGET]은 명시해야함
- 명령어 수행 시 어떤 옵션이 수행되는지 명시해야함

※ 과제 구현에 필요한 함수들

1. fork(), getpid()

- 리눅스 시스템에서 새로운 프로세스를 생성할 때 사용하는 함수
- fork()에 의해 생성된 자식 프로세스는 PID를 부여받으며, 부모 프로세스의 PID를 PPID로 가짐
- 상속받는 항목은 PGID, SID 상속받으며, 파일 디스크립터 테이블 및 시그널을 상속받음
- 리눅스에서 fork()는 copy-on-write 매커니즘에 의해 구현되어 있음
- getpid()를 통해서 현재 실행 중인 프로세스의 PID를 얻을 수 있음

```
#include <unistd.h>
pid_t fork(void);
```

반환값	== 0	자식 프로세스에게 전달되는 값.
	> 0	부모 프로세스에게 전달되는 자식 프로세스의 PID 값.
	< 0	더 이상 프로세스를 만들 수 없을 경우 -1이 반환 되며 errno가 설정됨.

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
```

반환값	호출한 프로세스의 PID를 반환함.

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. wait계열 함수들

- 부모 프로세스가 자식 프로세스의 종료를 기다릴 경우 사용할 수 있는 함수

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
```

인자	*status	자식 프로세스가 종료된 상태를 저장할 주소
반환값	성공 시 종료된 자식 프로세스의 PID를 반환. 실패 시 –1이 반환 됨	

```
pid_t waitpid(pid_t pid, int *status, int options);
```

인자	pid	기다릴 대상이 될 프로세스를 지정할 수 있음. man 2 wait 참조
	*status	자식 프로세스가 종료된 상태를 저장할 주소
	options	waitpid()가 반환 될 때 추가적인 행동을 수행하도록 설정 가능
반환값	성공 시 상태가 변화한 자식 프로세스의 PID를 반환. 실패 시 –1이 반환 됨	

fork_wait_example.c

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

int main(void)
{
    int pid, status;

    pid = fork();
    if (pid < 0) {
        fprintf(stderr,"FORK ERROR :");
        exit(0);
    }
    else if (pid == 0) {
        int i;
        for (i = 0; i < 5; i++) {
            printf("Child : %d\n", i);
            sleep(2);
        }
        exit(1);
    }
    else {
        printf("I wait Child(%d)\n", pid);
        wait(&status);
        printf("Child is exit (%d)\n", status);
    }
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./fork_wait_example
I wait Child(10739)
Child : 0
Child : 1
Child : 2
Child : 3
Child : 4
Child is exit (256)
```

3. access()

- 사용자를 기준으로 주어지는 경로에 해당하는 파일의 권한을 확인할 수 있는 함수

```
#include <unistd.h>
int access(const char *pathname, int mode);
```

인자	*pathname	접근 가능 여부를 확인할 path
	mode	읽기, 쓰기, 실행 가능 여부와 같은 접근성을 체크하기 위한 모드 설정 값
반환값	성공 시 0이 반환, 실패 시 –1이 반환 되며 errno가 설정 됨	

access_example.c

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

int main (int argc, char* argv[])
{
    char* path = argv[1];
    int rval;

    rval = access (path, F_OK);    // File existence checking.
    if (rval == 0)
        printf ("%s exists\n", path);
    else {
        if (errno == ENOENT)
            printf ("%s does not exist\n", path);
        else if (errno == EACCES)
            printf ("%s is not accessible\n", path);
        exit(0);
    }
```

```
    rval = access (path, R_OK);    // Read accessibility checking.
    if (rval == 0)
        printf ("%s is readable\n", path);
    else
        printf ("%s is not readable (access denied)\n", path);

    rval = access (path, W_OK);    // Write accessibility checking.
    if (rval == 0)
        printf ("%s is writable\n", path);
    else if (errno == EACCES)
        printf ("%s is not writable (access denied)\n", path);
    else if (errno == EROFS)
        printf ("%s is not writable (read-only filesystem)\n", path);
    exit(0);
}
```

실행결과

```
oslab@localhost:~$ ./access_example /etc/passwd
/etc/passwd exists
/etc/passwd is readable
/etc/passwd is not writable (access denied)
```

4. scandir()

- dirp 디렉토리의 파일 및 디렉토리 목록을 filter 함수에서 정제하여 compar의 비교 조건으로 sorting
- 이 함수는 opendir(3), readdir(3), closedir(3)을 한번에 처리하고 filter와 sorting 기능을 갖는 함수

```
#include <dirent.h>
int scandir(const char *dirp, struct dirent ***namelist, int  (*filter) (const struct dirent *) , int
(*compar) (const struct dirent **, const struct dirent **))
```

