Background

GDB

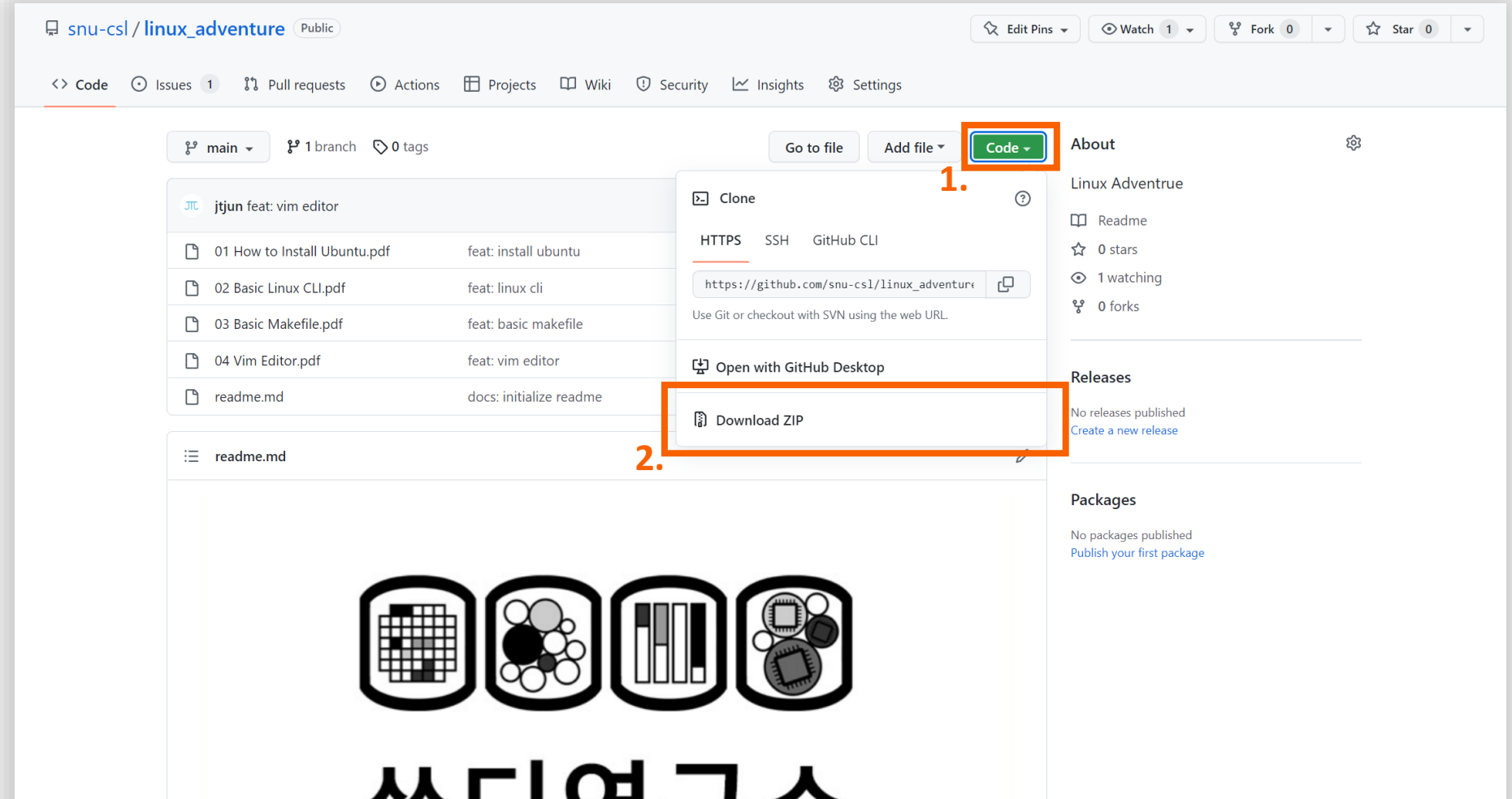
Segfault Debugging

GDB
GNU Debugger

github.com/snu-csl/linux_adventure

Preparation

- 실습 파일 https://github.com/snu-csl/linux_adventure



Example Codes

main.c

```c
#include <stdio.h>

int sum(int x, int y) {
  return x + y;
}

int main() {
  int x, y, res;
  scanf("%d %d\n", &x, &y);
  res = sum(x, y);
  printf("result is %d\n", res);
  return 0;
}
```

sum.h

```c
int sum_mat(int **mat, int len);
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

segfault.c

