

운영체제 실습 Report

실습 제목: Assignment 4

실습일자: 2022년 10월 11일 (금)

제출일자: 2022년 12월 01일 (목)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 금요일 5,6교시

학 번: 2018202065

성 명: 박 철 준

● Introduction

1. 제목

Assignment 4

2. 과제 요구사항

Assignment 4-1

- ID를 바탕으로 아래와 같은 프로세스 정보를 출력하는 Module 작성

1. 프로세스의 이름과 pid
2. 정보가 위치하는 가상 메모리 주소
3. 프로세스의 데이터 주소, 코드 주소, 힙 주소
4. 정보의 원본 파일의 전체 경로

- file_varea.c 프로그램

1. 2차 과제에서 작성한 ftrace 시스템 콜(336번)을 다음 함수로 wrapping 할것
2. Hooking 함수 명: file_varea
3. 테스트용 프로그램 명 : assign4
4. struct task_struct 분석
5. pid_task()사용

Assignment 4-2

- Dynamic Recompilation

1. Shared memory에서 컴파일 된 코드에 접근
2. Code section의 함수는 마음대로 수정 할 수 없음
3. 해당 영역에 write 권한이 없기 때문 (r-x 상태)
4. Write 권한이 있는 영역을 할당 (uint8_t *compiled_code, 전역 변수로 구현)
5. 그 영역에 함수를 복사 (rw- 상태)
6. 복사한 함수를 최적화
7. 하드코딩 방지한다.
8. 최적화할 명령어는 add, sub, imul, div
9. 0 division, -로 인한 음수에 대한 예외는 없다고 가정
10. 연산이 중복으로 나올 경우 하나로 합치기 (아래는 예 참조)
11. add instruction이 중복으로 나오면 합치기 (최적화)
12. sub instruction이 중복으로 나오면 합치기 (최적화)
13. multiple instruction이 중복으로 나오면 합치기 (최적화)
14. Division instructio이 중복으로 나오면 합치기 (최적화)
15. 수정한 함수를 실행한다.
16. 수정한 함수에는 execute 권한이 없으므로 권한 변경: r-x
17. 컴파일한 코드와 기존 코드의 실행 시간을 비교 보고서에 명시
18. 50 set 실험한 값을 표로 나타내고, 그 값에 평균을 취할 것
19. 1 set은 다음과 같은 과정을 가진다.
20. 캐시 및 버퍼 지우기
21. 최적화하기 전 결과

22. 캐시 및 버퍼 지우기

23. 최적화 후 결과

- 보고서 추가 작성

1. objdump 뜨는 과정을 거친다.
2. dump 뜬 파일의 화면을 보고 문제 해결과정에 대해 생각해본다.
3. Redirection을 사용하여 objdump 결과를 file로 저장한다.

제한조건

1. 메모리 할당은 mmap() 사용한다.
2. 할당한 메모리의 권한은 write와 execute를 동시에 부여하지 않는다.
3. DIV 연산 시 register는 dl을 사용
4. Makefile을 통해 조건부 컴파일을 진행한다.

