

설계과제 1 개요 : SSU-Cleanup

○ 개요

- 리눅스 시스템 상에서 사용자가 정리를 원하는 디렉토리를 확장자를 기준으로 정리하여 관리하는 프로그램

○ 목표

- 새로운 명령어를 시스템 함수를 사용하여 구현함으로써 쉘의 원리를 이해하고, 유닉스/리눅스 시스템에서 제공하는 여러 시스템 자료구조와 시스템 콜 및 라이브러리를 이용하여 프로그램을 작성함으로써 표준 입출력 및 파일 입출력에 대해 적응하고 이를 응용하여 디렉토리 구조를 링크드 리스트로 구현하며 시스템 프로그래밍 설계 및 응용 능력을 향상킴

- 개인별 프로젝트

○ 보고서 제출 방법 (강의계획서와 동일. #2 설계과제부터는 보고서 제출 방법은 명세에서 삭제 예정. 강의계획서 참고)

- 설계과제는 “P1_학번_버전번호.zip” (예. P1_20250000_V1.zip 또는 P1_2025000_V1.0.zip) 형태로 압축하여 classroom.google.com에 제출해야 함.
- 압축 파일 내 (1)구현보고서인 P설계과제번호.hwp (예. P2.hwp) (2)헤더파일 및 소스코드 등 해당 디렉토리에서 컴파일과 실행이 가능하도록 모든 파일(makefile, obj, *.c, *.h 등 컴파일하고 실행하기 위한 파일)을 포함시켜야 함. 단, 특정한 디렉토리에서 실행해야 할 경우는 디렉토리는 포함하지 않아 됨.
- 구현보고서인 “P설계과제번호.hwp”에는 1. 과제개요(명세에서 주어진 개요를 그대로 쓰면 안됨. 자기가 구현한 내용 요약) 2. 기능(구현한 기능 요약), 3. 상세설계(함수 및 모듈 구성, 순서도, 구현한 함수 프로토타입 등), 4. 실행 결과 (구현한 모든 기능 및 실행 결과 캡처)를 반드시 포함시켜야 함.
- 제출한 압축 파일을 풀었을 때 해당 디렉토리에서 컴파일 및 실행이 되어야 함 (모든 프로그램은 해당 디렉토리에서 컴파일 및 실행이 되어야 함(특정한 디렉토리에서 실행해야 할 경우는 제외). 해당 디렉토리에서 컴파일이나 실행되지 않을 경우, 설계과제 제출 방법(파일명, 컴파일 시에 포함되어야 할 파일에 포함되어야 할 파일 등)을 따르지 않는 경우 해당 과제 성적은 10점 감점.)
- 과제를 기한 내 새로 제출할 경우 기존 것은 삭제하지 않고 P설계과제번호_학번_V2.zip 형태로 버전번호를 증가시키면 됨. 버전 번호는 대문자 V와 함께 integer를 1부터 incremental 증가시키면서 부여하면 됨. (예. V1, V2, V3, ...). 첫 제출 시에도 버전 번호 V1(또는 V1.0)를 붙여야 하며, 두 번째는 V2(또는 V2.0)를 붙여 제출하면 됨. 단, P설계과제번호_학번_V1.zip 압축 파일 내 구현보고서에는 버전번호는 붙이지 말 것.

○ 배점 기준 (강의계획서와 동일. #2 설계과제부터는 배점 기준도 명세에서 삭제 예정. 강의계획서 참고)

- 구현 보고서 P설계과제번호.hwp (15점) : 개요 1점, 기능 1점, 상세설계 10점, 실행 결과 3점
- 소스코드 (85점) : 소스코드 주석 5점, 실행 여부 80점 (설계 요구에 따르지 않고 설계된 경우 소스코드 주석 및 실행 여부는 0점 부여. 설계 요구에 따라 설계된 경우 기능 미구현 부분을 설계명세서의 100점 기준에서 해당 기능 감점 후 이를 80점으로 환산)
- 각 설계과제의 완성 유무는 제출 여부로 판단하는 것이 아니라 주어진 과제에서 명시된 “필수구현”의 구현 여부로 판단.
 - ✓ 예. 특정 과제의 필수구현 중 일부만 구현했을 경우 해당 점수는 부여하나 과제는 미구현으로 판단하고 본 교과목 이수조건인 설계과제 최소 구현 개수 1개에 포함시키지 않음
- 구현 보고서 및 소스코드 배점
 - ✓ 보고서는 다음과 같은 양식으로 작성(강의계획서 FAQ 참고) (15점)

