운영 체제

Assignment 4 보고서

학과 : 컴퓨터정보공학부

학번 : 2018202018

이름 : 유승재

제출 일자 : 2023. 11. 22 (수)

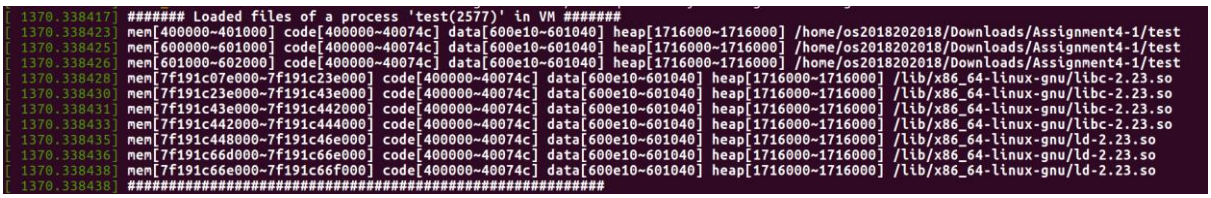
담당 교수 : 김태석 교수님

1. Introduction

이번 과제는 PID를 통해서 PID에 해당하는 프로세스의 이름, 가상 메모리 주소, 데이터 주소, 코드 주소, 힙 주소, 원본 파일의 전체 경로등을 얻어내는 방법을 알고 이를 출력해보는 4-1 과제와 컴파일된 코드를 공유 메모리에 복사하고 이를 write 권한이 있는 공간을 할당하여 해당하는 공간에 가져와 수정 후 실행해 어느 정도의 차이가 발생하는 지를 구분하는 두 과제를 수행하는 것입니다. 프로세스가 사용하는 메모리 주소는 어디에 저장하고 있는지와 직접 메모리 주소를 할당하고 사용해보는 과정에 대해서 이해하는 과정이 필요합니다.

2. Conclusion & Analysis

Assignment 4-1



Assignment 4-1의 결과 화면입니다. Test에서는 syscall 함수를 통해서 ftrace를 hooking한 모듈을 실행하게 됩니다. 따라서 이러한 출력 결과를 볼 수 있었습니다.

해당 과제에서는 실행한 프로세스의 pid에서 vm_area_struct를 찾고 이 vm_area_struct가 가리키는 mm_struct를 토대로 출력하게 됩니다. 이 때 경로를 출력하게 되는데 경로는 vm_area_struct가 가리키는 vm_file의 f_path에서 확인할 수 있었습니다. 하지만 이 경로가 NULL을 가리키는 경우 또한 존재했습니다.

이 경로가 NULL을 가리키는 경우에는 해당하는 프로세스의 코드 구역이 아닌 스택이나 힙 영역을 참조하였기 때문입니다. 스택과 힙 영역의 메모리는 파일에 직접 매핑되지 않는다는 것을 알게 되었습니다.

Assignment 4-2

Assignment 4-2에서는 dynamic compile을 구현한 코드와 구현하지 않은 코드를 반복하여 50번 수행하여 결과값을 평균으로 결과의 차이를 확인합니다.

또한 해당하는 함수, 즉 func(1)의 수행 시간이 단 한번 수행될 경우 큰 차이를 보이지 못하는 것을 확인했습니다.

```
os2018202018@ubuntu:~/Downloads/Assignment4-2$ make
gcc -o drecompile D_recompile.c
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./test
result: 15
0.003648
Data was filled to shared memory.
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./drecompile
15
total execution time : 0.000004 sec
os2018202018@ubuntu:~/Downloads/Assignment4-2$ make dynamic
gcc -Ddynamic -o drecompile D_recompile.c
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./test
result: 15
0.003648
Data was filled to shared memory.
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./drecompile
15
total execution time : 0.000002 sec
os2018202018@ubuntu:~/Downloads/Assignment4-2$
```

해당 사진과 같이 큰 차이를 보이지 못하는 것을 확인할 수 있었습니다.

```
for (int i = 0; i < 1000; i++) {
    result = func(1);
}
```