● Conclusion & Analysis

Assignment 4-1

- file_varea.c 프로그램

```
static asmlinkage pid_t file_varea(const struct pt_regs* regs)
{
    //hooking ㉠ ㉡ ㉢ ㉣ ㉤ ㉥ ㉦ ㉧ ㉨ ㉩ ㉪ ㉫ ㉬ ㉭ ㉮ ㉯ ㉰ ㉱ ㉲ ㉳ ㉴ ㉵ ㉶ ㉷ ㉸ ㉹ ㉺ ㉻ ㉼ ㉽ ㉾ ㉿
    struct task_struct* t;
    struct mm_struct* mm; //memory management struct
    struct vm_area_struct* mmap; // vm_area_struct
    char* file_path;
    char* buf;
    struct file* file;
    t = pid_task(find_vpid(regs->di), PIDTYPE_PID);
    mm = get_task_mm(t);
    buf = kmalloc(512, GFP_KERNEL); // using malloc in kernel
    printk("##### Loaded files of a process '%s([%d])' in VM #####\n", t->comm, t->pid);
    mmap = mm->mmap;
    while(mmap)
    {
        memset(buf, 0, 512);
        file = mmap->vm_file; // ㉠ ㉡ ㉢ ㉣ ㉤ ㉥ ㉦ ㉧ ㉨ ㉩ ㉪ ㉫ ㉬ ㉭ ㉮ ㉯ ㉰ ㉱ ㉲ ㉳ ㉴ ㉵ ㉶ ㉷ ㉸ ㉹ ㉺ ㉻ ㉼ ㉽ ㉾ ㉿
        if(file) // ㉠ ㉡ ㉢ ㉣ ㉤ ㉥ ㉦ ㉧ ㉨ ㉩ ㉪ ㉫ ㉬ ㉭ ㉮ ㉯ ㉰ ㉱ ㉲ ㉳ ㉴ ㉵ ㉶ ㉷ ㉸ ㉹ ㉺ ㉻ ㉼ ㉽ ㉾ ㉿
        {
            file_path = d_path(&file->f_path, buf, 512);
            printk(KERN_INFO "mem[%lx-%lx] code[%lx-%lx] data[%lx-%lx] heap[%lx-%lx] %s\n", mmap->vm_start, mmap->vm_end, mm->start_code, mm->end_code, mm->start_data, mm->end_data, mm->start_brk, mm->brk,
            file_path);
        }
        mmap = mmap->vm_next; // ㉠ ㉡ ㉢ ㉣ ㉤ ㉥ ㉦ ㉧ ㉨ ㉩ ㉪ ㉫ ㉬ ㉭ ㉮ ㉯ ㉰ ㉱ ㉲ ㉳ ㉴ ㉵ ㉶ ㉷ ㉸ ㉹ ㉺ ㉻ ㉼ ㉽ ㉾ ㉿
    }
    printk("#####\n");
    kfree(buf); // using free in kernel
    return 0;
}
```

stack로부터 PID 값을 받고 pid_task함수를 통해 해당 task를 가져온다. 이후 get_task_mm 함수를 통해 memory management struct를 가져오고 해당 struct에서 vm_area_struct를 나타내는 mmap을 추출한다. 이후 mmap의 vm_file struct를 통해 vma(Virtual Memory Area)에 해당하는 파일 구조체에 대한 포인터를 받아오고 해당 값이 NULL이 아닌 경우 mmap의 mem 주소 mm의 code, data, heap영역과 vm_file을 통해 받아온 file에 구조체에 대한 포인터에서 d_path 함수를 통해 file의 주소를 추출하는 과정을 이룬다.

