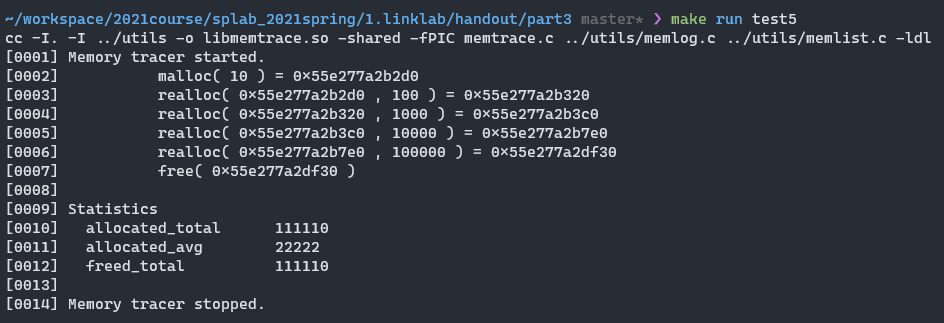
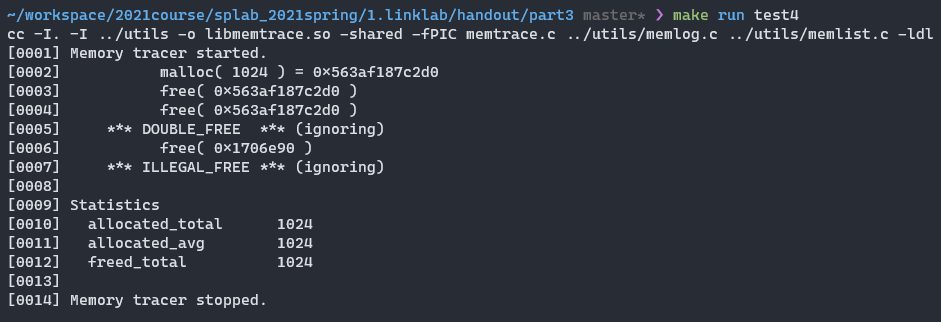
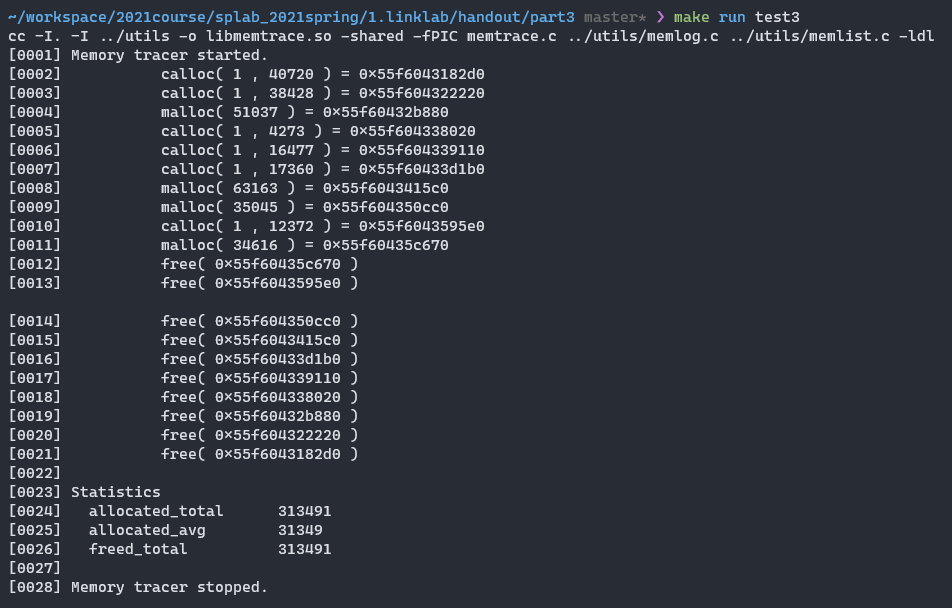
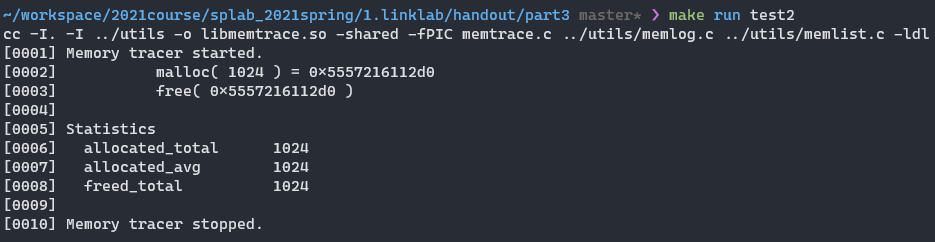
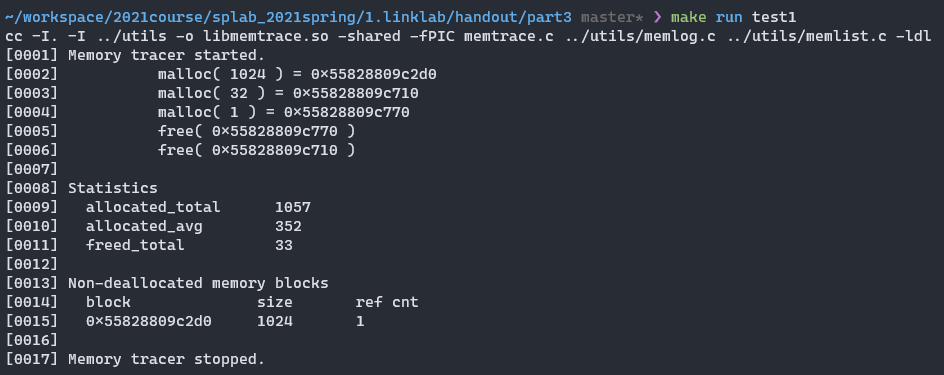
**2021 SYSTEM PROGRAMMING  
Lab1 Report – Link Lab**

