SCE212 Project #0: 파일 입출력에 대한 이해와 문자열 파싱

마감기한: 3월 21일 오후 11:59

1. 개요

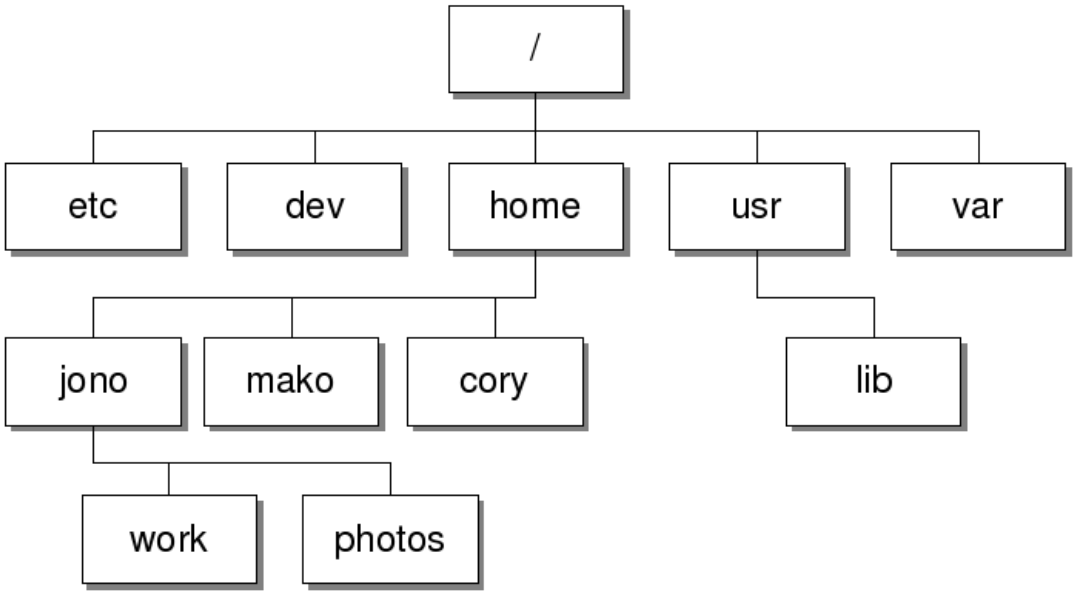
첫번째 과제는 특정 디렉토리의 파일들을 출력하는 프로그램을 구현하는 것입니다. 이 프로젝트의 주 목적은 학생들에게 C 프로그래밍 언어에서 문자열 파싱(parsing), 포인터(pointer), 구조체(structure)에 대한 이해를 돕기 위해서 준비하였습니다. 또한 앞으로 진행할 과제 환경(Linux) 및 제출 시스템 (pasubmit)에 대해 친숙해지는 시간이 될 것 입니다.

간략하게 구현할 프로그램을 설명하자면, 입력으로 파일들의 경로(path)가 주어지고, 이 정보를 가지고 특정 디렉토리 안에 있는 파일들을 출력하는 프로그램입니다.

2. 배경지식

2.1 디렉토리(directory) 구조

Linux는 기본적으로 사용자가 볼 수 있는 모든 것부터 보이지 않는 운영체제 및 하드웨어가 구성하고 있는 모든 것까지 모두 파일(file) 형태로 이루어져있습니다. (Linux 자체가 파일의 총 집합체라고 말해도 과언이 아닙니다.) 아래는 Linux kernel 이 구성하고 있는 디렉토리 구조입니다.



몇 가지 생략된 디렉토리 가 있지만 일반적으로 그림과 같은 계층적 구조를 이루고 있습니다. 모든 파일이나 디렉토리들은 `root`라고 불리는 디렉토리에서 부터 시작하며, `'/'` 로 표현됩니다.

2.2 경로 (Path)

Linux에는 Windows 와 동일하진 않지만 거의 비슷한 개념으로 경로(Path) 라는 것이 있습니다. 예를 들어 Windows 에서 "`C:\Program files\AppData\`" 로 path 를 나타낼 수 있다면 Linux 에선 비슷하게 "`/home/user/application`" 의 형태로 나타낼 수 있습니다. 경로를 나타내는 방식은 두 가지가 있는데 각각 표기하는 방법에 따라 불려지는 이름이 다릅니다.