따라서 실제 코드에서는 이를 1000번 반복하여 유의미한 결과 차이를 출력할 수 있도록 확인하는 방식을 택했습니다. 이는 테스트 중에서만 사용하였습니다.

```
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./test
result: 15
0.003648
Data was filled to shared memory.
os2018202018@ubuntu:~/Downloads/Assignment4-2$ make
gcc -o drecompile D_recompile.c
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./drecompile
15
total execution time : 0.001353 sec
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./test
result: 15
0.003648
Data was filled to shared memory.
os2018202018@ubuntu:~/Downloads/Assignment4-2$ make dynamic
gcc -Ddynamic -o drecompile D_recompile.c
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ./drecompile
15
total execution time : 0.000273 sec
os2018202018@ubuntu:~/Downloads/Assignment4-2$
```

이러한 방식으로 테스트했을 경우 오차를 감안한 차이를 확인할 수 있었습니다.

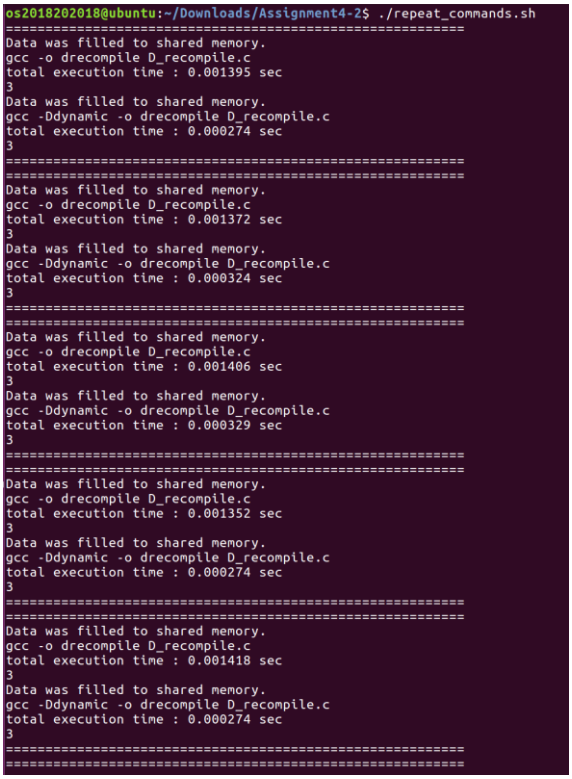
이번 테스트는 동적 컴파일을 수행하는 코드와 그렇지 않은 코드를 반복하여 50세트 수행한 후 이의 평균 결과를 출력하는 것이 목표입니다.

따라서 위의 명령어를 좀 더 빠르게 테스트하기 위해서 repeat_commands.sh 파일을 생성하였습니다.

```
$ repeat_commands.sh
cd ~/Downloads/Assignment4-2

for i in {1..50}
do
    ./test
    make
    ./drecompile
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
    ./test
    make dynamic
    ./drecompile
    sync
    echo 3 | sudo tee /proc/sys/vm/drop_caches
done
```

해당 그림과 같이 쉘 스크립트 파일을 생성한 후 이를 실행하여 나오는 결과값을 확인한 후 평균값을 계산하였습니다.



최적화 전 : 1395 + 1372 + 1406 + 1352 + 1418 = 6943μs

최적화 후 : 274 + 324 + 329 + 274 + 274 = 1475μs

```
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001447 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000315 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001388 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000293 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001417 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000294 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001374 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000275 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001372 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000296 sec
3
============================================================
============================================================
```

