2021 SYSTEM PROGRAMMING

Lab1 Report – Link Lab

자유전공학부 2012-13311 안 효 지

1. 실행 결과

- part3에 part1, 2의 결과가 모두 포함되므로 part3의 결과만 첨부하였다.

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/me
l
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0×5563364182d0
[0003]           malloc( 32 ) = 0×556336418710
[0004]           malloc( 1 ) = 0×556336418770
[0005]           free( 0×556336418770 )
[0006]           free( 0×556336418710 )
[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
[0015]   0×5563364182d0     1024       1
[0016]
[0017] Memory tracer stopped.
```

- test1 : allocated total은 malloc한 모든 byte수 더하면 1057 나온다. Allocated avg는 1057 / malloc 일어난 횟수 3 = 352. Freed total은 32+1 = 33. 맨 처음 할당된 1024는 dealloc되지 않았으므로 non-dealloc에 나오게 된다.

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test2
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/me
l
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0×55c4a1acb2d0
[0003]           free( 0×55c4a1acb2d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
```

- test2 : 특별한 설명이 필요 없으며, 모든 allocated block이 dealloc되었으므로 블록 결과도 나오지 않는다.

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test3
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/me
l
[0001] Memory tracer started.
[0002]           calloc( 1 , 36142 ) = 0x560ad9a4a2d0
[0003]           calloc( 1 , 47487 ) = 0x560ad9a53040
[0004]           malloc( 47567 ) = 0x560ad9a5ea00
[0005]           calloc( 1 , 43980 ) = 0x560ad9a6a410
[0006]           calloc( 1 , 27939 ) = 0x560ad9a75020
[0007]           calloc( 1 , 12169 ) = 0x560ad9a7bd80
[0008]           malloc( 309 ) = 0x560ad9a7ed50
[0009]           calloc( 1 , 25155 ) = 0x560ad9a7eec0
[0010]           malloc( 60166 ) = 0x560ad9a85140
[0011]           malloc( 38683 ) = 0x560ad9a93c80
[0012]           free( 0x560ad9a93c80 )
[0013]           free( 0x560ad9a85140 )
[0014]           free( 0x560ad9a7eec0 )
[0015]           free( 0x560ad9a7ed50 )
[0016]           free( 0x560ad9a7bd80 )
[0017]           free( 0x560ad9a75020 )
[0018]           free( 0x560ad9a6a410 )
[0019]           free( 0x560ad9a5ea00 )
[0020]           free( 0x560ad9a53040 )
[0021]           free( 0x560ad9a4a2d0 )
[0022]
[0023] Statistics
[0024]   allocated_total      339597
[0025]   allocated_avg        33959
[0026]   freed_total          339597
[0027]
[0028] Memory tracer stopped.
```

- test3 : calloc을 테스트한다. 특별히 첨언할 내용이 없다.

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/mem
l
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x55c283cbe2d0
[0003]           free( 0x55c283cbe2d0 )
[0004]           free( 0x55c283cbe2d0 )
[0005]     *** DOUBLE_FREE  *** (ignoring)
[0006]           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.
```

- test4: line3에서 free한 주소를 line4에서 또 free하려고 하므로 double free이다. Line6에서는 애초에 할당한 적 없었던 곳을 free하려 하므로 illegal free이다.

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test5
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/mem
l
[0001] Memory tracer started.
[0002]           malloc( 10 ) = 0x56268619e2d0
[0003]           realloc( 0x56268619e2d0 , 100 ) = 0x56268619e320
[0004]           realloc( 0x56268619e320 , 1000 ) = 0x56268619e3c0
[0005]           realloc( 0x56268619e3c0 , 10000 ) = 0x56268619e7e0
[0006]           realloc( 0x56268619e7e0 , 100000 ) = 0x5626861a0f30
[0007]           free( 0x5626861a0f30 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

- test5: 사이즈가 커지는 realloc의 경우를 다룬다. 사이즈가 커지므로 이전 블록은 dealloc하고 새 블록을 alloc해야 한다. 마지막에 free까지 있으므로 alloc total == freed total 이어야 하며, malloc 1회, realloc 4회에서 모두 alloc이 일어나므로 alloc avg는 total / 5를 하면 된다. Dealloc 후 alloc을 한 것은 realloc의 인자와 return value가 다른 것을 보면 알 수 있다.

