Link Lab

2016-12424 Chaewon Kim

This is a project about tracking memory in the process of dynamic memory allocation. In order to do this, we need to know about memory allocation and at-load-time library interpositioning needed. Most of the project was already implemented, so there was only few part to modify.

Getting Start

We have to start with file `linklab.tgz`. First, we need to unzip this file. We can use below command to unzip `.tgz` format.

```
tar xvf linklab.tgz
```

There are `Makefile`s at every location of directory inside. `Makefile`s have two major roll for this project. First, they give manual and help. Second, they are used as tools for builiding project. Every path for linking is provided, so we don't need to write long complex command to compile project.

Skeleton of Project

There are 6 folders provided, but 4 of them have same structure. They are just for implementing different part of project, so we can devide them into three roles.

1.Utils

`Utils` provide basic utils for project. It helps printing log and managing memory list.

2.Test

This part contains main function for this project. Actually, main function is not very important. There are several test cases, which is main function, just used for calling memory allocation.

3.Library

This is part we need to implement. We need to implement tools for tracking memories.

Tracking Memories

To track memories, we need to use user-define malloc function. To do this we need to understand concept of of at-load-time interpositioning. We can define malloc as we want and use it. But problem is we do such so, we can't access to original malloc function. Below code help us to use original malloc function.

```
static void *(*mallocp)(size_t size) = NULL;

if(!mallocp) {
```

```
    mallocp = dlsym(RTLD_NEXT, "malloc");
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
 }
```

Above code make us use original malloc function, but this function is rellocated during load time, so compiled program it-self doesn't contain any code about original malloc. By using such code we are able to us customed malloc function.

```
void *malloc(size_t size) {
    void* ptr;
    ptr = mallocp(size);

    n_malloc+=size;
    n_allocb++;

    LOG_MALLOC(size, ptr);
    return ptr;
}
```

If we call malloc function, above customed-malloc will called. Inside this function, it call original-malloc function and do same role, but it additionally track how many memories is used. Same can be done on other functions.

Additionally, we need to track memories which have been allocated, but not deallocated yet. To do this, we need to manage every allocation and deallocation. Data structure is already implemented in `Utils`, so we just need to call function provided.

```
void free(void* ptr) {
    item *cur = dealloc(list, ptr);
    n_freeb+=cur->size;
    LOG_FREE(ptr);
    freep(ptr);
    return;
}
```

Above alloc and dealloc is function provided by `Utils`. Just calling function let us to save and erase data in linked list.

Pinpointing Call Locations

To point out exact location of function call, we need to use library `unwind.h`. Using functions in library, we can get process name and offset of command.

```
int get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs) {
    unw_context_t context;
    unw_cursor_t cursor;
    unw_word_t off, ip, sp;
    unw_proc_info_t pip;
    char procname[256];
    int ret;

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

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

    unw_step(&cursor);
    unw_step(&cursor);
    unw_step(&cursor);

    if(unw_get_proc_name(&cursor, procname, 256, &off))
        return -1;

    strcpy(fname, procname);
    fnlen=strlen(procname);
    *ofs=off-5;

    return 0;
}
```

Because function call happened on exactly 3 function above(get_callinfo -> mlog -> malloc -> above), we

Exception Handling

Above codes are implemented for ideal situation. By adding some exception handling, I have handled two unordinary situation. First case is deallocating memory which is already deallocated. Second case is deallocating memory which is never been allocated. It is done easily by checking linked-list.

```
void *realloc(void *ptr, size_t size) {
    item *f = find(list, ptr);
    if(f==NULL) {
        LOG_ILL_FREE();
    } else if(f->cnt==0) {
        LOG_DOUBLE_FREE();
    } else {
        item *cur = dealloc(list, ptr);
        n_freeb+=cur->size;
    }

    ...
}
```

```
void free(void* ptr) {
    LOG_FREE(ptr);

    item *f = find(list, ptr);
    if(f==NULL) {
        LOG_ILL_FREE();
        return;
    }

    if(f->cnt==0) {
        LOG_DOUBLE_FREE();
        return;
    }

    ...
}
```

Result

test1

```
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0x157e060
[0003]         main:10 : malloc( 32 ) = 0x157e4c0
[0004]         main:1d : malloc( 1 ) = 0x157e540
[0005]         main:25 : free( 0x157e540 )
[0006]         main:2d : free( 0x157e4c0 )
[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]   0x157e060          1024       1         main:6
[0016]
[0017] Memory tracer stopped.
```

test2

```
[0001] Memory tracer started.
[0002]         main:9  : malloc( 1024 ) = 0x809060
[0003]         main:11 : free( 0x809060 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
```

```
[0009]
[0010] Memory tracer stopped.
```

test3

```
[0001] Memory tracer started.
[0002]         main:37 : malloc( 53791 ) = 0x1f03060
[0003]         main:6e : calloc( 1 , 46302 ) = 0x1f102e0
[0004]         main:37 : malloc( 12549 ) = 0x1f1b820
[0005]         main:6e : calloc( 1 , 14515 ) = 0x1f1e980
[0006]         main:37 : malloc( 61378 ) = 0x1f22290
[0007]         main:6e : calloc( 1 , 30681 ) = 0x1f312b0
[0008]         main:37 : malloc( 31021 ) = 0x1f38af0
[0009]         main:37 : malloc( 5423 ) = 0x1f40480
[0010]         main:37 : malloc( 32426 ) = 0x1f41a10
[0011]         main:6e : calloc( 1 , 13714 ) = 0x1f49920
[0012]         main:97 : free( 0x1f49920 )
[0013]         main:97 : free( 0x1f41a10 )
[0014]         main:97 : free( 0x1f40480 )
[0015]         main:97 : free( 0x1f38af0 )
[0016]         main:97 : free( 0x1f312b0 )
[0017]         main:97 : free( 0x1f22290 )
[0018]         main:97 : free( 0x1f1e980 )
[0019]         main:97 : free( 0x1f1b820 )
[0020]         main:97 : free( 0x1f102e0 )
[0021]         main:97 : free( 0x1f03060 )
[0022]
[0023] Statistics
[0024]   allocated_total      301800
[0025]   allocated_avg        30180
[0026]   freed_total          301800
[0027]
[0028] Memory tracer stopped.
```

test5

```
[0001] Memory tracer started.
[0002]         main:9  : malloc( 10 ) = 0x1b5f060
[0003]         main:16 : realloc( 0x1b5f060 , 100 ) = 0x1b5f0d0
[0004]         main:23 : realloc( 0x1b5f0d0 , 1000 ) = 0x1b5f190
[0005]         main:30 : realloc( 0x1b5f190 , 10000 ) = 0x1b5f5d0
[0006]         main:3d : realloc( 0x1b5f5d0 , 100000 ) = 0x1b5f5d0
[0007]         main:45 : free( 0x1b5f5d0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
```

```
[0013]
[0014] Memory tracer stopped.
```

Conclusion

From this project, I got a chance to try at-load-time interpositioning. Also, I have learned how to use library `unwind`.

It was not very hard assignment. But it was bit tricky to debug and fix the error. Project was divded into 3+1 steps, so if single mistake make us to fix same mistake 4 times.