시스템 프로그래밍 실습가이드

Malloc Lab

2021년 11월 22일

1. 개요

이 랩은 C프로그램을 위한 동적할당기를 작성하는 것이다. 즉, malloc, free, realloc 함수들을 여러분 만의 버전으로 작성해야 한다. 여러분은 이 함수들을 작성하면서 할당작업이 정확하고, 빠르고, 효율적으로 동작하도록 해야 한다.

2. 실습방법

여러분이 수정해야 하는 파일은 mm-naive.c, mm-implicit.c, mm-explicit.c 파일이다. mdriver.c 프로그램은 여러분이 구현한 코드를 시험하기 위한 프로그램이다. make 명령을 이용해서 mdriver를 만들고, ./mdriver -v 명령을 실행해보라.(-v 옵션은 간단한 정보를 출력한다)

여러분이 작성해야 하는 동적할당기는 다음의 네 개의 함수로 이뤄져 있으며 mm.h에 선언되어 있고, mm-*.c 파일에 구현되어 있다.

```
int mm_init(void)
void *mm_malloc(size_t size)
void mm_free(void *ptr);
void *mm_realloc(void *ptr, size_t size)
```

주어진 mm-*.c 파일은 아주 간단한, 그러나 정확히 동작하는 malloc 패키지를 이미 포함하고 있다. 이 코드를 참고해서 이 함수들을 수정해야 한다. 이 과정에서 추가로 함수를 몇 개 만들수도 있다. 각 함수들의 기능은 다음과 같다.

- mm_init : mm_malloc, mm_realloc, mm_free를 호출하기 전에 응용프로그램은(mdriver 프로그램) mm_init을 호출해서 필요한 초기화 작업을 수행한다. 초기화 작업에는 최초 힙 공간 할당이 포함된다. 만일 초기화 과정에서 오류가 발생하면 -1을 리턴하고, 그 외의 경우에는 0을 리턴한다.

- mm_malloc : mm_malloc함수는 적어도 size바이트 이상의 할당된 데이터 블록의 포인터를 리턴한다. 전체 할당된 블록은 힙 영역에 위치해야 만 하고, 다른 할당된 블록들과 겹쳐서는 안된다. 여러분이 작성한 malloc버전을 표준 C라이브러리(libc)에서 제공하는 malloc의 성능과 비교할 것이다. libc 의 malloc은 항상 8바이트로 정렬된(aligned) 데이터 포인터를 리턴하기 때문에, 여러분의 malloc 구현도 이처럼 동작해야 하고, 8바이트로 정렬된 포인터를 리턴해야 한다.

- mm_free : mm_free루틴은 ptr이 가리키는 블록을 반납한다. 그리고, 아무것도 리턴하지 않는다. 이 루틴은 이전의 mm_malloc과 mm_realloc가 포인터 ptr을 리턴한 경우에, 그리고, 아직 free되지 않은 경우에만 정상동작한다.

- mm_realloc : mm_realloc함수는 적어도 size바이트 이상의 할당된 영역의 포인터를 다음과 같은 제약사항을 준수하면서 리턴한다.

  - 만일 ptr이 NULL이면, mm_malloc(size)와 동일하다

  - 만일 size가 0이면, mm_free(ptr)과 동일하다

  - 만일 ptr이 NULL이 아니면, 이것은 이전에 mm_malloc이나 mm_realloc으로 리턴받은 값이다. mm_realloc을 호출하면 ptr이 가리키는 메모리 블록의 크기는(이전 블록의 크기) size 바이트로 변경되며, 새로운 블록의 주소가 리턴된다. 새로운 블록의 주소는 이전 블록의 주소와 동일할 수도 있으며, 다를 수도 있다.

    새로운 블록의 내용은 이전크기와 새블록의 크기 중에서 작은 값 만큼은 이전 ptr블록과 동일하다. 그 외의 부분은 초기화 되지 않는다. 예를 들어, 만일 이전 블록이 8바이트고, 새 블록이 12바이트라면, 새블록의 첫 8바이트는 이전 블록의 첫 8바이트와 동일하며, 나머지 4바이트는 초기화 되지 않은 상태다. 마찬가지로, 만일 이전 블록이 8바이트이고, 새 블록이 4바이트라면, 새 블록의 내용은 이전 블록의 첫 4바이트와 동일하다.