아래는 제공된 testcase에서 커버하지 못하는 case들을 커버하기 위해 만들었다.

File: test0.c

```c
#include <stdio.h>

int main() {
    return 0;
}

```

```
~/workspace/2021course/splab_2021spr
cc -I. -I ../utils -o libmemtrace.so
list.c -ldl
[0001] Memory tracer started.
[0002]
[0003] Statistics
[0004]   allocated_total      0
[0005]   allocated_avg        0
[0006]   freed_total          0
[0007]
[0008] Memory tracer stopped.
```

- test0은 아무런 입력이 들어오지 않는 경우이다. average를 단순히 total / count로 구하면 아무것도 없는 경우에는 count==0이 되어 divide by 0 가 일어나게 되는데, 이를 방지하기 위한 코드의 삽입이 필요함을 보여준다.

- 아래에 첨부된 test6은 test4에 나오는 free()의 ill/double free뿐만 아니라 realloc()에서도 나타날 수 있는 ill/double free를 캐치하기 위해 만들었다. 또한, realloc()에서 이전보다 사이즈가 줄어드는 경우, 아무일도 일어나지 않지만 log는 제대로 출력되는 것을 확인하는 코드도 삽입하였다.

File: test6.c

```c
// test4 revised
#include <stdlib.h>

int main(void)
{
  void *a;
  a = malloc(1024);
  free(a);
  free(a);                // double free
  free((void*)0xcafe);    // ill free
  realloc(a, 20);         // realloc double free
  void* b = malloc(40);
  realloc(b, 30);         // size가 줄어들어서 nothing happens
  realloc((void*)0xdeadbeef, 50); // realloc ill free

  /*statistics*/
  // total alloc : 1024 + 20 + 40 + 50 = 1134
  // alloc avg : 1134 / 4 (line7 + line11 + line12 + line14 = 4번)
  // freed bytes : line8에서 1024

  return 0;
}
```

