LAB_01 LINKLAB

2014-19498
독어교육과 정은주

part 1 - Load, Run Time Interpositioning

우선, load, run-time interpositioning을 통해서 실제 malloc을 memtrace.c에 intercept한다. 만들어둔 static function pointer mallocp를 이용해 현재 있는 라이브러리가 아니라 다음에 오는(RTLD_NEXT) 라이브러리 즉, standard library의 함수들을 부르고, 이를 이용해서 n_malloc은 1을, n_allocb은 size의 크기를 각각 더해주고, LOG_MALLOC size, ptr)을 이용해서 malloc이 불릴 때의 상황을 tracing 할 수 있다.

malloc과 마찬가지로, calloc, realloc, free의 경우도, 각각 n_calloc, n_realloc, n_free의 값을 더해주고, 그리고 n_allocb의 경우도 알맞게 더해간다. 더불어 malloc 함수와 마찬가지로 각각 LOG_CALLOC(nmemb, size, ptr), LOG_REALLOC(ptr, size, new_ptr), LOG_FREE(ptr)을 통해서 각 함수를 tracing할 수 있도록 한다. 그리고 마지막 fini(void)가 불릴 때, LOG_STATISTICS(n_allocb, avg, n_freeb)를 통해서 각각 함수가 불릴 때의 memory allocation size의 통계를 출력한다. 이때 free는 무시한다.

difficulties:

처음에 코드를 읽고 해석하는데 시간이 오래 걸렸다. 나누어져 있는 코드들을 조합하고, 논리 흐름에 맞게 처음부터 쭉 훑어서 빠진 부분들을 생각해내려 노력했다. 이미 존재하는 LOG 매크로 부분을 끌어다가 그대로 사용할 수 있었고, ppt에 intercept하는 부분이 있어서 몇 번의 시도 끝에 part1을 완성할 수 있었다.

```
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);
    n_malloc += 1;
    n_allocb += size;
    LOG_MALLOC(size, ptr);
    return ptr;
}
```

```
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);
   freep(ptr);
}
```

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

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

    new_ptr = reallocp(ptr, size);
    n_realloc += 1;
    n_allocb += size;
    LOG_REALLOC(ptr, size, new_ptr);
    return new_ptr;
}
```

```c
void *calloc(size_t nmemb, size_t size)
{
    char* error;
    void *ptr;

    if(!callocp) {
       callocp = dlsym(RTLD_NEXT, "calloc");
       if((error = dlerror()) != NULL) {
          fputs(error, stderr);
          exit(1);
       }
    }
    ptr = callocp(nmemb, size);
    n_calloc += 1;
    n_allocb += nmemb*size;
    LOG_CALLOC(nmemb, size, ptr);
    return ptr;
}
```

LOG_STATISCS(n_allocb, avg, n_freeb);

```c
__attribute__((destructor))
void fini(void)
{
  // ...
  static unsigned long avg = 0;
  if((n_malloc + n_calloc + n_realloc) != 0)
    avg = n_allocb/(unsigned long)(n_malloc + n_calloc + n_realloc);

  LOG_STATISTICS(n_allocb, avg, n_freeb);
  LOG_STOP();