3. 힙 안정성 체커

동적 메모리 할당기는 정확하게 구현하기가 매우 까다롭고 오류가 발생하기 쉽다. 그래서 다음과 같은 기능을 이용해서 힙의 상태를 체크해주면 좋다:

- free list의 모든 블록이 free로 표시되어 있는가?
- 블록연결을 하지 않은 연속된 가용블록들이 존재하는가?
- 모든 가용블록들은 실제로 free list에 들어가 있는가?
- free list의 포인터들이 유효한 가용 블록들인가?
- 할당된 블록들이 겹치지는 않는가?
- 힙 블록에서 포인터들은 유효한 힙의 주소를 가리키고 있는가?

힙 체커기능은 mm.c 파일의 int mm_check(void)에 구현할 수 있다. 만일 여러분의 힙에 문제가 있는 경우 0이 아닌 값을 리턴한다. 이 체커는 디버깅을 위해 여러분이 작성할 수 있는 함수다. 이 함수는 필요한 경우에 호출해서 사용하고, 최종 구현시에는 삭제해야 성능에 손해가 없다.

4. 지원 함수

memlib.c 패키지는 여러분의 동적메모리 할당기를 위한 메모리 시스템을 시뮬레이션 해준다. 여러분은 memlib.c의 다음 함수를 호출할 수 있다.

- void *mem_sbrk(int incr) : 힙을 incr바이트 만큼 늘려준다. 여기서 incr 는 양수이며, 새롭게 할당된 힙 영역의 첫 번째 바이트를 가리키는 포인터를 리턴한다. 이 기능은 리눅스의 sbrk함수와 거의 동일하다
- void *mem_heap_lo(void) : 힙의 첫 바이트의 포인터를 리턴
- void *mem_heap_hi(void) : 힙의 마지막 바이트의 주소를 리턴
- size_t mem_heapsize(void) : 힙의 현재 크기를 바이트수로 리턴
- size_t mem_pagesize(void) : 시스템의 페이지 크기를 바이트 수로 리턴(리눅스에서는 4K임)

5. 프로그램 작성규칙

- mm.c의 다른 인터페이스부분은 수정해서는 안된다
- malloc, calloc, free, realloc, sbrk, brk와 같이 메모리 관리 관련 라이브러리나 시스템콜은 절대로 사용할수 없다.

- 여러분은 배열, struct등을 mm.c 프로그램의 전역변수나 static변수로 새롭게 만들어서는 안된다. 그렇지만, 복합자료구조가 아닌 단일 변수를 전역변수로 선언해서 사용할 수는 있다.

6. 힌트

- 초기 개발시에는 mdriver -f 옵션을 이용해서 디버깅과 테스팅을 단순화 해라.
- mdriver -v와 -V 옵션을 이용하라. -v 옵션은 각 trace파일의 요약을 보여준다. -V는 각 trace파일이 읽혀질 때를 표시하고, 이를 이용하면 에러를 분리할 수 있게 된다.
- gcc -g 옵션을 이용해서 gdb를 이용하라. 디버거를 이용하면 메모리 참조 오류를 찾아내는데 유용하다.
- 교재에 구현된 malloc 코드의 모든 라인을 다 이해하라. 책에 간접리스트 방식의 코드가 모두 공개되어 있다.(9.9.12절) 여기서부터 출발 하는게 좋다. 간단한 간접리스트 방식의 코드 모두를 이해하기 전에는 여러분의 할당기 구현을 시작하지 말아라.
- 포인터 연산은 C 전처리기 매크로에 구현하라. C언어에서 포인터 사용은 에러가 발생할 가능성이 높다. 따라서, 교재의 코드를 참고하면서 여러분 만의 매크로로 구현하라.
- 단계적으로 구현하라. 처음의 9개의 trace들은 malloc과 free 요청들을 포함하고 있다. 마지막 2개의 trace는 realloc, malloc, free를 포함하고 있다. 먼저 malloc과 free를 9개의 trace에 대해 완벽하게 구현할 것을 추천한다. 그 이후에 realloc구현을 하는게 좋다. 처음에는 작성한 malloc과 free를 이용해서 realloc을 구현할 수 있다. 그러나, 정말로 우수한 성능을 얻기 위해서는 독립적인 realloc을 구현해야 한다.
- 성능을 최적화 하려면 gprof도구를 이용하면 좋다.
- 일찍 시작하라!!! 간단해 보이지만, 막상 구현을 시작하면 코딩이 꼬인다... 바로 시작해라.. 지금..