- `Absolute path`: 직역하면 절대 경로라는 뜻으로 `root`부터 선택된 파일, 디렉토리 까지의 전체 경로를 뜻합니다. 예를 들어 앞의 예제와 같은 것이 Absolute path 입니다.

- `Relative path`: 직역하면 상대 경로라는 뜻으로 선택된 파일 및 디렉토리의 시점에서 보여지는 경로를 의미합니다.

이번 과제에서는 Path를 상대 경로 없이 **절대 경로**만을 사용하여 표현하였습니다.

3. 구현 사항

본 과제에서 구현해야 될 2가지 문제에 대해서 아래 자세히 설명하도록 한다.

3.1 Task #1: utils.c의 parse_str_to_list() 작성

```
// This function splits the input string (const char* str) to tokens
// and put the tokens in token_list. The return value must be the number
// of tokens in the given input string.
int parse_str_to_list(const char* str, char** token_list) {
     /* Fill this function */
}
```

이 함수는 첫번째 매개변수 (`str`) 를 '/' 문자 또는 개행문자 (`\n`)를 단위로 하여 분리(tokenize) 시키고 그 결과로 만들어진 토큰(token)을 두번째 매개변수 (`token_list`)에 넣고 토큰의 갯수를 반환하는 함수이다. 아래는 입력 예시에 대한 출력 예시를 보여주는 그림이다.

예시)
`str = "/home/user/text.txt"`
`token_list =`

"home"	"user"	"text.txt"

힌트: `strtok()` 함수를 사용하면 쉽게 문자열을 tokenize할 수 있으며, `token_list`의 마지막 인덱스는 파일 이름, 나머지 인덱스는 디렉토리 이름이 담긴다.

3.2 Task #2: dir_file.c의 make_dir_and_file() 작성

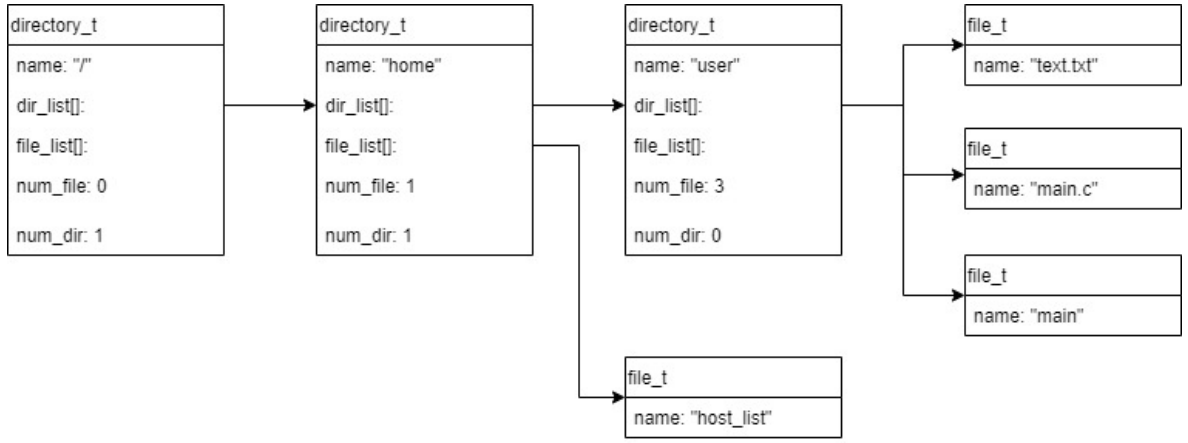
```
 // This function creates the hierarchy of files and directories
 // which are indicated by token_list. Everything starts in root_dir.
 // You can implement this function using the above find_create_dir()
 // and find_create_file() functions.
 void make_dir_and_file(directory_t* root_dir, char** token_list, int
num_tokens) {
     /* Fill this function */
 }
```

이 함수는 `root_dir` 디렉토리 (첫번째 매개변수)를 시작으로하여 `token_list`에 있는 토큰을 이용하여 디렉토리와 파일들의 계층적 구조로 만드는 함수이다. `token_list`에 얼마만큼의 토큰이 있는지는 3번째 매개변수 `num_tokens`에 기록 되어 있다. 단, `num_tokens`는 첫번째 task를 정상적으로 구현했을 때 `main` 함수 에서 전달 가능하다.

