System Programming

Programming Assignment #1 Linklab

2013-11826 임주경

- Part 1

Malloc, calloc, realloc, free의 네 함수는 memtrace.c에서 함수형 포인터로 선언된 mallocp, callocp, reallocp, freep가 dlsym 함수를 사용해 원래의 stdlib.h에 담긴 함수들에 접근할 수 있도록 dynamic linking 하였다. 이때, 새로 만든 malloc 함수는 원래 함수에 접근해 그대로 실행한 뒤, memlog를 이용해 LOG_MALLOC으로 실행 결과를 출력한다. 글로벌 변수 n_malloc을 하나 증가시켜 카운터 하며, n_allocb를 할당된 메모리 size만큼 증가시킨다. Free함수도 마찬가지로 dynamic linking 하며, 원래 함수의 기능을 마치고, LOG_FREE로 실행결과를 출력한다. Calloc, realloc 또한 마찬가지로 linking하며, 각각 memlog의 macro를 이용해 실형 결과를 출력한다. 이때, 각각의 글로벌 변수 n_calloc, n_realloc를 실행마다 카운터 하고, 할당된 메모리 크기만큼 n_allocb를 증가시킨다. 프로그램을 마칠 때, fini 함수 내부에 LOG_STATISTICS를 사용해서 할당된 메모리 크기의 총합과 그 크기를 할당 함수 카운터들의 합으로 나눠 평균 낸 크기를 출력하도록 구현하였다.

Part 1을 진행하며, library interpositioning을 이해할 수 있었고, load time에서 shared library를 이용한 dynamic linking의 개념을 배울 수 있었다.

- Part 2

Memtrace.c 내부에서 구현된 item의 pointer를 사용해서 할당된 메모리 포인터들의 정보를 갖는 구조체의 리스트를 구성하였다. 구현 과정에서 memlist.c의 alloc, dealloc, find 함수를 사용하였다.

Malloc과 calloc은 memlist.c의 alloc함수로 item list에 추가하였고, free 함수로 할당 해제된 메모리 포인터는 dealloc을 사용해서 list에서 ref cnt가 하나 감소하여 0을 갖는 item으로 수정된다. 또한, find 함수를 사용해서 free에서 할당 해제되는 메모리 포인터를 item list에서 찾아 그 size만큼을 n_freeb (할당 해제된 메모리 크기 총합)에 더하도록 하였다. Realloc 함수의 구현은 우선, 기존에 할당되어 있던 크기만큼 find 함수를 이용해 찾아내 n_freeb에 더하고, dealloc, alloc을 차례로 진행하여, item list 내부에서도 실제 realloc 함수가 free된 후, 다시 malloc이 진행된 결과가 반영되도록 구현하였다. Non-deallocated memory 출력을 위해서 fini 함수 내부에 몇 가지 구현을 추가하였다. 우선, n_allocb 와 n_freeb가 같지 않을 경우, Non-deallocated memory blocks table의 title을 출력하도록 설정하여, 할당중인 메모리가 존재하지 않으면 아예 list table을 띄우지 않게 하였다. 이 list는 while 함수를 이용해 item 포인터의 ref cnt값이 0이 아닌, 즉, 아직 메모리가 할당된 item들은 LOG_BLOCK을 이용해 출력되도록 설정하였다.

- Part 3

Part 3의 구현을 위해 callinfo.c 내부의 get_callinfo 함수를 구현하였다. 구현을 위해 libunwind.h 헤더 파일을 사용하였으며, #define UNW_LOCAL_ONLY를 선언하여 라이브러리 내부의 함수들을 원활히 사용할 수 있도록 하였다. Unw_context_t 타입의 context 변수와 unw_cursor_t cursor 변수를 선언하였고, unw_getcontext 함수와 unw_init_local 함수를 이용해서 cursor 값을 초기화해주었다. Unw_step 함수를 이용해서 stack frame을 tracking하며, main 함수에 cursor가 존재하도록 3번의 step을 진행하였다. 이때, 동적 할당 함수들이 항상 main함수에서 call된다는 전제에 따라서 구현하였다. Cursor가 main함수에 위치하게 한 뒤, unw_get_proc_name 함수를 이용해서 get_callinfo의 인자들인 fname, fnlen, ofs에 현재 프로시져의 정보를 입력하였다. 이때, callq의 instruction이 실습 환경에서 5byte라는 전제로, off set 보정 값으로 5를 빼주었다. 위의 구현이 원활하게 완료되면 0 값을 return 해주도록 구현하였다. 위의 구현으로 각 동적 할당 함수들이 call 되었을 때, 상대적 pc값과 호출되는 함수의 위치를 올바르게 tracing할 수 있었다. Unwind 헤더 파일이 생소하였고, 별도의 return이 필요 없이 함수에 변수로 전달하는 것만으로 프로시져의 정보를 각 변수에 저장할 수 있다는 점이 처음에 와 닿지 않아서 구현하는 데 어려움을 느꼈다. 또한, 구현 과정에서 debugging에도 unwind 함수들을 유용하게 사용할 수 있다는 점을 배울 수 있었다.

- Bonus

Bonus 구현은 free, realloc 두 함수의 수정으로 완성하였다. 우선 free의 경우, memlist의 find함수를 사용하여 할당 해제하려는 포인터가 기존에 존재하지 않는 Illegal 포인터라면, LOG_ILL_FREE를 실행하며, 기존에 존재하는 포인터지만, ref cnt가 0으로 이미 해제된 포인터라면, LOG_DOUBLE_FREE를 실행하도록 한다. 그 외에는 정상적으로 free가 실행되도록 하였다.

Realloc도 마찬가지로, find를 이용해 포인터를 리스트에서 체크하고, Illegal 포인터라면 원래의 realloc함수에 입력 주소 값을 NULL로 실행한다. Double free도 마찬가지이며, 각각의 경우에 맞게 mlog 매크로를 구현하였다. Free가 제대로 실행되지 않더라도, 원래의 realloc 함수로 구현하였기 때문에, malloc은 일어난다. 따라서 n_allocb 값의 증가와 alloc을 이용한 list 갱신은 그대로 진행돼야 한다.