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

// ...
```

part2 - Tracing unfreed memory

free되지 않은 메모리 부분을 출력하기 위해서는 이미 구현되어 있는 mem_list를 이용하여 linked list에 메모리가 할당될 때마다 메모리를 담고, free할 때는 ref_cnt를 0으로 만들어주어 free되었다는 것을 표시해준다. 그리고 ref_cnt가 0보다 큰 경우는 free가 적절히 되지 않았다는 것을 의미한다. 따라서 LOG_NONFREED_START() 와 LOG_BLOCK(ptr, size, cnt, fname, ofs)를 통해서 해당 부분을 출력했다.

difficulties:

먼저 mem_list의 alloc, dealloc 등의 함수를 사용하기 위해서는 linked list의 인자들은 무엇이 있고, 또 각 함수가 어떻게 동작하는지 파악해야했다. 처음에 freed total을 구하기 위해서 어떻게 코드를 짜야하는지에 대해서 많이 고민했다. 그리고 alloc 함수와 dealloc 함수 이외에 find(list, ptr)을 이용하여 size를 더해주면 된다는 것을 발견했다. 여기까지는 순조로웠으나, realloc 부분에서 free part를 구현하는데 꽤나 많은 시간이 소요되었다.

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

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

    new_ptr = reallocp(ptr, size);
    n_realloc += 1;
    n_allocb += size;
    LOG_REALLOC(ptr, size, new_ptr);
    n_freeb += find(list, ptr)->size;
    dealloc(list, ptr);
    alloc(list, new_ptr, size);
    return new_ptr;
}
```

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

   if(!ptr)
     return;

   if(!freep) {
     freep = dlsym(RTLD_NEXT, "free");
     if ((error = dlerror()) != NULL) {
       fputs(error, stderr);
       exit(1);
     }
   }
   LOG_FREE(ptr);
   n_freeb += find(list, ptr)->size;
   dealloc(list, ptr);
   freep(ptr);

}
```

list를 훑으면서 list의 ref_cnt가 0이 아닌 경우 즉, allocate된 memory가 해제되어 있지 않은 요소들의 경우는 LOG_NONFREED_START()를 한 번 출력한 후에 각 요소들을 차례대로 각각의 block 주소, size, ref cnt, caller를 출력하도록 한다.

```c
__attribute__((destructor))
void fini(void)
{
  // ...
   static unsigned long avg = 0;
  if((n_malloc + n_calloc + n_realloc) != 0)
    avg = n_allocb /(unsigned long)(n_malloc+n_calloc+n_realloc);

  LOG_STATISTICS(n_allocb, avg, n_freeb);
  int log_check = 0;

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

  LOG_STOP();

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

part3 Pinpointing call locations

get_callinfo(char *fname, size_t fnlen, unsigned long long *ofs) 함수를 libunwind 라이브러리를 이용하여 각 stack에 저장되어 있는 함수들의 주소, offset 등을 이용하여 ???로 표시되어 있는 부분을 알맞은 caller(여기서는 모두 main이라 가정) 그리고 각각의 offset을 출력하도록 함수를 고쳤다.

함수 코드 flow는 다음과 같다. 우선 unw_getcontext(&context)를 통해서 현재의 machine state를 얻는다. 그리고 unw_init_local(&cursor, &context)를 통해서 local unwinding을 위한 cursor를 초기화한다. 그 후에 while문을 통해서 unw_step(&cursor)를 통해 다음 fname으로 이동한다. 본 과제에서는 fname이 main인 경우라고 설정해두었으므로, unw_get_proc_name(&cursor, procname, 256, &off)를 통해서 받은 procname이 main이 되는 순간 while문을 빠져나온다. 그러면 off에 담기는 값은 다음 instruction 수행과 관련된 시작점의 값이므로, callq 인스트럭션의 주소가 항상 고정된 5 바이트라 가정했으므로 이 값을 빼주면 해당 caller 즉, main의 offset이 설정된다.

difficulties:

libunwind의 동작 방법을 파악하는 것 자체가 어려웠다. 구글링을 통해 각각 function들이 어떻게 작동하는지를 먼저 파악했고, 각각의 unw_proc_info_t pip, 혹은 unw_word_t off 등등이 어떻게 사용되어야 하는지를 고민했다. 우선 github에 제시되어진 struct를 보고 각각의 기능을 파악한 후에 코드를 짤 수 있었다.

```c
#include <stdlib.h>
#include <stdio.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;
  unw_proc_info_t pip;
  char procname[256];
  int ret;

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

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

