**System Programming**

**Linker Lab: Memtrace**

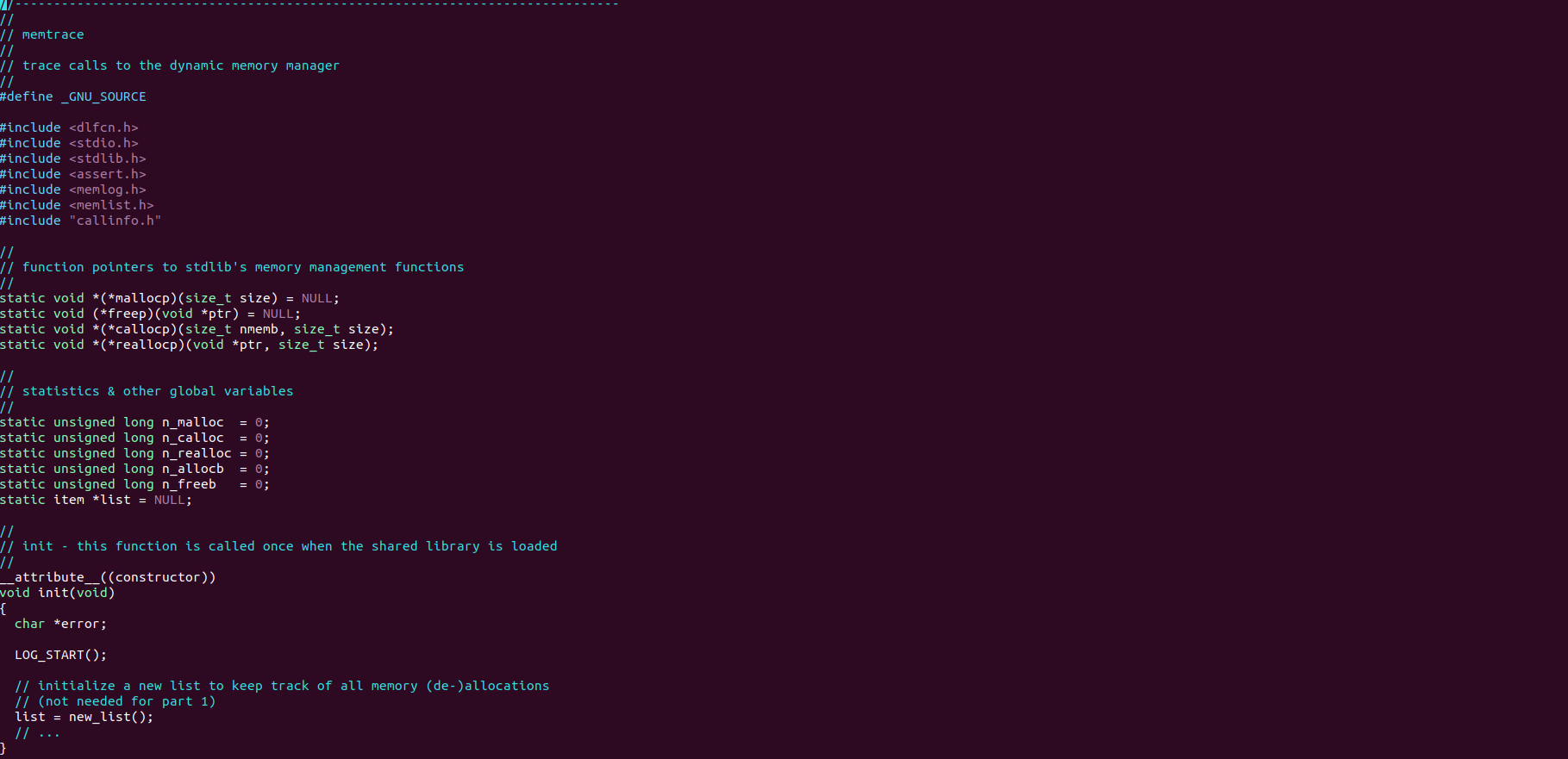
2017-18538 황선영

1. Introduction

Linker lab을 통해 library interpositioning과shared library의 간단한 data 관리를 이해할 수 있다. 그리고 debugger library를 사용함으로써 실제 프로그램을 pinpoint하는 법을 배운다.