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test6
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/meml
l
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x5653716292d0
[0003]           free( 0x5653716292d0 )
[0004]           free( 0x5653716292d0 )
[0005]     *** DOUBLE_FREE  *** (ignoring)
[0006]           free( 0xcafe )
[0007]     *** ILLEGAL_FREE *** (ignoring)
[0008]           realloc( 0x5653716292d0 , 20 ) = 0x565371629710
[0009]     *** DOUBLE_FREE  *** (ignoring)
[0010]           malloc( 40 ) = 0x565371629760
[0011]           realloc( 0x565371629760 , 30 ) = 0x565371629760
[0012]           realloc( 0xdeadbeef , 50 ) = 0x5653716297c0
[0013]     *** ILLEGAL_FREE *** (ignoring)
[0014]
[0015] Statistics
[0016]   allocated_total      1134
[0017]   allocated_avg        283
[0018]   freed_total          1024
[0019]
[0020] Non-deallocated memory blocks
[0021]   block              size       ref cnt
[0022]   0x565371629710     20         1
[0023]   0x565371629760     40         1
[0024]   0x5653716297c0     50         1
[0025]
[0026] Memory tracer stopped.
```

-test6: line4에서는 직전에 free한 주소를 또 free하려고 하므로 double free가 발생한다. Line6에서는 한 번도 alloc되지 않았던 곳을 free하려 하므로 illegal free가 발생한다. Line8에서는 이미 line3에서 정상적으로 free가 된 곳에 realloc을 하려고 하므로 double free가 발생하지만, 이를 무시하고 20을 할당한 후 그 주소를 반환한다. Line11에서는 line10에서 40만큼 할당했던 곳을 30으로 재할당하라고 하는 것이기 때문

에 아무런 일도 일어나지 않는다. 이는 realloc에 인자로 들어온 주소와 return으로 나오는 주소가 같은 것을 보면 확인할 수 있다. Line 12에서는 한 번도 alloc되지 않은 곳에 realloc을 시키므로 illegal free가 나타나지만, 이를 무시하고 50만큼을 alloc한 후에 그 주소를 반환한다. Statistics에 관한 설명은 test6.c에 주석으로 설명해놓았다.

File: test7.c

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    // 1. malloc size = 0인 경우 -> either null or a unique pointer
    //      그냥 null 출력하게 만듦.
    void* a = malloc(0);

    // 2. calloc ptr, size 둘 중 최소 하나 0인 경우.
    void* b = calloc(0, 20);    // null return하게 만듦.
    b = calloc(4, 0);           // 상동.
    b = calloc(0, 0);           // 상동.

    // 3. realloc ptr null인 경우 -> malloc(size)역할.
    void* c = realloc(NULL, 20);
    c = realloc(NULL, 0);

    // 4. realloc size == 0인 경우. free() 역할 함.
    void* d = malloc(5);
    void* e = realloc(d, 0);            // 직전 malloc을 free시켜야 함.
    e = realloc((void*)0xdeadbeef, 0);  // ill free
    e = realloc(d, 0);                  // double free

}
// line 15, 19에서만 실제로 allocated되므로 total / 2하면 됨.
```

```
~/workspace/2021course/splab_2021spring/1.linklab/handout/part3 master* ↑ ❯ make run test7
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/mem
l
[0001] Memory tracer started.
[0002]          malloc( 0 ) = (nil)
[0003]          calloc( 0 , 20 ) = (nil)
[0004]          calloc( 4 , 0 ) = (nil)
[0005]          calloc( 0 , 0 ) = (nil)
[0006]          malloc( 20 ) = 0x563d5aa422d0
[0007]          malloc( 0 ) = (nil)
[0008]          malloc( 5 ) = 0x563d5aa42320
[0009]          realloc( 0x563d5aa42320 , 0 ) = (nil)
[0010]          realloc( 0xdeadbeef , 0 ) = (nil)
[0011]     *** ILLEGAL_FREE *** (ignoring)
[0012]          realloc( 0x563d5aa42320 , 0 ) = (nil)
[0013]     *** DOUBLE_FREE  *** (ignoring)
[0014]
[0015] Statistics
[0016]   allocated_total      25
[0017]   allocated_avg        12
[0018]   freed_total          5
[0019]
[0020] Non-deallocated memory blocks
[0021]   block              size       ref cnt
[0022]   0x563d5aa422d0     20         1
[0023]
[0024] Memory tracer stopped.
```

-test7에서는 malloc(0), calloc(0, size), calloc(ptr, 0), calloc(0, 0), realloc(null, size), realloc(ptr, 0), realloc(null, 0)의 코너케이스들을 테스트한다.

- malloc(0)은 매뉴얼에 보면 null이나 나중에 free될 수 있는 pointer를 반환하라고 나와있으므로 여기서는 임의로 NULL을 반환하게 만들었다. 결과 사진의 line2 참고.

- calloc()에서도 size==0 | ptr==0 인 경우에 null을 반환하게 해놓았다.

- 결과사진의 line6, 7은 realloc(null, size)를 확인하기 위한 코드이다. 컴파일러가 realloc의 포인터가 null인 경우에는 malloc으로 치환하므로 결과에는 malloc으로 나오게 된다.

- 결과사진의 line9는 line8에서 malloc(5)로 allocated된 블록이 realloc(ptr, 0)을 통해 free되는지 확인할 수 있다. free가 되었으므로 return value는 null로 설정했다.

- 결과사진의 line10은 한 번도 할당되지 않았던 곳을 realloc(ptr, 0)해서 free시키라는 의미이므로 illegal free가 나오며, 이 역시 return value는 null이다.

- 결과사진의 line12는 이미 line9에서 realloc(ptr, 0)에 의해 free된 곳을 또 realloc(ptr, 0)으로 free하라고 하는 것이므로 double free가 일어난다.

