Link Lab

수리과학부 2019-16022 박채연

1. 실행결과

(1) part 1

part 1의 test 1, test 2, test 3 수행결과는 다음과 같다. freed_total은 실제 할당해제된 크기와 관계없이 0으로 출력되고 allocated_total과 allocated_avg는 정상적으로 잘 출력되는 모습을 확인할 수 있다.

```
stu9@sp05:~/lab1/handout/part1$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x5629512192d0
[0003]          malloc( 32 ) = 0x5629512196e0
[0004]          malloc( 1 ) = 0x562951219710
[0005]          free( 0x562951219710 )
[0006]          free( 0x5629512196e0 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
```

```
stu9@sp05:~/lab1/handout/part1$ make run test2
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x55c29e4df2d0
[0003]          free( 0x55c29e4df2d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          0
[0009]
[0010] Memory tracer stopped.
```

```
stu9@sp05:~/lab1/handout/part1$ make run test3
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          calloc( 1 , 55024 ) = 0x55a024c582d0
[0003]          malloc( 41310 ) = 0x55a024c659d0
[0004]          calloc( 1 , 50458 ) = 0x55a024c6fb40
[0005]          calloc( 1 , 31023 ) = 0x55a024c7c070
[0006]          calloc( 1 , 12789 ) = 0x55a024c839b0
[0007]          malloc( 22919 ) = 0x55a024c86bb0
[0008]          calloc( 1 , 15610 ) = 0x55a024c8c540
[0009]          malloc( 63834 ) = 0x55a024c90250
[0010]          malloc( 43593 ) = 0x55a024c9fbc0
[0011]          malloc( 32471 ) = 0x55a024caa620
[0012]          free( 0x55a024caa620 )
[0013]          free( 0x55a024c9fbc0 )
[0014]          free( 0x55a024c90250 )
[0015]          free( 0x55a024c8c540 )
[0016]          free( 0x55a024c86bb0 )
[0017]          free( 0x55a024c839b0 )
[0018]          free( 0x55a024c7c070 )
[0019]          free( 0x55a024c6fb40 )
[0020]          free( 0x55a024c659d0 )
[0021]          free( 0x55a024c582d0 )
[0022]
[0023] Statistics
[0024]   allocated_total      369031
[0025]   allocated_avg        36903
[0026]   freed_total          0
[0027]
[0028] Memory tracer stopped.
```

(2) part 2

part 2의 test 1, test 2, test 3 수행결과는 다음과 같다. allocated_total과 allocated_avg 뿐만 아니라 freed_total 역시 실제 값과 동일하게 잘 출력되는 모습을 확인할 수 있다. 또한, 할당해제되지 않은 메모리 블럭이 한 개 이상일 경우 할당해제되지 않은 메모리 블럭에 대한 정보 역시 정상적으로 출력되는 모습을 확인할 수 있다.

```
stu9@sp05:~/lab1/handout/part2$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x560171fc92d0
[0003]          malloc( 32 ) = 0x560171fc9710
[0004]          malloc( 1 ) = 0x560171fc9770
[0005]          free( 0x560171fc9770 )
[0006]          free( 0x560171fc9710 )
[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]   0x560171fc92d0     1024       1
[0016]
[0017] Memory tracer stopped.
```

```
stu9@sp05:~/lab1/handout/part2$ make run test2
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x5643823142d0
[0003]          free( 0x5643823142d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
```

```
stu9@sp05:~/lab1/handout/part2$ make run test3
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 63016 ) = 0x5581117742d0
[0003]          malloc( 63316 ) = 0x558111783930
[0004]          calloc( 1 , 56807 ) = 0x5581117930c0
[0005]          calloc( 1 , 64165 ) = 0x5581117a0ee0
[0006]          calloc( 1 , 46616 ) = 0x5581117b09c0
[0007]          malloc( 9751 ) = 0x5581117bc010
[0008]          calloc( 1 , 1405 ) = 0x5581117be660
[0009]          malloc( 57771 ) = 0x5581117bec20
[0010]          calloc( 1 , 31700 ) = 0x5581117cce10
[0011]          malloc( 30484 ) = 0x5581117d4a20
[0012]          free( 0x5581117d4a20 )
[0013]          free( 0x5581117cce10 )
[0014]          free( 0x5581117bec20 )
[0015]          free( 0x5581117be660 )
[0016]          free( 0x5581117bc010 )
[0017]          free( 0x5581117b09c0 )
[0018]          free( 0x5581117a0ee0 )
[0019]          free( 0x5581117930c0 )
[0020]          free( 0x558111783930 )
[0021]          free( 0x5581117742d0 )
[0022]
[0023] Statistics
[0024]   allocated_total      425031
[0025]   allocated_avg        42503
[0026]   freed_total          425031
[0027]
[0028] Memory tracer stopped.
```

