시스템 프로그래밍 과제 1 linklab 레포트

Linker Lab: Memtrace

2019-13674
양현서
바이오시스템소재학부

Contents

1 Introduction

수업 시간에 Library interpositioning을 배웠다. 이것은 어떤 프로그램이 실행될 때 사용되는 외부 공유 라이브러리의 함수 호출을 중간에서 가로채 임의의 다른 함수를 실행할 수 있게 하는 기법이다. Compile time, Link time, Load/Run time에 가능한데, 이번 lab에서는 Load/Run time에 사용하였다.

구체적으로는 `malloc`, `free`, `calloc`, 그리고 `realloc`을 interpositioning하여 테스트 프로그램들의 메모리 할당과 해제를 추적하고, 해제되지 않은 메모리와 그 메모리를 할당한 위치 등과 같은 유용한 정보도 출력하는 라이브러리를 만든다.

2 Part 1: Tracing Dynamic Memory Allocation

Part 1에서는 테스트 프로그램들의 `malloc`, `free`, `calloc`, 그리고 `realloc` 함수들의 호출과 그 결과값을 출력하고, 할당된 메모리와 한번 호출당 할당된 메모리의 평균을 출력한다. part 1은 library interpositioning 실습의 몸풀기라고 볼 수 있다. 처음에는 `malloc`과 `free` 만이 대상인 줄 잘못 이해하였으나 나중에 `calloc`과 `realloc`도 구현하였다. handout에 나온 결과 예시를 보면 `malloc`과 `free`의 호출 정보와 결과값을 화면에 나타내고, 마지막에 총 할당 정보를 표시한다. 처음에는 `mlog` 함수를 직접 호출하여 화면과 비슷하게 출력하게 하려 하였으나, 곧 memlog.h를 다시 살펴보고 나서 `LOG_MALLOC`등과 같은 매크로들을 이용하면 된다는 것을 알게 되었다.

library interpositioning을 성공적으로 수행하기 위해, `init` 함수에 실제 `malloc`, `free`, `calloc`, 그리고 `realloc` 함수들에 대한 포인터를 초기화하는 루틴을 넣었다.

```
86   mallocp = dlsym(RTLD_NEXT, "malloc");
87   if ((error = dlerror()) != NULL) {
88      fputs(error,stderr);
89      exit(1);
90   }
91   freep = dlsym(RTLD_NEXT, "free");
92   if ((error = dlerror()) != NULL) {
93      fputs(error,stderr);
94      exit(1);
95   }
96   callocp = dlsym(RTLD_NEXT, "calloc");
97   if ((error = dlerror()) != NULL) {
98      fputs(error,stderr);
```

```
99        exit(1);
100     }
101     reallocp = dlsym(RTLD_NEXT, "realloc");
102     if ((error = dlerror()) != NULL) {
103        fputs(error,stderr);
104        exit(1);
105     }
```

여기서 RTLD_NEXT를 사용하여 현재 라이브러리가 아닌 다음 라이브러리에서 함수 심볼들을 찾으라고 링커에게 명령하였다.
part 1의 가로채어 대신 실행되는 함수들은 아래와 같이 작성하였다.

```
35  void * malloc(size_t size)
36  {
37     n_allocb += size;
38     n_malloc++;
39     void * resultP = mallocp(size);
40     LOG_MALLOC(size, resultP);
41     return resultP;
42  }
```

간단하게 총 할당 바이트 수를 나타내는 n_mallocb에 size만큼 더하고 n_malloc을 1 증가시키고 끝난다.

```
44  void free(void * ptr)
45  {
46     LOG_FREE(ptr);
47     freep(ptr);
48  }
```

free의 경우는 더 간단하게 LOG_FREE만 이용하면 된다.

```
50  void * calloc(size_t nmemb, size_t size)
51  {
52     n_calloc++;
53     void * resultP = callocp(nmemb, size);
54     LOG_CALLOC(nmemb, size, resultP);
55     if(resultP) {
56       n_allocb += size*nmemb;
57     }
58     return resultP;
59  }
```

calloc의 경우는 할당되는 최종 바이트 크기가 nmemb×size 인 것만 주의하면 malloc과 비슷하다. if(resultP) 부분은 할당의 성공을 체크하는 부분인데, calloc과 realloc을 구현하면서 오류 확인 목적으로 넣었다.

```
60  void * realloc(void *ptr, size_t new_size)
61  {
62     n_realloc++;
63     void * resultP = reallocp(ptr, new_size);
64     LOG_REALLOC(ptr, new_size, resultP);
65     if(resultP) {
66        n_allocb+= new_size;
67     }
68     return resultP;
69  }
```

realloc도 아직까지는 특별하게 처리할 것이 없다.

```
113 void fini(void)
114 {
115   // ...
```

```
116   int n = n_malloc + n_calloc + n_realloc;
117   int avg = n ? n_allocb / n : 0;
118   LOG_STATISTICS(n_allocb, avg, n_freeb);
119
120   LOG_STOP();
121
122   // free list (not needed for part 1)
123   free_list(list);
124 }
```

라이브러리가 언로드될 때 호출되는 `fini` 함수에서는 `LOG_STATISTICS` 매크로를 이용해 총 메모리 할당량과 평균 메모리 할당량을 표시한다. 평균 메모리 할당량을 구할 때 처음에는 `n_malloc`으로만 나누었다가, `n_malloc+n_calloc+n_realloc`으로 나누는 것으로 수정하였다.

2.1 테스트 결과

test1

```
user102@SystemProgramming:~/handout/part1$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : malloc( 1024 ) = 0x16e1060
[0003]           (nil) : malloc( 32 ) = 0x16e1470
[0004]           (nil) : malloc( 1 ) = 0x16e14a0
[0005]           (nil) : free( 0x16e14a0 )
[0006]           (nil) : free( 0x16e1470 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
```

아직 함수 호출자 부분은 nil로 나오고, `freed_total`도 0인 것을 볼 수 있다.