2. Part1: Tracing Dymamic Memory Allocation

① 구현

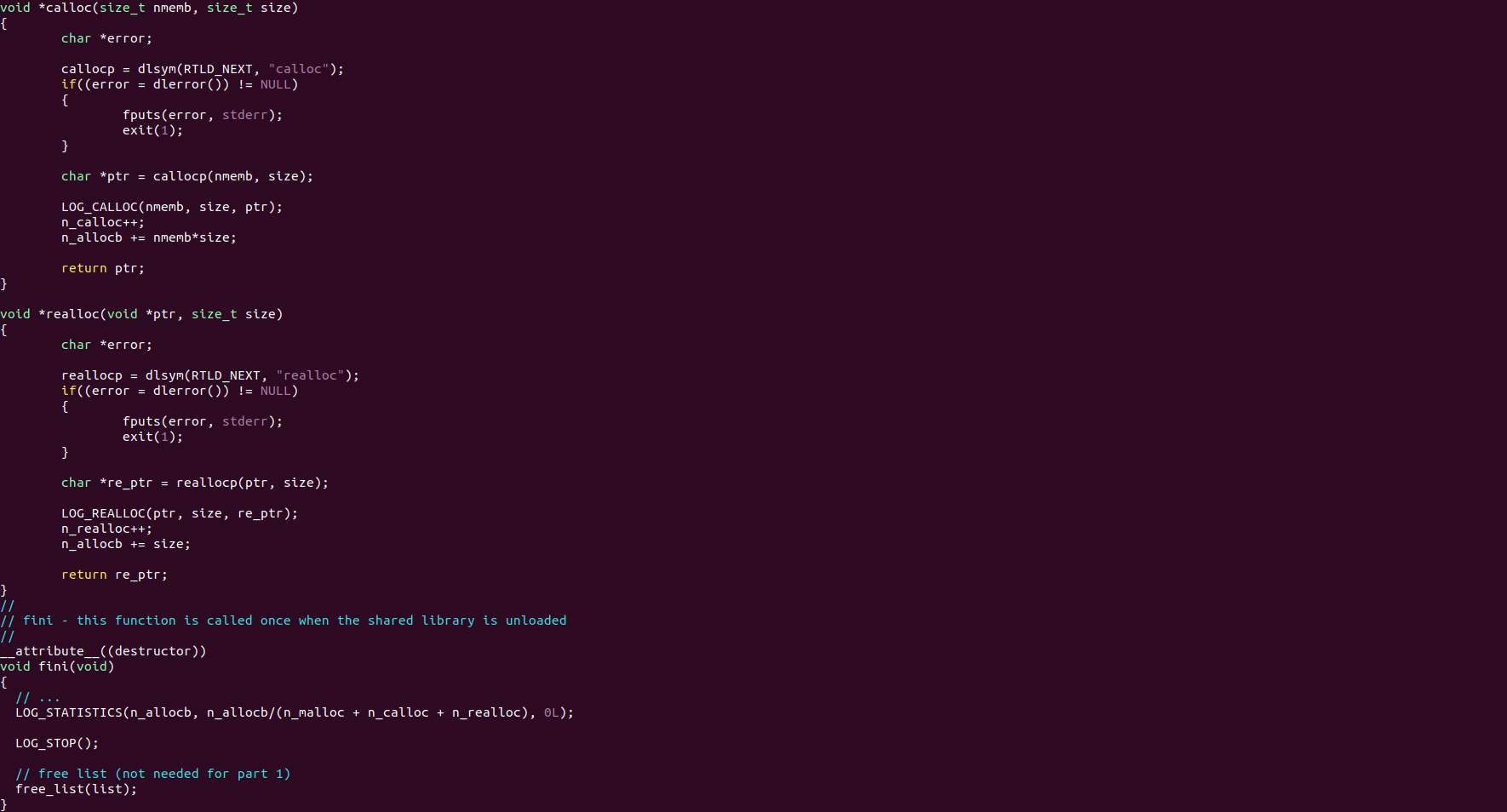


먼저 위의 코드는 memtrace.c의 초반이다. 여러 헤더파일을 include한다. 그리고 standard library를 가리키는 function pointer를 선언한다. 통계값을 나타내기 위한 long 변수도 선언한다. init함수는 shared library가 load될 때 한번만 call되며 init함수 안에는 기록을 시작하는 함수와 allocation 기록을 남기기위한 list가 있다. 앞으로 이부분은 수정하지 않으므로 다음 내용에 싣지 않을 것이다.