인자	dirp	디렉토리에 대한 절대 또는 상대 path 의 디렉토리
	namelist	namelist 에 디렉토리에 있는 파일 및 디렉토리 목록이 저장됨 내부적으로 malloc(3) 또는 realloc(3)으로 할당되므로 사용후에는 반드시 free(3) 해야 함 다중 pointer 변수를 선언해서 사용하므로 건별 free(3) 후 namelist 자체도 free(3)를 해야함 예제 참조
	filter	namelist에 포함시킬 것인지 여부를 판단하는 함수에 대한 pointer 이 filter 함수의 return 값이 0 이면 namelist에 포함시키지 않고 0이 아니면 포함시킴 NULL이면 filter 없이 파일 및 디렉토리 전체가 namelist 에 저장됨 . 및 .. 디렉토리도 포함되어 있음
	compar	데이터를 sort 할 비교함수에 대한 포인터 내부적으로 qsort(3)를 사용하므로 이 함수를 통하여 sorting 함 만약 이름(struct dirent 구조체의 d_name)으로 sorting하려고 한다면 이미 구현된 alphasort(3) 함수를 사용할 수 있음 만약 이 compar를 NULL 로 설정하면 sorting 없이 출력됨
반환값	성공 시 0이상 반환하며 namelist에 저장된 struct dirent * 의 개수가 반환 , 실패 시 –1이 반환 되며 errno가 설정 됨	

scndir_example.c

```
#include <dirent.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
const char *path = ".";
int main(void)
{    struct dirent **namelist;
    int count;
    int i;
    if((count = scandir(path, &namelist , NULL, alphasort)) == -1)
    {
        fprintf(stderr, "%s Directory Scan Error : %s\n", path, strerror(errno));
        return 1;
    }
    for (i =0 ; i < count ; i++)
    {
        printf("%s\n", namelist[i]->d_name);
    }
    for (i = 0 ; i < count ; i++)
        free(namelist[i]);
    free(namelist);
    return 0;
}
```

실행결과

```
 oslab@localhost:~$ ./scandir
.
..
Makefile
a1.txt
```

8. 설계 구성 요소

1) 목표 설정
   - ○ 주어진 요구 조건을 이해하고 명확한 설계 목표를 설정한다.
   - ○ 설계의 목표 및 요구조건을 문서화한다.
2) 분석
   - ○ 목표 설정에서 명시한 요구 조건을 분석하고 해결을 위한 기본 전략을 수립한다.
   - ○ 문제 해결을 위한 배경 지식을 이론 강의에서 듣고 개별적으로 학생들이 다양한 경로를 통해 자료를 찾아 분석한다.
3) 합성 (구조 설계)
   - ○ 분석 결과를 토대로 적절한 구조를 도출한다.
   - ○ 도출된 구조를 토대로 모듈을 작성한다.
4) 제작 (구현)
   - ○ 각 모듈의 입출력을 명세한다.
   - ○ 구조와 모듈을 C 언어로 구현하고 이를 컴파일하여 실행 파일을 만든다.
5) 시험 및 평가(성능 평가)
   - ○ 모듈의 입출력 명세에 따라, 다양한 입력 데이타를 작성하고 모듈을 테스트한다.
   - ○ 작성된 실행 파일이 안정적으로 수행되는지 다양하게 테스트하고 이를 평가한다.

6) 결과 도출
   ○ 안정적으로 수행되는 최종 결과(Output)를 캡처하여 최종 결과물은 보고서에 반영한다.

9. 평가 도구

1) 설계 보고서 평가
2) 실행 평가 (컴파일 및 실행)

10. 평가 준거(방법)

1) 평가 도구 1)에 대한 평가 준거
   ○ 소스 코드 분석 및 새로운 모듈의 설계가 제대로 이루어졌는가?
      • 설계 요구 사항을 제대로 분석하였는가?
      • 설계의 제약 조건을 제대로 반영하였는가?
      • 설계 방법이 적절한가?
   ○ 문서화
      • 소스 코드에 주석을 제대로 달았는가?
      • 설계 보고서가 잘 조직화되고 잘 쓰여졌는가?
2) 평가 도구 2)에 대한 평가 준거
   ○ 요구조건에 따라 올바르게 수행되는가?

3) 기타
4) 보고서 제출 마감은 제출일 자정까지
5) 지연 제출 시 감점
   ○ 1일 지연 시 마다 30% 감점
   ○ 3일 지연 후부터는 미제출 처리
6) 압축 오류, 파일 누락
   ○ 50% 감점 처리 (추후 확인)
7) copy 발견시
   ○ F 처리

11. 점수 배점

▶ ssu_cp 80
1. ssu_cp 기본 명령어 12
2. 오류처리 출력 4
3. 실행 시 입력 받은 인자 처리 구현 8
4. -s 옵션 구현 8
5. -i 옵션 구현 7
6. -l 옵션 구현 8
7. -n 옵션 구현 3
8. -p 옵션 구현 7
9. -r 옵션 구현 10
10. -d 옵션 구현 10
11. Makefile 사용 3

※ 필수적으로 구현해야 할 기능 : 1 ,2, 3, 8 ,9, 11