예시)

```
/home/user/text.txt
/home/user/main.c
/home/user/main
/home/host_list
```

〈파일들의 경로〉



〈프로그램에서 설계된 구조〉

위의 그림은 `token_list`에 있는 `token`에 대해서 생성된 디렉토리 및 파일의 구조를 도식화하여 보여준다. 그림에서 나와있는 `directory_t` 및 `file_t`에 대한 구조체는 `include/dir_file.h`에 정의되어 있으니 해당 부분을 참고하면 쉽게 구현이 가능하다.

*중요: `utils.c` 그리고 `dir_file.c` 외 다른 파일들(예를들어 `main.c`)은 고치면 안됩니다.

4. 시작하기

* Linux 환경에 익숙하지 않은 경우 아래 내용을 읽어보는 것을 강력히 추천
  - Linux Shell: https://missing-semester-kr.github.io/2020/course-shell/
  - vim editor: https://missing-semester-kr.github.io/2020/editors/
  - Makefile: https://missing-semester-kr.github.io/2020/metaprogramming/

4.1 제공되는 코드 다운로드

```
$] git clone https://github.com/csl-ajou/sce212-project0.git
Cloning into 'sce212-project0'...
remote: Enumerating objects: 48, done.
remote: Counting objects: 100% (48/48), done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 48 (delta 1), reused 48 (delta 1), pack-reused 0
Unpacking objects: 100% (48/48), done.
Checking connectivity... done.
```

`git`을 통한 코드 다운로드 방법

4.2 소스 코드 컴파일

```
$] pwd
/home/csl

$] cd sce212-project0/
$] pwd
/home/csl/sce212-project0/
```

코드가 있는 위치로 이동

```
$] make
gcc -std=c99 -g -I ./ -I ./include   -c -o utils.o utils.c
gcc -std=c99 -g -I ./ -I ./include   -c -o main.o main.c
gcc -std=c99 -g -I ./ -I ./include   -c -o dir_file.o dir_file.c
gcc -g -o pa0 ./utils.o ./main.o ./dir_file.o
```

`make`를 이용하여 컴파일 하면 `pa0` 실행 파일을 생성 할 수 있다.

4.3 프로그램 실행

```
$] ls
dir_file.c      include  main.o      my_outputs  README.md
sample_outputs  utils.o  dir_file.o  main.c      Makefile
pa0             sample_inputs   utils.c
```

`ls` 명령어를 통해 살펴보면 `pa0` 실행 파일이 생성된 것을 확인할 수 있다. 아래는 `pa0`를 실행하는 방법을 보여준다. 참고로 본 과제에서 사용되는 입력 파일은 `sample_inputs`에 존재하고 해당 `sample_inputs`에 대응되는 올바른 출력 결과는 `sample_outputs`에 존재한다.

```
# example0 실행
# ./pa0 <file_input> <dir_input>
$] ./pa0 sample_inputs/input0/file_input  \
> sample_inputs/input0/dir_input

main.c
README.md
.gitignore
figure1.png
figure2.png
```

위 예시처럼 실행파일을 실행하기 위해서는 실행파일 앞에 '`./`'를 붙여 실행할 수 있다. `pa0`는 프로그램 인자로 `file_input`과 `dir_input`를 가진다. 그렇기 때문에 프로그램을 정상적으로 실행하기 위해서는 인자를 위 예시처럼 같이 입력해야한다.

4.3.1 입력 예시: `sample_inputs/input0/`

dir_input	file_input
/home/jinu/git/project0 /home/jinu/ /home	/home/jinu/git/project0/main.c /home/jinu/git/project0/README.md /home/jinu/git/project0/.gitignore /home/jinu/figure1.png /home/jinu/figure2.png

입력은 두 개의 텍스트 파일로 받으며, `dir_input`과 `file_input`이 있습니다.

- `dir_input`: 디렉토리들의 절대 경로를 담은 텍스트 파일
- `file_input`: 파일들의 절대 경로를 담은 텍스트 파일

위 예제의 경우, '`/home/jinu/git/project0`' 디렉토리안에 `main.c`, `README.md`, 그리고 `.gitignore` 파일이 존재하고, '`/home/jinu`' 디렉토리안에는 `figure1.png` 그리고 `figure2.png` 파일이 존재하는 경우입니다.

