System Programming Lab #1 Report

2019-18499 김준혁

실행 결과

/* Part 1 */

Part1 – test1

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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 ) = 0x7fffeab65290
[0003]          malloc( 32 ) = 0x7fffeab656a0
[0004]          malloc( 1 ) = 0x7fffeab656d0
[0005]          free( 0x7fffeab656d0 )
[0006]          free( 0x7fffeab656a0 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part1$
```

malloc과 free가 잘 돌아가며, allocate한 byte 수와 횟수를 구하여 평균을 구하는 것까지 잘 작동함을 확인

Part1 – test2

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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 ) = 0x7fffbf596290
[0003]          free( 0x7fffbf596290 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          0
[0009]
[0010] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part1$
```

test2 또한 정상적인 작동 확인

Part1 – test3

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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]          malloc( 17172 ) = 0x7fffd34b1290
[0003]          malloc( 47834 ) = 0x7fffd34b55b0
[0004]          malloc( 17272 ) = 0x7fffd34c10a0
[0005]          malloc( 59275 ) = 0x7fffd34c5420
[0006]          malloc( 29322 ) = 0x7fffd34d3bc0
[0007]          calloc( 1 , 34408 ) = 0x7fffd34dae60
[0008]          malloc( 38814 ) = 0x7fffd34e34d0
[0009]          malloc( 62136 ) = 0x7fffd34ecc80
[0010]          calloc( 1 , 58431 ) = 0x7fffd34fbf40
[0011]          calloc( 1 , 39536 ) = 0x7fffd350a390
[0012]          free( 0x7fffd350a390 )
[0013]          free( 0x7fffd34fbf40 )
[0014]          free( 0x7fffd34ecc80 )
[0015]          free( 0x7fffd34e34d0 )
[0016]          free( 0x7fffd34dae60 )
[0017]          free( 0x7fffd34d3bc0 )
[0018]          free( 0x7fffd34c5420 )
[0019]          free( 0x7fffd34c10a0 )
[0020]          free( 0x7fffd34b55b0 )
[0021]          free( 0x7fffd34b1290 )
[0022]
[0023] Statistics
[0024]   allocated_total      404200
[0025]   allocated_avg        40420
[0026]   freed_total          0
[0027]
[0028] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part1$
```

calloc 또한 작동을 확인할 수 있었다.

Part1 – test5 (Realloc 확인용)

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part1$ 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 ) = 0x7fffe882b290
[0003]          realloc( 0x7fffe882b290 , 100 ) = 0x7fffe882b290
[0004]          realloc( 0x7fffe882b290 , 1000 ) = 0x7fffe882b290
[0005]          realloc( 0x7fffe882b290 , 10000 ) = 0x7fffe882b290
[0006]          realloc( 0x7fffe882b290 , 100000 ) = 0x7fffe882b290
[0007]          free( 0x7fffe882b290 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          0
[0013]
[0014] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part1$
```

추가적으로 realloc의 작동 확인을 위해 test5를 이용하여 실시함.
realloc 또한 잘 작동함을 확인할 수 있었음.

/* Part 2 */

Part2 - test1

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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 ) = 0x7fffdf21f290
[0003]          malloc( 32 ) = 0x7fffdf21f6d0
[0004]          malloc( 1 ) = 0x7fffdf21f730
[0005]          free( 0x7fffdf21f730 )
[0006]          free( 0x7fffdf21f6d0 )
[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]   0x7fffdf21f290     1024       1
[0016]
[0017] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part2$
```

코드에 따르면, 1024바이트만큼 동적 할당한 것이 free되지 않는데, 그거에 맞게 Non-deallocated memory block이 정상적으로 출력되는 것을 확인할 수 있다. 또한 그 외의 나머지 할당받은 메모리는 free되어 총 33바이트가 나오는 것을 확인할 수 있다.

Part2 - test2

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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 ) = 0x7fffe05bf290
[0003]          free( 0x7fffe05bf290 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part2$
```

동적 할당받은 메모리가 모두 free 되었기 때문에, “Non-deallocated...” 메시지가 로그에 출력이 안되는 것을 확인할 수 있다.