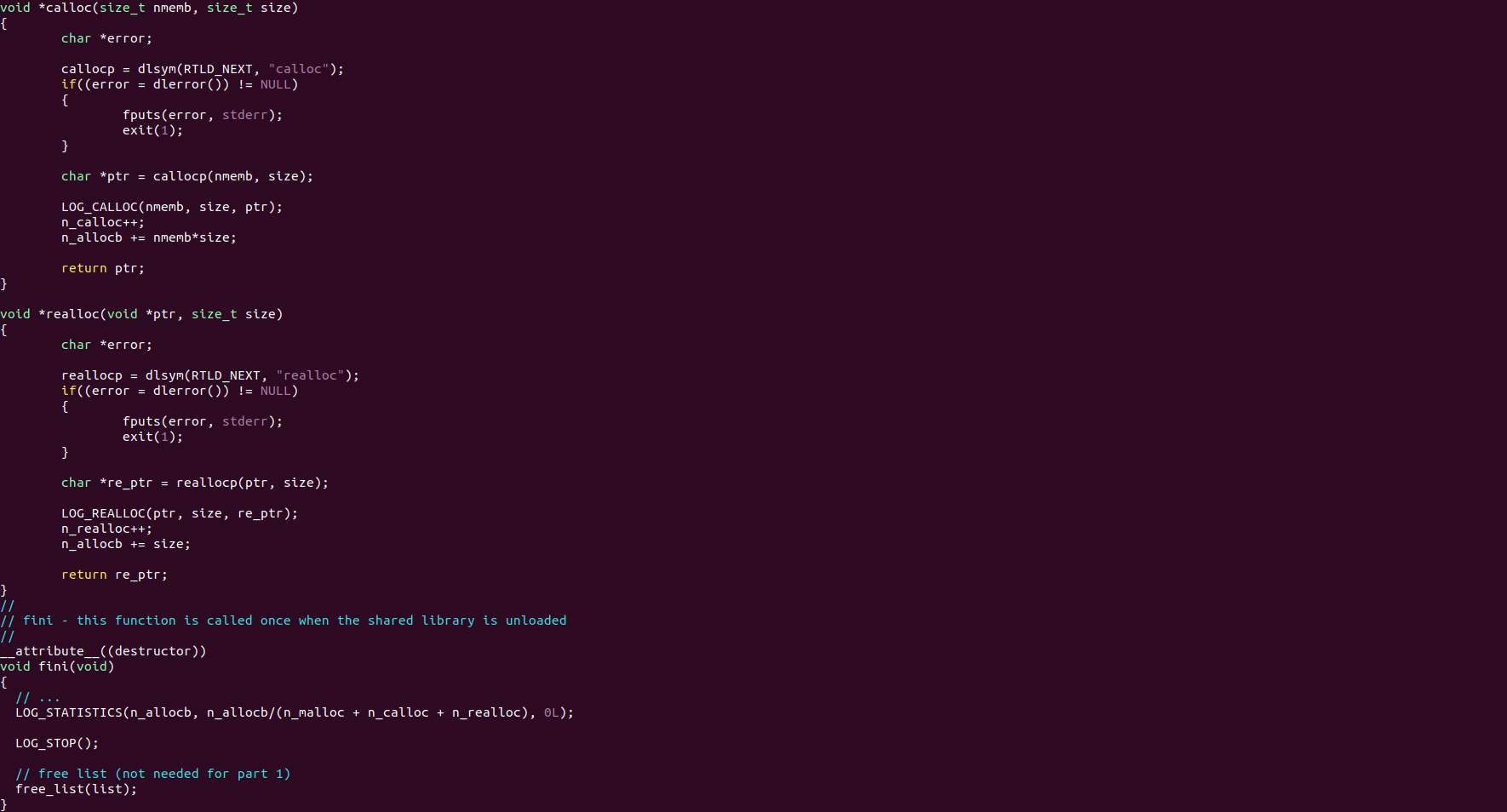


malloc함수에서 mallocp 포인터는 dlsym함수로부터 memory에 로드된 심볼의 주소를 받는다. 만약 에러가 나면 stderr가 생기고, 그렇지 않으면 if문을 빠져나가 stdlib의 malloc을 수행한다. 그리고 프로그래머가 의도하는 기능을 추가한다. LOG\_MALLOC함수를 이용하여 할당 크기와 주소를 기록하고 malloc 횟수를 늘리고 alloc byte에 size를 더한다. 그리고 할당 주소 ptr을 return한다.

free함수는 malloc함수와 같은 방식으로 에러처리를 하고 freep를 이용하여 stdlib의 free함수를 call하고 그 내용을 기록한다.



calloc과 realloc function pointer도 전체적인 맥락은 malloc function pointer와 같다. 다른점은 calloc의 경우 alloc byte에 nmemb와 size를 곱한 값을 넣어주어야 한다. 그것이 전체 할당 사이즈이기 때문이다. 이 부분을 놓치고 있다가 나중에서야 깨닫고 수정했다. realloc의 경우 argument인 ptr과 함수 내에서 선언한 포인터가 다르다. 그 이유는 realloc함수가 call될 때 다른 주소에 할당하는 경우가 종종 생기기 때문이다. (물론 두 포인터의 값이 같을 때도 있다.) 그래서 LOG\_REALLOC함수를 보면 첫번째 argument는 ptr이고 할당 결과 주소인 re\_ptr은 세번째 위치한다. Return 값도 re\_ptr이다.