4.3.2 출력 예시

4.3.1에서 사용한 예시를 입력으로 하였을 때, 나오는 출력은 아래와 같습니다.

```
main.c
README.md
.gitignore
```

```
figure1.png
figure2.png
```

여기서 주의할 점은 **디렉토리가 아닌 파일만을 출력한다는 점입니다.** 또한, 디렉토리는 존재하지만 그 안에 파일이 존재하지 않을 경우에는 출력하지 않습니다. 출력 순서는 다음과 같습니다.

1. `dir_input`에 적힌 디렉토리 순서로 출력
2. 디렉토리 내부에 여러 파일이 존재하는 경우, `file_input`에 적힌 순서로 출력 (알파벳 순서가 아님)

4.3.3 결과 비교

`pa0`이 올바른 결과값을 출력했는지 판별하기 위해서는 제공되는 `sample_outputs`에 있는 파일 내용과 출력값이 일치하는 지 확인하면 된다. 하지만 이러한 작업을 눈으로 하나씩 비교하는 것은 매우 번거러울 수 있다. 다행히도 `Linux`에서는 이러한 작업을 도와주는 유틸리티 `diff`라는 것을 제공해주고 있다. 참고로 `diff`는 비교 대상으로 파일만 입력으로 받기 때문에 `diff`를 사용하기 위해서는 `pa0`에서 나오는 출력값을 파일에 저장해서 비교해야한다. `Linux`에서는 '>'(이를 `Redirection`이라고 부름)를 사용하여 출력값을 파일에 저장할 수 있다. 아래 예시에서 첫번째 라인을 실행하면 `my_outputs/output0`라는 파일이 생기고 그 파일에 `pa0`의 출력값이 저장된다. 그런 다음 `diff`를 통해 `sample_outputs`에 있는 파일과 출력값이 저장된 파일을 비교하면 된다.

```
# ./pa0의 출력값이 my_outputs/output0 파일에 저장됨
# Usage: ./executable > file_name
$] ./pa0 sample_inputs/input0/file_input  \
> sample_inputs/input0/dir_input > my_outputs/output0

$] ls my_outputs/output0
output0

$] diff -Naur my_outputs/output0 sample_outputs/output0
```

위의 명령어는 `diff`를 통해서 2개의 파일을 비교하는 것으로 일치하면 아무런 메시지를 출력하지 않고 불일치 할 경우 틀린 부분에 대해서 출력해준다.아래는 과제에서 제공해주는 `Makefile`에서 0번째 테스트 (`test_0`)에 대하여 실행하는 경우 `diff`를 통해 출력된 결과를 보여준다.

```
$] make test_0
Testing example0
--- my_outputs/output0     2020-03-20 14:37:18.789646635 +0900
+++ sample_outputs/output0 2020-03-20 14:37:09.329573792 +0900
@@ -0,0 +1,5 @@
+main.c
+README.md
+.gitignore
+figure1.png
```

```
+figure2.png
      Results not identical, check the diff output
```

Makefile에 정의해놓은 테스트 케이스 (test_0 ~ test_4)를 이용하여 reference 프로그램의 output과 비교하여 자신의 출력결과가 어디서 잘못되었는지 알 수 있다. 'make test' 명령어를 이용하면 테스트케이스 0번 부터 4번까지 모두 테스트한 결과를 보여준다.

만약 자신의 출력결과가 reference 프로그램의 출력과 동일하면 아래와 같이 나타난다.

```
$] make test_0
Testing example0
      Test seems correct
```

5. 결과 제출

본 숙제에서 작성한 utils.c 그리고 dir_file.c 파일 2개를 zip 혹은 tar.gz 형태로 압축하여 제출한다. 압축을 도와주기 위하여 Makefile에 submission이라는 명령어를 추가하였고 아래에서 처럼 2개의 파일을 submission명령어를 통해 압축하여 pa0-submission.tar.gz 파일로 만들어 낼 수 있다.