자유전공학부 2012-13311 안 효 지

**1. 실행 결과** part3에 part1, 2의 결과가 모두 포함되므로 part3의 결과만 첨부한다.



**2. 구현방법**

**<Part1>** Dynamic linking은 .so에 대한 정보만 가지고 있는 partially linked executable file이 실제로 메모리에 load되고 실행될 때, 메모리에 올라와있는 .so에서 필요한 definition들을 링킹하여 fully linked executable file로 되는 것을 의미한다. 이 과정에서 LD\_PRELOAD 환경변수가 shared library의 경로로 되어있으면 stdlib보다 그 경로에 먼저 가서 찾는 성질을 이용하여 interpositioning 하는 것이 load/run time interposition이다. 따라서 실제 우리가 호출하는 malloc, calloc, realloc, free 함수의 이름과 동일한 wrapper 함수를 만들어주어야 한다.  
 **(1) libc의 함수 주소 받아오기**: 각 wapper 함수에서 dlsym(RTLD\_NEXT, “…”); 를 통해 libc에 존재하는 original 함수의 주소를 받아온다. 함수명은 주소이기 때문에 그 주소에 인자들을 넣어 return value를 받는다.  
 **(2) Tracing dynamic memory allocation**: malloc, calloc, realloc의 경우에는 할당되는 size를 Statistics에 print되는 allocated\_total(n\_allocb)에 더해준다. 또한, allocated\_avg를 구하기 위해 각 wrapper함수가 호출될 때마다 n\_malloc, n\_calloc, n\_realloc에 ++를 해준다.   
 **(3) Statistics 출력 코드 넣기**: 계산된 변수들을 이용하여 destructor에서는 LOG\_STATISTICS가 출력되는데, alloc\_avg = n\_allocb / (n\_malloc+n\_calloc+n\_realloc) 을 계산하는 과정에서 divided by 0가 발생하지 않도록 divisor == 0인 경우엔 평균도 0으로 출력되도록 구현하였다. 이미 제공된 매크로함수를 이용하였다.

**<Part2**>  
 **(1) allocated, freed memory 계산하기**: memory tracing을 위해 linked list가 사용된다. Malloc, calloc의 경우에는 memlist.c에 있는 alloc()을 사용해 list에 추가를 한다. Realloc의 경우에는 일단 기존의 할당된 공간을 없애야하므로 dealloc()을 해서 list node의 cnt를 줄여서 free가 되었다는 것을 표시해주고 기존 공간의 size를 n\_freeb에 더해준다. 그리고 새로 할당된 공간의 주소를 받아 alloc()하여 list에 추가하고 n\_allocb에 해당 사이즈만큼 더해준다. Free의 경우에는 인자로 전달받은 주소를 dealloc()으로 보내고, 그 노드의 사이즈를 n\_freeb에 더해서 total\_freed\_byte를 구한다.  
 **(2) unfreed memory 출력하기**: total\_allocated\_byte와 total\_freed\_byte가 같아야 unfreed memory가 없는 것이다. 따라서 두 값이 같지 않은 경우에만 “Non-deallocated memory block”가 출력되도록 하였다. List를 순서대로 iterate하며 non-freed block을 의미하는, cnt가 0이 아닌 노드의 정보만을 출력하게 하였다.