1. 과제 개요 (1점) // 명세에 주어진 개요를 더 상세하게 작성
2. 구현 기능 (1점) // 함수 프로토타입 반드시 포함
3. 상세 설계 (10점) // 함수 기능별 흐름도(순서도) 반드시 포함
4. 실행 결과 (3점) // 테스트 프로그램의 실행 결과 캡처 및 분석
5. (텍스트 형태의) 소스코드 // 주석 포함, 캡처해서 넣지 말고, 텍스트 형태로.

- ✓ 소스코드 및 실행 여부 (85점) // 주석 (5점), 실행 여부 (80점)

○ 제출 기한 : 3월 25일(화) 오후 11시 59분 59초

○ ssu_cleanup 프로그램 기본 사항

- 리눅스 상에서 파일 경로의 최대 크기는 4,096 바이트이며, 파일 이름의 최대 크기는 255 바이트임
- 프로그램 실행 시 내장명령어(tree, arrange, help, exit)에 따라 해당 기능 실행
- ssu_cleanup 프로그램을 통해 정리 가능한 경로들은 사용자 홈 디렉토리(/home/사용자아이디) 하위 경로여야 함
- 본 명세에서 특별히 설명하거나 지정하지 않은 것들은 모두 학생들이 스스로 판단하여 구현하면 됨.
- 프로그램 전체에서 system() 절대 사용 불가. 사용 시 본 과제는 0점 처리

○ 설계 및 구현

1. ssu_cleanup

1) Usage : ssu_cleanup

- ssu_cleanup 프로그램의 내장명령어는 디렉토리 트리 구조 출력(tree), 디렉토리 정리(arrange), 내장명령어 설명(help), 종료(exit)로 이루어져 있음
- 각 내장명령어에 따라 필요한 인자 및 옵션 또한 프로그램 인자를 통해 입력받음

2) 실행 결과

- ssu_cleanup 실행 시 프롬프트 “학번)” (예. 20230000) 출력
- ssu_cleanup의 실행 결과는 각 명령어의 실행 결과를 출력함

예제 1-1. ssu_cleanup 실행 결과
% ./ssu_cleanup 20230000)
예제 1-2. ssu_cleanup 실행 결과(help를 통해 명령어 목록을 출력한 모습)
20230000) help Usage: > tree <DIR_PATH> [OPTION]... <none> : Display the directory structure recursively if <DIR_PATH> is a directory -s : Display the directory structure recursively if <DIR_PATH> is a directory, including the size of each file -p : Display the directory structure recursively if <DIR_PATH> is a directory, including the permissions of each directory and file > arrange <DIR_PATH> [OPTION]... <none> : Arrange the directory if <DIR_PATH> is a directory -d <output_path> : Specify the output directory <output_path> where <DIR_PATH> will be arranged if <DIR_PATH> is a directory -t <seconds> : Only arrange files that were modified more than <seconds> seconds ago -x <exclude_path1, exclude_path2, ...> : Arrange the directory if <DIR_PATH> is a directory except for the files inside <exclude_path> directory -e <extension1, extension2, ...> : Arrange the directory with the specified extension <extension1, extension2, ...> > help [COMMAND] > exit

3) 예외 처리(미 구현 시 아래 점수만큼 감점, 필수구현은 아님)

- 프롬프트 상에서 지정한 내장명령어 외 기타 명령어 입력 시 help 명령어의 결과를 출력 후 프롬프트 재출력(3점)
- 프롬프트 상에서 엔터만 입력 시 프롬프트 재출력(2점)

2. 내장명령어 1. tree

1) Usage : tree <DIR_PATH> [OPTION]...