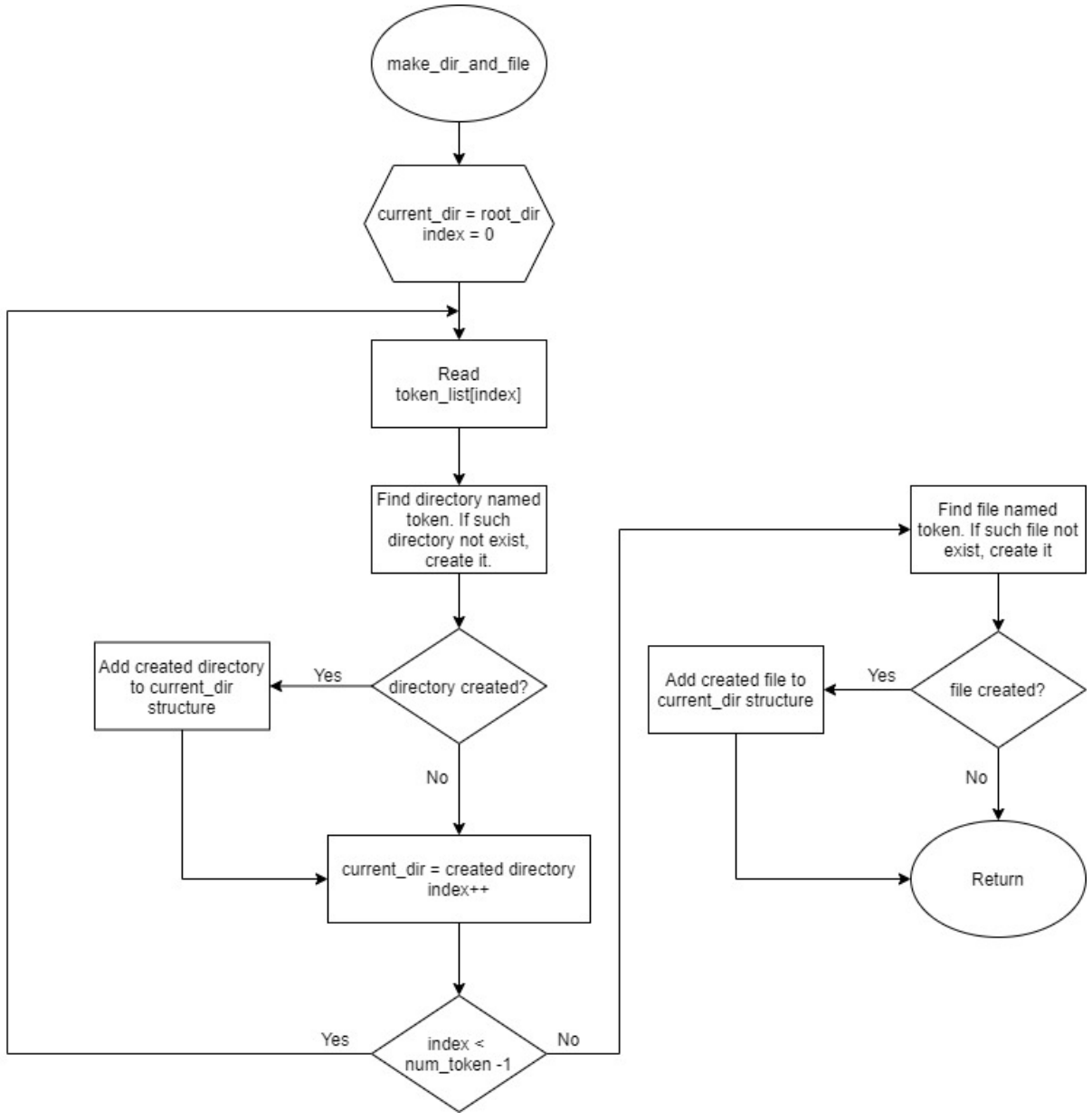
```
$] pwd
/home/csl/sce212-project0/

$] make submission
Generating a compressed file (pa0-submission.tar.gz) including
utils.c and dir_file.c
a utils.c
a dir_file.c

$] ls pa0-submission.tar.gz
pa0-submission.tar.gz
```

만들어진 pa0-sumission.tar.gz 파일을 **https://sslab.ajou.ac.kr/pasubmit**에서 제출하고 나서 반드시 Test를 통해서 본인이 테스트한 결과와 동일한 결과가 나오는지 확인을 진행한다.

6. 참고

아래는 구현을 할 때 참고하면 되는 부분이고 본인의 방법대로 진행해도 상관 없음

6.1 `make_dir_and_file`에 대한 Flow Chart



6.2 Segmentation Fault가 발생했을 때, 디버깅 방법

코드를 작성하다 보면 `Segmentation Fault` 라는 에러를 만나실 수 있는데, 해당 에러가 발생했을 때 디버깅하는 법을 알려드리려고 합니다. 우선 `Segmentation Fault`에 대해서 소개한 뒤 디버깅 방법에 대해서 설명하겠습니다.