Part2 - test3

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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]          calloc( 1 , 17115 ) = 0x7fffd9b31290
[0003]          malloc( 3299 ) = 0x7fffd9b355b0
[0004]          calloc( 1 , 19451 ) = 0x7fffd9b362d0
[0005]          malloc( 7846 ) = 0x7fffd9b3af10
[0006]          malloc( 4576 ) = 0x7fffd9b3cdf0
[0007]          malloc( 8175 ) = 0x7fffd9b3e010
[0008]          malloc( 52866 ) = 0x7fffd9b40040
[0009]          malloc( 16787 ) = 0x7fffd9b4cf00
[0010]          calloc( 1 , 54601 ) = 0x7fffd9b510d0
[0011]          calloc( 1 , 29076 ) = 0x7fffd9b5e660
[0012]          free( 0x7fffd9b5e660 )
[0013]          free( 0x7fffd9b510d0 )
[0014]          free( 0x7fffd9b4cf00 )
[0015]          free( 0x7fffd9b40040 )
[0016]          free( 0x7fffd9b3e010 )
[0017]          free( 0x7fffd9b3cdf0 )
[0018]          free( 0x7fffd9b3af10 )
[0019]          free( 0x7fffd9b362d0 )
[0020]          free( 0x7fffd9b355b0 )
[0021]          free( 0x7fffd9b31290 )
[0022]
[0023] Statistics
[0024]   allocated_total      213792
[0025]   allocated_avg        21379
[0026]   freed_total          213792
[0027]
[0028] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part2$
```

총 10번의 동적 할당과 해당 할당에 맞는 포인터의 메모리를 해제했기 때문에, 모든 할당 메모리가 free되었으며, 이에 따라 Non-dealloc 메시지가 출력이 되지 않았음을 확인할 수 있다.

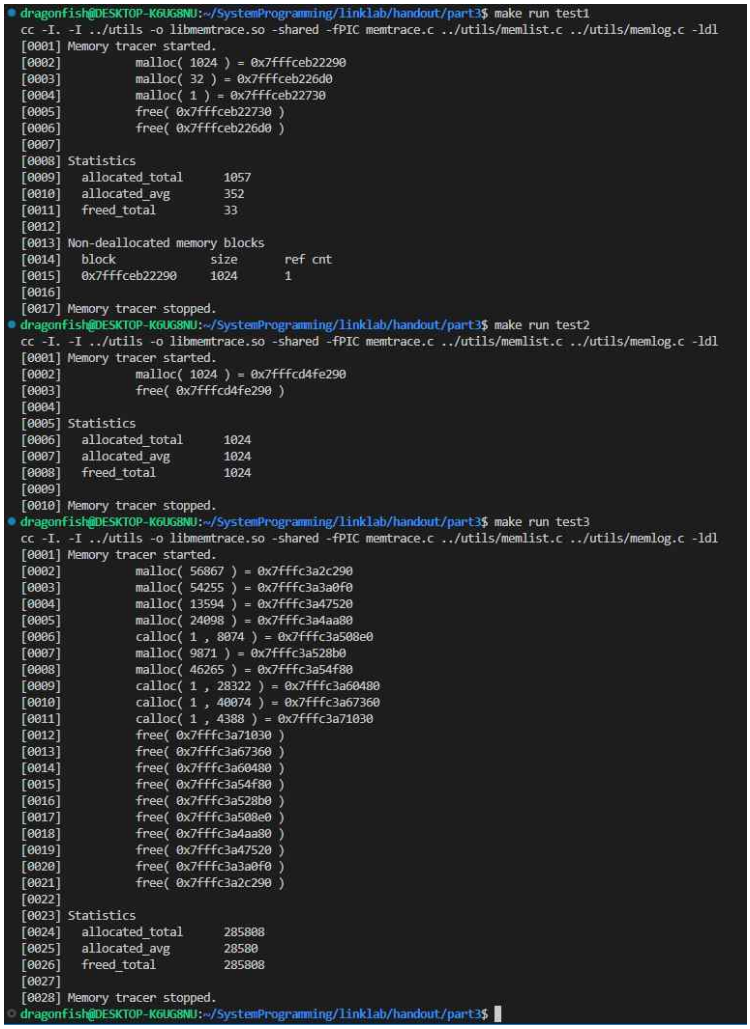
Part2 - test5 (realloc 확인용)

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part2$ 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 ) = 0x7fffe3598290
[0003]          realloc( 0x7fffe3598290 , 100 ) = 0x7fffe35982e0
[0004]          realloc( 0x7fffe35982e0 , 1000 ) = 0x7fffe3598380
[0005]          realloc( 0x7fffe3598380 , 10000 ) = 0x7fffe35987a0
[0006]          realloc( 0x7fffe35987a0 , 100000 ) = 0x7fffe359aef0
[0007]          free( 0x7fffe359aef0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part2$
```