test2

```
  user102@SystemProgramming:~/handout/part1$ make run test2
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
  [0002]           (nil) : malloc( 1024 ) = 0x24c3060
  [0003]           (nil) : free( 0x24c3060 )
  [0004]
  [0005] Statistics
  [0006]   allocated_total      1024
  [0007]   allocated_avg        1024
  [0008]   freed_total          0
  [0009]
  [0010] Memory tracer stopped.
```

test3

```
  user102@SystemProgramming:~/handout/part1$ make run test3
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : malloc( 42213 ) = 0x1d51060
[0003]           (nil) : malloc( 22581 ) = 0x1d5b550
[0004]           (nil) : calloc( 1 , 727 ) = 0x1d60d90
```

```
[0005]           (nil) : calloc( 1 , 64048 ) = 0x1d61070
[0006]           (nil) : calloc( 1 , 50720 ) = 0x1d70ab0
[0007]           (nil) : malloc( 43080 ) = 0x1d7d0e0
[0008]           (nil) : calloc( 1 , 61740 ) = 0x1d87930
[0009]           (nil) : malloc( 37447 ) = 0x1d96a70
[0010]           (nil) : calloc( 1 , 37103 ) = 0x1d9fcc0
[0011]           (nil) : calloc( 1 , 59380 ) = 0x1da8dc0
[0012]           (nil) : free( 0x1da8dc0 )
[0013]           (nil) : free( 0x1d9fcc0 )
[0014]           (nil) : free( 0x1d96a70 )
[0015]           (nil) : free( 0x1d87930 )
[0016]           (nil) : free( 0x1d7d0e0 )
[0017]           (nil) : free( 0x1d70ab0 )
[0018]           (nil) : free( 0x1d61070 )
[0019]           (nil) : free( 0x1d60d90 )
[0020]           (nil) : free( 0x1d5b550 )
[0021]           (nil) : free( 0x1d51060 )
[0022]
[0023] Statistics
[0024]   allocated_total      419039
[0025]   allocated_avg        41903
[0026]   freed_total          0
[0027]
[0028] Memory tracer stopped.
```

test4

```
  user102@SystemProgramming:~/handout/part1$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : malloc( 1024 ) = 0x107c060
[0003]           (nil) : free( 0x107c060 )
[0004]           (nil) : free( 0x107c060 )
*** Error in `../test/test4': double free or corruption (top): 0x000000000107c060
↪  ***
[0005]           (nil) : malloc( 36 ) = 0x7f4b8c0008c0
[0006]           (nil) : calloc( 1182 , 1 ) = 0x7f4b8c0008f0
[0007]           (nil) : malloc( 36 ) = 0x7f4b8c000da0
[0008]           (nil) : malloc( 56 ) = 0x7f4b8c000dd0
[0009]           (nil) : calloc( 15 , 24 ) = 0x7f4b8c000e10
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f4b930647e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7f4b9306d37a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7f4b9307153c]
./libmemtrace.so(free+0x39)[0x7f4b933b7d8a]
../test/test4[0x40048e]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f4b9300d830]
../test/test4[0x4004c9]
======= Memory map: ========
00400000-00401000 r-xp 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
00600000-00601000 r--p 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
00601000-00602000 rw-p 00001000 ca:01 657555
↪  /home/user102/handout/test/test4
0107c000-0109d000 rw-p 00000000 00:00 0                                  [heap]
7f4b8c000000-7f4b8c021000 rw-p 00000000 00:00 0
7f4b8c021000-7f4b90000000 ---p 00000000 00:00 0
```

```
7f4b92bd3000-7f4b92be9000 r-xp 00000000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
... (중략)
7f4b933b1000-7f4b933b3000 rw-p 001c4000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
7f4b933b3000-7f4b933b7000 rw-p 00000000 00:00 0
7f4b933b7000-7f4b933b9000 r-xp 00000000 ca:01 657531
↪  /home/user102/handout/part1/libmemtrace.so
7f4b933b9000-7f4b935b8000 ---p 00002000 ca:01 657531
↪  /home/user102/handout/part1/libmemtrace.so
7f4b935b8000-7f4b935b9000 r--p 00001000 ca:01 657531
↪  /home/user102/handout/part1/libmemtrace.so
7f4b935b9000-7f4b935ba000 rw-p 00002000 ca:01 657531
↪  /home/user102/handout/part1/libmemtrace.so
7f4b935ba000-7f4b935e0000 r-xp 00000000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7f4b937d5000-7f4b937d8000 rw-p 00000000 00:00 0
7f4b937dd000-7f4b937df000 rw-p 00000000 00:00 0
7f4b937df000-7f4b937e0000 r--p 00025000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7f4b937e0000-7f4b937e1000 rw-p 00026000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7f4b937e1000-7f4b937e2000 rw-p 00000000 00:00 0
7ffd8755e000-7ffd8757f000 rw-p 00000000 00:00 0                          [stack]
7ffd87581000-7ffd87584000 r--p 00000000 00:00 0                          [vvar]
7ffd87584000-7ffd87586000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
↪  [vsyscall]
Aborted (core dumped)
Makefile:37: recipe for target 'run' failed
make: *** [run] Error 134
```

아직 double free와 illegal free 처리를 하지 않기 때문에 크래시되는 것을 볼 수 있다.

test5

```
  user102@SystemProgramming:~/handout/part1$ make run test5
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : malloc( 10 ) = 0x154c060
[0003]           (nil) : realloc( 0x154c060 , 100 ) = 0x154c060
[0004]           (nil) : realloc( 0x154c060 , 1000 ) = 0x154c060
[0005]           (nil) : realloc( 0x154c060 , 10000 ) = 0x154c060
[0006]           (nil) : realloc( 0x154c060 , 100000 ) = 0x154c060
[0007]           (nil) : free( 0x154c060 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          0
[0013]
[0014] Memory tracer stopped.
```

3 Part 2: Tracing Unfreed Memory

Part 2에서는 Part 1의 정보와 더불어 해제되는 메모리에 대한 정보도 출력하며, 해제되지 않은 메모리의 양과 할당 위치에 대한 정보를 제공한다. 이제부터는 part 1과 다르게 각 메모리 할당을 개개 함수 스코프를 넘어 추적해야 한다. 그러기 위하여 자료구조가 필요한데, 마침 utils/memlist.h를 이용하여 memlist.c에 구현되어 있는 링크드 리스트 자료구조를 이용할 수 있었다.

```
35  void * malloc(size_t size)
36  {
37     n_allocb += size;
38     n_malloc++;
39     void * resultP = mallocp(size);
40     LOG_MALLOC(size, resultP);
41     item * allocated = alloc(list, resultP, size);
42     return resultP;
43  }
```

part 2에서 추가된 코드는 `alloc` 함수를 호출하는 것이다. `alloc` 함수를 호출하면 자동으로 새로운 item을 생성하거나 이미 존재할 경우 reference count를 증가시켜 준다.

```
45  void free(void * ptr)
46  {
47     LOG_FREE(ptr);
48     item * deallocated = dealloc(list, ptr);
49     n_freeb += deallocated->size;
50     freep(ptr);
51  }
```

`free`에는 `dealloc`을 이용하여 reference count를 감소시켜준다. 또 이 함수의 리턴값인 그 `item`의 주소를 이용하여 해당 주소 메모리 영역의 크기를 `n_freeb`에 더한다.

```
53  void * calloc(size_t nmemb, size_t size)
54  {
55     n_calloc++;
56     void * resultP = callocp(nmemb, size);
57     LOG_CALLOC(nmemb, size, resultP);
58     if(resultP) {
59       n_allocb += size*nmemb;
60       item * allocated = alloc(list, resultP, nmemb * size);
61     }
62     return resultP;
63  }
```