- 경로(PATH)를 입력받아 디렉토리와 파일들을 트리 구조로 출력함

2) 인자 설명

- 첫 번째 인자 <DIR_PATH>는 출력할 디렉토리의 경로이며 상대경로와 절대경로 모두 입력 가능해야 함
- 두 번째 인자 [OPTION]은 ‘-s’, ‘-p’가 있으며 생략할 수 있음(‘-s’, ‘-p’ 옵션은 아래 설명 확인). 동시에 사용할 수도 있음

3) 실행 결과

- 첫 번째 인자 <DIR_PATH>을 계층적으로 출력한 뒤, 디렉토리와 파일의 수를 출력함
- 디렉토리의 이름 끝에는 슬래시(/)를 붙여서 출력함
- 리눅스 및 윈도우의 tree 명령어와 매우 유사한 구조를 가짐
- 특정 디렉토리에서 자식의 이름 순서는 사전 순으로 정렬되어야 함
- 파일 혹은 디렉토리의 깊이가 <DIR_PATH>와 차이 나는 만큼 아래의 문자를 통해 시각적으로 출력하며, 아래의 문자들 사이에는 공백문자 ' ' 3개를 출력함
- ✓ | — : 출력할 파일 혹은 디렉토리의 가장 가까운 조상인 부모 디렉토리의 마지막 자식이 아닐 때 이름 왼쪽에 출력
- ✓ | — : 이 문자가 출력되어야 하는 깊이를 기준으로 부모가 되는 디렉토리가 마지막 원소의 자식이 아닌 경우 이 문자를 출력, 부모 디렉토리가 마지막 원소일 때는 이 문자를 공백문자 ' ' 으로 대체
- ✓ | — : 출력할 파일 혹은 디렉토리의 가장 가까운 조상인 부모 디렉토리의 마지막 자식일 때 파일의 이름 왼쪽에 출력

그림 1. /home/oslab/test1 디렉토리 트리 구조

```
/home/oslab/test1
|— a/
|   |— b/
|   |— c.txt
|   |— d/
|— e.txt
|— f/
|   |— g/
|   |   |— h.txt
|   |— i.txt
```

예제 1-3. tree 내장명령어 실행(그림 1의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/test1” 일 때)

```
% ./ssu_cleanup
20230000> tree .
```

```
.
|— a/
|   |— b/
|   |— c.txt
|   |— d/
|— e.txt
|— f/
|   |— g/
|   |   |— h.txt
|   |— i.txt
```

6 directories, 4 files

```
20230000> tree /home/oslab/test1/f
```

```
/home/oslab/test1/f
|— g/
|   |— h.txt
|— i.txt
```

2 directories, 2 files

- ‘-s’ 옵션 입력시에는 파일 또는 디렉토리의 크기와 함께 출력

예제 1-4. tree 내장명령어의 -s 옵션(그림 1의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/test1” 일 때)

```
% ./ssu_cleanup
20230000> tree . -s
[4096] .
|— [4096] a/
|   |— [4096] b/
|   |— [25] c.txt
|   |— [4096] d/
```

```
├─ [14] e.txt
└─ [4096] f/
    ├─ [4096] g/
    │   └─ [13] h.txt
    └─ [10] i.txt
```

6 directories, 4 files

20230000> tree /home/oslab/test1/f -s

[4096] /home/oslab/test1/f

```
├─ [4096] g/
│   └─ [13] h.txt
└─ [10] i.txt
```

2 directories, 2 files

- ‘-p’ 옵션 입력시에는 파일 또는 디렉토리의 권한과 함께 출력

예제 1-5. tree 내장명령어의 -p 옵션(그림 1의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/test1/” 일 때)

% ./ssu_cleanup

20230000> tree . -p

```
[drwxr-xr-x] .
├─ [drwxr-xr-x] a/
│   ├─ [drwxr-xr-x] b/
│   └─ [-rw-r--r--] c.txt
└─ [drwxr-xr-x] d/
    ├─ [-rw-r--r--] e.txt
    └─ [drwxr-xr-x] f/
        ├─ [drwxr-xr-x] g/
        │   └─ [-rw-r--r--] h.txt
        └─ [-rw-r--r--] i.txt
```

6 directories, 4 files

20230000> tree /home/oslab/test1/f -p

```
[drwxr-xr-x] /home/oslab/test1/f
├─ [drwxr-xr-x] g/
│   └─ [-rw-r--r--] h.txt
└─ [-rw-r--r--] i.txt
```

2 directories, 2 files

- ‘-sp’ 옵션 입력시에는 파일 또는 디렉토리의 권한, 크기와 함께 출력

예제 1-6. tree 내장명령어의 -sp 옵션(그림 1의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/test1/” 일 때)