- 결과

```
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ make clean
make -C /lib/modules/4.19.67-2018202065/build SUBDIRS=/home/os2018202065/os_4_2018202065_C/4-1 clean
make[1]: Entering directory '/home/os2018202065/Downloads/linux-4.19.67'
CLEAN /home/os2018202065/os_4_2018202065_C/4-1/.tmp_versions
CLEAN /home/os2018202065/os_4_2018202065_C/4-1/Module.symvers
make[1]: Leaving directory '/home/os2018202065/Downloads/linux-4.19.67'
rm -f assign4
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ make
make -C /lib/modules/4.19.67-2018202065/build SUBDIRS=/home/os2018202065/os_4_2018202065_C/4-1 modules
make[1]: Entering directory '/home/os2018202065/Downloads/linux-4.19.67'
CC [M] /home/os2018202065/os_4_2018202065_C/4-1/file_varea.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/os2018202065/os_4_2018202065_C/4-1/file_varea.mod.o
LD [M] /home/os2018202065/os_4_2018202065_C/4-1/file_varea.ko
make[1]: Leaving directory '/home/os2018202065/Downloads/linux-4.19.67'
gcc -o assign4 test.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ sudo insmod file_varea.ko
[sudo] password for os2018202065:
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ ./assign4
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ sudo rmmod file_varea
os2018202065@ubuntu:~/os_4_2018202065_C/4-1$ dmesg

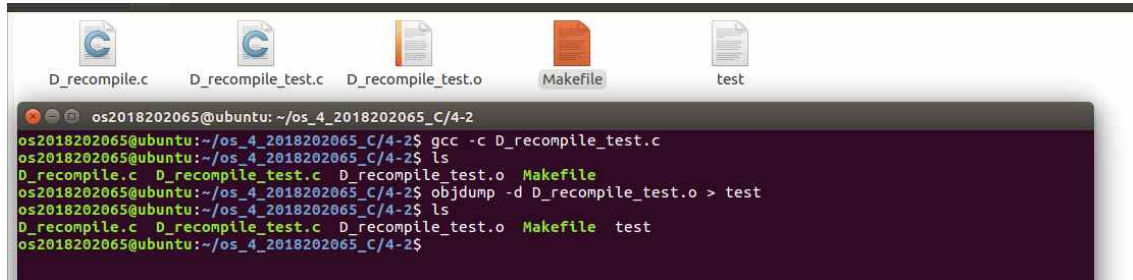
125.186351 ##### Loaded files of a process 'assign4([3547])' in VM #####
125.186357 mem[400000-401000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /home/os2018202065/os_4_2018202065_C/4-1/assign4
125.186359 mem[600000-601000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /home/os2018202065/os_4_2018202065_C/4-1/assign4
125.186361 mem[601000-602000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /home/os2018202065/os_4_2018202065_C/4-1/assign4
125.186363 mem[7f115ab4d000-7f115ad0d000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/libc-2.23.so
125.186365 mem[7f115ad0d000-7f115af0d000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/libc-2.23.so
125.186367 mem[7f115af0d000-7f115af11000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/libc-2.23.so
125.186368 mem[7f115af11000-7f115af13000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/libc-2.23.so
125.186370 mem[7f115af17000-7f115af3d000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/ld-2.23.so
125.186372 mem[7f115b13d000-7f115b13d000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/ld-2.23.so
125.186374 mem[7f115b13d000-7f115b13e000] code[400000-40074c] data[600e10-601040] heap[25d2000-25d2000] /lib/x86_64-linux-gnu/ld-2.23.so
125.186375 #####
```

요구하는 결과가 잘나옴을 알 수 있다.

- Dynamic Recompilation

1. 과제 해결을 위해 dump 파일 생성

-명령어 입력 후 파일 생성



```
os2018202065@ubuntu: ~/os_4_2018202065_C/4-2
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ gcc -c D_recompile_test.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ls
D_recompile.c  D_recompile_test.c  D_recompile_test.o  Makefile  test
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ objdump -d D_recompile_test.o > test
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ls
D_recompile.c  D_recompile_test.c  D_recompile_test.o  Makefile  test
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$
```

- 파일의 화면

D_recompile_test.o: file format elf64-x86-64

Disassembly of section .text:

```
0000000000000000 <Operation>:
0: 55                push    %rbp
1: 48 89 e5          mov     %rsp,%rbp
4: 89 7d fc          mov     %edi,-0x4(%rbp)
7: 8b 55 fc          mov     -0x4(%rbp),%edx
a: 89 d0             mov     %edx,%eax
c: b2 02            mov     $0x2,%dl
e: 83 c0 01          add     $0x1,%eax
11: 83 c0 01          add     $0x1,%eax
14: 83 c0 01          add     $0x1,%eax
17: 83 c0 01          add     $0x1,%eax
1a: 83 c0 02          add     $0x2,%eax
1d: 83 c0 03          add     $0x3,%eax
20: 83 c0 01          add     $0x1,%eax
23: 83 c0 02          add     $0x2,%eax
26: 83 c0 01          add     $0x1,%eax
29: 83 c0 01          add     $0x1,%eax
2c: 6b c0 02          imul    $0x2,%eax,%eax
2f: 6b c0 02          imul    $0x2,%eax,%eax
32: 6b c0 02          imul    $0x2,%eax,%eax
35: 83 c0 01          add     $0x1,%eax
38: 83 c0 01          add     $0x1,%eax
3b: 83 c0 03          add     $0x3,%eax
3e: 83 c0 01          add     $0x1,%eax
41: 83 c0 01          add     $0x1,%eax
44: 83 c0 01          add     $0x1,%eax
47: 83 c0 03          add     $0x3,%eax
4a: 83 c0 01          add     $0x1,%eax
4d: 83 c0 01          add     $0x1,%eax
50: 83 c0 02          add     $0x2,%eax
53: 83 c0 01          add     $0x1,%eax
56: 83 c0 01          add     $0x1,%eax
59: 83 c0 01          add     $0x1,%eax
5c: 83 c0 01          add     $0x1,%eax
5f: 83 c0 01          add     $0x1,%eax
62: f6 f2            div     %dl
64: f6 f2            div     %dl
66: 83 e8 01          sub     $0x1,%eax
69: 83 e8 01          sub     $0x1,%eax
6c: 83 e8 03          sub     $0x3,%eax
6f: 83 e8 01          sub     $0x1,%eax
72: 83 e8 01          sub     $0x1,%eax
75: 83 e8 01          sub     $0x1,%eax
78: 83 e8 03          sub     $0x3,%eax
7b: 83 e8 01          sub     $0x1,%eax
7e: 83 e8 01          sub     $0x1,%eax
```