(3) part 3

part 3의 test 4, test 5 수행결과는 다음과 같다. 특히 test 4에서 이미 할당 해제된 메모리 공간을 한번 더 할당해제 시도할 경우 Double Free 관련 문구를 정상적으로 출력하고, 할당되지 않은 메모리 공간을 할당해제 시도할 경우 Ill Free 관련 문구를 정상적으로 출력하는 것을 확인할 수 있다. 또한, 두 경우 모두 error를 invoke하지 않고 로그만을 정상적으로 출력하는 것을 확인할 수 있었다.

```
stu9@sp05:~/lab1/handout/part3$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x55a2830802d0
[0003]          free( 0x55a2830802d0 )
[0004]          free( 0x55a2830802d0 )
[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.
```

```
stu9@sp05:~/lab1/handout/part3$ make run test5
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlist.c ../utils/memlog.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 10 ) = 0x55f834b802d0
[0003]          realloc( 0x55f834b802d0 , 100 ) = 0x55f834b80320
[0004]          realloc( 0x55f834b80320 , 1000 ) = 0x55f834b803c0
[0005]          realloc( 0x55f834b803c0 , 10000 ) = 0x55f834b807e0
[0006]          realloc( 0x55f834b807e0 , 100000 ) = 0x55f834b82f30
[0007]          free( 0x55f834b82f30 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

2. 어떻게 구현했는지(파트별로)

(1) part 1

본 과제의 구현은 크게 생성자, 소멸자, 그외 함수들로 나눌 수 있다.

```
37  __attribute__((constructor))
38  void init(void)
39  {
40    char *error;
41
42    LOG_START();
43
44    // initialize a new list to keep track of all memory (de-)allocations
45    // (not needed for part 1)
46    list = new_list();
47
48    // ...
49    mallocp=dlsym(RTLD_NEXT, "malloc");
50    if ((error=dlerror())!=NULL)
51      exit(1);
```

```
53    freep=dlsym(RTLD_NEXT, "free");
54    if ((error=dlerror())!=NULL)
55      exit(1);
56
57    callocp=dlsym(RTLD_NEXT, "calloc");
58    if ((error=dlerror())!=NULL)
59      exit(1);
60
61    reallocp=dlsym(RTLD_NEXT, "realloc");
62    if ((error=dlerror())!=NULL)
63      exit(1);
64  }
```

우선 생성자를 먼저 살펴볼 것이다. 생성자 내부에서 LOG_START()가 호출되었는데, 이는 "Memory tracer started."이라는 문자열을 출력한다. 다음으로 list=new_list()를 통해 linked list를 생성한다. 이는 part 1의 구현과는 관련 없는 부분이지만 part 2와 part 3에서 할당해제되지 않은 메모리 블럭 관련 정보을 출력하거나 메모리가 double free 혹은 ill free 되었는지 여부 등을 체크할 때 필요하다.

mallocp, freep, callocp, reallocp는 stdlib의 malloc(), free(), calloc(), realloc()에 대한 함수 포인터 역할을 한다. mallocp=dlsym(RTLD_NEXT, "malloc")를 통해 mallocp가 표준 C 라이브러리의 malloc 함수를 참조하도록 하였다. 이때 만약 에러가 발생하면 exit(1)을 하게 된다. 마찬가지로 freep, callocp, reallocp가 표준 C 라이브러리의 free, calloc, realloc 함수를 참조하도록 하고 만약 에러가 발생하면 exit(1)을 호출하도록 하였다.

