lab1_2019-11730

Part 1

실행 결과

```
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x55adf55242d0
[0003]           malloc( 32 ) = 0x55adf55246e0
[0004]           malloc( 1 ) = 0x55adf5524710
[0005]           free( 0x55adf5524710 )
[0006]           free( 0x55adf55246e0 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
```

test1

```
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x558c218412d0
[0003]           free( 0x558c218412d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          0
[0009]
[0010] Memory tracer stopped.
```

test2

```
[0001] Memory tracer started.
[0002]           calloc( 1 , 10244 ) = 0x55e9965ed2d0
[0003]           calloc( 1 , 10652 ) = 0x55e9965efae0
[0004]           calloc( 1 , 38832 ) = 0x55e9965f2490
[0005]           calloc( 1 , 43012 ) = 0x55e9965fbc50
[0006]           calloc( 1 , 10654 ) = 0x55e996606460
[0007]           malloc( 35782 ) = 0x55e996608e10
[0008]           calloc( 1 , 24006 ) = 0x55e9966119e0
[0009]           calloc( 1 , 35514 ) = 0x55e9966177b0
[0010]           calloc( 1 , 13892 ) = 0x55e996620280
[0011]           malloc( 65172 ) = 0x55e9966238d0
[0012]           free( 0x55e9966238d0 )
[0013]           free( 0x55e996620280 )
```

```
[0014]           free( 0x55e9966177b0 )
[0015]           free( 0x55e9966119e0 )
[0016]           free( 0x55e996608e10 )
[0017]           free( 0x55e996606460 )
[0018]           free( 0x55e9965fbc50 )
[0019]           free( 0x55e9965f2490 )
[0020]           free( 0x55e9965efae0 )
[0021]           free( 0x55e9965ed2d0 )
[0022]
[0023] Statistics
[0024]   allocated_total      287760
[0025]   allocated_avg        28776
[0026]   freed_total          0
[0027]
[0028] Memory tracer stopped.
```

test3

구현 방법

Load/Run-time Interpositioning을 이용해 기존 함수에

- name, arguments, return value를 print하는 코드
- allocation size와 count를 집계하는 코드
  위 두 가지 코드를 삽입합니다.

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

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

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

        ptr = mallocp(size);

        LOG_MALLOC(size, ptr);

        n_malloc++;
        n_allocb += size;

        return ptr;
}
```

위 코드는 `malloc` 함수를 Load/Run-time Interpositioning한 코드입니다.

1. `dlsym(RTLD_NEXT, "malloc")`로 `mallocp`가 라이브러리 함수를 가리키도록 합니다.

2. `mallocp(size)`로 실제로 malloc의 역할을 수행하도록 합니다.
3. `LOG_MALLOC(size, ptr)`로 name, arguments, return value를 print합니다.
4. `n_malloc++; n_allocb += size;`로 allocation size와 count를 집계합니다.

동일한 방법으로 calloc, realloc, free도 Interpositioning 합니다.

```
long n_alloc = n_malloc + n_calloc + n_realloc;
long avg_allocb = n_alloc > 0 ? n_allocb / n_alloc : 0;

LOG_STATISTICS(n_allocb, avg_allocb, n_freeb);
```

`void fini(void)` 함수에 위 코드를 추가해서, 각 함수에서 집계한 통계를 출력하도록 합니다.
`n_alloc`은 malloc, calloc, realloc이 호출된 횟수, `n_allocb`는 총 할당된 바이트 수를 의미하므로 `n_allocb / n_alloc` 으로 평균 바이트 수를 구할 수 있습니다.

Part2

실행 결과

```
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x559ab7c7c2d0
[0003]           malloc( 32 ) = 0x559ab7c7c710
[0004]           malloc( 1 ) = 0x559ab7c7c770
[0005]           free( 0x559ab7c7c770 )
[0006]           free( 0x559ab7c7c710 )
[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]   0x559ab7c7c2d0     1024       1
[0016]
[0017] Memory tracer stopped.
```

test1

```
[0001] Memory tracer started.
[0002]           malloc( 1024 ) = 0x563a23ea52d0
[0003]           free( 0x563a23ea52d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
```

test2