- 최적화 문제 해결과정

먼저 dump된 파일을 통해 asm 코드대해 알 수 있었고 이는 다음과 같다.

dl 레지스터(b2), add(83 c0), imul(6b, c0), sub (83 e8), div(f6 f2), retq(c3)

이를 통해 최적화하기 위한 compiled_code에는 해당 16진수의 숫자들이 들어가야 하며 중복된 asm 코드값이 나온 경우 상수 부분을 더하고 명령어 개수를 축소하여 저장하는 과정과 이러한 과정의 반복중 retq를 통해 asm 코드의 끝을 알 수 있게 하는 방법에 대해 생각해 보고 이를 통해 구현하였다.

2. D_recompile.c

- 전역 변수 및 함수

```
uint8_t* Operation;
uint8_t* compiled_code;

int segment_id;
int fd;
int pagesize = 0;
```

- main 함수

```
int main(void)
{
    pagesize = getpagesize();
    int (*func)(int a);
    int i;
    struct timespec begin, end; //시간을 측정하기 위한 변수
    time_t sec;
    long nsec;
    sharedmem_init();
    drecompile_init();
    func = (int (*)(int a))drecompile(Operation);
    clock_gettime(CLOCK_MONOTONIC, &begin);
    func(1); //compiled code excute
    clock_gettime(CLOCK_MONOTONIC, &end);
    sec = end.tv_sec - begin.tv_sec;
    nsec = end.tv_nsec - begin.tv_nsec;
    if (nsec < 0) nsec += 1000000000;
    printf("total excution time: %3ld.%09lds\n", sec, nsec);
    drecompile_exit();
    sharedmem_exit();
    return 0;
}
```

시간을 측정하기 위해 다음과 같은 방식으로 설정

- sharedmem_init, sharedmem_exit, drecompile_init, drecompile_exit 함수

```
49 void sharedmem_init()
50 {
51     segment_id = shmget(1234, pagesize, 0); // get shared memory
52     Operation = (uint8_t*)shmat(segment_id, NULL, 0); // attach
53 }
54
55 void sharedmem_exit()
56 {
57     shmdt(Operation); // detach
58     shmctl(segment_id, IPC_RMID, NULL); // remove shared memory
59 }
60
61 void drecompile_init(uint8_t* func)
62 {
63     char temp[pagesize];
64     fd = open("Operation.txt", O_RDWR|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);
65     /*이래쓰 bus error 방지 과정*/
66     for(int n = 0; n < PAGE_SIZE; n++) temp[n] = 'M';
67     write(fd, temp, pagesize);
68     lseek(fd, 0, 0);
69     /*-----*/
70     compiled_code = mmap(0, pagesize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0); // memory mapping
71     assert(compiled_code != MAP_FAILED); //error detection
72     memset(compiled_code, 0, pagesize); //메모리 매핑된 공간 초기화
73     msync(compiled_code, pagesize, MS_SYNC); //메모리 매핑된 변경된 사항을 파일에 반영 (동기화) MS_SYNC 정책
74 }
75
76
77 void drecompile_exit()
78 {
79     munmap(compiled_code, pagesize); // unmap
80 }
81
```