`calloc` 은 앞서 언급하였듯이 할당되는 최종 바이트 크기가 `nmemb`×`size` 인 것만 주의하면 `malloc`과 동일한 내용이다.

```
64  void * realloc(void *ptr, size_t new_size)
65  {
66     n_realloc++;
67     void * resultP = reallocp(ptr, new_size);
68     item * deallocated = dealloc(list, ptr);
69     n_freeb += deallocated->size;
70     LOG_REALLOC(ptr, new_size, resultP);
71     if(resultP) {
72        n_allocb+= new_size;
73        alloc(list, resultP, new_size);
74     }
75     return resultP;
76  }
```

`realloc`은 앞의 `malloc`과 `free`의 내용을 둘 다 가지고 있다.

```
120 void fini(void)
121 {
122   // ...
123   int n = n_malloc + n_calloc + n_realloc;
124   int avg = n? n_allocb/n : 0;
125   LOG_STATISTICS(n_allocb, avg, n_freeb);
126   item * i=list->next;
```

```
127   if(i && i->cnt >0) {
128     LOG_NONFREED_START();
129   }
130   //LOG_BLOCK();
131   while(i) {
132     if(i->cnt >0) {
133        LOG_BLOCK(i->ptr, i->size, i->cnt, i->fname, i->ofs);
134     }
135     i=i->next;
136   }
137
138   LOG_STOP();
139
140   // free list (not needed for part 1)
141   free_list(list);
142 }
```

`fini`는 non freed 블록이 존재할 경우 그에 대한 정보를 출력한다.

3.1 테스트 결과

test1

```
  user102@SystemProgramming:~/handout/part2$ make run test1
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪   ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
  [0002]           (nil) : malloc( 1024 ) = 0x1d20060
  [0003]           (nil) : malloc( 32 ) = 0x1d204c0
  [0004]           (nil) : malloc( 1 ) = 0x1d20540
  [0005]           (nil) : free( 0x1d20540 )
  [0006]           (nil) : free( 0x1d204c0 )
  [0007]
  [0008] Statistics
  [0009]   allocated_total      1057
  [0010]   allocated_avg        352
  [0011]   freed_total          33
  [0012]
  [0013] Non-deallocated memory blocks
  [0014]   block              size       ref cnt    caller
  [0015]   0x1d20060          1024       1          ???:0
  [0016]
  [0017] Memory tracer stopped.
```

test1의 소스코드를 보면 첫 번째 `malloc` 호출로 할당한 메모리를 해제하지 않는 것을 볼 수 있는데, 이 테스트 결과도 그것을 잘 나타내고 있는 것을 볼 수 있다.

test2

```
  user102@SystemProgramming:~/handout/part2$ make run test2
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪   ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
  [0002]           (nil) : malloc( 1024 ) = 0x1298060
  [0003]           (nil) : free( 0x1298060 )
  [0004]
  [0005] Statistics
  [0006]   allocated_total      1024
  [0007]   allocated_avg        1024
  [0008]   freed_total          1024
  [0009]
  [0010] Memory tracer stopped.
```

1024바이트의 메모리를 할당받고 그것을 그대로 해제하는 것이 잘 나타나 있다.

test3

```
  user102@SystemProgramming:~/handout/part2$ make run test3
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : calloc( 1 , 43776 ) = 0x19b3060
[0003]           (nil) : calloc( 1 , 190 ) = 0x19bdbc0
[0004]           (nil) : calloc( 1 , 13781 ) = 0x19bdce0
[0005]           (nil) : calloc( 1 , 43393 ) = 0x19c1310
[0006]           (nil) : calloc( 1 , 58232 ) = 0x19cbcf0
[0007]           (nil) : malloc( 39935 ) = 0x19da0c0
[0008]           (nil) : malloc( 31759 ) = 0x19e3d20
[0009]           (nil) : malloc( 30749 ) = 0x19eb990
[0010]           (nil) : calloc( 1 , 33536 ) = 0x19f3210
[0011]           (nil) : calloc( 1 , 36193 ) = 0x19fb570
[0012]           (nil) : free( 0x19fb570 )
[0013]           (nil) : free( 0x19f3210 )
[0014]           (nil) : free( 0x19eb990 )
[0015]           (nil) : free( 0x19e3d20 )
[0016]           (nil) : free( 0x19da0c0 )
[0017]           (nil) : free( 0x19cbcf0 )
[0018]           (nil) : free( 0x19c1310 )
[0019]           (nil) : free( 0x19bdce0 )
[0020]           (nil) : free( 0x19bdbc0 )
[0021]           (nil) : free( 0x19b3060 )
[0022]
[0023] Statistics
[0024]   allocated_total      331544
[0025]   allocated_avg        33154
[0026]   freed_total          331544
[0027]
[0028] Memory tracer stopped.
```