realloc 작동 확인을 위해 test5도 돌렸으며, realloc을 통해 free된 메모리 또한 모두 잡아내고, 모든 할당 메모리가 free되었음을 확인, Non-dealloc 메시지를 출력하지 않았음을 확인.

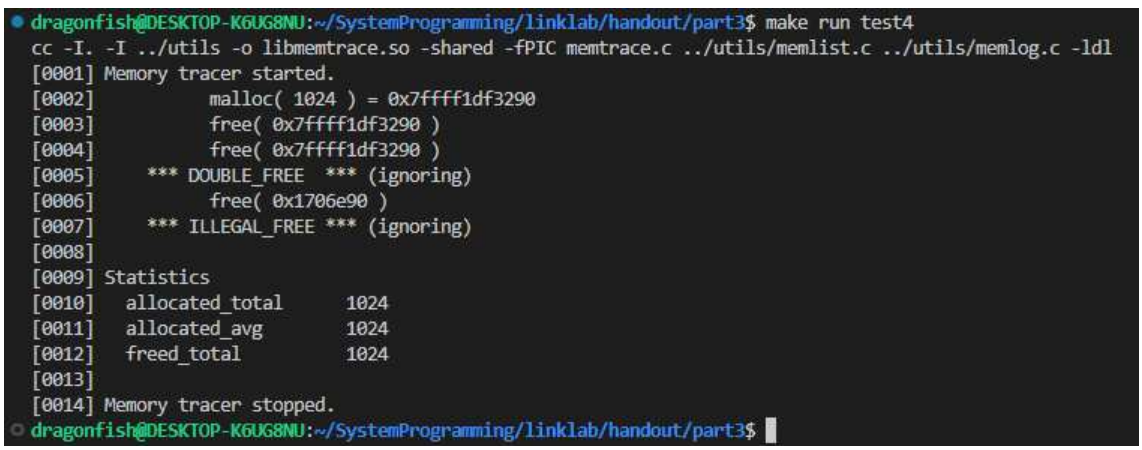
/* Part 3 */

Part3 - test1~3



Part2 와 같은 결과 출력을 하여, 문제 없음을 확인하였다.

Part3 - test4



test4를 돌려 나온 결과, double free와 illegal free가 걸러지고, ignoring 되는 것을 확인할 수 있다.