6.2.1 Segmentation Fault 이란?

Segmentation Fault는 컴퓨터 소프트웨어의 실행 중에 일어날 수 있는 특수한 오류이며, 발생 원인은 프로그램이 허용되지 않은 메모리 영역에 접근을 시도하거나, 허용되지 않은 방법으로 메모리 영역에 접근을 시도할 경우 발생합니다.

예를 들어, 포인터 변수로 부터 할당하지 않은 메모리를 접근했을 때 또는 할당 범위를 넘어서 접근했을 때 주로 발생합니다.

6.2.2 GDB

GDB는 잘 알려진 디버거로써 다른 디버거들과 마찬가지로 프로그램의 내부에서 진행되고 있는 함수 혹은 코드라인을 추적할 수 있고 그와 동시에 해당 지점에서의 변수 혹은 결과값들을 확인하여 프로그램이 정상적으로 동작하는 확인할 수 있다.

6.2.2.1 GDB 실행

아래 명령어들은 GDB에서 자주 사용되는 명령어들이며 과제를 수행하는데 충분할 것이라고 생각한다.

실행 프로그램 앞에 `'gdb'`를 붙여서 실행하면 해당 프로그램을 GDB를 통해 디버깅할 수 있게 된다. 만약 실행하려는 프로그램이 인자를 받는 다면 `'gdb'`에 `'--args'` 옵션을 사용해야한다 (아래 예시가 있으니 참고). GDB는 사용자가 프로그램을 제어할 수 있도록 prompt를 제공하여 입력되는 명령어에 따라 동작한다. GDB 명령어는 뒤에서 더 자세히 설명하고 여기서는 GDB 안에서의 프로그램 실행과 GDB를 나가는 것을 알려주겠다. GDB prompt에 `'run'` 혹은 `'r'`을 입력하게 되면 프로그램이 실행이 된다. GDB가 처음 시작할 때 프로그램도 같이 실행되는 것이 아니기 때문에, 추가적으로 run 명령어를 실행해야한다. 만약 중간에 특정 에러가 발생하게 되면 문제가 발생한 부분에서 프로그램이 멈추게 될 것이고, 반대로 에러가 발생하지 않는다면 정상적으로 프로그램이 끝날 것이다. 그리고 GDB에서 나가고 싶다면 `'quit'` 혹은 `'q'`을 prompt에 입력하면 정상적으로 나갈 수 있다.

```
$] pwd
/home/csl/sce212-project0/

# gdb <executable>
$] gdb --args ./pa0 sample_inputs/input0/file_input \
> sample_inputs/input0/dir_input
(gdb) run
Starting program:
/home/jinu/project/sce212-project0-reference/pa0
sample_inputs/input0/file_input sample_inputs/input0/dir_input
main.c
README.md
.gitignore
```

```
figure1.png
figure2.png
[Inferior 1 (process 2934329) exited normally]

(gdb) r
Starting program:
/home/jinu/project/sce212-project0-reference/pa0
sample_inputs/input0/file_input sample_inputs/input0/dir_input
main.c
README.md
.gitignore
figure1.png
figure2.png
[Inferior 1 (process 2935148) exited normally]

(gdb) quit

$] # shell로 다시 돌아옴
```

6.2.2.2 GDB 명령어

- **`run / r` (프로그램 실행)**
  *run*은 프로그램을 시작 혹은 재시작할 수 있다.

- **`break / b` (breakpoint 설정)**
  `break`는 프로그램의 stop 포인트를 설정한다. 즉 breakpoint를 설정하는 명령어이다. `break function_name` 을 통해 특정 함수에 breakpoint를 걸 수 있다. 이 외에도 `break linenumber` 또는 `break filename:linenumber` 를 통해 특정 소스파일에 있는 해당 라인에 breakpoint를 설정할 수 있다.

- **`delete / d` (breakpoit 삭제)**
  `delete` 혹은 `d` 를 통해 모든 breakpoint를 삭제할 수 있다. 또한 `delete number`를 통해 해당 넘버에 해당하는 breakpoint를 삭제할 수 있다. breakpoint의 정보는 `info breakpoints`를 통해 확인할 수 있다.