```
while(unw_step(&cursor) > 0) {
  if(unw_get_proc_info(&cursor, &pip))
     return -1;

  ret = unw_get_proc_name(&cursor, procname, 256, &off);
  if(ret && ret != -UNW_ENOMEM) {
     procname[0] = '?';
     procname[1] = 0;
  }

  if(strcmp(procname,  "main") == 0)
    break;
}

if(strcmp(procname, "main") != 0)
  return -1;

*ofs = (unsigned long int) (off -5);
strcpy(fname, procname);
return 0;
```

BONUS part

realloc의 경우에는 이미 malloc 혹은 calloc되어 있는 부분에 다시 size를 재조정할 때에 사용되는 함수이다. 즉 free 후 alloc이기 때문에 만약 free할 부분이 없다면, 에러가 발생하게 된다. 따라서, 본 과제에서는 이미 메모리 해제를 한 부분을 다시 해제하는 경우인 double free 혹은 메모리가 할당되지 않은 부분을 해제하려고 하는 경우인 illegal free를 적절히 알려주고, 에러 핸들링을 해준다.

difficulities:

free 함수의 경우는 문제에서 요구하는바 그대로 메모리가 설정되어 있지 않은, 즉 리스트에 존재하지 않은 경우는 illegal free의 log를 출력하였고, 만약 mem list에 존재하지만 ref cnt가 0인 경우는 이미 메모리가 해제된 경우이므로 double free를 출력한다. 그리고 이때, find(list, ptr)을 이용하여 만약 NULL을 return 한다면, 메모리가 할당된 적이 없고, 만약 NULL이 아니지만, cnt 값이 0이라면, double free가 된다. 그리고 다른 경우는 정상적으로 메모리가 할당되었고, 다시 해제할 수 있는 경우이다.

realloc의 경우는 고려해야 하는 경우가 많아 처음에 여러 번 코드를 수정했다. 일단 code flow를 보면, interpositioning을 한 reallocp를 이용해 새로운 pointer를 new_ptr에 받는다. 그리고 새로 받은 pointer와 기존의 pointer를 출력한다. LOG를 할 때, LOG_REALLOC(ptr, size, new_ptr)을 하고 그 후에 free가 ILL_FREE 혹은 DOUHBLE_FREE인 경우를 출력한다. 따라서 find(list, ptr)를 이용해서 mem list를 이용해 double free 혹은 illegal free인지를 확인하려면, LOG를 한 후, realloc을 위한 mem list의 dealloc과 alloc을 진행한다.

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

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

   if(ptr == NULL || find(list, ptr) == NULL) {
     LOG_ILL_FREE();
   } else if(find(list,ptr)->cnt <=0) {
     LOG_DOUBLE_FREE();
   } else {
     n_freeb += find(list, ptr)->size;
     dealloc(list, ptr);
     freep(ptr);
```

```c
void *realloc(void *ptr, size_t size) {
     char *error;
     void *new_ptr = ptr;
     void *ori_ptr = ptr;

     if(!reallocp) {
       reallocp = dlsym(RTLD_NEXT, "realloc");
       if((error = dlerror()) != NULL) {
         fputs(error, stderr);
         exit(1);
       }
     }
     n_realloc += 1;
     n_allocb += size;
     int pt_ch = 0;

     if(find(list, ptr) == NULL){
          pt_ch = 1;
          ptr = NULL;
     }
     else if(find(list, ptr)->cnt == 0){
          pt_ch = 2;
          ptr = NULL;
     }
     else{
          n_freeb += find(list, ptr)->size;
          dealloc(list, ptr);
     }
     new_ptr = reallocp(ptr, size);
     if(size != 0)
          alloc(list, new_ptr, size);
     LOG_REALLOC(ori_ptr, size, new_ptr);

     if(pt_ch == 1)
       LOG_ILL_FREE();
     else if(pt_ch == 2)
       LOG_DOUBLE_FREE();

     return new_ptr;
```