sharedmem_init을 통해 D_recompile_test.c의 어셈블리 함수들을 저장한 shared memory에 접근한다. 방식은 shmget 함수를 통해 커널에 shared memory segment를 위한 공간 요청을 하고 shmat 함수를 통해 현재의 프로세스가 생성된 shared memory를 사용할 수 있도록 id 값을 이용하여 덧붙임(attach) 작업 수행한다. drecompile_init 함수를 통해 Operation.txt 파일에 대한 메모리 매핑을 진행한다. 방식은 mmap 함수를 통해 읽고 쓰기가 가능한 권한의 메모리 매핑영역을 할당하고 해당 크기는 pagesize 4096임 다음으로 메모리 매핑 공간을 초기화하여 저장에 용이하게 하였다. msync를 통해 변경된 사항을 파일에 반영한다. 정책은 MS_SYNC로 동기화를 요청하고 끝날 때까지 대기한다. 모든 과정과 최적화 한 함수를 실행하고 결과를 추출하였으면 sharedmem_exit을 통해 공유 메모리를 해제한다. 방식은 shmdt를 통해 프로세스와 공유 메모리를 detach하고 drecompile_exit 함수의 munmap함수를 통해 메모리 매핑 영역을 unmap한다.

- drecompile 함수

```
82 void* drecompile(uint8_t* func)
83 {
84     int dynamic_check=0;
85     #ifdef dynamic
86         dynamic_check=1;
87         int i = 0;
88         int j = 0;
89         // for all function instruction
90         uint8_t dl=0;
91         int dl_register=0;
92
93         while (func[i] != 0xc3)
94         {
95             if(func[i]==0xb2) //finding dl register
96             {
97                 dl=func[i+1];
98                 dl_register=i;
99             }
100
101             if (func[i] != 0x83 && func[i] != 0x6b && func[i] != 0xf6)
102             {
103                 // operation, source register
104                 // op is not target of optimization
105                 compiled_code[j++] = func[i++];
106             }
107             else
108             {
109                 uint8_t add_count=0;
110                 uint8_t sub_count=0;
111                 uint8_t imul_count=1;
112                 uint8_t div_count=1;
113                 int k = 0;
114                 if (func[i] == 0x83)
115                 {
116                     if (func[i + 1] == 0xc0) //add
117                     {
118                         k = i + 3;
119                         add_count+=func[i+2];
120                         while (func[k] == func[i] && func[k+1] == func[i+1])
121                         {
122                             add_count+=func[k+2];
123                             k = k + 3;
124                         }
125                         compiled_code[j++] = func[i];
126                         compiled_code[j++] = func[i+1];
127                         compiled_code[j++] = add_count;
128                         i=k;
```



```

129         continue;
130     }
131     else if (func[i + 1] == 0xe8)//sub
132     {
133         k = i + 3;
134         sub_count += func[i+2];
135         while (func[k] == func[i] && func[k+1] == func[i+1])
136         {
137             sub_count += func[k+2];
138             k = k + 3;
139         }
140
141         compiled_code[j++] = func[i];
142         compiled_code[j++] = func[i+1];
143         compiled_code[j++] = sub_count;
144         i = k;
145         continue;
146     }
147     else//exception handling
148     {
149         compiled_code[j++] = func[i++];
150         continue;
151     }
152 }
153 else if (func[i] == 0x6b)
154 {
155     if (func[i + 1] == 0xc0)//imul
156     {
157         k = i + 3;
158         imul_count *= func[i+2];
159         while (func[k] == func[i] && func[k+1] == func[i+1])
160         {
161             imul_count *= func[k+2];
162             k = k + 3;
163         }
164         compiled_code[j++] = func[i];
165         compiled_code[j++] = func[i+1];
166         compiled_code[j++] = imul_count;
167         i = k;
168         continue;
169     }
170     else//exception handling
171     {
172         compiled_code[j++] = func[i++];
173         continue;
174     }
175 }