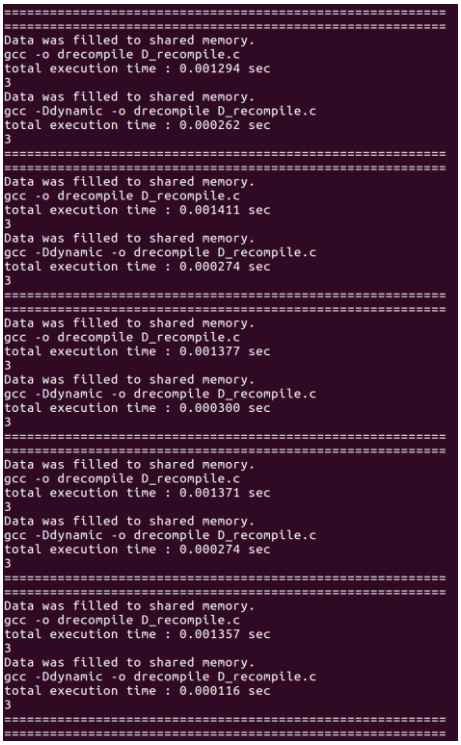
최적화 전 : 1447 + 1388 + 1417 + 1374 + 1372 = 6998μs

최적화 후 : 315 + 293 + 294 + 275 + 296 = 1473μs

```
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001384 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001333 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000423 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001364 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000243 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001343 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001353 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000305 sec
3
============================================================
============================================================
```

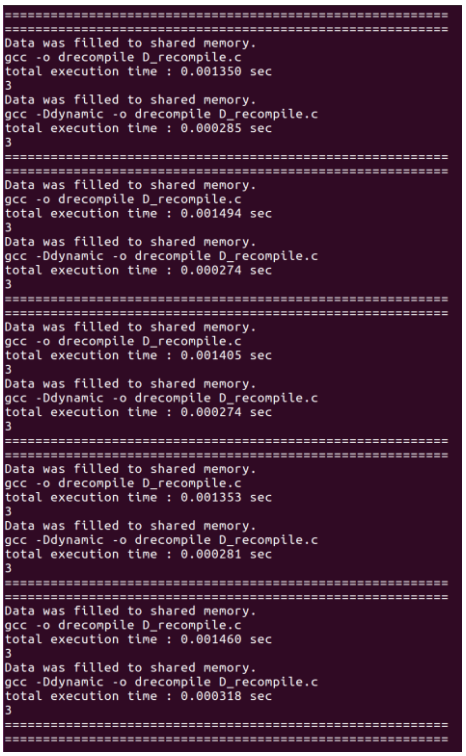
최적화 전 : 1384 + 1333 + 1364 + 1343 + 1353 = 6777μs

최적화 후 : 274 + 423 + 243 + 274 + 305 = 1519μs



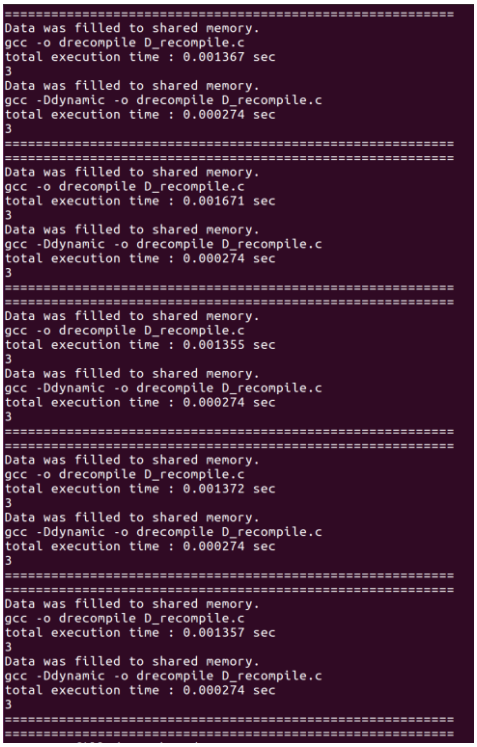
최적화 전 : 1294 + 1411 + 1377 + 1371 + 1357 = 6810μs