```
84  void *malloc(size_t size){
85    void *tp=mallocp(size);
86    LOG_MALLOC(size, tp);
87    n_malloc++;
88    n_allocb+=size;
89
90    return tp;
91  }
92
93  void *calloc(size_t nmemb, size_t size){
94    void *tp=callocp(nmemb, size);
95    LOG_CALLOC(nmemb, size, tp);
96    n_calloc++;
97    n_allocb+=nmemb*size;
98
99    return tp;
100 }
101
102 void *realloc(void *ptr, size_t size){
103   void *tp=reallocp(ptr, size);
104   LOG_REALLOC(ptr, size, tp);
105   n_realloc++;
106   n_allocb+=size;
107
108   return tp;
109 }
110
111 void free(void *ptr){
112   LOG_FREE(ptr);
113   freep(ptr);
114 }
```

다음으로 malloc(), calloc(), realloc(), free()함수를 살펴볼 것이다. 이들 함수는 표준 C 라이브러리에 포함된 실제 함수들의 wrapper 역할을 하는 함수들이다. 생성자 호출로 인해 mallocp, callocp, reallocp, freep가 실제 malloc(), calloc(), realloc(), free() 함수들에 대한 포인터가 되었으므로 이들을 wrapper 함수 내부에서 호출하여 사용할 것이다.

malloc()함수 내부에서는 void *tp=mallocp(size)의 호출로 size 크기의 메모리를 할당받고 이를 포인터인 tp가 가리키게 된다. LOG_MALLOC(size, tp)의 호출을 통해 할당 받은 메모리 크기와 메모리 시작 주소를 출력한다. 또한, malloc()함수가 호출된 횟수를 의미하는 n_malloc의 값을 1만큼 증가시키고 할당받은 메모리의 총 바이트 수를 의미하는 n_allocb의 값을 size만큼 증가시키며 할당받은 메모리 시작 주소를 return한다.

calloc()함수 내부에서는 void *tp=callocp(nmemb, size)의 호출로 size 크기의 메모리를 nmemb개 할당받고 이를 포인터인 tp가 가리키게 된다. LOG_CALLOC(nmemb, size, tp)의 호출을 통해 nmemb와 size을 출력하고, 메모리 시작 주소인 tp도 출력한다. 또한, calloc()함수가 호출된 횟수를 의미하는 n_calloc의 값을 1만큼 증가시키고 할당받은 메모리의 총 바이트 수를 의미하는 n_allocb의 값을 nmemb*size만큼 증가시키며 할당 받은 메모리 시작 주소를 return한다.

realloc()함수 내부에서는 void *tp=reallocp(ptr, size)의 호출로 ptr이 가리키는, 기존에 동적으로 할당했던 메모리 공간을 size 크기로 재할당하고 포인터인 tp가 해당 메모리 공간을 가리키게 된다. LOG_REALLOC(ptr, size, tp)를 호출하여 기존 메모리 공간의 시작 주소, 재할당한 메모리 공간의 크기, 재할당한 메모리의 시작주소를 출력한다. 또한, realloc()함수가 호출된 횟수를 의미하는 n_realloc의 값을 1만큼 증가시키고 할당받은 메모리의 총 바이트 수를 의미하는 n_allocb의 값을 size만큼 증가시키며 재할당받은 메모리 시작 주소를 return한다.

마지막으로, free()함수 내부에서는 LOG_FREE(ptr)의 호출로 할당 해제할 메모리의 시작 주소를 출력하고 freep(ptr)의 호출로 ptr이 가리키는 메모리 공간을 할당 해제한다.

```
69  __attribute__((destructor))
70  void fini(void)
71  {
72    // ...
73
74    LOG_STATISTICS(n_allocb, n_allocb/(n_malloc+n_calloc+n_realloc), 0L);
75
76    LOG_STOP();
77
78    // free list (not needed for part 1)
79    free_list(list);
80  }
```

마지막으로 소멸자 부분을 살펴볼 것이다. LOG_STATISTICS()를 호출하여 프로그램이 실행되는 동안 할당된 메모리의 총 바이트 수, 할당된 메모리의 평균 바이트 수, 할당 해제된 메모리의 총 바이트 수를 각각 출력하였다, 이때, n_allocb가 할당된 메모리의 총 바이트 수를 의미하고, n_malloc+n_calloc+n_realloc은 malloc(), calloc(), realloc() 함수가 호출된 횟수의 합을 의미한다. 따라서 n_allocb/(n+malloc+n_calloc+n_realloc)은 할당된 메모리의 평균 바이트 수를 의미한다. 한편,

part 1에서는 할당 해제된 메모리의 총 바이트 수를 따로 기록하지 않았으므로 알 수 없고, 과제 설명에서도 실제 값과 상관없이 0으로 출력하라고 되어있으므로 마지막 인자로 0을 전달하였다. 다음으로 LOG_STOP()을 호출하여 "Memory tracer stopped."를 출력하도록 하였다. 마지막으로 part 1과는 관련이 없지만 free_list(list)를 통해 linked list를 모두 할당 해제하였다.

(2) part 2

part 2에서는 linked list 자료구조를 활용하여 동적 할당된 메모리 관련 정보를 관리한다. 생성자 함수는 part 1과 달라진 부분이 없다. 반면, linked list의 도입으로 malloc(), calloc(), realloc(), free() wrapper 함수의 구현이 약간 달라졌는데 해당 코드는 다음과 같다.