Part3 - test5

```
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/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 ) = 0x7fffc5e9c290
[0003]          realloc( 0x7fffc5e9c290 , 100 ) = 0x7fffc5e9c2e0
[0004]          realloc( 0x7fffc5e9c2e0 , 1000 ) = 0x7fffc5e9c380
[0005]          realloc( 0x7fffc5e9c380 , 10000 ) = 0x7fffc5e9c7a0
[0006]          realloc( 0x7fffc5e9c7a0 , 100000 ) = 0x7fffc5e9eef0
[0007]          free( 0x7fffc5e9eef0 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
dragonfish@DESKTOP-K6UG8NU:~/SystemProgramming/linklab/handout/part3$
```

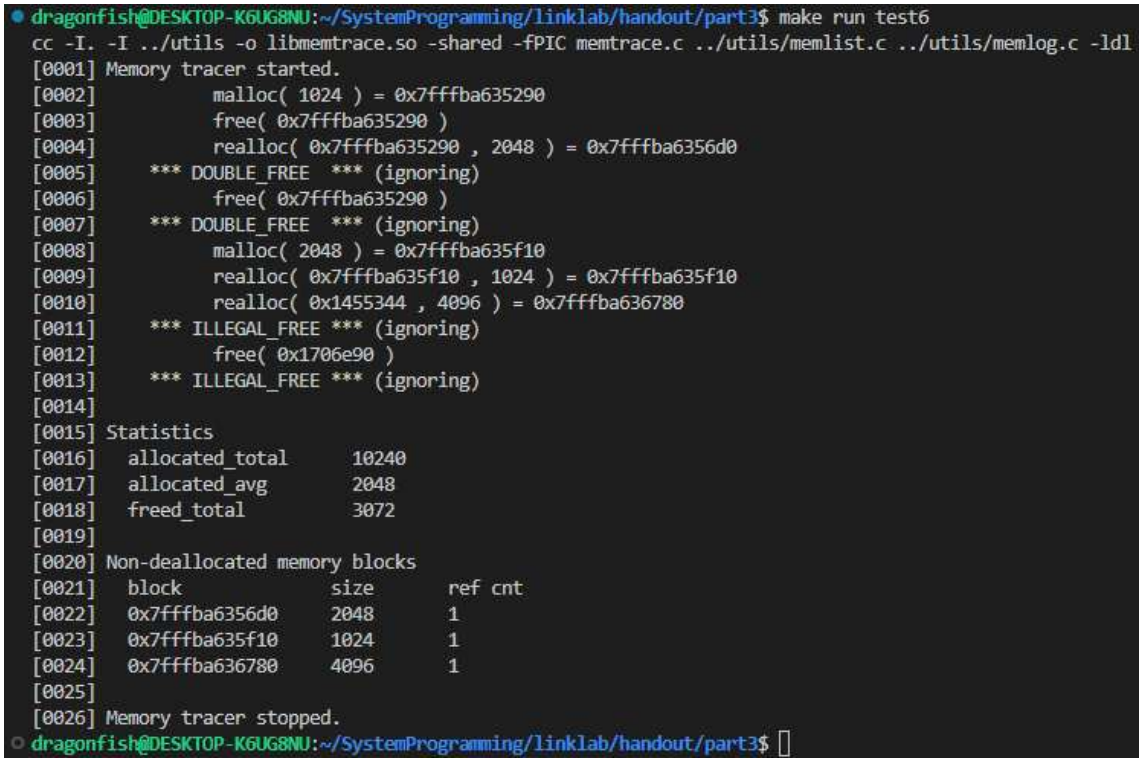
test5 에서는 illegal한 명령이 없기 때문에 에러 메시지 없이 정상적으로 작동했다.

Part3 - test6 (자체적으로 만든 testcase다. 제출 시에 삭제함)

SystemProgramming > linklab > handout > test > C test6.c

```c
#include <stdlib.h>

int main(void)
{
  void *a;

  a = malloc(1024);
  free(a);
  void *b = realloc(a,2048);
  free(a);
  a = malloc(2048);
  a = realloc(a,1024);

  a = realloc((void*)0x1455344,4096);
  free((void*)0x1706e90);

  return 0;
}

```

test6.c 코드를 다음과 같이 짰다. realloc에서 나오는 double free, illegal free를 확인하면서 ignoring하고 다른 주소로 동적 할당 받으며, Non-deallocated 된 메모리도 총 3개 나오게 하는 코드를 짜보았다.