% ./ssu_cleanup

20230000> tree . -sp

```
[drwxr-xr-x 4096] .
├─ [drwxr-xr-x 4096] a/
│   ├─ [drwxr-xr-x 4096] b/
│   └─ [-rw-r--r-- 25] c.txt
└─ [drwxr-xr-x 4096] d/
    ├─ [-rw-r--r-- 14] e.txt
    └─ [drwxr-xr-x 4096] f/
        ├─ [drwxr-xr-x 4096] g/
        │   └─ [-rw-r--r-- 13] h.txt
        └─ [-rw-r--r-- 10] i.txt
```

6 directories, 4 files

20230000> tree /home/oslab/test1/f -sp

```
[drwxr-xr-x 4096] /home/oslab/test1/f
├─ [drwxr-xr-x 4096] g/
│   └─ [-rw-r--r-- 13] h.txt
└─ [-rw-r--r-- 10] i.txt
```

2 directories, 2 files

4) 예외 처리(미 구현 시 아래 점수만큼 감점, 필수구현은 아님)

- 첫 번째 인자로 올바르지 않은 경로(존재하지 않는 디렉토리) 입력 시 Usage 출력 후 프롬프트 재출력(2점)
- 첫 번째 인자로 입력받은 경로(절대경로 또는 상대경로)가 길이 제한(4096 Byte)을 넘어선 경우, 그리고 절대경로 내 한 파일 및 디렉토리 이름의 길이 제한(256 Byte)을 넘어선 경우 에러 처리. 또한 해당 경로가 디렉토리가 아니거나 입력받은 경로(절대 경로 또는 상대경로)가 사용자의 홈 디렉토리(\$HOME 또는 ~)를 벗어나는 경우 에러 처리 후 프롬프트 재출력(2점)
- 첫 번째 인자로 입력받은 경로(절대 경로)가 사용자의 홈 디렉토리(\$HOME, ~)를 벗어나는 경우 “<입력받은 경로> is outside the home directory” 표준 출력 후 프롬프트 재출력(2점)
- 두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력(2점)

3. 내장명령어 2. arrange

1) Usage : arrange <DIR_PATH> [OPTION] ...

- 정리할 디렉토리 경로(DIR_PATH)를 입력받아 해당 디렉토리 내의 파일들을 확장자 기준으로 분류하여 결과물을 현재 작업 디렉토리 하위의 <DIR_PATH>_arranged에 저장

2) 인자 설명

- 첫 번째 인자 <DIR_PATH>는 정리할 디렉토리의 경로이며 상대경로와 절대경로 모두 입력 가능해야 함
- 두 번째 인자 [OPTION]은 ‘-d’, ‘-t’, ‘-x’, ‘-e’, 가 있으며 모두 생략 가능함

3) 실행 결과

- 현재 작업 디렉토리에 <DIR_PATH>_arranged 디렉토리가 존재하지 않을 경우 생성됨
- arranged 디렉토리 내에 확장자별 디렉토리(txt, c, cpp 등)가 생성되며, 해당 확장자를 가진 파일들이 각 디렉토리로 복사됨
- 디렉토리 정리 후 프롬프트 재출력
- 정리할 디렉토리 내에서는 파일명이 중복되는 경우, 파일 선택, diff, vi, 정리 안함 중 하나를 수행할 수 있음

그림 2. /home/oslab/test2/ 디렉토리 트리 구조의 예	그림 3. /home/oslab/test3 디렉토리 트리 구조의 예	그림 4. /home/oslab/test4 디렉토리 트리 구조의 예
<pre>/home/oslab/test2 ├─ a/ │ ├── b/ │ │ └─ c.c │ └─ d/ ├─ e.c └─ f/ ├── g/ │ └─ h.c └─ i.c</pre>	<pre>/home/oslab/test3 ├─ a/ │ ├── b/ │ │ └─ c.txt │ └─ d/ ├─ e.txt └─ f/ ├── g/ │ └─ h.c └─ i.c</pre>	<pre>/home/oslab/test4 ├─ a/ │ ├── abc.txt │ └─ hello.txt └─ b/ └─ hello.txt</pre>

예제 2-1. arrange 내장명령어 실행(그림 2, 3, 4의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/” 일 때)

```
% ./ssu_cleanup
20230000> arrange
Usage : arrange <DIR_PATH> [OPTION]

20230000> arrange hello
hello does not exist

20230000> arrange test2/e.c
test2/e.c is not a directory

20230000> arrange test2
```

test2 arranged

20230000> tree test2_arranged

```
test2_arranged
├── c/
│   ├── c.c
│   ├── e.c
│   ├── h.c
│   └── i.c
```