RESULTS

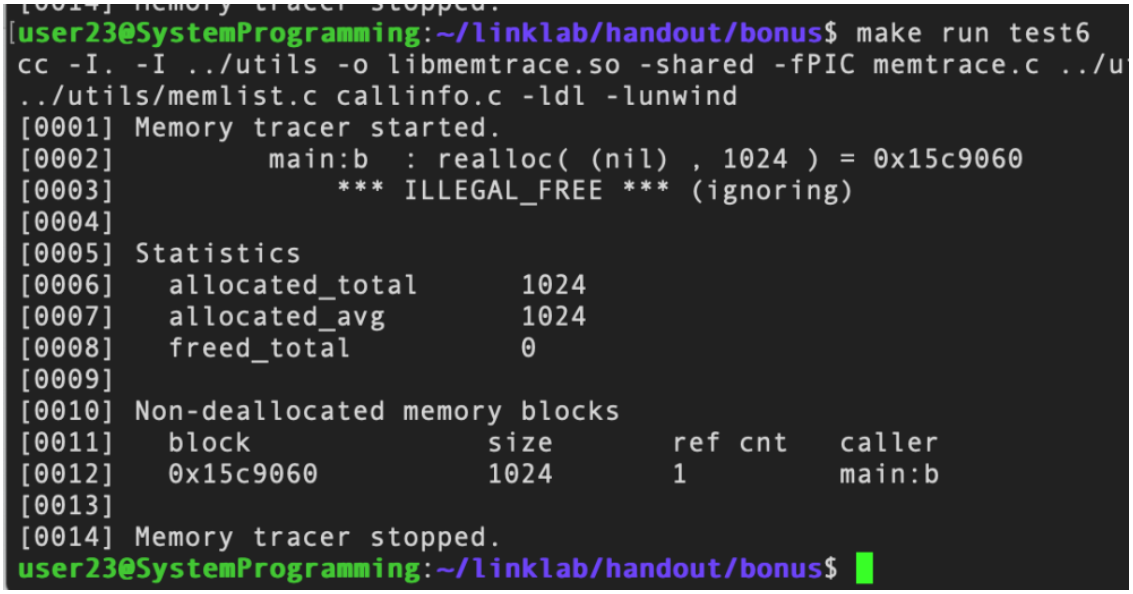
```
user23@SystemProgramming:~/linklab/handout/bonus$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0xe00060
[0003]         main:10 : malloc( 32 ) = 0xe004c0
[0004]         main:1d : malloc( 1 ) = 0xe00540
[0005]         main:25 : free( 0xe00540 )
[0006]         main:2d : free( 0xe004c0 )
[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]   0xe00060           1024       1          main:6
[0016]
[0017] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$
```