최적화 후 : 262 + 274 + 300 + 274 + 116 = 1226μs



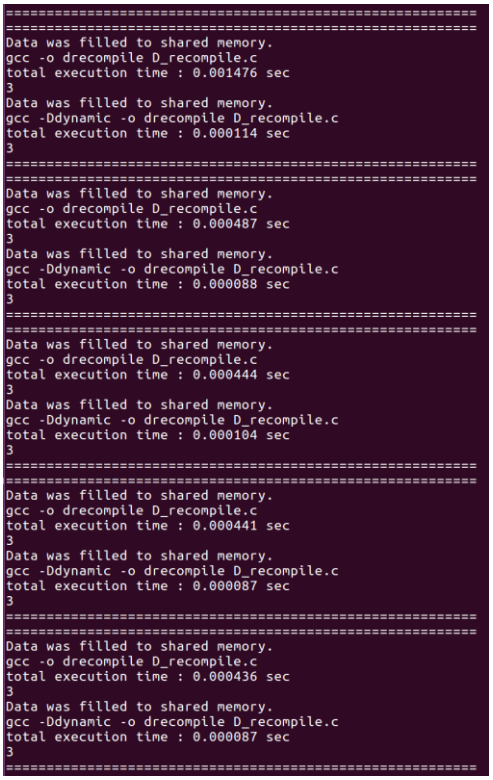
최적화 전 : 1350 + 1494 + 1405 + 1353 + 1460 = 7062μs

최적화 후 : 285 + 274 + 274 + 281 + 318 = 1432μs



최적화 전 : 1367 + 1671 + 1355 + 1372 + 1357 = 7122μs

최적화 후 : 274 + 274 + 274 + 274 + 274 = 1370μs



최적화 전 : 1476 + 487 + 444 + 441 + 436 = 3284μs

최적화 후 : 114 + 88 + 104 + 87 + 87 = 480μs

```
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.000425 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001428 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001051 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.000607 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000273 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001384 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
```

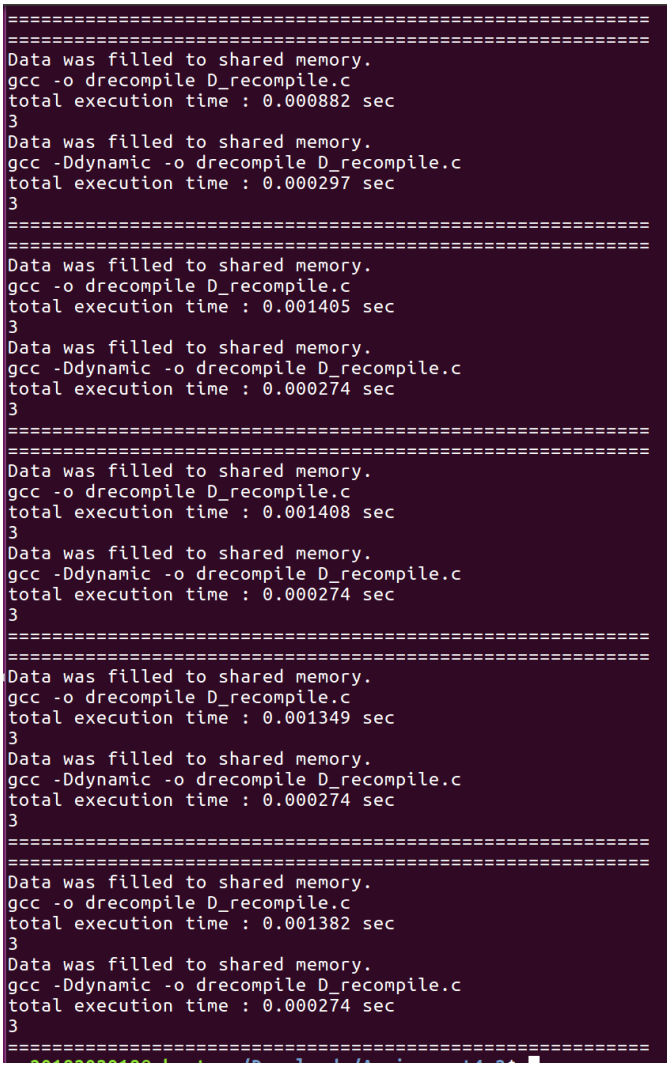
최적화 전 : 425 + 1428 + 1051 + 607 + 1384 = 4895μs

최적화 후 : 274 + 274 + 274 + 273 + 274 = 1369μs

```
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001216 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000273 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001413 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.001194 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000279 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.000503 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000087 sec
3
============================================================
============================================================
Data was filled to shared memory.
gcc -o drecompile D_recompile.c
total execution time : 0.000425 sec
3
Data was filled to shared memory.
gcc -Ddynamic -o drecompile D_recompile.c
total execution time : 0.000274 sec
3
============================================================
```