test4

```
  user102@SystemProgramming:~/handout/part2$ make run test4
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
  [0002]           (nil) : malloc( 1024 ) = 0xf90060
  [0003]           (nil) : free( 0xf90060 )
  [0004]           (nil) : free( 0xf90060 )
  *** Error in `../test/test4': double free or corruption (!prev):
↪  0x0000000000f90060 ***
  [0005]           (nil) : malloc( 36 ) = 0x7f85a80008c0
  [0006]           (nil) : calloc( 1182 , 1 ) = 0x7f85a8000940
  [0007]           (nil) : malloc( 36 ) = 0x7f85a8000e40
  [0008]           (nil) : malloc( 56 ) = 0x7f85a8000ec0
  [0009]           (nil) : calloc( 15 , 24 ) = 0x7f85a8000f50
  ======= Backtrace: =========
  /lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f85aef9f7e5]
  /lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7f85aefa837a]
  /lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7f85aefac53c]
  ./libmemtrace.so(free+0x6c)[0x7f85af2f2e2b]
  ../test/test4[0x40048e]
  /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f85aef48830]
  ../test/test4[0x4004c9]
  ======= Memory map: ========
```

```
  00400000-00401000 r-xp 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
  00600000-00601000 r--p 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
  00601000-00602000 rw-p 00001000 ca:01 657555
↪  /home/user102/handout/test/test4
  00f90000-00fb1000 rw-p 00000000 00:00 0                                  [heap]
  7f85a8000000-7f85a8021000 rw-p 00000000 00:00 0
  7f85a8021000-7f85ac000000 ---p 00000000 00:00 0
  7f85aeb0e000-7f85aeb24000 r-xp 00000000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
  7f85aeb24000-7f85aed23000 ---p 00016000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
  7f85aed23000-7f85aed24000 rw-p 00015000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
  7f85aed24000-7f85aed27000 r-xp 00000000 ca:01 2097667
↪  /lib/x86_64-linux-gnu/libdl-2.23.so
  7f85aed27000-7f85aef26000 ---p 00003000 ca:01 2097667
↪  /lib/x86_64-linux-gnu/libdl-2.23.so
  7f85aef26000-7f85aef27000 r--p 00002000 ca:01 2097667
↪  /lib/x86_64-linux-gnu/libdl-2.23.so
  7f85aef27000-7f85aef28000 rw-p 00003000 ca:01 2097667
↪  /lib/x86_64-linux-gnu/libdl-2.23.so
  7f85aef28000-7f85af0e8000 r-xp 00000000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
  7f85af0e8000-7f85af2e8000 ---p 001c0000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
  7f85af2e8000-7f85af2ec000 r--p 001c0000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
  7f85af2ec000-7f85af2ee000 rw-p 001c4000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
  7f85af2ee000-7f85af2f2000 rw-p 00000000 00:00 0
  7f85af2f2000-7f85af2f4000 r-xp 00000000 ca:01 657556
↪  /home/user102/handout/part2/libmemtrace.so
  7f85af2f4000-7f85af4f4000 ---p 00002000 ca:01 657556
↪  /home/user102/handout/part2/libmemtrace.so
  7f85af4f4000-7f85af4f5000 r--p 00002000 ca:01 657556
↪  /home/user102/handout/part2/libmemtrace.so
  7f85af4f5000-7f85af4f6000 rw-p 00003000 ca:01 657556
↪  /home/user102/handout/part2/libmemtrace.so
  7f85af4f6000-7f85af51c000 r-xp 00000000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
  7f85af711000-7f85af714000 rw-p 00000000 00:00 0
  7f85af719000-7f85af71b000 rw-p 00000000 00:00 0
  7f85af71b000-7f85af71c000 r--p 00025000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
  7f85af71c000-7f85af71d000 rw-p 00026000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
  7f85af71d000-7f85af71e000 rw-p 00000000 00:00 0
  7ffe57c0d000-7ffe57c2e000 rw-p 00000000 00:00 0                          [stack]
  7ffe57da8000-7ffe57dab000 r--p 00000000 00:00 0                          [vvar]
  7ffe57dab000-7ffe57dad000 r-xp 00000000 00:00 0                          [vdso]
  ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
↪  [vsyscall]
  Aborted (core dumped)
  Makefile:37: recipe for target 'run' failed
  make: *** [run] Error 134
```

아직 illegal free와 double free를 처리하지 않기 때문에 여전히 크래시가 발생하는 것을 볼 수 있다.

test5

```
  user102@SystemProgramming:~/handout/part2$ make run test5
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]           (nil) : malloc( 10 ) = 0x1a75060
[0003]           (nil) : realloc( 0x1a75060 , 100 ) = 0x1a750d0
[0004]           (nil) : realloc( 0x1a750d0 , 1000 ) = 0x1a75190
[0005]           (nil) : realloc( 0x1a75190 , 10000 ) = 0x1a755d0
[0006]           (nil) : realloc( 0x1a755d0 , 100000 ) = 0x1a755d0
[0007]           (nil) : free( 0x1a755d0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

`realloc` 으로 해제되는 메모리도 제대로 추적되는 것을 볼 수 있다.

4 Part 3: Pinpointing Call Locations

Part 3에서는 Part 2의 정보에 더불어, libunwind 라이브러리를 이용하여 이 메모리 관리 함수들의 호출 위치를 추적한다. 이번 파트에서는 memtrace.c의 내용은 part 2와 동일하다. 그러나 이번에는 다른 파일인 callinfo.c를 구현하는 것이 관건이다.
callinfo.h를 보면 설명이 나와 있다.

```
7  // return the PC of the callsite to the dynamic memory management function
8  //
9  //   fname      pointer to character array to hold function name
10 //   fnlen      length of character array
11 //   ofs        pointer to offset to hold PC offset into function
12 //
13 // returns
14 //    0         on success
15 //   <0         on error
16 //
17 int get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs);
```

이 함수는 인자 3개를 받고 성공시 0, 실패시 음수를 반환해야 한다고 쓰여 있다. 이 인자 3개가 IN 용인지 OUT용인지 판단하기 위해 memlog.c를 살펴본다.

```
17     if (get_callinfo(&buf[0], sizeof(buf), &ofs) != -1) {
18       res += fprintf(stderr, "%12s:%-3llx: ", buf, ofs);
19     } else {
20       res += fprintf(stderr, "%5c%10p : ", ' ', NULL);
21     }
```

callinfo.h의 내용과 다르게 실제 사용하는 측은 리턴값이 -1인지 아닌지를 검사하고 있다. 이 점을 유의하며 callinfo.c의 내용을 작성하였다. 그리고 17행을 보면 `get_callinfo`의 인자들은 전부 OUT임을 알 수 있다. 즉, 정보를 `get_callinfo` 내부에서 생산하여 호출자에게 전달해 주어야 한다는 것이다. 과제 pdf에 libunwind를 이용하여 stack trace를 출력하는 예제가 있어 이용하였다.

```
1  #include <stdlib.h>
2  #define UNW_LOCAL_ONLY
3  #include <libunwind.h>
4
5  int get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs)
6  {
7    unw_context_t context;
```

```
8     unw_cursor_t cursor;
9     unw_word_t off, ip, sp;
10    unw_proc_info_t pip;
11    char procname[256];
12    int ret;
13    if(unw_getcontext(&context)) {
14      return -1;
15    }
16    if(unw_init_local(&cursor, &context)) {
17      return -1;
18    }
```

이 부분은 예제에 따라 libunwind를 사용하기 위한 변수들을 초기화한 것이다. 이제 cursor를 적당히 이동시켜 get_callinfo를 호출한 mlog를 호출한 우리가 가로챈 함수를 호출한 위치를 찾아야 한다. 따라서 `unw_step` 함수를 3번 호출하여 커서를 이동시킨다.

```
19    if(unw_step(&cursor)<=0) {
20      return -1;
21    }
22    if(unw_step(&cursor)<=0) {
23      return -1;
24    }
25    if(unw_step(&cursor)<=0) {
26      return -1;
27    }
28    if(unw_get_proc_info(&cursor, &pip)) {
29      return -1;
30    }
```

이제 `unw_get_proc_name` 함수를 호출하여 함수 이름과 그 함수의 처음 주소로부터의 오프셋을 구하면 된다.