- **`continue / c` (프로그램 재개)**
  `continue`는 멈춰있는 프로그램을 다시 재개한다. 만약 뒤에 breakpoint가 없다면 프로그램이 끝날때까지 프로그램이 실행된다.

- **`next / n` (다음 라인으로 넘어가기)**
  `next`는 현재 멈춰있는 상태를 실행하고 다음 라인에서 멈춘다. line-by-line으로 프로그램을 실행할 수 있어 디버깅할 때 유용하게 사용된다.

- **`list / c` (주변 소스코드 출력)**
  list linenumber 를 통해 `linenumber` 주변 소스 코드를 출력할 수 있다. 만약 `linumber` 없이 `list`만 사용할 경우 현재 위치 주변의 소스 코드를 출력한다. `list`를

반복해서 입력할 수 있으며 다음에 출력되는 소스코드는 이전 list에서 출력된 소스코드의 다음 소스코드들을 출력한다.

- **print / p (변수 출력)**
  print variable은 해당 변수안에 있는 값을 출력한다. 또한 배열 인덱스에 대해서도 적용가능하다.
  e.g. list라는 배열이 있고 첫 번째 인덱스 값이 25일 때, 아래처럼 접근하여 인덱스 0번에 있는 값을 출력할 수 있다.

```
(gdb) print list[0]
$1 = 25
```

또한 변수 뿐만 아니라 표현식에도 적용가능하다.

```
(gdb) print list[0]+list[0]
$1 = 50 # 25 + 25
```

- **up (이전함수로 이동)**
  up은 상위 프레임으로 이동한다. 다시 말해, 현재 속한 함수에서 해당 함수를 호출한 부분으로 이동한다.

6.2.3 Segmentation Fault 발생 원인 찾기

위 세션에서 GDB에 대해 간단히 소개하였습니다. 그럼 이번 세션에는 실제 과정의 일부를 수정하여 segmentation fault가 발생하게 만들고, 그 프로그램에서 에러가 발생되는 곳을 GDB로 찾는 방법을 소개하겠습니다.

```
$] pwd
/home/csl/sce212-project0/

$] gdb --args ./pa0 sample_inputs/input0/file_input \
> sample_inputs/input0/dir_input
(gdb) r
Starting program:
/home/jinu/project/sce212-project0-reference/pa0
sample_inputs/input0/file_input sample_inputs/input0/dir_input

Program received signal SIGSEGV, Segmentation fault.
__memmove_avx_unaligned_erms () at
../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:374
374     ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:
No such file or directory.
# 현재 segmentation fault가 memmove-vec-unaligned-erms.S 에서 발생한다고
```

나온다. 하지만 해당 파일을 우리가 가지고 있는 소스코드가 아닌 다른 library에 속한 파일이기 때문에 코드를 열람할 수가 없어 디버깅이 힘들다. 그렇기 때문에 자신이 가지고 있는 코드 중 해당 파일을 호출한 곳으로 가서 디버깅하는 것이 더 효율적이다. 이전 함수를 이동하기 위한 명령어로 `up`을 사용하면 된다.

```
(gdb) up
#1  0x00005555555558e7 in create_file (name=0x55555555d3d0
"main.c") at dir_file.c:60
60          memcpy(file->name, name, MAX_NAME_SIZE);
```

`#` 이전 `error`가 발생한 부분을 호출한 곳이 `dir_file.c`의 60번째 라인에 있는 `memcpy(file->name, name, MAX_NAME_SIZE);` 라고 나온다. 그렇다면 우선적으로 확인해봐야할 것은 해당 함수에서 사용된 인자들이다. `p` 라는 명령어를 통해 하나씩 찾아보자.

```
(gdb) p file
$1 = (file_t *) 0x55563810
(gdb) p file->name
Cannot access memory at address 0x55563810
```

`file->name`에 접근이 불가능하다는 메시지가 나왔다. 즉, `file->name`에 메모리 할당되지 않았다는 것을 알 수 있다. 이를 알았으니 디버거를 나가서 소스코드 중 `file->name`을 할당하거나 사용하는 부분을 유심히 확인하여 코드를 수정하면 생각보다 쉽게 `segmentation fault`를 해결할 수 있을 것이다.