```c
#include <stdlib.h>
#include "sum.h"

int main() {
  int *mat[3], res;
  for (int i=0; i<=len; i++) {
    mat[i] = malloc(sizeof(int)*3);
    for (int j=0; j<=len; j++)
      mat[i][j] = j + 1;
  }
  res = sum_mat(mat, 3);
  for (int i=0; i<=3; i++) free(mat[i]);
  return 0;
}
```

GDB 란?

-g 옵션 필요성

gdb 실행

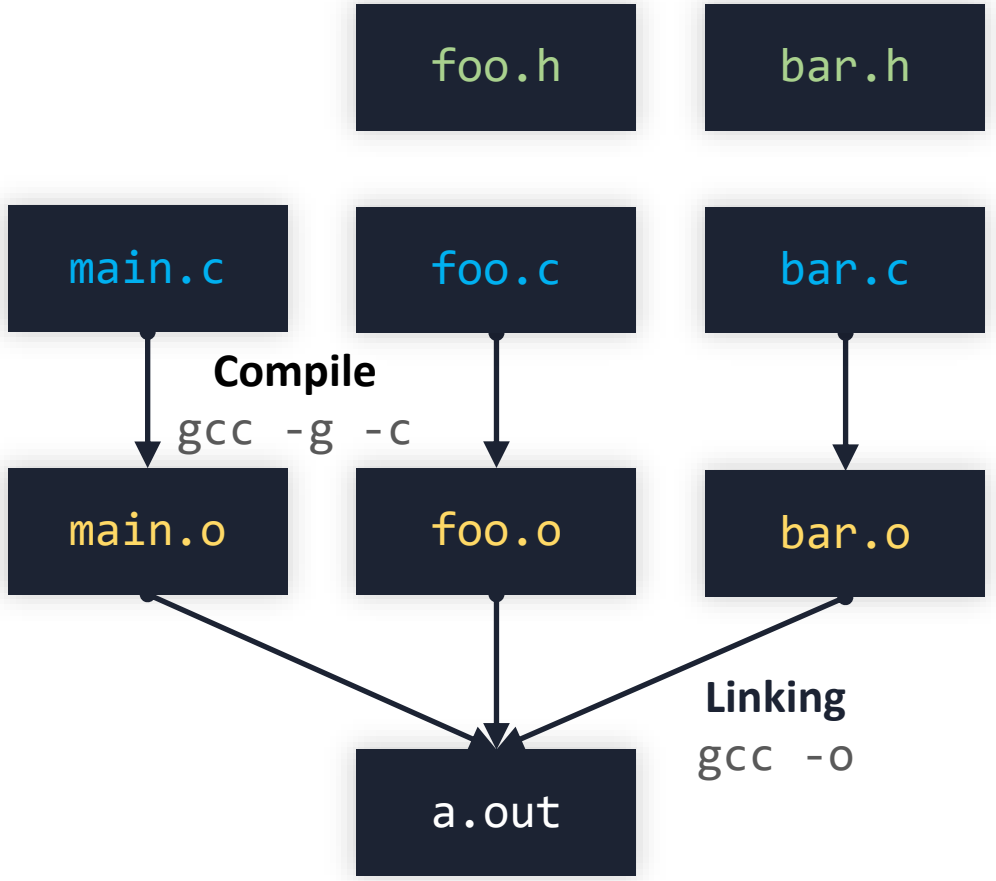
Background

GDB 란?

- **GNU Debugger**
  - **GNU**: “GNU is Not Unix”, 컴퓨터 소프트웨어의 모음집
    GPL(General Public License) 하에 배포되는 Free Software
  - **GDB**: GNU 소프트웨어를 위한 기본 **디버거**
  - 다양한 유닉스 기반 시스템에서 동작 (C, C++, 포트란 등의 언어 지원)
  - 프로그램의 실행을 **추적**하고 **수정**할 수 있는 기능 제공
  - 개발하는 과정에서 발생하는 버그를 **line-by-line**으로 찾을 수 있음

-g 옵션 필요성

- 어셈블리 : 기계어, 1:1 매칭
- -g 없이는 어셈블리만 확인
  소스코드의 symbol 확인 불가능
- → 디버깅을 위해 symbol 추가
  - gcc -g 옵션
  - -O 최적화를 함께할 경우, 일부 유실



GDB 실행

- **gcc**와 **gdb** 설치
- 컴파일시 **-g** 옵션

  `$ gcc -g -o [exe] [src]`

  실행파일에 symbol 추가
- `$ gdb [exe]` 로 시작