```

```

175
176     }
177     else if (func[i] == 0xf6)
178     {
179         if (func[i + 1] == 0xf2) //div
180         {
181             k = i + 2;
182             div_count*=dl;
183             while (func[k] == func[i] && func[k+1] == func[i+1])
184             {
185                 div_count*=dl;
186                 k = k + 2;
187             }
188             compiled_code[j++]=func[i];
189             compiled_code[j++]=func[i+1];
190             compiled_code[dl_register+1]=div_count;
191             i=k;
192             continue;
193         }
194         else //exception handling
195         {
196             compiled_code[j++] = func[i++];
197             continue;
198         }
199     }
200 }
201 }
202 // end of func
203 compiled_code[j] = func[i]; //end
204 msync(compiled_code, pagesize, MS_SYNC);
205 #endif

```

```

206 if(dynamic_check==0) // case of no dynamic
207 {
208     //copiled_code 기존 그대로 매핑된 영역에 저장
209     int b=0;
210     while (func[b] != 0xc3)
211     {
212         compiled_code[b] = func[b];
213         b++;
214     }
215     compiled_code[b] = func[b];
216     msync(compiled_code, pagesize, MS_SYNC); //메모리 매핑 후 변경된 사항을 파일에 반영 (동기화) MS_SYNC 정책
217 }
218 if(mprotect(compiled_code, pagesize, PROT_READ | PROT_EXEC)) // set protection RX
219 {
220     printf("set protection RX error using mprotect\n");
221 }
222 return compiled_code;
223 }
224
225
226

```

ifdef dynamic ~ endif 을 통해 조건부 컴파일이 가능하도록하였고 dynamic recompilation 이 적용될 경우 메모리 맵핑 공간인 compiled_code 영역에 기존의 공유 메모리가 가지고 있는 operation을 최적화하여 저장하는데 과정은 다음과 같다.

1. add의 경우 처음 add를 발견한 경우 while문을 통해 다음 operation이 add인지 확인하고 카운트 후 while문 종료 후 add operation의 asm 코드를 compiled_code에 한 번 적고 카운트하면서 합산한 상수를 다음으로 적는다.

2. sub의 경우 처음 sub를 발견한 경우 while문을 통해 다음 operation이 sub인지 확인하고 3번째 인자인 상수를 카운트 후 while문 종료 후 sub operation의 asm 코드를 compiled_code에 한 번 적고 카운트하면서 합산한 상수를 다음으로 적는다.

3. imul의 경우 처음 imul를 발견한 경우 while문을 통해 다음 operation이 imul인지 확인하고 3번째 인자인 상수를 카운트 후 while문 종료 후 imul operation의 asm 코드를 compiled_code에 한 번 적고 카운트하면서 합산한 상수를 다음으로 적는다.

4. div의 경우 처음 div를 발견한 경우 while문을 통해 div연산 횟수를 카운트하고 while문이 끝나면 dl 레지스터에 저장되어있던 2값에 횟수를 곱하여 저장한다. 이후 div 연산 asm 코드를 한번 적는다.

해당 과정을 while문을 통해 retq : c3를 만날 때 까지 반복하였다.

중간에 최적화 대상인 operation 대신 다른 operation이 발견될 경우 모두 `compiled_code`에 바로 저장하는 방식으로 구현하였다. 이후 `compile_code`를 리턴한다.

dynamic recompilation이 적용이 되지 않을 시 endif 다음 영역으로 바로 넘어오게 되는데 dynamic_check 변수를 통해 이를 감지하고 compiled_code에 원래의 Operation을 저장하여 compile_code를 리턴한다.

-최적화 증거

1. 매핑한 파일인 Operation.txt 내부

[illegible]

최적화 전 내부 내용의 양과 최적화 후 내부의 내용의 양이 확연한 차이가 있음을 알 수 있다.

2. asm 코드를 실행한 결과값을 출력한 모습

```
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./test2
Data was filled to shared memory.
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ make
gcc -o drecompile D_recompile.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./drecompile
result : 15
total excution time: 0.000162094 sec
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./test2
Data was filled to shared memory.
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ make dynamic
gcc -Ddynamic -o drecompile D_recompile.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./drecompile
result : 15
total excution time: 0.000070236 sec
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$
```

최적화 전의 result와 최적화 후 result가 같음을 알 수 있다.