```
34    if(unw_get_proc_name(&cursor, fname, fnlen, &off)) {
35      return -1;
36    }
```

이렇게 하고 나니 결과값이 objdump로 예측한 결과와 5 차이가 난다. 그 이유는 libunwind가 알려주는 offset은 함수의 호출 후 돌아갈 return address 즉 함수를 호출하는 `callq` 명령 다음 인스트럭션의 주소에 대한 offset을 알려주기 때문이다. 이는 x64 아키텍쳐의 `callq` 명령이 스택에 현재 `rip` 즉 PC 값을 푸시하고 타깃으로 점프하는 식으로 작동하기 때문이다.
objdump에 의한 `callq` 명령의 크기는 5바이트이다. 그러므로 `unw_get_proc_name`의 결과값에서 5를 빼준 값을 `*ofs`에 넣는다.

```
37    *ofs = off - 5; //5 is the len of the call instruction
38    //*ofs = off;
39    return 0;
40  }
```

4.1 테스트 결과

test1

```
user102@SystemProgramming:~/handout/part3$ objdump -d ../test/test1
(중략)
Disassembly of section .text:

0000000000400470 <main>:
  400470:       53                      push   %rbx
  400471:       bf 00 04 00 00          mov    $0x400,%edi
  400476:       e8 d5 ff ff ff          callq  400450 <malloc@plt>
  40047b:       bf 20 00 00 00          mov    $0x20,%edi
  400480:       e8 cb ff ff ff          callq  400450 <malloc@plt>
  400485:       bf 01 00 00 00          mov    $0x1,%edi
```

```
  40048a:       48 89 c3                mov    %rax,%rbx
  40048d:       e8 be ff ff ff          callq  400450 <malloc@plt>
  400492:       48 89 c7                mov    %rax,%rdi
  400495:       e8 96 ff ff ff          callq  400430 <free@plt>
  40049a:       48 89 df                mov    %rbx,%rdi
  40049d:       e8 8e ff ff ff          callq  400430 <free@plt>
  4004a2:       31 c0                   xor    %eax,%eax
  4004a4:       5b                      pop    %rbx
  4004a5:       c3                      retq
  4004a6:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  4004ad:       00 00 00
  (후략)
```

main:6, main:10, main:1d, main:25, main:2d에서 메모리 할당과 해제 함수가 호출되는 것을 볼 수 있다.

```
  user102@SystemProgramming:~/handout/part3$ make run test1
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
  [0002]         main:6  : malloc( 1024 ) = 0x1286060
  [0003]         main:10 : malloc( 32 ) = 0x12864c0
  [0004]         main:1d : malloc( 1 ) = 0x1286540
  [0005]         main:25 : free( 0x1286540 )
  [0006]         main:2d : free( 0x12864c0 )
  [0007]
  [0008] Statistics
  [0009]   allocated_total      1057
  [0010]   allocated_avg        352
  [0011]   freed_total          33
  [0012]
  [0013] Non-deallocated memory blocks
  [0014]   block              size       ref cnt   caller
  [0015]   0x1286060          1024       1         main:6
  [0016]
  [0017] Memory tracer stopped.
```

objdump로 예측한 위치와 일치하는 것을 볼 수 있었다.

test2

```
  user102@SystemProgramming:~/handout/part3$ objdump -d ../test/test2
  (중략)
  Disassembly of section .text:

0000000000400470 <main>:
  400470:       48 83 ec 08             sub    $0x8,%rsp
  400474:       bf 00 04 00 00          mov    $0x400,%edi
  400479:       e8 d2 ff ff ff          callq  400450 <malloc@plt>
  40047e:       48 89 c7                mov    %rax,%rdi
  400481:       e8 aa ff ff ff          callq  400430 <free@plt>
  400486:       31 c0                   xor    %eax,%eax
  400488:       48 83 c4 08             add    $0x8,%rsp
  40048c:       c3                      retq
  40048d:       0f 1f 00                nopl   (%rax)
  (후략)
```

main:9, main:11에서 각각 메모리 할당과 해제가 일어나는 것을 볼 수 있다.

```
  user102@SystemProgramming:~/handout/part3$ make run test2
  cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
  [0001] Memory tracer started.
```

```
[0002]         main:9  : malloc( 1024 ) = 0x147a060
[0003]         main:11 : free( 0x147a060 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
```

objdump로 예측한 것과 일치한다.

test3

```
user102@SystemProgramming:~/handout/part3$ objdump -d ../test/test3
(중략)
Disassembly of section .text:

0000000000400600 <main>:
  400600:       41 54                   push   %r12
  400602:       55                      push   %rbp
  400603:       31 ff                   xor    %edi,%edi
  400605:       53                      push   %rbx
  400606:       48 83 ec 60             sub    $0x60,%rsp
  40060a:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  400611:       00 00
  400613:       48 89 44 24 58          mov    %rax,0x58(%rsp)
  400618:       31 c0                   xor    %eax,%eax
  40061a:       e8 a1 ff ff ff          callq  4005c0 <time@plt>
  40061f:       89 c7                   mov    %eax,%edi
  400621:       48 89 e3                mov    %rsp,%rbx
  400624:       48 8d 6c 24 50          lea    0x50(%rsp),%rbp
  400629:       e8 72 ff ff ff          callq  4005a0 <srand@plt>
  40062e:       eb 15                   jmp    400645 <main+0x45>
  400630:       4c 89 e7                mov    %r12,%rdi
  400633:       48 83 c3 08             add    $0x8,%rbx
  400637:       e8 94 ff ff ff          callq  4005d0 <malloc@plt>
  40063c:       48 89 43 f8             mov    %rax,-0x8(%rbx)
  400640:       48 39 eb                cmp    %rbp,%rbx
  400643:       74 37                   je     40067c <main+0x7c>
  400645:       e8 96 ff ff ff          callq  4005e0 <rand@plt>
  40064a:       99                      cltd
  40064b:       c1 ea 10                shr    $0x10,%edx
  40064e:       8d 34 10                lea    (%rax,%rdx,1),%esi
  400651:       0f b7 f6                movzwl %si,%esi
  400654:       29 d6                   sub    %edx,%esi
  400656:       4c 63 e6                movslq %esi,%r12
  400659:       e8 82 ff ff ff          callq  4005e0 <rand@plt>
  40065e:       a8 01                   test   $0x1,%al
  400660:       75 ce                   jne    400630 <main+0x30>
  400662:       4c 89 e6                mov    %r12,%rsi
  400665:       bf 01 00 00 00          mov    $0x1,%edi
  40066a:       48 83 c3 08             add    $0x8,%rbx
  40066e:       e8 3d ff ff ff          callq  4005b0 <calloc@plt>
  400673:       48 89 43 f8             mov    %rax,-0x8(%rbx)
  400677:       48 39 eb                cmp    %rbp,%rbx
  40067a:       75 c9                   jne    400645 <main+0x45>
  40067c:       48 8d 5c 24 48          lea    0x48(%rsp),%rbx
  400681:       48 8d 6c 24 f8          lea    -0x8(%rsp),%rbp
  400686:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40068d:       00 00 00
```

