시스템 프로그래밍 Linker lab 보고서

2017-12751 컴퓨터공학부 이동학

이번 lab에서는 load time interpositioning을 통해 malloc, calloc, realloc, free 4개의 함수를 intercept하여 memory trace를 통해 name, arguments, return value 및 각종 statistics를 출력하고, 더 나아가 illegal deallocation을 감지하는 프로그램을 작성하였습니다.

Part 1. Tracing dynamic memory allocation

```c
void *malloc(size_t size)
{
    char *error;
    void *ptr;

    if(!mallocp) {
        mallocp = dlsym(RTLD_NEXT, "malloc");
        if ((error = dlerror()) != NULL) {
            mlog(0, error);
            exit(1);
        }
    }

    ptr = mallocp(size);
    n_allocb += size;
    n_malloc++;
    LOG_MALLOC(size, ptr);
    return ptr;
}
```

```
[0001] Memory tracer started.
[0002]           (nil) : malloc( 1024 ) = 0xad7060
[0003]           (nil) : malloc( 32 ) = 0xad7470
[0004]           (nil) : malloc( 1 ) = 0xad74a0
[0005]           (nil) : free( 0xad74a0 )
[0006]           (nil) : free( 0xad7470 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
```

```c
void fini(void)
{
  // ...

  LOG_STATISTICS(n_allocb, n_allocb / (n_malloc + n_calloc + n_realloc), 0L);

  LOG_STOP();

  // free list (not needed for part 1)
  free_list(list);
}
```

Malloc를 예로 들면 dlsym을 이용해 원래 함수를 받아와서 메모리 할당량, 함수 호출 횟수를 저장하는 기능을 추가로 구현해주었습니다. 그 후 shared library가 unload되면 총 메모리 할당량, 평균 메모리 할당량을 출력하도록 하였습니다.

Part 2. Tracing unfreed memory

```c
void free(void *ptr)
{
    char *error;

    if(!freep) {
        freep = dlsym(RTLD_NEXT, "free");
        if((error = dlerror()) != NULL) {
            mlog(0, error);
            exit(1);
        }
    }

    freep(ptr);
    n_freeb += find(list, ptr)->size;
    dealloc(list, ptr);
    LOG_FREE(ptr);
}
```

```
[0001] Memory tracer started.
[0002]          (nil) : malloc( 1024 ) = 0xe08060
[0003]          (nil) : malloc( 32 ) = 0xe084c0
[0004]          (nil) : malloc( 1 ) = 0xe08540
[0005]          (nil) : free( 0xe08540 )
[0006]          (nil) : free( 0xe084c0 )
[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]   0xe08060           1024       1          ???:0
[0016]
[0017] Memory tracer stopped.
```

```c
void fini(void)
{
  LOG_STATISTICS(n_allocb, n_allocb / (n_malloc + n_calloc + n_realloc), n_freeb);

    item *i = list->next;
    int check = 0;

    while(i != NULL) {
        if(i->cnt > 0) {
            if(check == 0) {
                LOG_NONFREED_START();
                check = 1;
            }
            LOG_BLOCK(i->ptr, i->size, i->cnt, i->fname, i->ofs);
        }

        i = i->next;
    }

  LOG_STOP();

  // free list (not needed for part 1)
  free_list(list);
}
```

파트2에서는 총 메모리 해제량과 해제되지 않은 메모리 블럭을 찾기 위해 메모리 리스트를 이용하였습니다. 메모리를 할당될때마다 리스트에 추가하고 해제될때마다 리스트에서 삭제하는 대신, cnt값을 0으로 바꾸고 메모리 해제량을 저장했습니다. 리스트에서 삭제하지 않은 이유는 후에 Bonus에서 Double free와 Illegal free를 구분하기 위함입니다. 그 후 shared library가 unload되면 non-deallocated memory block이 있는 경우에 해당 블록의 address, size, ref cnt, caller를 출력하도록 하였습니다.

Part 3. Pinpointing call locations

```c
#include <stdlib.h>
#define UNW_LOCAL_ONLY
#include <libunwind.h>
#include <string.h>

int get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs)
{
    unw_context_t context;
    unw_cursor_t cursor;
    unw_word_t off;
    char procname[256];
    int ret = 0;

    if(unw_getcontext(&context)) {
        return -1;
    }

    if(unw_init_local(&cursor, &context)) {
        return -1;
    }

    while(unw_step(&cursor) > 0) {
        if(unw_get_proc_name(&cursor, procname, 256, &off)==0) {
            strcpy(fname, procname);
            *ofs = off - 5;
            if(strcmp(fname, "main") == 0){
                break;
            }
        }
    }

    return 0;
}
```

```
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0x7feb04cb7168
[0003]         main:10 : malloc( 32 ) = 0x249e060
[0004]         main:1d : malloc( 1 ) = 0x249e4c0
[0005]         main:25 : free( 0x249e540 )
[0006]         main:2d : free( 0x249e4c0 )
[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]   0x249e060          1024       1         main:6
[0016]
[0017] Memory tracer stopped.
```

파트 3 에서는 allocation 과 deallocation 이 memory 의 어느 부분에서 발생하는지 알아보기 위해 libunwind 를 이용하였습니다. unw_getcontext 와 unw_init_local 을 이용해 변수를 초기화한 후,

unw_step 을 통해 역으로 거슬러 올라가면서 function name 과 PC offset 을 검색합니다. 과제 스펙에서 alloc, dealloc 은 main 함수에서만 이루어지고 callq 는 5 바이트라고 가정했기 때문에 strcmp 를 이용해 function name 이 main 이면 그때 PC offset 에서 5 를 뺀 값을 반환하도록 했습니다.

Bonus: detect and ignore illegal deallocations

```
void *realloc(void *ptr, size_t size)
{
    char *error;
    void *ptr2;

    if(!reallocp) {
        reallocp = dlsym(RTLD_NEXT, "realloc");
        if((error = dlerror()) != NULL) {
            mlog(0, error);
            exit(1);
        }
    }

    LOG_REALLOC(ptr, size, ptr2);

    if(find(list, ptr) == NULL) {
        LOG_ILL_FREE();
        ptr2 = reallocp(NULL, size);
    }
    else if(find(list, ptr)->cnt == 0) {
        LOG_DOUBLE_FREE();
        ptr2 = reallocp(NULL, size);
    }
    else {
        ptr2 = reallocp(ptr, size);
        n_freeb += find(list, ptr)->size;
        dealloc(list, ptr);
    }
    n_allocb += size;
    n_realloc++;
    alloc(list, ptr2, size);
    return ptr2;

}
```

```
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0x7f97eff46168
[0003]         main:11 : free( 0xe11060 )
[0004]         main:19 : free( 0xe11060 )
[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.
```

보너스 파트에선 메모리 해제가 잘못 이루어지고 있는지 판단하기 위해 part2의 리스트를 활용했습니다. 해제하고자 하는 주소가 list에 없다면 illegal free로 판단하고 list에 있지만 cnt가 0인 경우에는 double free로 판단합니다. Realloc의 경우 illegal deallocation으로 판단하면 malloc과 똑같이 기능하도록 구현했습니다.

후기: 주로 자료구조, 알고리즘 처럼 high level한 코드를 접하고 작성하다보니 메모리나 어셈블리와 같은 low level을 들여다보면 항상 새롭고 신비한 기분이 듭니다. 이번 과제는 크게 어렵지 않게 memory allocation의 다양한 정보를 볼 수 있어서 재미있었고, backtracing library도 처음 사용하면서 흥미를 느낄 수 있었습니다.