2 directories, 4 files

20230000> arrange test3

test3 arranged

20230000> tree test3_arranged

```
test3_arranged
├── c/
│   ├── h.c
│   └── i.c
└── txt/
    ├── c.txt
    └── e.txt
```

3 directories, 4 files

20230000> arrange test4

1. test4/a/hello.txt
2. test4/b/hello.txt

choose an option:

0. select [num]
1. diff [num1] [num2]
2. vi [num]
3. do not select

20230000> diff 1 2

diff test4/a/hello.txt test4/b/hello.txt 수행 결과 출력

choose an option:

0. select [num]
 1. diff [num1] [num2]
 2. vi [num]
 3. do not select
- 20230000> do not select

test4 arranged

- '-d' 옵션 입력 시에는 <DIR_PATH> 디렉토리를 정리하여 <output_path>에 저장함

예제 2-2. arrange 내장명령어의 -d 옵션(그림 3의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/” 일 때)

20230000> arrange test2 -d hello

test2 arranged

```
20230000> tree hello
hello
├─ c/
│   ├── c.c
│   ├── e.c
│   ├── h.c
│   └─ i.c
```

2 directories, 4 files

- ‘-t’ 옵션 입력 시에는 <DIR_PATH> 내에 있는 파일들 중 마지막으로 수정한 시간이 <seconds> 초 이내인 파일들만 정리함

예제 2-3. arrange 내장명령어의 -t 옵션(그림 6의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/” 일 때)

```
20230000> arrange test2 -t 10 // test2/a/c.c와 test2/e.c 파일은 수정한 지 10초가 지났다고 가정
test2 arranged
```

```
20230000> tree test2_arranged
test2_arranged
├─ c/
│   └─ h.c
│   └─ i.c
```

2 directories, 2 files

- ‘-x’ 옵션 입력 시에는 <exclude_path1, exclude_path2, ...> 디렉토리의 하위 파일들을 제외하고 <DIR_PATH> 디렉토리를 정리함

예제 2-4. arrange 내장명령어의 -x 옵션(그림 2의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/” 일 때)

```
20230000> arrange test2 -x f
test2 arranged
```

```
20230000> tree test2_arranged
test2_arranged
├─ c/
│   ├── c.c
│   └─ e.c
```

2 directories, 2 files

- ‘-e’ 옵션 입력 시에는 <extension1, extension2, ...> 내에 있는 확장자를 가진 파일들만 정리함

예제 2-5. arrange 내장명령어의 -e 옵션(그림 3의 상황에서 현재 작업 디렉토리(pwd)가 “/home/oslab/” 일 때)

```
20230000> arrange test3 -e txt
test3 arranged
```

```
20230000> tree test3_arranged
test3_arranged
├─ txt/
│   ├── c.txt
│   └─ e.txt
```

2 directories, 2 files

- 예외 처리(미 구현 시 아래 점수만큼 감점, 필수구현은 아님)

- 첫 번째 인자 입력이 없는 경우 Usage 출력 후 프롬프트 재출력(3점)
- 첫 번째 인자로 입력받은 경로(절대경로 또는 상대경로)가 길이 제한(4096 Byte)을 넘어서는 경우, 그리고 절대경로 내 한 파일 및 디렉토리 이름의 길이 제한(256 Byte)을 넘어서는 경우 에러 처리. 또한 해당 경로가 디렉토리가 아니거나 입력받은 경로(절대 경로 또는 상대경로)가 사용자의 홈 디렉토리(\$HOME 또는 ~)를 벗어나는 경우 에러 처리 후 프롬프트 재출력(2점)
- 두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력(5점)

4. 내장명령어 3. help

- 1) Usage : help
 - 프로그램 내장명령어에 대한 설명(Usage) 출력
- 2) 실행 결과
 - 프로그램 내장명령어 help 실행