```
$ sudo apt install gcc
$ sudo apt install gdb
$ gcc -g -o main main.c↲
$ gdb main↲
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
...
Reading symbols from main...
(gdb)
```

gdb 예시
gdb 명령어

GDB

gdb 예시

```
$ gcc -g -o main main.c↲
$ gdb main↲
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation,
Inc.
...
Reading symbols from main...
(gdb) run
1 2
Result is 3
[Inferior 1 (process 615) exited normally]
(gdb)
```

main.c

```c
#include <stdio.h>

int sum(int x, int y) {
  return x + y;
}

int main() {
  int x, y, res;
  scanf("%d %d\n", &x, &y);
  res = sum(x, y);
  printf("result is %d\n", res);
  return 0;
}
```

gdb 명령어

```
(gdb) b main
Breakpoint 1 at 0x11a1:
file main.c, line 7.
(gdb) r
Starting program: /gdbasic/main

Breakpoint 1, main () at main.c:7
7       int main() {
(gdb) wa res
Hardware watchpoint 2: res
(gdb) d main
(gdb)
```

- **r : run {arg1} {arg2}**
  프로그램 처음부터 실행 (breakpoint 일시정지)

- **b : break [breakpoint]**
  특정 라인(10, +2)이나 함수에 breakpoint 설정
  tb: 1회용 break point

- **cl : clear {breakpoint}**
  breakpoint 삭제, 인자 없으면 모두 삭제

- **wa : watch [var_name]**
  var_name 변수가 변경될 때마다
  break하고 변경 사항 보여줌

- **d : delete [target]**
  <var_name> 또는 <#breakpoint>
  또는 display <#display> 삭제

gdb 명령어

```
(gdb) n 2
1 2
10      res = sum(x, y);
(gdb) s
sum (x=21845, y=1431655021) at main.c:3
7       int sum(int x, int y) {
(gdb) finish
Run till exit from #0
sum (x=21845, y=1431655021) at main.c:3
0x5555555551e7 in main () at main.c:10
10      res = sum(x, y);
Value returned is $1 = 3
(gdb) c
```

- **n : next [num]**
  함수를 들어가지 않고 한 행 씩 실행
  [num] 인자를 넘기는 경우 [num] 만큼 실행

- **s : step [num]**
  함수 내부로 들어가서 한 행 씩 실행

- **finish**
  함수 return 까지 실행

- **return {ret}**
  반환 값 ret로 현재 함수를 빠져나감
  (실제로 함수를 실행하진 않음)

- **c : continue**
  현재 위치에서 다음 breakpoint까지 실행

gdb 명령어

```
(gdb) c
Continuing.
Hardware watchpoint 2: res

Old value: 32767
New value: 3
main () at main.c:11
11      printf("result is %d\n", res);
(gdb) bt
#0 main () at main.c:11
(gdb) p x
$2 = 1
(gdb)
```

- **bt**
  back-trace 로, 프로그램 호출 스택 확인
- **k : kill**
  디버깅 중인 프로그램의 실행 취소
- **p : print**
  디버깅 중 원하는 값 출력
- **display {var_name}**
  var_name 변수의 값을 계속 보여줌
  인자 없으면, 현재 display 목록 보여줌
- **l : list**
  현재 위치에서 +-5줄의 소스코드 출력

gdb 명령어

```
(gdb) i locals
x = 1
y = 2
res = 3
(gdb) set res = 10
(gdb) i locals
x = 1
y = 2
res = 10
(gdb) n
result is 10
12      return 0;
(gdb) q
```

- **i : info [target]**
  - locals : 지역 변수
  - variables : 전역 변수
  - break : break point 정보
  - func : function 정보

- **set [setting]**
  메모리 특정 영역에 값 설정, gdb 출력 형식 설정

- **h : help**
  도움말

- **q : quit**
  gdb 종료
  또는 <Ctrl> + d

Segfault 분석

Debug Segfault

Segfault Debugging

Segfault 분석

- **gdbasic** 폴더에서
  Makefile 을 통해 빌드 가능

- **segfault.c** 에서 **sum.c** 의
  `sum_mat` 함수 호출
  `len` by **`len`** 행렬의 총합 구함

- **`sum_mat`** 함수의 **`for`** 문에서
  `i<=len`이 **`len`**을 포함 (line 5)
  행렬의 범위를 넘어가는 메모리 참조,
  Segmentation Fault 발생

sum.c