```
  400690:       48 8b 3b                mov    (%rbx),%rdi
  400693:       48 83 eb 08             sub    $0x8,%rbx
  400697:       e8 d4 fe ff ff          callq  400570 <free@plt>
  40069c:       48 39 eb                cmp    %rbp,%rbx
  40069f:       75 ef                   jne    400690 <main+0x90>
  4006a1:       31 c0                   xor    %eax,%eax
  4006a3:       48 8b 4c 24 58          mov    0x58(%rsp),%rcx
  4006a8:       64 48 33 0c 25 28 00    xor    %fs:0x28,%rcx
  4006af:       00 00
  4006b1:       75 09                   jne    4006bc <main+0xbc>
  4006b3:       48 83 c4 60             add    $0x60,%rsp
  4006b7:       5b                      pop    %rbx
  4006b8:       5d                      pop    %rbp
  4006b9:       41 5c                   pop    %r12
  4006bb:       c3                      retq
  4006bc:       e8 bf fe ff ff          callq  400580 <__stack_chk_fail@plt>
  4006c1:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  4006c8:       00 00 00
  4006cb:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  (후략)
```

main:37, main:6e에서 메모리를 할당받고, main:97에서 해제하는 것을 볼 수 있다.

```
  user102@SystemProgramming:~/handout/part3$ make run test3
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:6e : calloc( 1 , 47824 ) = 0x25f3060
[0003]         main:6e : calloc( 1 , 11452 ) = 0x25feb90
[0004]         main:37 : malloc( 10845 ) = 0x26018b0
[0005]         main:6e : calloc( 1 , 32498 ) = 0x2604370
[0006]         main:37 : malloc( 29647 ) = 0x260c2c0
[0007]         main:37 : malloc( 31076 ) = 0x26136f0
[0008]         main:6e : calloc( 1 , 30023 ) = 0x261b0b0
[0009]         main:37 : malloc( 28743 ) = 0x2622650
[0010]         main:37 : malloc( 45171 ) = 0x26296f0
[0011]         main:6e : calloc( 1 , 11093 ) = 0x26347c0
[0012]         main:97 : free( 0x26347c0 )
[0013]         main:97 : free( 0x26296f0 )
[0014]         main:97 : free( 0x2622650 )
[0015]         main:97 : free( 0x261b0b0 )
[0016]         main:97 : free( 0x26136f0 )
[0017]         main:97 : free( 0x260c2c0 )
[0018]         main:97 : free( 0x2604370 )
[0019]         main:97 : free( 0x26018b0 )
[0020]         main:97 : free( 0x25feb90 )
[0021]         main:97 : free( 0x25f3060 )
[0022]
[0023] Statistics
[0024]   allocated_total      278372
[0025]   allocated_avg        27837
[0026]   freed_total          278372
[0027]
[0028] Memory tracer stopped.
```

objdump의 결과와 마찬가지로 `malloc`과 `calloc`, `free`의 호출을 제대로 표시하는 것을 볼 수 있다.

test4

```
  user102@SystemProgramming:~/handout/part3$ objdump -d ../test/test4
  (중략)
```

```
0000000000400470 <main>:
400470:       53                      push   %rbx
400471:       bf 00 04 00 00          mov    $0x400,%edi
400476:       e8 d5 ff ff ff          callq  400450 <malloc@plt>
40047b:       48 89 c3                mov    %rax,%rbx
40047e:       48 89 c7                mov    %rax,%rdi
400481:       e8 aa ff ff ff          callq  400430 <free@plt>
400486:       48 89 df                mov    %rbx,%rdi
400489:       e8 a2 ff ff ff          callq  400430 <free@plt>
40048e:       bf 90 6e 70 01          mov    $0x1706e90,%edi
400493:       e8 98 ff ff ff          callq  400430 <free@plt>
400498:       31 c0                   xor    %eax,%eax
40049a:       5b                      pop    %rbx
40049b:       c3                      retq
40049c:       0f 1f 40 00             nopl   0x0(%rax)
```

main:6, main:11, main:19, main:23 에서 메모리를 할당받고 세 번 해제하는 것을 볼 수 있다.

```
  user102@SystemProgramming:~/handout/part3$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0x969060
[0003]         main:11 : free( 0x969060 )
[0004]         main:19 : free( 0x969060 )
*** Error in `../test/test4': double free or corruption (!prev):
↪  0x0000000000969060 ***
[0005]      realloc:2e95: malloc( 36 ) = 0x7fc2cc0008c0
[0006]           (nil) : calloc( 1182 , 1 ) = 0x7fc2cc000940
[0007]           (nil) : malloc( 36 ) = 0x7fc2cc000e40
[0008]           (nil) : malloc( 56 ) = 0x7fc2cc000ec0
[0009] _dl_debug_state:1059: calloc( 15 , 24 ) = 0x7fc2cc000f50
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7fc2d30257e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7fc2d302e37a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7fc2d303253c]
./libmemtrace.so(free+0x6c)[0x7fc2d3378feb]
../test/test4[0x40048e]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7fc2d2fce830]
../test/test4[0x4004c9]
======= Memory map: ========
00400000-00401000 r-xp 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
00600000-00601000 r--p 00000000 ca:01 657555
↪  /home/user102/handout/test/test4
00601000-00602000 rw-p 00001000 ca:01 657555
↪  /home/user102/handout/test/test4
00969000-0098a000 rw-p 00000000 00:00 0                                  [heap]
7fc2cc000000-7fc2cc021000 rw-p 00000000 00:00 0
7fc2cc021000-7fc2d0000000 ---p 00000000 00:00 0
7fc2d297c000-7fc2d2992000 r-xp 00000000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
7fc2d2992000-7fc2d2b91000 ---p 00016000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
7fc2d2b91000-7fc2d2b92000 rw-p 00015000 ca:01 2097679
↪  /lib/x86_64-linux-gnu/libgcc_s.so.1
7fc2d2b92000-7fc2d2b9e000 r-xp 00000000 ca:01 1453561
↪  /usr/local/lib/libunwind.so.8.0.1
7fc2d2b9e000-7fc2d2d9e000 ---p 0000c000 ca:01 1453561
↪  /usr/local/lib/libunwind.so.8.0.1
```