```
94   void *malloc(size_t size){
95
96     void *tp=mallocp(size);
97     alloc(list, tp, size);
98
99     LOG_MALLOC(size, tp);
100    n_malloc++;
101    n_allocb+=size;
102
103    return tp;
104  }
105
106  void *calloc(size_t nmemb, size_t size){
107
108    void *tp=callocp(nmemb, size);
109    alloc(list, tp, nmemb*size);
110
111    LOG_CALLOC(nmemb, size, tp);
112    n_calloc++;
113    n_allocb+=nmemb*size;
114
115    return tp;
116  }
117
118  void *realloc(void *ptr, size_t size){
119
120    item *cur=find(list, ptr);
121    if (cur&&cur->cnt>0){
122      n_freeb+=cur->size;
123      dealloc(list, ptr);
124    }
125
126    void *tp=reallocp(ptr, size);
127    alloc(list, tp, size);
128
129    LOG_REALLOC(ptr, size, tp);
130    n_realloc++;
131    n_allocb+=size;
132
133    return tp;
134  }
```

```
136  void free(void *ptr){
137    LOG_FREE(ptr);
138
139    item *cur=find(list, ptr);
140    if (cur&&cur->cnt>0){
141      n_freeb+=cur->size;
142      dealloc(list, ptr);
143    }
144    freep(ptr);
145  }
```

'utils/memlist.c'에 정의된 alloc()과 dealloc()는 linked list를 관리해주는 함수이다. 할당해제된 메모리의 총 바이트 수를 구하거나 프로그램이 종료될 때까지 할당해제되지 않은 메모리 블럭에 관한 정보를 출력하기 위해서 linked list가 사용되는데, alloc()은 메모리 할당 시 list에 해당 메모리 관련 정보를 포함하는 블록을 추가하거나 업데이트하고 dealloc()은 메모리 할당해제 시 list에서 해당 메모리 관련 블록을 찾아 cnt를 1만큼 감소시키는 역할을 한다. 또한 find()함수는 linked list를 탐색하며 ptr이 가리키는 메모리 공간이 프로그램 실행 중 할당된 적이 있는지 확인하고 있다면 linked list에서 해당 메모리 관련 블록을 가리키는 포인터를, 없다면 NULL을 리턴한다.

malloc()과 calloc()이 part 1과 달라진 부분은 alloc()의 호출이다. 메모리 공간 할당 후 malloc()과 calloc() 내부에서 각각 alloc(list, tp, size)과 alloc(list, tp, nmemb*size)를 호출하여 linked list에 할당된 메모리 관련 정보(할당 크기, 메모리 시작주소, 참조 개수, 다음 블록에 대한 포인터)를 포함하는 블록을 추가하였다.

본 과제에서는 할당해제된 메모리의 총 바이트 수를 계산할 때 realloc()이 기존 메모리의 할당해제와 새로운 메모리의 할당을 모두 수행한다고 가정한다. 따라서 realloc() 내부에서 item *cur=find(list, ptr)를 실행했을 때 cur이 NULL이 아니고 cur->cnt가 0보다 크다면 해당 메모리 공간이 할당되어 있다는 것을 의미하므로 n_freeb에 기존 메모리의 크기인 cur->size를 더한다. 또한 void *tp=reallocp(ptr, size)를 통해 새로운 메모리 공간을 할당받았다. 여기에서도 앞에서와 마찬가지로 dealloc()과 alloc()를 통해 linked list에 메모리 할당 및 해제 관련 정보를 업데이트 하였다. 그 외 부분은 part 1과 동일하다.

free() 내부에서는 item *cur=find(list, ptr)을 실행했을 때 cur이 NULL이 아니고 cur->cnt가 0보다 크다면 해당 메모리 공간이 할당되어 있다는 것을 의미하므로 n_freeb에 할당해제하고자 하는 메모리 크기인 cur->size를 더한다. dealloc()을 통해 linked list에 메모리 해제 관련 정보를 업데이트하고 freep(ptr)을 통해 ptr이 가리키는 메모리 공간을 정상적으로 할당해제하였다.