**<Part3> Illegal/Double free  
@ free()  
 (1) illegal free check**: free()에 argument로 들어온 ptr이 list에 있는지 유무를 find()를 통해 확 인해보면 된다. 존재하지 않는다면 LOG\_ILL\_FREE()를 호출하여 메시지를 출력하도록 한다.  
 **(2) double free check**: illegal free는 아니라면 이미 free된 곳인지 확인해야 한다. (1)에서 find()를 했던 return value를 변수에 저장해놓고, 해당 변수의 cnt값이 0인지 확인하여 0이라면 double free 메시지를 출력하도록 한다.  
 **(3)** **normal free**: illegal free도, double free도 아니라면 part2처럼 동작하도록 두면 된다.   
**@ realloc()  
 (1) illegal free check:** realloc()은 기존block dealloc() 후 새 block을 alloc()하는 것이 원칙이기 때문에 dealloc() 과정에서 illegal free가 발생할 수 있다. Free()에서 했던 것과 마찬가지로, find()를 통해 argument로 들어온 ptr이 list에 존재하지 않는지 확인한다. 존재하지 않으면 illegal free에 해당한다. 하지만 이 오류를 무시하고 alloc() 단계는 제대로 수행되어야 하므로 reallocp(NULL, size)를 이용해 malloc(size)처럼 작동되도록 코드를 짜고, alloc()을 통해 list에 추가 및 total\_allocated bytes에 할당된 size를 더해주고, LOG\_REALLOC을 먼저 출력한 후 LOG\_ILL\_FREE를 출력한다.   
 **(2) double free check**: free()와 마찬가지로, find()를 했던 결과를 받아놓고, 해당 node의 cnt 값이 0이면 이미 free(dealloc)이 된 상태를 의미하므로 이를 확인한다. 이후 작업 역시 realloc의 illegal free check과 동일하다.   
 **(3) normal realloc**: 정상적으로 작동하면 되므로 part2의 realloc의 작업을 수행한다.

**3. 어려웠던 점** 강의시간에 library interposition에 대해 배우긴 했지만, 프로그래머의 필요 때문에 코드를 중간에 탈취하는 방법 정도로 추상적으로 이해하고 있을 뿐이었다. 이것이 정확히 왜 필요하고 어디에 쓰이는지 직접 경험한 적이 없었으므로 개념 자체가 낯선 것 자체가 과제 수행 과정에서 어려웠던 점이다. 하지만 과제 수행이라는 구체적인 상황이 주어진 상태에서 csapp 교재와 강의pdf를 참고해 shared library, dynamic linking의 개념을 확실히 복습하고 library interposition 부분을 읽으니 compile, link, load/run time interposition의 원리와 특성이 확실히 이해가 되었다.   
 함수 포인터에 대해 C언어 개념을 공부할 때 한 번 듣고 넘어간 적이 있었지만 컴퓨터구조 수업에서도 직접 사용해본 적이 없었기에 함수포인터 문법이 낯설었다. 하지만 개념을 다시 살피고 과제를 하며 직접 사용해 보았으므로 앞으로는 잊을 일이 없을 듯하다.

**4. 새롭게 배운 점**  
 part2의 test5를 통과하기 위해 list를 이용해 memory tracing을 하는데 realloc에서만 자꾸 오류가 났다. Realloc에 대한 잘못된 이해가 원인이었다. `void\* rptr = realloc(ptr, size);` 에서, 무조건 기존에 allocated된 주소(ptr)에 새로운 size를 할당하여 항상 rptr==ptr인 줄 착각하고 있던 것이었다. 한참 코드를 손보다가 `$ man realloc`을 해보니 return value와 argument는 같을 수도 있고 다를 수도 있으며, size가 0인 경우에는 free()의 역할을, ptr이 NULL이면 malloc(size)의 역할을 할 수도 있다는 것을 알게 되었다.