```
7fc2d2d9e000-7fc2d2d9f000 r--p 0000c000 ca:01 1453561
↪  /usr/local/lib/libunwind.so.8.0.1
7fc2d2d9f000-7fc2d2da0000 rw-p 0000d000 ca:01 1453561
↪  /usr/local/lib/libunwind.so.8.0.1
7fc2d2da0000-7fc2d2daa000 rw-p 00000000 00:00 0
7fc2d2daa000-7fc2d2dad000 r-xp 00000000 ca:01 2097667
↪  /lib/x86_64-linux-gnu/libdl-2.23.so
(중략)
7fc2d3372000-7fc2d3374000 rw-p 001c4000 ca:01 2097653
↪  /lib/x86_64-linux-gnu/libc-2.23.so
7fc2d3374000-7fc2d3378000 rw-p 00000000 00:00 0
7fc2d3378000-7fc2d337b000 r-xp 00000000 ca:01 657537
↪  /home/user102/handout/part3/libmemtrace.so
7fc2d337b000-7fc2d357a000 ---p 00003000 ca:01 657537
↪  /home/user102/handout/part3/libmemtrace.so
7fc2d357a000-7fc2d357b000 r--p 00002000 ca:01 657537
↪  /home/user102/handout/part3/libmemtrace.so
7fc2d357b000-7fc2d357c000 rw-p 00003000 ca:01 657537
↪  /home/user102/handout/part3/libmemtrace.so
7fc2d357c000-7fc2d35a2000 r-xp 00000000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7fc2d3797000-7fc2d379a000 rw-p 00000000 00:00 0
7fc2d379d000-7fc2d37a1000 rw-p 00000000 00:00 0
7fc2d37a1000-7fc2d37a2000 r--p 00025000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7fc2d37a2000-7fc2d37a3000 rw-p 00026000 ca:01 2097629
↪  /lib/x86_64-linux-gnu/ld-2.23.so
7fc2d37a3000-7fc2d37a4000 rw-p 00000000 00:00 0
7ffe44d79000-7ffe44d9a000 rw-p 00000000 00:00 0                          [stack]
7ffe44de2000-7ffe44de5000 r--p 00000000 00:00 0                          [vvar]
7ffe44de5000-7ffe44de7000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
↪  [vsyscall]
Aborted (core dumped)
Makefile:37: recipe for target 'run' failed
make: *** [run] Error 134
```

아직 part1, 2와 마찬가지로 illegal free와 double free로 크래시되는 것을 볼 수 있다.

test5

```
user102@SystemProgramming:~/handout/part3$ objdump -d ../test/test5
(중략)
00000000004004c0 <main>:
  4004c0:       48 83 ec 08             sub    $0x8,%rsp
  4004c4:       bf 0a 00 00 00          mov    $0xa,%edi
  4004c9:       e8 c2 ff ff ff          callq  400490 <malloc@plt>
  4004ce:       be 64 00 00 00          mov    $0x64,%esi
  4004d3:       48 89 c7                mov    %rax,%rdi
  4004d6:       e8 c5 ff ff ff          callq  4004a0 <realloc@plt>
  4004db:       be e8 03 00 00          mov    $0x3e8,%esi
  4004e0:       48 89 c7                mov    %rax,%rdi
  4004e3:       e8 b8 ff ff ff          callq  4004a0 <realloc@plt>
  4004e8:       be 10 27 00 00          mov    $0x2710,%esi
  4004ed:       48 89 c7                mov    %rax,%rdi
  4004f0:       e8 ab ff ff ff          callq  4004a0 <realloc@plt>
  4004f5:       be a0 86 01 00          mov    $0x186a0,%esi
  4004fa:       48 89 c7                mov    %rax,%rdi
  4004fd:       e8 9e ff ff ff          callq  4004a0 <realloc@plt>
  400502:       48 89 c7                mov    %rax,%rdi
  400505:       e8 66 ff ff ff          callq  400470 <free@plt>
```

```
  40050a:       31 c0                   xor    %eax,%eax
  40050c:       48 83 c4 08             add    $0x8,%rsp
  400510:       c3                      retq
  400511:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  400518:       00 00 00
  40051b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  (후략)
```

main:9, main:16, main:23, main:30, main:3d, main:45에서 메모리를 할당받고 해제하는 것을 볼 수 있다.

```
  user102@SystemProgramming:~/handout/part3$ make run test5
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:9  : malloc( 10 ) = 0x176c060
[0003]         main:16 : realloc( 0x176c060 , 100 ) = 0x176c0d0
[0004]         main:23 : realloc( 0x176c0d0 , 1000 ) = 0x176c190
[0005]         main:30 : realloc( 0x176c190 , 10000 ) = 0x176c5d0
[0006]         main:3d : realloc( 0x176c5d0 , 100000 ) = 0x176c5d0
[0007]         main:45 : free( 0x176c5d0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

objdump의 결과와 일치한다.

5 Bonus: Detect and Ignore Illegal Deallocations

Bonus에서는 part 1 3을 test4에 대해 적용했을 때 double free와 illegal free 시 크래시되는 것을 예방하고 대신 각각 DOUBLE FREE와 ILLEGAL FREE 정보를 출력하도록 한다. 이번에는 `free`와 `realloc`에 처리가 추가된다. 기존 `list`에서 현재 free 하려는 메모리 영역에 대한 item을 검색하고, 존재하지 않을 경우 Illegal free 오류를, 존재는 하나 이미 reference count가 0일 경우 Double free 오류를 출력하고 무시하는 코드가 추가되었다.

```
45  void free(void * ptr)
46  {
47     LOG_FREE(ptr);
48     item * tobedeallocated = find(list, ptr);
49     if(tobedeallocated) {
50        if(tobedeallocated -> cnt <=0) {
51           LOG_DOUBLE_FREE();
52           return;
53        }
54        n_freeb += tobedeallocated->size;
55        freep(ptr);
56        dealloc(list, ptr);
57     } else {
58       LOG_ILL_FREE();
59     }
60  }
```

`free`에서는 위의 내용이 그대로 구현되어 있다.

```
74  void * realloc(void *ptr, size_t new_size)
75  {
76     void * resultP = NULL;
```

```
77      int double_free = 0;
78      int illegal_free = 0;
79      n_realloc++;
80      item * tobedeallocated = find(list, ptr);
81      if(tobedeallocated) {
82         if(tobedeallocated->cnt<=0) {
83            double_free = 1;
84            resultP = reallocp(NULL, new_size);
85            // LOG_DOUBLE_FREE();
86         } else {
87            n_freeb += tobedeallocated->size;
88            dealloc(list, ptr);
89            resultP = reallocp(ptr, new_size);
90            // LOG_REALLOC(ptr, new_size, resultP);
91         }
92      } else {
93        illegal_free = 1;
94        resultP = reallocp(NULL, new_size);
95      }
96      LOG_REALLOC(ptr, new_size, resultP);
97      if(resultP) {
98         n_allocb += new_size;
99         alloc(list, resultP, new_size);
100     }
101     if(double_free) {
102        LOG_DOUBLE_FREE();
103     }
104     if(illegal_free) {
105        LOG_ILL_FREE();
106     }
107     return resultP;
108  }
```

realloc에서는 처리가 좀 더 복잡한데, 그 이유는 etl에 올라온 realloc 예시 출력을 보면 realloc 호출 인자와 결과값을 먼저 출력하고 나서 illegal free나 double free 오류를 출력하기 때문이다. 그렇다고 illegal free나 double free 검출보다 realloc을 먼저 할 수는 없을 것이므로, 판단 결과를 저장하는 변수인 double_free 변수와 illegal_free 변수를 사용하였다.