최적화 전 : 1216 + 1413 + 1194 + 503 + 425 = 4751μs

최적화 후 : 273 + 274 + 279 + 87 + 274 = 1187μs



최적화 전 : 882 + 1405 + 1408 + 1349 + 1382 = 6426μs

최적화 후 : 297 + 274 + 274 + 274 + 274 = 1393μs

쉘 스크립트 파일을 실행한 결과는 이러한 방식으로 50번 출력되는 것을 확인할 수 있었습니다.

각 평균을 구해봤을 때

최적화 전 : 6943 + 6998 + 6777 + 6810 + 7062 + 7122 + 3284 + 4895 + 4751 + 6426 = 61068μs / 50 = 1221.36μs 소요

최적화 후 : 1475 + 1473 + 1519 + 1226 + 1432 + 1370 + 480 + 1369 + 1187 + 1393 = 12924μs / 50 = 258.48μs 소요

가 나타나는 것을 확인할 수 있었습니다.

최적화 전보다 최적화 후가 대략 4.72배 더 빠르게 수행되는 것을 확인할 수 있었습

니다.

Objdump 분석 방법

```
os2018202018@ubuntu:~/Downloads/Assignment4-2$ objdump -d D_recompile_test.c > disassembly
```

```
os2018202018@ubuntu:~/Downloads/Assignment4-2$ ls
disassembly  drecompile  D_recompile.c  D_recompile_test.c  D_recompile_test.o  Makefile  repeat_commands.sh  test
os2018202018@ubuntu:~/Downloads/Assignment4-2$
```

Disassembly 파일을 objdump를 통해 만들었습니다. 이는 D_recomplie_test.c를 Disassembly한 결과입니다.

disassembly

```
1
2   D_recompile_test.o:     file format elf64-x86-64
3
4
5   Disassembly of section .text:
6
7   0000000000000000 <Operation>:
8      0: 55                    push   %rbp
9      1: 48 89 e5              mov    %rsp,%rbp
10     4: 89 7d fc              mov    %edi,-0x4(%rbp)
11     7: 8b 55 fc              mov    -0x4(%rbp),%edx
12     a: 89 d0                 mov    %edx,%eax
13     c: b2 02                 mov    $0x2,%dl
14     e: 83 c0 01              add    $0x1,%eax
15    11: 83 c0 01              add    $0x1,%eax
16    14: 83 c0 01              add    $0x1,%eax
17    17: 83 c0 01              add    $0x1,%eax
18    1a: 83 c0 02              add    $0x2,%eax
19    1d: 83 c0 03              add    $0x3,%eax
20    20: 83 c0 01              add    $0x1,%eax
21    23: 83 c0 02              add    $0x2,%eax
22    26: 83 c0 01              add    $0x1,%eax
23    29: 83 c0 01              add    $0x1,%eax
24    2c: 6b c0 02              imul   $0x2,%eax,%eax
25    2f: 6b c0 02              imul   $0x2,%eax,%eax
26    32: 6b c0 02              imul   $0x2,%eax,%eax
27    35: 83 c0 01              add    $0x1,%eax
28    38: 83 c0 01              add    $0x1,%eax
29    3b: 83 c0 03              add    $0x3,%eax
30    3e: 83 c0 01              add    $0x1,%eax
31    41: 83 c0 01              add    $0x1,%eax
32    44: 83 c0 01              add    $0x1,%eax
33    47: 83 c0 03              add    $0x3,%eax
34    4a: 83 c0 01              add    $0x1,%eax
35    4d: 83 c0 01              add    $0x1,%eax
36    50: 83 c0 02              add    $0x2,%eax
37    53: 83 c0 01              add    $0x1,%eax
38    56: 83 c0 01              add    $0x1,%eax
39    59: 83 c0 01              add    $0x1,%eax
40    5c: 83 c0 01              add    $0x1,%eax
41    5f: 83 c0 01              add    $0x1,%eax
42    62: f6 f2                 div    %dl
43    64: f6 f2                 div    %dl
44    66: 83 e8 01              sub    $0x1,%eax
```

이 파일은 이러한 실제로 어떠한 바이트가 메모리에 할당되고 이는 무슨 어셈블리 명령어를 수행하는지를 나타내주고 있습니다.