- 결과 도출

```

os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ sync
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ echo 3| sudo tee /proc/sys/vm/drop_caches
3
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ gcc -o test2 D_recompile_test.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./test2
Data was filled to shared memory.
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ make
gcc -o drecompile D_recompile.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./drecompile
total excution time: 0.000002123 sec
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ sync
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ echo 3| sudo tee /proc/sys/vm/drop_caches
3
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./test2
Data was filled to shared memory.
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ make dynamic
gcc -Ddynamic -o drecompile D_recompile.c
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ ./drecompile
total excution time: 0.000000888 sec
os2018202065@ubuntu:~/os_4_2018202065_C/4-2$ █

```

해당 최적화 구현 방식이 실행시간을 줄여주었기 때문에 유의미한 결과값을 가짐을 알 수 있다.

해당 방식을 50번 진행한 결과는 아래와 같다.

횟수	최적화 전	최적화 후
1	0.000000596	0.000000299
2	0.000000718	0.000000286
3	0.000000573	0.000000264
4	0.000000598	0.000000272
5	0.000000624	0.000000222
6	0.000000743	0.000000332
7	0.000000644	0.000000265
8	0.000000718	0.000000262
9	0.000000580	0.000000297
10	0.000000975	0.000000334
11	0.000000698	0.000000256
12	0.000000596	0.000000290
13	0.000000636	0.000000263
14	0.000000608	0.000000330
15	0.000000638	0.000000289
16	0.000000702	0.000000270
17	0.000000602	0.000000238
18	0.000000627	0.000000247
19	0.000000630	0.000000266
20	0.000000604	0.000000278
21	0.000000558	0.000000294
22	0.000000693	0.000000262
23	0.000000687	0.000000351
24	0.000000595	0.000000273
25	0.000000707	0.000000278
26	0.000000624	0.000000288
27	0.000000655	0.000000351
28	0.000000611	0.000000338
29	0.000000719	0.000000268
30	0.000000765	0.000000257

31	0.000000615	0.000000357
32	0.000000661	0.000000331
33	0.000000775	0.000000371
34	0.000000640	0.000000287
35	0.000000775	0.000000276
36	0.000000661	0.000000265
37	0.000000595	0.000000290
38	0.000000647	0.000000327
39	0.000000649	0.000000272
40	0.000000593	0.000000219
41	0.000000630	0.000000348
42	0.000000592	0.000000272
43	0.000000574	0.000000317
44	0.000000601	0.000000259
45	0.000000703	0.000000332
46	0.000000633	0.000000273
47	0.000000547	0.000000267
48	0.000000719	0.000000361
49	0.000000613	0.000000264
50	0.000000650	0.000000312
평균	0.00000065194	0.0000002904

평균을 구하여 나타내어 볼 때 최적화 후 실행시간이 최적화 전 실행시간에 비해 약 2배 빠른 것을 알 수 있다.

● 고찰

이번 과제를 구현하면서 발생한 이슈는 다음과 같다.

1. 메모리 �핑 공간에 memcpy, memset함수를 사용함에 있어 bus error가 뜨는 이슈
2. 최적화한 부분이 잘 적용되었음을 알기 위한 방법에 대한 이슈

이슈를 해결했던 방식은 다음과 같다.

1. 빈 파일에 맵핑한 경우 해당 맵핑 공간을 사용할 경우 bus에러가 뜬다 이를 해결하기 위해 빈 파일의 사이즈를 설정할 필요가 있다. 때문에 파일에 해당 맵핑의 길이만큼의 크기의 문자를 쓰는 과정을 넣어 사이즈를 설정해 주어 해결하였다.

2. 메모리 맵핑에 한 공간에 Operation을 저장함에 있어 최적화 전에는 더 많은 정보들이 저장되어 있을 것이고 msync를 통해 파일을 동기화 하여 변경된 사항을 반영하면 최적화 후 보다 더 많은 내용이 출력될 것임을 예상하여 해당 실험을 해보았고 이를 증명하였다. 이에 대한 자세한 과정은 보고서 결과 부분에 서술하였다.