5.1 테스트 결과

test1 test3, test5는 앞의 part3의 테스트 결과와 같을 것이므로 part1 part3에서 크래시되었던 test4에 대해 테스트를 해 본다.

test4

```
5   void *a;
6
7   a = malloc(1024);
8   free(a);
9   free(a);
10  free((void*)0x1706e90);
```

double free와 illegal free가 일어남을 볼 수 있다.

```
user102@SystemProgramming:~/handout/bonus$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
↪  ../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]          main:6  : malloc( 1024 ) = 0x12d8060
[0003]          main:11 : free( 0x12d8060 )
[0004]          main:19 : free( 0x12d8060 )
```

```
[0005]             *** DOUBLE_FREE  *** (ignoring)
[0006]         main:23 : free( 0x1706e90 )
[0007]             *** ILLEGAL_FREE *** (ignoring)
[0008]
[0009] Statistics
[0010]   allocated_total      1024
[0011]   allocated_avg        1024
[0012]   freed_total          1024
[0013]
[0014] Memory tracer stopped.
```

double free와 illegal free를 발견하고 무시하여 크래시를 예방하며 로그를 잘 출력하는 것을 볼 수 있다.

6 Conclusion

6.1 어려웠던 점

익숙한 환경의 부재

작년 수업에서 처음 배워 사용하던 리눅스 텍스트 기반 에디터인 emacs 에디터가 없어서 `i`와 `:wq`밖에 모르던 vi 에디터를 사용하였는데, vi 환경에 적응하는 데 시간이 걸렸다. 다행히 검색을 통해 필요한 기능들을 익힐 수 있었다.

libunwind

Part 3에서 libunwind를 이용해 메모리 할당 함수들의 호출 위치를 추적하는데, objdump로 예측했던 offset과 libunwind가 `unw_get_proc_name`으로 알려주는 offset이 서로 달라 혼란이 있었다. 기본적으로 libunwind도 스택에 저장된 리턴 어드레스를 이용하는 것이라 해서 callq의 인스트럭션 크기만큼 보정을 해 주어야 할 것 같다는 생각은 들었지만, 예제들을 아무리 찾아 봐도 보정에 대한 이야기는 없어서 혼란스러웠다. 결국 그 추측에 확신을 얻기 위해 libunwind의 소스 코드를 찾아 보았는데, 파일이 너무 많아서 어려움을 겪었다. 다행히 질의응답에 누군가 질문을 올린 것을 발견하여 5의 보정값을 빼는 것에 확신을 얻을 수 있었다.

6.2 놀라웠던 점

library interpositioning의 간단함

이렇게 간단한 코드로 어떤 프로그램의 메모리 사용을 해당 프로그램을 수정하지 않고 추적할 수 있다는 점이 놀라웠다. 또한 라이브러리를 조금 이용하여 실제 함수 호출 위치와 그 호출자 함수의 이름 등 유용한 정보들을 손쉽게 얻을 수 있다는 사실이 놀라웠다.

call이 없는 main과 최적화

objdump로 test 들을 분석해 보는 과정에서 testx도 분석해 보았는데, testx의 `main` 코드를 보면 `foo` 를 호출하는 부분이 전혀 없었다. Makefile을 살펴보아 -O2 플래그를 확인하였고, 컴파일러 최적화 과정의 함수 인라이닝 때문이라는 것을 알게 되었다.
`foo` 함수는 `main`에 인라인 처리가 되었음에도 불구하고 바이너리 파일 안에 존재했는데, `static`이 붙지 않아 심볼이 남았기 때문이라고 생각할 수 있다.
`main` 함수에도 마지막 함수 호출은 `jmp` 인스트럭션으로 대체되어 있고 제대로 된 `return`이 없던 것을 볼 수 있었는데, 이것도 컴파일러 최적화의 결과이며, `call` 후 타깃 함수의 `ret`, 그 후 `main`의 `ret`을 간소화한 것으로 생각이 든다. 이와 더불어 스택에 대한 걱정을 잠시 했었는데(원래는 리턴 어드레스 푸시로 인해 스택이 약간 변화하므로), 이 시스템은 64비트라 적은 매개변수는 레지스터로 전달하여 상관이 없구나 하는 생각이 들었다. 그에 이어서 든 생각인데, 매개변수도 쉽게 레지스터에 전달하는데 리턴 어드레스도 ARM 프로세서와 같이 새로운 레지스터에 저장하는 것은 어떨까 하는 생각도 잠깐 들었다.

`hlt` 인스트럭션

레포트를 작성할 때 objdump의 출력을 복사하다가 우연히 `_start`함수에 `hlt` 인스트럭션이 존재하는 것을 발견하였다.

```
00000000004004a0 <_start>:
4004a0:        31 ed                   xor    %ebp,%ebp
4004a2:        49 89 d1                mov    %rdx,%r9
4004a5:        5e                      pop    %rsi
4004a6:        48 89 e2                mov    %rsp,%rdx
4004a9:        48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
4004ad:        50                      push   %rax
4004ae:        54                      push   %rsp
4004af:        49 c7 c0 10 06 40 00    mov    $0x400610,%r8
4004b6:        48 c7 c1 a0 05 40 00    mov    $0x4005a0,%rcx
4004bd:        48 c7 c7 70 04 40 00    mov    $0x400470,%rdi
4004c4:        e8 77 ff ff ff          callq  400440 <__libc_start_main@plt>
4004c9:        f4                      hlt
4004ca:        66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
```

분명 알고 있는 바로는 `hlt` 인스트럭션은 ring 0 레벨에서만 실행 가능한 것으로 알고 있는데, 바로 다음의 `nopw 0x0(%rax, %rax, 1)`과 같은 단순한 패딩은 아닌 것 같아서 검색해 보았다. Github의 코드를 찾아보니 이것은 프로그램의 크래시를 의도한 것이었다.

```
  hlt                     /* Crash if somehow `exit' does return.  */
```

이 `hlt` 명령은 단지 General Protection Fault를 이용해 프로그램을 종료시키기 위한 것이었다. exit()함수가 실패했으므로 확실하게 프로그램을 종료시키기 위한 것이었다.