<p>예 3-1. help 내장명령어 실행</p> <pre> 20230000> help Usage: > tree <DIR_PATH> [OPTION]... <none> : Display the directory structure recursively if <DIR_PATH> is a directory -s : Display the directory structure recursively if <DIR_PATH> is a directory, including the size of each file -p : Display the directory structure recursively if <DIR_PATH> is a directory, including the permissions of each directory and file > arrange <DIR_PATH> [OPTION]... <none> : Arrange the directory if <DIR_PATH> is a directory -d <output_path> : Specify the output directory <output_path> where <DIR_PATH> will be arranged if <DIR_PATH> is a directory -t <seconds> : Only arrange files that were modified more than <seconds> seconds ago -x <exclude_path1, exclude_path2, ...> : Arrange the directory if <DIR_PATH> is a directory except for the files inside <exclude_path> directory -e <extension1, extension2, ...> : Arrange the directory with the specified extension <extension1, extension2, ...> > help [COMMAND] > exit </pre>

5. 내장명령어 4. exit

- 1) Usage : exit
 - 현재 실행중인 ssu_cleanup 프로그램 종료
- 2) 실행 결과
 - 프로그램 종료

<p>예 4-1. exit 실행 시 프로그램 종료</p> <pre> 20230000> exit % </pre>
--

○ 과제 구현을 위한 참고 함수 (필수적으로 사용해야 하는 것은 아님)

- 1. getopt() : 프로그램 실행 시 입력한 인자를 처리하는 라이브러리 함수

<pre> #include <unistd.h> int getopt(int argc, char * const argv[], const char *optstring); // _POSIX_C_SOURCE #include <getopt.h> int getopt_long(int argc, char * const argv[], const char *optstring, const struct option *longopts, int *longindex); // _GNU_SOURCE </pre>

- 2. scandir : 디렉토리에 존재하는 파일 및 디렉토리 전체 목록 조회하는 라이브러리 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 **));
```

-1 : 오류가 발생, 상세한 오류 내용은 errno에 설정
0 이상 : 정상적으로 처리, namelist에 저장된 struct dirent *의 개수가 return

- 3. realpath : 상대경로를 절대경로로 변환하는 라이브러리 함수

```
#include <stdlib.h>
char *realpath(const char *path, char *resolved_path);
```

NULL : 오류가 발생, 상세한 오류 내용은 errno 전역변수에 설정
NULL이 아닌 경우 : resolved_path가 NULL이 아니면, resolved_path를 return,
resolved_path가 NULL이면, malloc(3)으로 할당하여 real path를 저장한 후에 return

- 4. strtok : 특정 문자 기준으로 문자열을 분리하는 함수

```
#include <string.h>
char *strtok(char *restrict str, const char *restrict delim);
```

return a pointer to the next token, or NULL if there are no more tokens.

- 5. exec()류 함수 : 현재 프로세스 이미지를 새로운 프로세스 이미지로 대체하는 함수(라이브러리, 시스템콜)

```
#include <unistd.h>
int execl(const char *pathname, const char *arg, .../* (char *) NULL */);
int execv(const char *pathname, char *const argv[]);
int execlp(const char *pathname, const char *arg, .../* (char *) NULL */);
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg, .../* (char *) NULL */);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

The exec() family of functions replaces the current process image with a new process image.
<https://man7.org/linux/man-pages/man3/exec.3.html> 또는 교재 참고

○ make와 Makefile

- make : 프로젝트 관리 유틸리티
- ✓ 파일에 대한 반복 명령어를 자동화하고 수정된 소스 파일만 체크하여 재컴파일 후 종속된 부분만 재링크함
- ✓ Makefile(규칙을 기술한 파일)에 기술된 대로 컴파일 명령 또는 쉘 명령을 순차적으로 실행함
- Makefile의 구성
- ✓ Macro(매크로) : 자주 사용되는 문자열 또는 변수 정의 (컴파일러, 링크 옵션, 플래그 등)
- ✓ Target(타겟) : 생성할 파일
- ✓ Dependency(종속 항목) : 타겟을 만들기 위해 필요한 파일의 목록
- ✓ Command(명령) : 타겟을 만들기 위해 필요한 명령(shell)

Macro

Target : Dependency1 Dependency2 ...
<-Tab->Command 1
<-Tab->Command 2
<-Tab->...