원하는대로 9번, 10번 줄에서 double free를 걸러냈으며, 14번, 15번 줄에서는 illegal free를 걸러냈다. 그러면서도 총 allocated memory 와 freed memory 값을 통해, realloc 명령이 illegal/double free는 무시하면서 다른 주소에 allocate 했다는 것을 알 수 있으며, Non-deallocated memory blocks 가 복수인 경우에도 잘 출력했다는 것을 확인할 수 있다.

파트 별 구현 과정

/* Part 1 */

void Init()

```
__attribute__((constructor))
void init(void)
{
  char *error;

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

init 함수에는 load/run-time interpositioning을 해주는 코드를 넣었다. 수업 pdf를 참고하였으며, malloc 뿐만 아니라 calloc, realloc, free 또한 변수 이름만 다르게 하여 같은 형식으로 넣었다.

void *malloc(size_t size)

```
void *malloc(size_t size) {

  void *res;

  res = mallocp(size);
  n_allocb += (unsigned long) size;
  n_malloc++;

  LOG_MALLOC(size, res);

  return res;
}
```

실제로 할당 받은 메모리의 주소값을 받기 위해 void *res를 선언하고, 실제 malloc으로 얻은 주소값을 저장하였다. n_allocb를 allocate된 총 byte를,n_malloc은 malloc을 call한 횟수를 의미하는 변수로 판단하여 각각 size와 1을 더해주었고, 할당받은 size와 주소값을 로그 출력하도록 하였고, res를 return 하였다.

void *calloc(size_t nmemb, size_t size)

```
void *calloc(size_t nmemb, size_t size) {

  void *res;

  res = callocp(nmemb, size);
  n_allocb += (unsigned long) nmemb * (unsigned long) size;
  n_calloc++;

  LOG_CALLOC(nmemb, size, res);

  return res;
}
```

void *malloc 함수와 매우 유사하게 짰지만, 총 할당받는 메모리가 (nmemb * size)이기 때문에 해당 부분만 다르고 나머지는 비슷하게 작성했다.

void *realloc(void *ptr, size_t size)

```
void *realloc(void *ptr, size_t size) {

  void *res;

  res = reallocp(ptr, size);
  n_allocb += (unsigned long) size;
  n_realloc++;

  LOG_REALLOC(ptr, size, res);

  return res;
}
```

malloc, calloc과 유사하며, ptr만 추가된 형태다. part 1에서는 freeb 는 구하지 않기 때문에 해당 코드를 part 1에서 더미로 넣지는 않았다.

void free(void *ptr)

```
void free(void *ptr) {

  freep(ptr);

  LOG_FREE(ptr);

}
```

free 함수는 part 1에서는 큰 기능이 없어 할당받은 메모리를 free하고 로그를 출력하는 것만 작성하였다.

void fini()

```
__attribute__((destructor))
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);
}
```

LOG_STATISTICS에 총 allocated byte값과 평균 allocated byte 값을 넣어주었다.

/* Part 2 */

void Init()

init 함수의 목적은 interpositioning을 위한 dlsym과 list 초기화밖에 없기 때문에 part 1에서의 변경점이 없다.

void *malloc(size_t size)

```
void *malloc(size_t size) {

  void *res;

  res = mallocp(size);
  n_allocb += (unsigned long) size;
  n_malloc++;

  alloc(list, res, size);

  LOG_MALLOC(size, res);

  return res;
}
```

할당받은 메모리의 주소를 추적하기 위힌 linked list에 item을 저장하는 alloc 함수를 추가해줬다.

void *calloc(size_t nmemb, size_t size)