마지막으로 fini함수가 있다. 이 함수 내에서는 전체 할당 바이트 수, alloc함수 call 횟수 당 바이트 수, free한 전체 바이트 수를 출력하는 LOG\_STATISTICS함수가 있고, 기록을 끝내는 LOG\_STOP함수가 있다. 마지막으로 할당 정보를 담고 있던 list를 free해준다.

② 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

test1.c와 test2.c를 실행했을 때 나오는 결과다. Test1에서 malloc한 size모두가 allocated\_total에 반영되었고 malloc함수를 3번 call했으므로 average도 맞다.

Test2에서는 malloc후 free를 한다. 함수의 정보가 잘 출력되고 통계값도 잘 나오는 것을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Test3,4,5를 실행한 결과다. Test3의 경우 alloc함수를 10번 call 했기 때문에 allocated\_total은 average의 10배이다.

Test4의 경우 segfault가 생긴다. 이는 free함수에서 예외처리를 해주지 않았기 때문이다.

Test5의 경우 여러 번 realloc을 하는데 잘 수행되는 것을 볼 수 있다.

3. Part2: Tracing Unfreed Memory

① 구현



Part2는 free되지 않은 memory를 trace하는 부분이다. 할당정보(포인터, reference 횟수, 함수이름, size 등)가 담긴 list에서 free의 argument인 포인터가 있는 item을 찾아 그 item의 element인 size를 n\_freeb(the number of freed byte)에 더한다. 그리고 그 아이템의 요소들을 delloc함수로 업데이트한다.



malloc함수에서는 alloc함수를 이용해 ptr의 값을 가진 item의 요소들을 업데이트한다. 이 때 size는 malloc함수의 argument인 size이다.



calloc함수도 마찬가지로 alloc함수를 이용해 ptr의 값을 가진 item의 요소들을 업데이트한다. 이 때 size는 nmemb\*size이다. 크기가 size인 메모리 할당이 nmemb개 이루어지기 때문이다.



realloc함수는 free 후 alloc을 한다. Realloc을 하기 전에 할당되었던 size를 n\_freeb에 더하고 그 주소의 item을 dealloc한다. 그 후 새로운 주소(변했는지 안 변했는지는 모른다.)와 함께 alloc함수를 이용하여 item을 업데이트한다.

이 부분에서 re\_ptr 대신 ptr을 넣어서 segmetation fault가 생겼다. 그 부분을 해결하기 위해 realloc에 대한 정보가 필요했고, 그래서 realloc을 call할 때 주소가 변할 수도 있다는 사실을 알게 되었다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

free되지 않은 부분을 출력하기 위해 구현한 부분이다. List의 모든 item을 살펴보되, free되지 않아서 reference counter가 0이 아닌 item의 요소만 출력한다.

② 결과

텍스트, 전화이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Test1, 2를 실행한 결과다. Part1과 달리 freed\_total과 free되지 않은 block의 정보가 표시된다. free되지 않은 block이 없는 경우 아무 정보도 표시하지 않는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Test3에서 allocate한 만큼 올바르게 free한 것을 통계를 통해 알 수 있다.

앉아있는, 테이블, 대형, 검은색이(가) 표시된 사진

자동 생성된 설명

Test4는 free에 대한 예외처리를 해주지 않았기 때문에 error가 발생한다.

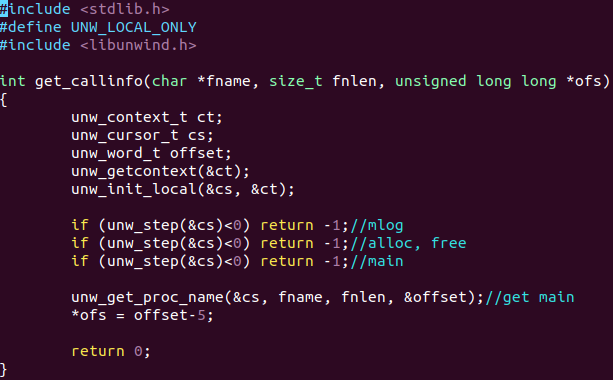
텍스트, 테이블, 앉아있는, 검은색이(가) 표시된 사진

자동 생성된 설명

Test5는 alloc한만큼 free도 잘 수행했음을 알 수 있다.