```
1  #include "sum.h"
2
3  int sum_mat(int **mat, int len) {
4    int res = 0;
5    for (int i=0; i<=len; i++)
6      for (int j=0; j<len; j++)
7        res += mat[i][j];
8    return res;
9  }
```

segfault.c

```
1  #include <stdlib.h>
2  #include "sum.h"
3
4  int main() {
5    int *mat[3], res;
6    for (int i=0; i<=len; i++) {
7      mat[i] = malloc(sizeof(int)*3);
8      for (int j=0; j<=len; j++)
9        mat[i][j] = j + 1;
10   }
11   res = sum_mat(mat, 3);
12   for (int i=0; i<=3; i++) free(mat[i]);
13   return 0;
14 }
```

Debug Segfault

```
$ make↲
cc -g -c -o segfault.o sefault.c
cc -g -c -o sum.o sum.c
cc -o seg segfault.o sum.o
$ ./seg↲
Segmentation fault
$ gdb seg↲
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
...
Reading symbols from seg...
(gdb)
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) run
Starting program: /gdbasic/seg

Program received signal SIGSEGV, Segmentation fault.
0x5555555552a0 in sum_mat (mat=0x7fffffffdd90, len=3) at sum.c:7
7       res += mat[i][j];
(gdb) bt
#0 0x5555555552a0 in sum_mat (mat=0x7fffffffdd90, len=3) at sum.c:7
#1 0x5555555552a0 in main () at segfault.c:11
(gdb) b sum_mat
Breakpoint 1 at 0x555555555254: file sum.c, line 3.
(gdb)
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) run
The program being debugged has been started already.
Start it from the biegginning? (y or n) y
Starting program: /gdbasic/seg

Breakpoint 1,
sum_mat (mat=0x7fffff7e6154 <__GI___libc_malloc+116>, len=3) at sum.c:3
3       int sum_mat(int **mat, int len);
(gdb) s
4       int res = 0;
(gdb) display res
1: res = 21845
(gdb)
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) s
5       for (int i=0; i<=len; i++);
1: res = 0
(gdb) display i
2: i = -8784
(gdb) watch i
Hardware watchpoint 2: i
(gdb) display
1: res = 0
2: i = -8784
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb)
Continuing.

Hardware watchpoint 2: i

Old value = 2
New value = 3
0x5555555552b5 in sum_mat (mat=0x7fffffffdd90, len=3) at sum.c:5
5       for (int i=0; i<=len; i++);
1: res = 18
2: i = 3
(gdb)
```

sum.c

```
1  #include "sum.h"
2
3  int sum_mat(int **mat, int len) {
4    int res = 0;
5    for (int i=0; i<=len; i++)
6      for (int j=0; j<len; j++)
7        res += mat[i][j];
8    return res;
9  }
```

Debug Segfault

```
(gdb)
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x5555555552a0 in sum_mat (mat=0x7fffffffdd90, len=3) at sum.c:7
7       res += mat[i][j];
1: res = 18
2: i = 3
(gdb)
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) run
The program being debugged has been started already.
Start it from the biegginning? (y or n) y
Starting program: /gdbasic/seg

Breakpoint 1,
sum_mat (mat=0x7ffff7e6154 <__GI___libc_malloc+116>, len=3) at sum.c:3
3       int sum_mat(int **mat, int len);
1: res = 21845
(gdb) s
4       int res = 0;
1: res = 21845
(gdb)
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) set len = 2
(gdb) c
Continuing.
[Inferior 1 (process 9758) exited normally]
(gdb) q
$ vi sum.c↲
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i<=len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

Debug Segfault

```
(gdb) set len = 2
(gdb) c
Continuing.
[Inferior 1 (process 9758) e
(gdb) q
$ vi sum.c↲
```

sum.c

```
1   #include "sum.h"
2
3   int sum_mat(int **mat, int len) {
4     int res = 0;
5     for (int i=0; i<=len; i++)
6       for (int j=0; j<len; j++)
7         res += mat[i][j];
8     return res;
9   }
```

Debug Segfault

```
(gdb) set len = 2
(gdb) c
Continuing.
[Inferior 1 (process 9758) exited normally]
(gdb) q
$ vi sum.c↲
$ make↲
cc -g -c -o sum.o sum.c
cc -o seg segfault.o sum.o
$ ./seg↲
$ make clean↲
rm *.o seg
$
```

sum.c

```c
#include "sum.h"

int sum_mat(int **mat, int len) {
  int res = 0;
  for (int i=0; i< len; i++)
    for (int j=0; j<len; j++)
      res += mat[i][j];
  return res;
}
```

감사합니다.