```
void *calloc(size_t nmemb, size_t size) {

  void *res;
  unsigned long total_size;

  res = callocp(nmemb, size);
  total_size = (unsigned long) nmemb * (unsigned long) size;
  n_allocb += total_size;
  n_calloc++;

  alloc(list, res, (size_t) total_size);

  LOG_CALLOC(nmemb, size, res);

  return res;
}
```

alloc 함수에 size를 전달하기 위해 nmemb*size를 2번하지 않기 위해 total_size라는 unsigned long 타입의 변수를 선언하고, linked list에 item을 추가하는 명령을 추가했다.

void *realloc(void *ptr, size_t size)

```
void *realloc(void *ptr, size_t size) {

  void *res;
  item *cur;

  cur = dealloc(list, ptr);
  n_freeb += (unsigned long) cur->size;

  res = reallocp(ptr, size);
  n_allocb += (unsigned long) size;
  n_realloc++;

  alloc(list, res, size);

  LOG_REALLOC(ptr, size, res);

  return res;
}
```

realloc할 때, 기존 메모리를 표면적으로 free함과 동시에 free된 메모리를 계산하기 위해 dealloc 함수를 이용하였고, 그를 통해 얻은 item *cur의 size를 n_freeb 에 더해주었다.

realloc은 void *ptr과 다른 곳에서 realloc이 될 수 있기 때문에, linked list에 넣는 alloc 함수에는 실제 realloc 함수를 통해 얻은 주소값을 넣어주는 식으로 수정했다.

void free(void *ptr)

```
void free(void *ptr) {

  item *cur;

  freep(ptr);

  cur = dealloc(list, ptr);
  n_freeb += cur->size;

  LOG_FREE(ptr);

}
```

linked list에서 dealloc함과 동시에 n_freeb에 free한 memory size를 더해주었다.

void fini()

```
__attribute__((destructor))
void fini(void)
{
  // ...

  LOG_STATISTICS(n_allocb, n_allocb / (n_malloc + n_calloc + n_realloc), n_freeb);

  if(n_allocb > n_freeb) {
    LOG_NONFREED_START();
    item *i;
    i = list->next;
    while(i != NULL) {
      if(i->cnt > 0) LOG_BLOCK(i->ptr, i->size, i->cnt);
      i = i->next;
    }
  }

  LOG_STOP();

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

allocated byte가 덜 free되어 freed byte보다 클 경우, “Non-deallocated...”를 1번 출력하고, linked list를 순회하여 cnt가 0보다 큰, 즉 free되지 않은 메모리 블록들의 정보를 출

력하도록 수정하였다.

/* Part 3 */
void Init()
void *malloc(size_t size)
void *calloc(size_t nmemb, size_t size)
void fini()
이 네 함수는 part 2에서 추가적인 변경사항이 없다.

void *realloc(void *ptr, size_t size)

```
void *realloc(void *ptr, size_t size) {

  void *res;
  item *cur;

  // Finding ptr before reallocation
  cur = find(list, ptr);
  if(cur == NULL) {
    res = mallocp(size);
    LOG_REALLOC(ptr, size, res);
    LOG_ILL_FREE();
  } else if(cur->cnt == 0) {
    res = mallocp(size);
    LOG_REALLOC(ptr, size, res);
    LOG_DOUBLE_FREE();
  } else {
    cur = dealloc(list, ptr);
    n_freeb += (unsigned long) cur->size;

    res = reallocp(ptr, size);
    LOG_REALLOC(ptr, size, res);
  }

  n_allocb += (unsigned long) size;
  n_realloc++;

  alloc(list, res, size);

  return res;
}
```

double/illegal free를 걸러내기 위해 먼저 find() 함수를 통해 linked list에 해당하는 메모

리 주소가 없는지/이미 free 되어 cnt가 0인지/어느 것에도 해당하지 않는지를 확인한다.