```
69  __attribute__((destructor))
70  void fini(void)
71  {
72    // ...
73
74    LOG_STATISTICS(n_allocb, n_allocb/(n_malloc+n_calloc+n_realloc), n_freeb);
75    if (n_allocb-n_freeb > 0){
76      LOG_NONFREED_START();
77
78      item *cur = list->next;
79      while (cur){
80        if (cur->cnt>0){
81          LOG_BLOCK(cur->ptr, cur->size, cur->cnt);
82        }
83        cur=cur->next;
84      }
85    }
86    LOG_STOP();
87
88    // free list (not needed for part 1)
89    free_list(list);
90  }
```

part 2의 소멸자에서는 LOG_STATISTICS()의 마지막 인자로 할당해제된 메모리의 총 바이트수를 의미하는 n_freeb를 전달하여 freed_total이 정상적으로 출력되도록 하였다. 또한 n_allocb-n_freeb가 0보다 큰지 조사하고, 만약 0보다 크다면 할당해제되지 않은 메모리 블록이 있다는 의미이므로 우선 LOG_NONFREED_START()를 호출하여 "Non-deallocated memory blocks"를 출력하도록 하였다. 다음으로 linked list를 탐색하며 각 블록에 대해 참조 개수(cnt)가 0보다 큰지 확인하고 0보다 크면 할당해제되지 않았다는 의미이므로 LOG_BLOCK()를 통해 메모리 시작주소, 해당 메모리 블록의 바이트 수, 참조 개수를 출력하였다. 마지막으로 LOG_STOP()을 호출하여 "Memory tracer stopped."가 출력되도록 하였다.

(3) part 3

```
135 void free(void *ptr){
136   LOG_FREE(ptr);
137
138   item *cur=find(list, ptr);
139   if (!cur){
140     LOG_ILL_FREE();
141   }
142   else{
143     if (cur->cnt>0){
144       n_freeb+=cur->size;
145       freep(ptr);
146       dealloc(list, ptr);
147     }
148     else
149       LOG_DOUBLE_FREE();
150   }
151 }
```

part 3의 구현에서 part 2와 달라진 부분은 free() 함수 뿐이다. item *cur=find(list, ptr)를 실행했을 때 cur이 NULL이면 인자로 주어진 포인터가 가리키는 메모리 공간이 이전에 할당된 적이 없었다

는 것을 의미하므로 LOG_ILL_FREE()를 호출하여 관련 로그를 출력하도록 하였다. 만약 cur이 NULL이 아니더라도 cur->cnt가 0이면 해당 메모리 공간이 이미 할당 해제되었다는 것을 의미하므로 LOG_DOUBLE_FREE()를 호출하여 관련 로그를 출력하도록 하였다. 만약 이 두 가지 경우에 해당하지 않는다면 n_freeb를 cur->size만큼 증가시키고 freep(ptr)를 호출하여 정상적으로 메모리 공간을 할당해제하였다. 마지막으로 dealloc(list, ptr)을 호출하여 linked list에 메모리 할당 해제 관련 정보를 업데이트하였다.

3. 어려웠던 점

이론 수업시간에 Dynamic Linking을 다루었지만 본 실습을 하기 전에는 compile time, link time, load/run time interpositioning이 일어나도록 하는 코드를 구분하기 어려웠었다. 처음 접하는 개념이라 어렵다고 느껴졌는데 종류도 어려가지라 더 헷갈렸다. 그래서 이번 실습이 load/run time interpositioning에 관한 것이라는 설명을 들었을 때 걱정이 앞섰지만 하나하나 구현해보면서 load/run time interpositioning 개념에 대해 더 정확히 알 수 있는 기회가 되었다. 또한, 세 가지 종류 중 하나에 대해 정확히 이해하고 나니 이론 수업 당시 이해하지 못하고 넘어갔던 compile time과 link time interpositioning에 대해서도 정확히 이해할 수 있었다.

약간의 시행착오를 겪은 것을 제외하고 코드 구현에서 크게 어렵다고 느껴지는 부분은 없었다. part 2에서 소멸자 함수 내에서 할당되지 않은 메모리 블록을 출력하는 부분을 아래의 스크린샷과 같이 while 문으로 구현했는데 cur=cur->next를 실수로 if문 안에 포함시켰더니 프로그램이 종료되지 않는 경우가 있었다. "Memory tracer stopped."가 출력되지 않아서 LOG_STOP() 호출 전에 문제가 발생했을 것이라고 생각하여 소멸자 함수를 살펴보던 중 해당 오류를 발견하였고, cur=cur->next를 if문 밖으로 빼냈더니 정상적으로 로그가 출력되는 것을 확인할 수 있었다.