4. Part3: Pinpointing Call Location

① 구현



Part3에서는 libunwind library를 이용하여 callinfo.c를 작성했다. Libunwind의 함수를 이용하여 backtracing했다. 먼저 context, cursor, offset 변수를 선언한다. unw\_getcontext함수로 현재의 machine state를 get하고 unw\_init\_local함수로 cursor를 초기화한다. 그 후 커서를 이동시키기 위하여 unw\_step함수를 이용한다. 이 함수는 stack frame에서 이전의 stack frame으로 이동하는 함수다. 커서가 처음에는 get\_callinfo에 위치하다가 한번 call하면 mlog에 위치하고, 한번 더 call하면 alloc과 free함수에 위치하고, 한번 더 call하면 main함수에 위치하게 된다. 총 3번 call하면 cursor가 main에 위치하게 된다. 그 후 unw\_get\_proc\_name함수를 call하여 현재 커서가 위치하는 procedure의 이름을 얻는다. 그 후 offset 값을 보정하는 과정을 거쳤는데, callq instruction의 크기가 항상 5bytes라서 offset의 값이 실제 위치보다 5가 더 크게 나와서 5를 빼서 보정해주었다. 출력되는 offset값은 callq instruction의 offset 값이다.

② 결과

텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명

Alloc과 free함수는 모두 main함수에서 일어난다. 그러므로 항상 함수가 call되는 위치는 main으로 표시된다. 그리고 함수가 call되는 위치의 offset이 표시되는데, 이는 함수 call의 주소에 main의 주소를 뺀 값이다. Non-deallocated된 block을 출력하는 과정에서도 caller의 이름 main과 callee의 offset값을 출력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Test2,3을 실행했을 때의 결과다. caller함수의 name과 offset값이 출력된다.

병, 사진, 검은색, 표지판이(가) 표시된 사진

자동 생성된 설명

텍스트, 테이블, 앉아있는, 쥐고있는이(가) 표시된 사진

자동 생성된 설명

Test3, 4를 실행했을 때의 결과다. caller함수의 name과 offset값이 출력된다.

5. Bonus: Detect and Ignore Illegal Deallocations

① 구현

전화, 휴대폰, 앉아있는, 테이블이(가) 표시된 사진

자동 생성된 설명

Illegal free를 한 후에 double free를 처리한다. 왜냐하면 double을 먼저 처리할 경우 item이 NULL면 segmentation fault가 뜨기 때문이다. Illegal free는 free하고자하는 주소가 list에 없을 때 발생한다. Allocation이 된 적 없는 주소에 free를 하는 셈이다. 그래서 메시지를 표시하고 return하도록 했다. Double free는 이미 free된 주소를 다시 free하는 것이다. 할당되었다가 free되어 reference count가 이미 0인 것을 다시 free하면 오류가 생긴다. 이러한 오류를 막기 위해 메시지를 띄우고 return하여 함수를 끝냈다.

앉아있는, 주차, 검은색, 대형이(가) 표시된 사진

자동 생성된 설명

Realloc은 free했다가 alloc하는 함수이므로 수정해주어야 한다. Illegal free와 같은 맥락으로 주소가 list에 없는 경우, double free와 같은 맥락으로 이미 free가 되어 또다시 free를 할 필요가 없는 경우 두가지에 대하여 생각해보아야 한다. 첫번째 경우와 두번째 경우 모두 free에 대한 처리는 하지 않고 alloc에 대한 처리만 해준다. 첫번째 경우와 두번째 경우 모두 free를 해줄 필요가 없기 때문이다. reallocp(NULL, size)를 하면 malloc과 똑 같은 기능을 한다.

② 결과

텍스트, 테이블, 컴퓨터, 앉아있는이(가) 표시된 사진

자동 생성된 설명

Double free와 illegal free가 제대로 작동한 모습이다. 같은 주소에 한번 더 free를 할 때 double free가 발생하여 free를 무시한다. 할당한 적 없는 주소에 free를 할 때는 illegal free가 발생하여 free를 무시한다.

6. Conclusion

Linker lab을 통해 Load/run time interpositioning에 대하여 제대로 이해하게 되었다. 까다로운 과정이었지만 직접 코딩해보며 프로그램이 어떻게 동작하는지 알아볼 수 있는 좋은 기회였다. 이번 랩을 통하여 terminal을 조작하는 방법, segmentation fault를 다루는 방법을 알게 되었다. 값진 경험이었다.