메모리 주소가 없을 경우 illegal free로 판단. mallocp를 통해 다른 주소로 메모리를 할당 받은 후, realloc 로그와 illgeal free 로그를 출력한다.

이미 free되어 cnt가 0인 경우 double free로 판단. mallocp를 통해 할당 받고, realloc 로그, double free 로그를 출력한다.

위의 두 상황에 해당하지 않는 경우, 정상적인 realloc 실행 가능으로 판단. 기존 part2와 비슷한 과정이 수행되도록: linked list에서 dealloc을하고, reallocp로 메모리를 할당받고, realloc 로그를 출력한다.

세 상황 모두 free에는 차이가 있지만, 결국 메모리를 할당받기 때문에 공통적으로 할당받은 byte 값들을 더해주고, 횟수도 realloc으로 증가시키며, linked list에 alloc해준다.

void freevoid *ptr)

```
void free(void *ptr) {

  item *cur;

  LOG_FREE(ptr);

  // Finding ptr before free
  cur = find(list, ptr);
  if(cur == NULL) {
    LOG_ILL_FREE();
    return;
  } else if(cur->cnt == 0) {
    LOG_DOUBLE_FREE();
    return;
  }

  cur = dealloc(list, ptr);
  n_freeb += cur->size;

  freep(ptr);

}
```

기본적인 double/illegal free의 판단은 realloc에서와 동일하지만, 해당 오류의 경우, free를 진행하지 않기 때문에 로그를 출력하고 바로 return하도록 하며, 정상적인 경우에 대해서만 n_freeb를 더하고, linked list에서 dealloc 되도록 하였다.

어려웠던 점

첫 과제였다보니 구현하는 쪽에서는 큰 어려움은 없었다. 다만, test 코드들이 기본적인 구현에 있어서 충분한 feedback을 제공해주지 않아(특히 realloc에 대해) 직접 따로 testN.c를 만들어서 하는 것이 맞는지 헷갈렸다. 또한 pdf에서 제공해주는 정보로는 realloc에서 발생하는 double/illegal free에 대해서 어떻게 기능해야할지 알려주지 않아서, QnA에 관련된 내용이 나올 때까지, 내가 문제 내용을 덜 이해했나하고 미루고 고민했던 것이 나한테는 까다롭게 느껴졌다.

그리고, part1 이후 2, 3까지 코드를 복사하면서 진행을 하는데, part3에서 최적화한 부분이 part2에서도 최적화가 가능한 부분일 때, 이를 참지 못하고 part2에 되돌아가서 다시 수정하고 혹시나 에러가 발생하지 않는지 다시 빌드하고 테스트하는 과정을 거치는 것이 조금 힘들었다. 개인적인 강박으로 발생한 일이나, 이에 대한 최적화 내용을 보고서에 다 반영하기에는 설명이 길어질 것 같아 이를 방지하기 위한 작업에서 피로를 느낄 수밖에 없었다.

새롭게 배운 점

load/run-time interpositioning이 어떻게 구현되고 실행이 되는지를 실습을 통해 더욱 직관적으로 느껴볼 수 있었고, 동적 할당된 memory가 non-free되는 것을 막기 위해 관련 함수들을 추적하면서, linked list 등에 저장한 주소값을 통해 non-free된 위치를 추적 및 관리할 수 있다는 것을 깨달았다.

지금까지 프로그래밍할 때 메모리를 관리할 필요가 없긴 했다. 막연한 두려움으로 동적 할당된 메모리 관리를 하는 것이 쉽지는 않을 것이라 생각이 들었지만, 이번에 실습한 것을 통해 메모리를 관리하는 법을 어느 정도 깨달았다고 생각한다. 추후에 프로그래밍하면서 메모리에 대한 관리가 필요할 때, 이보다 더 효율적으로(특히 자료구조적인 면에서) 관리할 수 있는 방법을 생각하여 이용해야겠다.