```
78      item *cur = list->next;
79      while (cur){
80        if (cur->cnt){
81          LOG_BLOCK(cur->ptr, cur->size, cur->cnt);
82        }
83        cur=cur->next;
84      }
```

그 외로, realloc 함수에 첫번째 인자로 Null 포인터가 전달될 때 어떻게 구현해야할지 몰라서 어려움을 겪었다. 'man realloc'을 통해 realloc의 동작 방식에 대해 살펴보니 첫번째 인자로 Null 포인터가 전달되어도 malloc(size)과 같이 정상적으로 동작한다고 되어 있어서 이미 할당해제된 메모리를 가리키는 포인터나 할당된 적 없는 메모리를 가리키는 포인터가 전달될 때 Double Free 관련 로그나 Ill Free 관련 로그를 출력해야 하는건지 헷갈렸는데 QnA 게시판에서 이 경우는 고려하지 않아도 된다고 하셔서 해당 어려움을 해결할 수 있었다.

4. 새롭게 배운 점

Dynamic linking은 일어나는 시기에 따라 compile time, link time, load/run time으로 구분할 수 있는데 구현 방법이 어떻게 다른지 정확히 이해하게 되었다. Compile time에서의 interpositioning은 전처리기를 이용하는 방법으로, 해더에 #define malloc(size) mymalloc(size)와 같이 선언을 하고 소스 파일 내에 wrapper함수인 mymalloc()를 정의하면 malloc()가 호출될 때 실제로는 mymalloc()이 호출된다. 또한, 리눅스의 정적 링커는 --wrap 플래그를 통해 link time interpositioning을 지원하는데, --wrap,malloc이라고 하면 프로그램 내부에서 malloc()을 호출할 때 __wrap_malloc()이 호출되고 __real_malloc(size)이 실제 malloc()을 참조한다. 마지막으로 본 과제에서 직접적으로 다룬 load/run time에서의 Interpositioning은 dlsym()이 프로그램 내부에서 호출되는 것이 특징이다. 이 때, dlsym()이란 동적라이브러리를 open하여 symbol의 시작 번지를 return 하는 함수를 의미한다. LD_PRELOAD 환경변수는 dynamic linker에게 탐색 순서를 알려주는데, 예를 들어 LD_PRELOAD=/usr/lib64/libdl.so로 지정하면 프로그램 내부에서 mallocp=dlsym(RTLD_NEXT, "malloc") 실행시 가장 먼저 /usr/lib64/libdl.so를 탐색하게 된다.

본 과제 설명 pdf에 다른 사람의 코드를 읽는 것에 익숙해져야 한다는 문구가 있어서 처음에는 '다른 사람과 협업을 하라는 의미인가?'라는 생각이 들었다. 그러나 본격적으로 과제를 수행하면서 해당 문구가 의미하는 바를 정확히 이해할 수 있었다. memlog.h, memlog.c, memlist.h, memlist.c에는 이미 다른 사람이 구현해둔 자료구조와 함수들이 굉장히 많은데, 이들을 적절히 사용하면 과제를 어렵지 않게 해결할 수 있다. 그러나 정확하게 동작하는 프로그램을 만들기 위해서는 구현되어 있는 자료구조를 어떻게 활용할 수 있는지, 구현되어 있는 함수들이 어떤 역할을 수행하는지 정확히 이해해야 하는데 이를 위해서는 다른 사람이 구현한 코드를 읽는 것이 필수적이다. 본 과제를 수행하면서 '다른 사람의 코드를 효율적으로 읽는 법'을 배울 수 있었다. main 내에서 함수가 호출될 때마다 해당 함수가 어떤 역할을 하는지 찾아보기보다 다른 사람이 구현한 자료구조나 함수들이 어떤 역할을 하는지 쭉 한 번에 모두 다 훑어보며 전체적인 틀을 이해하고 프로그램 내에서 함수를 호출할 때 기억이 나지 않으면 이전으로 돌아가 해당 함수의 역할을 다시 살펴보는 것이 나에게 맞는 방법임을 알 수 있었다.