2. 구현방법

<Part1. Tracing allocated memory>

Dynamic linking은 .so에 대한 정보만 가지고 있는 partially linked executable file이 실제로 메모리에 load되고 실행될 때, 메모리에 올라와있는 .so에서 필요한 definition들을 링킹하여 fully linked executable file로 되는 것을 의미한다. 이 과정에서 LD_PRELOAD 환경변수가 shared library의 경로로 되어있으면 stdlib보다 그 경로에 먼저 가서 찾는 성질을 이용하여 interpositioning 하는 것이 load/run time interposition이다. 따라서 실제 우리가 호출하는 malloc, calloc, realloc, free 함수의 이름과 동일한 wrapper 함수를 만들어주어야 한다.

(1) libc의 함수 주소 받아오기

각 wapper 함수에서 dlsym(RTLD_NEXT, "..."); 를 통해 libc에 존재하는 original 함수의 주소를 받아온다. 함수명은 주소이기 때문에 그 주소에 인자들을 넣어 return value를 받는다.

(2) Tracing dynamic memory allocation

malloc, calloc의 경우에는 할당되는 size를 Statistics에 print되는 allocated_total(n_allocb)에 더해준다. 또한, allocated_avg를 구하기 위해 각 wrapper함수가 호출될 때마다 n_malloc, n_calloc 에 ++를 해준다. (realloc은 후술)

(3) Statistics 출력 코드 넣기

계산된 변수들을 이용하여 destructor에서는 LOG_STATISTICS가 출력된다. alloc_avg = n_allocb / (n_malloc+n_calloc+n_realloc) 을 계산하는 과정에서 divided by 0가 발생하지 않도록 divisor == 0인 경우엔 평균도 0으로 출력되도록 구현하였다. 이미 제공된 매크로함수를 이용하였다.

(4) realloc 함수 구현할 때 주의할 점

과제에서는 old size < new size인 경우에만 dealloc -> alloc을 해야하므로 크기 비교를 먼저 해야 한다. 크기 비교를 위해서는 list 사용이 필수이므로 모든 wrapper함수에 alloc 및 dealloc을 집어넣었다. Statistics는 alloc/dealloc이 일어날 때에만 update 되어야 하므로 old < new 인 경우에만 n_realloc ++ 및 n_allocb+=size를 해주도록 한다. → spec이 수정되어 list를 사용하지 않고 그냥 무조건 alloc을 하게 코드를 변경하였다.

<Part2. Tracing freed memory>

(1) allocated, freed memory 계산하기

memory tracing을 위해 linked list가 사용된다. ①Malloc, ②calloc의 경우에는 memlist.c에 있는 alloc()을 사용해 list에 추가를 한다. ③Realloc의 경우에는 먼저, size가 0이면 free의 역할을 하도록 한다. 0이 아닌 경우에는 <part1>에서도 언급했다시피 크기 비교를 먼저 해야하므로, list에서 인자로 들어온 ptr을 find하여 old < new 인 경우에만 기존의 할당된 공간을 없애기 위해 dealloc()을 한다. list node의 cnt을 감소시켜 free가 된 사실을 표시하고 기존 공간의 size를 n_freeb에 더해준다. 그리고 새로 할당된 공간의 주소를 받아 alloc()하여 list에 추가하고 n_allocb에 해당 사이즈만큼 더해준다. New가 old보다 작거나 같은 경우에는 statistics에 아무런 일도 일어나지 않는다. ④Free의 경우에는 인자로 전달받은 주소를 dealloc()으로 보내고, 그 노드의 사이즈를 n_freeb에 더해서 total_freed_byte를 구한다.

(2) unfreed memory 출력하기

total_allocated_byte와 total_freed_byte가 같아야 unfreed memory가 없는 것이다. 따라서 두 값이 같지 않은 경우에만 "Non-deallocated memory block"가 출력되도록 하였다. List를 순서대로 iterate하며 non-freed block을 의미하는, cnt가 0이 아닌 노드의 정보만을 출력하게 하였다.

<Part3> Illegal/Double free

@ free()

(1) illegal free check

free()에 argument로 들어온 ptr이 list에 있는지 유무를 find()를 통해 확 인해보면 된다. 존재하지 않는다면 LOG_ILL_FREE()를 호출하여 메시지를 출력하도록 한다.

(2) double free check

illegal free는 아니라면 이미 free된 곳인지 확인해야 한다. (1)에서 find()를 했던 return value를 변수에 저장해놓고, 해당 변수의 cnt값이 0인지 확인하여 0이라면 double free 메시지를 출력하도록 한다.