```
[0001] Memory tracer started.
[0002]           calloc( 1 , 43023 ) = 0x561154caf2d0
[0003]           malloc( 3215 ) = 0x561154cb9b20
[0004]           calloc( 1 , 881 ) = 0x561154cba7f0
[0005]           calloc( 1 , 21908 ) = 0x561154cbaba0
[0006]           calloc( 1 , 48419 ) = 0x561154cc0170
[0007]           malloc( 13033 ) = 0x561154ccbed0
[0008]           calloc( 1 , 11762 ) = 0x561154ccf200
[0009]           malloc( 64169 ) = 0x561154cd2030
[0010]           malloc( 41521 ) = 0x561154ce1b20
[0011]           calloc( 1 , 62541 ) = 0x561154cebd90
[0012]           free( 0x561154cebd90 )
[0013]           free( 0x561154ce1b20 )
[0014]           free( 0x561154cd2030 )
[0015]           free( 0x561154ccf200 )
[0016]           free( 0x561154ccbed0 )
[0017]           free( 0x561154cc0170 )
[0018]           free( 0x561154cbaba0 )
[0019]           free( 0x561154cba7f0 )
[0020]           free( 0x561154cb9b20 )
[0021]           free( 0x561154caf2d0 )
[0022]
[0023] Statistics
[0024]   allocated_total      310472
[0025]   allocated_avg        31047
[0026]   freed_total          310472
[0027]
[0028] Memory tracer stopped.
```

test3

구현 방법

Part1의 구현에 추가로, Load/Run-time Interpositioning을 이용해

- memlist에 정의된 alloc, dealloc을 호출하는 코드
- free된 byte를 집계하는 코드
  위 두 가지 코드를 삽입합니다.

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

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

```
        ptr = mallocp(size);

        LOG_MALLOC(size, ptr);
        alloc(list, ptr, size);

        n_malloc++;
        n_allocb += size;

        return ptr;
}
```

위 코드는 `malloc` 함수를 Load/Run-time Interpositioning한 코드입니다.

1. `alloc(list, ptr, size)`로 alloc 정보를 저장합니다.

동일한 방법으로 `calloc`도 Interpositioning 합니다.

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

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

        LOG_FREE(ptr);
        item *deallocated_item = dealloc(list, ptr);

        n_freeb += deallocated_item->size;

        freep(ptr);
}
```

위 코드는 `free` 함수를 Load/Run-time Interpositioning한 코드입니다.

1. `dealloc(list, ptr)`로 dealloc 정보를 저장합니다.
2. `n_freeb += deallocated_item->size`로 free된 byte를 집계합니다.

realloc에서는 기존 ptr은 `free`처럼 `dealloc` 후 free byte로 집계하고, 새로운 ptr은 `malloc`처럼 `alloc` 합니다.

Part 3

실행 결과

```
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x557536eee2d0
[0003]          free( 0x557536eee2d0 )
[0004]          free( 0x557536eee2d0 )
[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

```
[0001] Memory tracer started.
[0002]           malloc( 10 ) = 0x56469d0362d0
[0003]           realloc( 0x56469d0362d0 , 100 ) = 0x56469d036320
[0004]           realloc( 0x56469d036320 , 1000 ) = 0x56469d0363c0
[0005]           realloc( 0x56469d0363c0 , 10000 ) = 0x56469d0367e0
[0006]           realloc( 0x56469d0367e0 , 100000 ) = 0x56469d038f30
[0007]           free( 0x56469d038f30 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

test5

구현 방법

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

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

        LOG_FREE(ptr);
        item *curr = list;
        while ((curr != NULL) && (curr->ptr != ptr)) {
                curr = curr->next;
        }

        if (curr == NULL) {
                LOG_ILL_FREE();
                return;
        }
        if (curr->cnt == 0) {
```

```
                LOG_DOUBLE_FREE();
                return;
        }

        item *deallocated_item = dealloc(list, ptr);

        n_freeb += deallocated_item->size;

        freep(ptr);
}
```

위 코드는 `free` 함수를 Load/Run-time Interpositioning한 코드입니다.

1. list에서 free할 ptr로 메모리 블록을 찾습니다.
2. `curr == NULL`, 즉 list에 메모리 블록이 없다면 할당된 적 없는 메모리를 free하는 것이므로 Illegal free 입니다.
3. `curr->cnt == 0`이라면 이미 해제되었으므로 Double free 입니다.

어려웠던 점

- C 프로그래밍이 익숙하지 않아 다른 사람이 작성한 코드를 읽는 것이 어려웠습니다.

새롭게 배운 점

- 메모리 할당(해제) 함수를 Interpositioning하고, 메모리 할당과 해제에 대한 정보를 추적하는 방법을 배웠습니다. 이를 통해 메모리 누수를 탐지하고 디버깅할 수 있게 되었습니다.
- `dlfcn.h` 라이브러리를 사용하여 동적 라이브러리를 로드하고 함수 포인터를 얻는 방법도 배울 수 있었습니다.