위의 <Operation>은 D_recompile_test.c의 어셈블리어로 이루어진 Operation 함수를 의미하고 이는 저러한 바이트들로 구성되어있고, 어떠한 동작을 하는지를 나타내어주는 파

일이라고 분석했습니다. 또한 이 Operation 함수가 실제로 컴파일되는 코드이기 때문에 이 코드에서 어떠한 바이트가 들어갈 시 어떠한 명령어를 수행하는 지를 파악했습니다.

Add : 83 c0

Sub : 83 e8

Imul : 6b c0

Div : f6 f2

이러한 비트는 해당 명령어를 나타낸다는 것을 알아냈습니다.

그리고 add, sub, imul은 뒤에 상수가 따라 붙어 이 상수와 %eax 레지스터와 해당 연산을 진행하고 결과값을 %eax 레지스터에 저장하는 것을 알게 되었습니다. Div는 다른 명령어와는 다르게 %eax의 값을 %dl의 값에 저장되어있는 값으로 나누게 되고, 이 값을 %eax에 저장하게 되는 연산입니다. 따라서 저는 %dl에 값을 저장하는 과정에서 recompile시에 그 값을 divc라는 변수에 저장했습니다. 이렇게 명령어를 어떻게 바이트 단위로 저장할지 분석을 하게 되었습니다. 따라서 recompile 시에 같은 명령어가 반복되는 과정을 거칠 경우 add를 여러 번 하는 것이 아닌 add 명령어를 통해 여러 번 반복하여 더해질 숫자를 하나의 변수에 저장하고, 이를 add가 아닌 다른 명령어가 나타나게 될 경우 그 변수의 수만큼 add 명령어를 수행하는 방식으로 진행했습니다. 이렇게 진행할 경우 같은 명령어를 반복하여 진행하지 않고 한 번의 명령어 수행을 통해 같은 기능을 하도록 구현할 수 있었습니다. Add와 sub는 해당 명령어가 수행될 때 사용되는 상수만큼 변수를 더해주었고, imul과 div는 상수만큼 변수를 곱해주었습니다.

```
int i = 0;
uint8_t add = 0;
uint8_t sub = 0;
uint8_t mul = 1;
uint8_t div = 1;
uint8_t divc = 1;
```

이런 방식으로 첫 변수를 초기화하고 사용했습니다.

3. Consideration

이번 과제는 프로세스가 실행될 때 메모리와 관련된 정보들을 어디에 저장하고 있는지를 확인하고 출력하는 Assignment 4-1과 어셈블리 코드를 공유 메모리에 저장하고 이를 새로운 메모리 공간을 할당하여 복사한 후 write 권한을 부여해 수정, 실행 권한을 할당하고 이를 실행한 코드를 최적화하지 않은 코드와 어느 정도의 성능 차이를 가지는 지를 확인하는 Assignment 4-2를 진행했습니다. 4-1같은 경우에는 지금까지 사용했던

task_struct의 새로운 변수들을 활용하는 과정이었지만 크게 어려운 점이 없었습니다. 4-2에서는 Disassembly code를 해석하는 과정에서부터 꽤 오랜 시간을 소요하게 되었습니다. 공유 메모리에 코드를 저장하는 과정이 이러한 바이트 단위로 나누어 메모리 공간에 저장된다는 것을 생각하는 것이 오래 걸렸습니다. 또한 4-2에서는 함수 포인터를 사용하게 되는 과정에서 함수 포인터라는 문법적인 부분에서 이것이 정확하게 어떠한 역할을 하게 되는 것인지 알지 못하여 처음에 공유된 코드를 잘못 이해하여 구현에 많은 시간을 들였던 것 같습니다.

4. Reference

강의자료를 이용했습니다.