(3) normal free

illegal free도, double free도 아니라면 part2처럼 동작하도록 두면 된다.

@ realloc()

1) size == 0 인 경우

size가 0이면 alloc은 이루어지지 않고 free의 역할만을 한다. Ptr이 한 번도 입력된 적 없는 것이면 바로 illegal free를, 이미 free한 곳이면 double free를, 현재 할당되어 있는 곳이면 free를 하게 한다. 이 때에는 비록 realloc()이 호출되었지만 실제로 작동하는 것은 free이므로, realloc의 호출횟수를 의미하는 n_realloc은 ++해주지 않는다. 다만 dealloc이 되었으니 n_freeb에는 사이즈만큼을 더해준다.

2) size != 0인 경우

(1) illegal free check

Free()에서 했던 것과 마찬가지로, find()를 통해 argument로 들어온 ptr이 list에 존재하지 않는지 확인한다. 존재하지 않으면 illegal free에 해당한다. 하지만 이 오류를 무시하고 alloc() 단계는 제대로 수행되어야 하므로 reallocp(NULL, size)를 이용해 malloc(size)처럼 작동되도록 코드를 짜고, alloc()을 통해 list에 추가 및 total_allocated bytes에 할당된 size를 더해주고, LOG_REALLOC을 먼저 출력한 후 LOG_ILL_FREE를 출력한다.

(2) double free check

free()와 마찬가지로, find()를 했던 결과를 받아놓고, 해당 node의 cnt 값이 0이면 이미 free(dealloc)이 된 상태를 의미하므로 이를 확인한다. 이후 작업 역시 realloc의 illegal free check과 동일하다.

(3) normal realloc

정상적으로 작동하면 되므로 part2의 realloc의 작업을 수행한다.

3. 어려웠던 점

강의시간에 library interposition에 대해 배우긴 했지만, 프로그래머의 필요 때문에 코드를 중간에 탈취하는 방법 정도로 추상적으로 이해하고 있을 뿐이었다. 이것이 정확히 왜 필요하고 어디에 쓰이는지 직접 경험한 적이 없었으므로 개념 자체가 낯선 것 자체가 과제 수행 과정에서 어려웠던 점이다. 하지만 과제 수행이라는 구체적인 상황이 주어진 상태에서 csapp 교재와 강의pdf를 참고해 shared library, dynamic linking의 개념을 확실히 복습하고 library interposition 부분을 읽으니 compile, link, load/run time interposition의 원리와 특성이 확실히 이해가 되었다.

함수 포인터에 대해 C언어 개념을 공부할 때 한 번 듣고 넘어간 적이 있었지만 컴퓨터구조 수업에서도 직접 사용해본 적이 없었기에 함수포인터 문법이 낯설었다. 하지만 개념을 다시 살피고 과제를 하며 직접 사용해 보았으므로 앞으로는 잊을 일이 없을 듯하다.

4. 새롭게 배운 점

part2의 test5를 통과하기 위해 list를 이용해 memory tracing을 하는데 realloc에서만 자꾸 오류가 났다. Realloc에 대한 잘못된 이해가 원인이었다. `void* rptr = realloc(ptr, size);` 에서, 무조건 기존에 allocated된 주소(ptr)에 새로운 size를 할당하여 항상 rptr==ptr인 줄 착각하고 있던 것이었다. 한참 코드를 손보다가 `$ man realloc`을 해보니 return value와 argument는 같을 수도 있고 다를 수도 있으며, size가 0인 경우에는 free()의 역할을, ptr이 NULL이면 malloc(size)의 역할을 할 수도 있다는 것을 알게 되었다.

또한, realloc에 관한 과제의 스펙을 파악하는 과정에서 실제 컴퓨터에서는 realloc이 어떻게 돌아가는지 많은 공부를 할 수 있었다. 그 과정에서, 구현체에 따라 다르기는 하지만, 가령 malloc(100)을 한 경우에 할당된 사이즈 정보를 저장하기 위해 실제로는 100byte 이상을 할당하기도 한다는 사실을 새로이 알게 되었다.