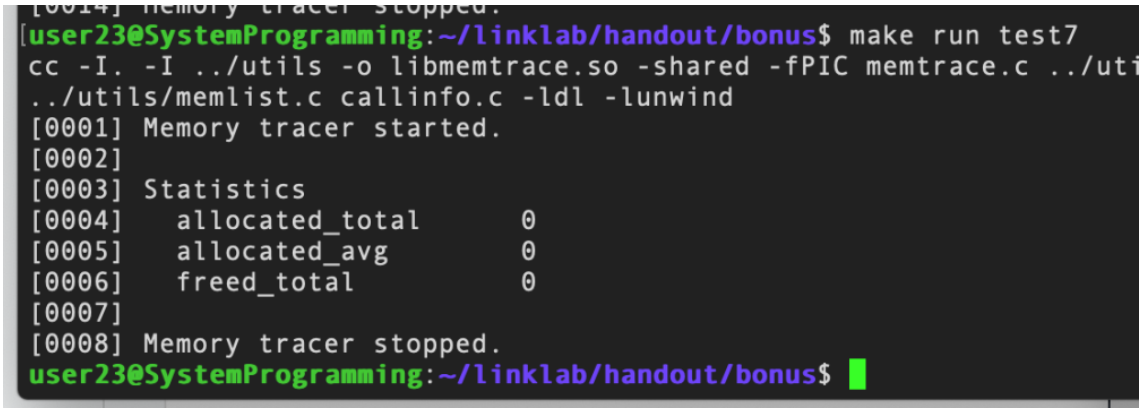
```
user23@SystemProgramming:~/linklab/handout/bonus$ make run test2
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../uti
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:9  : malloc( 1024 ) = 0x1c14060
[0003]         main:11 : free( 0x1c14060 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$
```

```
user23@SystemProgramming:~/linklab/handout/bonus$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:6  : malloc( 1024 ) = 0xf54060
[0003]         main:11 : free( 0xf54060 )
[0004]         main:19 : free( 0xf54060 )
[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.
user23@SystemProgramming:~/linklab/handout/bonus$
```

```
user23@SystemProgramming:~/linklab/handout/bonus$ make run test3
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:6e : calloc( 1 , 4198 ) = 0x1a2a060
[0003]         main:37 : malloc( 11099 ) = 0x1a2b120
[0004]         main:6e : calloc( 1 , 3866 ) = 0x1a2dce0
[0005]         main:6e : calloc( 1 , 293 ) = 0x1a2ec60
[0006]         main:37 : malloc( 3468 ) = 0x1a2ede0
[0007]         main:6e : calloc( 1 , 33284 ) = 0x1a2fbd0
[0008]         main:6e : calloc( 1 , 17615 ) = 0x1a37e30
[0009]         main:6e : calloc( 1 , 53823 ) = 0x1a3c360
[0010]         main:37 : malloc( 42327 ) = 0x1a49600
[0011]         main:6e : calloc( 1 , 32407 ) = 0x1a53bb0
[0012]         main:97 : free( 0x1a53bb0 )
[0013]         main:97 : free( 0x1a49600 )
[0014]         main:97 : free( 0x1a3c360 )
[0015]         main:97 : free( 0x1a37e30 )
[0016]         main:97 : free( 0x1a2fbd0 )
[0017]         main:97 : free( 0x1a2ede0 )
[0018]         main:97 : free( 0x1a2ec60 )
[0019]         main:97 : free( 0x1a2dce0 )
[0020]         main:97 : free( 0x1a2b120 )
[0021]         main:97 : free( 0x1a2a060 )
[0022]
[0023] Statistics
[0024]   allocated_total      202380
[0025]   allocated_avg        20238
[0026]   freed_total          202380
[0027]
[0028] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$
```

```
user23@SystemProgramming:~/linklab/handout/bonus$ make run test5
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../u
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]         main:9  : malloc( 10 ) = 0xdc3060
[0003]         main:16 : realloc( 0xdc3060 , 100 ) = 0xdc30d0
[0004]         main:23 : realloc( 0xdc30d0 , 1000 ) = 0xdc3190
[0005]         main:30 : realloc( 0xdc3190 , 10000 ) = 0xdc35d0
[0006]         main:3d : realloc( 0xdc35d0 , 100000 ) = 0xdc35d0
[0007]         main:45 : free( 0xdc35d0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$
```


[user23@SystemProgramming:~/linklab/handout/bonus$ make run test6
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../u
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002] main:b : realloc((nil) , 1024) = 0x15c9060
[0003] *** ILLEGAL_FREE *** (ignoring)
[0004]
[0005] Statistics
[0006] allocated_total 1024
[0007] allocated_avg 1024
[0008] freed_total 0
[0009]
[0010] Non-deallocated memory blocks
[0011] block size ref cnt caller
[0012] 0x15c9060 1024 1 main:b
[0013]
[0014] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$


[user23@SystemProgramming:~/linklab/handout/bonus$ make run test7
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../uti
../utils/memlist.c callinfo.c -ldl -lunwind
[0001] Memory tracer started.
[0002]
[0003] Statistics
[0004] allocated_total 0
[0005] allocated_avg 0
[0006] freed_total 0
[0007]
[0008] Memory tracer stopped.
user23@SystemProgramming:~/linklab/handout/bonus$