- Makefile의 동작 순서
- ✓ make 사용 시 타겟을 지정하지 않으면 제일 처음의 타겟을 수행

- ✓ 타겟과 종속 항목들은 관습적으로 파일명을 명시
- ✓ 명령 항목들이 충족되었을 때 타겟을 생성하기 위해 명령 (command 라인의 맨 위부터 순차적으로 수행)
- ✓ 종속 항목의 마지막 수정 시간(st_mtime)을 비교 후 수행
- Makefile 작성 시 참고사항
- ✓ 명령의 시작은 반드시 Tab으로 시작해야함
- ✓ 한 줄 주석은 #, 여러 줄 주석은 `##`
- ✓ 자동 매크로 : 현재 타겟의 이름이나 종속 파일을 표현하는 매크로

매크로	설명
<code>\$?</code>	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서 사용 불가능)
<code>\$\$</code>	현재 타겟의 종속 항목 (확장자 규칙에서 사용 불가능)
<code>\$<</code>	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서만 사용 가능)
<code>\$*</code>	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서만 사용 가능)
<code>\$@</code>	현재 타겟의 이름

- Makefile 작성 예시

(예 1). Makefile	(예 2). 매크로를 사용한 경우	(예 3). 자동 매크로를 사용한 경우
<pre>test : test.o add.o sub.o gcc test.o add.o sub.o -o test test.o: test.c gcc -c test.c add.o: add.c gcc -c add.c sub.o: sub.c gcc -c sub.c clean : rm test.o rm add.o rm sub.o rm test</pre>	<pre>OBJECTS = test.o add.o sub.o TARGET = test CC = gcc \$(TARGET) : \$(OBJECTS) \$(CC) -o \$(TARGET) \$(OBJECTS) test.o: test.c \$(CC) -c test.c add.o: add.c \$(CC) -c add.c sub.o: sub.c \$(CC) -c sub.c</pre>	<pre>OBJECTS = test.o add.o sub.o TARGET = test CC = gcc \$(TARGET) : \$(OBJECTS) \$(CC) -o \$@ \$^ test.o: test.c \$(CC) -c \$^ add.o: add.c \$(CC) -c \$^ sub.o: sub.c \$(CC) -c \$^</pre>

- (예 4). Makefile 수행 예시

<pre>oslab@a-VirtualBox:~\$ make gcc -c test.c gcc -c add.c gcc -c sub.c gcc test.o add.o sub.o -o test oslab@a-VirtualBox:~\$</pre>	<pre>oslab@a-VirtualBox:~\$ make clean rm test.o rm add.o rm sub.o rm test oslab@a-VirtualBox:~\$</pre>
--	---

○ 보고서 제출 시 유의 사항

- 보고서 제출 마감은 3월 25일(화) 11:59:59 PM까지 (구글 서버시간)

※ 구글클래스룸에서 11:59:00에서 1초라도 지연되면 1일 지연제출로 나올 것임. 이 부분은 추후 제출 시간 확인 후 11:59:59 이내 인 경우 정상 제출로 인정 예정임.

- 1초 지연제출도 0점 처리

- 압축 오류, 파일 누락 관련 감점은 강의계획서(Syllabus) 참고

- 필수구현 : 1. ssu_cleanup, 2. 내장명령어 tree, help, exit
- 배점(100점 만점. 실행 여부 배점 80점으로 최종 환산하며, 보고서 15점과 소스코드 주석 5점은 별도, 강의계획서 확인)

1. ssu_cleanup : 10점 (필수구현)

2. 내장명령어 1. tree : 25점 (필수구현. 단, 아래 2개의 옵션은 필수 구현 아님), 아래 옵션 미구현 시 : 15점

✓ '-s' 옵션 : 5점

✓ '-p' 옵션 : 5점

3. 내장명령어 2. arrange : 50점, 아래 옵션 미구현 시 : 20점

✓ '-d' 옵션 : 5점

✓ '-t' 옵션 : 5점

✓ '-e' 옵션 : 10점

✓ '-x' 옵션 : 10점

4. 내장명령어 3. help : 5점 (필수구현)

5. 내장명령어 4. exit : 5점 (필수구현)

6. makefile 작성(필수구현. 단, 매크로 사용하지 않아도 됨) : 5점

※ (가산점 부여기준 1) 3월 16일 밤 11시59분까지 명세서에서 언급한 모든 기능(필수구현 외 기타구현 모두)을 구현하고 보고서를 제출한 경우 본과제 총점에 +30점, 3월 23일 밤 11시59분까지 제출한 경우 본과제 총점(100점)에 +10점을 (가산점으로)추가 부여 함.