Assignment2

2020년 11월 15일 일요일 오후 11:08

20180486 이인희

0. 전역변수

기능 제작의 편의를 위해서 전역변수를 몇 개 정의하여 사용하였다.
데이터 구조의 정보를 저장하기 위한 meta_addr와 관련 변수들이 대표적이며, const 를 통해 자주 사용할 값들을 저장해두었다.

1-1. Radix 관련 구조체.

Radix_root는 Radix tree의 root와 height 등의 정보를 담는 일종의 호출자이다.
Radix_node는 radix tree의 노드인데, 64bit 시스템에서 48bit으로 주소가 들어와, PAGE INDEX는 36bit라 총 6층의 radix tree가 구성되게 되었다.

구현의 편의를 위해서, 6 level로 고정을 하고서 구현을 하였다.

1-2. thread cache 관련 구조체.

Thread_cache는 thread cache를 나타내는 구조체로 thread id와 free list들의 구조체인 thread_free_list로 구성되어 있다. 또한 Thread_cache는 Linked List로 구현되어 있기 때문에 front, back이 구조체에 포함되어 있다.

1-3. 기타 구조체들.

Central_cache는 thread_free_list에 대응되는 구조체로 볼 수 있다. 다만 약간의 차이가 존재하는데, thread_free_list의 경우 list들이 small size의 주소를 담고 있지만, Central_cache의 경우 CentralFreeList라는 구조체의 주소를 담고 있다. 이 CentralFreeList의 경우 5개의 요소로 구성되어 있는데, 이중 2개는 Span object를 나타내기 위한 주소이고, 남은 3개는 small object가 thread cache에서 central cache로 migrate할 때 최초로 저장하기 위한 부분이다. Span object의 경우 비어있는 것과 그렇지 않은 것을 따로 doubly linked list로 구현을 해두어 allocation 속도를 높이고자 했다. CentralFreeList의 re_head_obj, re_tail_obj는 thread_cache가 커져서 넘겨받은 freed small object들이 담겨 있는 곳으로, tail쪽으로 지속적으로 넣어준다. num_re는 small object의 개수를 나타낸다.

Page_Heap의 경우 pagemap으로 256개의 Span object를 가리키는 포인터와, radix tree root를 나타내는 DB로 구성되어 있다. pagemap의 경우 large object allocation이나 small object allocation에서 central cache의 Span이 비었을 때 페이지 단위로 메모리를 얻기 위해 탐색하는 곳이다. 이속에서 원하는 Span을 찾지 못하면 system call을 통해서 (mmap) 메모리를 할당받고 Span을 정의하게 된다.

Meta의 경우 Page_Heap과 Central_cache, Thread_cache를 담아두는 구조체로, 지금까지 정의한 모든 자료구조, 구조체 instance들을 접근할 수 있게 해주는 meta_data라는 전역변수를 위해 만들어 둔 구조체이다. 따라서 meta_data를 통해 모든 데이터에 접근이 가능하다.

2-1. Radix 관련 함수.

- search_tree() : radix tree에서 주어진 pointer가 속한 Span을 찾는다.
- add_one_node() : radix tree에 주어진 pointer가 속한 Span을 적절한 node에 추가한다.
- add_Span() : add_one_node()를 통해서 크기가 N*PAGE_SIZE인 Span object의 포인터를 N개의 위치에 추가를 한다.

2-2. small allocation 관련 함수

alloc_small_1, alloc_small_2, alloc_small_3는 모두 같은 함수로 작은 메모리의 allocation을 위해 정의된 함수이다. 3가지로 정의하게 된 이유는 Thread_cache에서 보면 알 수 있듯이 list를 3가지로 나눠서 정의했기 때문이다. 이 함수 안에서는 아래와 같은 순서로 allocation이 진행된다.

1. small object가 TC(Thread Cache)안에 남아있는가?
2. 없으면, central cache에서 불러오자. 그런데 CC(Central Cache)안에 small object가 있는가?
3. 없으면, CC의 Span에서 가져오자. 그런데 CC Span이 non-empty한가?
4. non-empty한 Span이 없으면 alloc_pages()를 통해서 새로운 Span을 가져오자, (100pages로 고정해두었다)
5. TC에 100개의 small object를 할당해준다.
6. TC에서 1개를 뽑아서 return한다.

위의 과정으로 진행되며, CC에 접근하는 모든 부분에는 spin lock이 걸려있다.

2-3. free 관련 함수.

merge_spans는 tc_free()에서 호출되는 함수로 large object deallocation중에 일어난다. 들어온 2개의 Span을 이어서 하나의 Span으로 합쳐 돌려준다.
tc_free()는 들어온 pointer를 free하는 함수이다.

1. 우선 들어온 pointer가 어디에 속하고 크기가 어떤지 알기 위해서 search_tree()를 통해 속한 Span을 확인한다.
2. 만약 Span에서 Size가 large object라고 나타내면, 다음과 같이 처리한다.
    a. 일단 Span의 바로 앞 주소, 그리고 바로 뒤 주소가 비어있는지 allocated되어있는지, 한번도 호출되지 않았는지를 search_tree()로 찾는다.
    b. 만약 비어있으면, merge_spans를 통해서 합성한다.
    c. 합성하면 Radix tree가 변경되기에, add_Span으로 tree를 갱신해준다.
3. 만약 Span에서 Size가 small object라고 나타내면, 다음과 같이 처리한다.
    a. 일단 현재 속한 TC를 확인한다.
    b. TC에 삽입한다
    c. tc_garbage_collect를 실행한다.

tc_garbage_collect는 TC에 특정 small object가 한계값 이상 있는지 확인하고, 그럴 경우 central cache로 migration을 하는 함수이다.
(아직 구현을 완료하지 않았지만, central cache에서 일정 한계값을 넘으면 Span으로 돌려주고, 이 Span이 fully non-empty가 되면 pagemap으로 돌리게 구현할 것이다.)

2-4. allocation관련 함수.
alloc_pages()는 num_page개의 page로 구성된 Span을 돌려주는 함수이다.

1. 일단 pagemap에 해당하는 Span이 있는지 탐색한다.
    a. 만약 더 큰 Span이 있다면 그 차이를 계산해 넣어준다. (이때 radix tree를 갱신해준다.)
    b. 255PAGES까지고 없으면, 256번째 pagemap을 탐색한다. 이때 Linked List는 앞쪽이 작은 object이고 뒤쪽이 큰 object이다.
2. pagemap에서 찾지 못하면 NULL이 그대로 유지되어있으니 이를 통해서 mmap()의 필요성을 판단한다.
    a. mmap()으로 할당한다.
3. size를 통해서 small object로 쪼개야 하는지, 그냥 return해도 되는지 판단하고, 쪼개야 하면 쪼개서 돌려준다.

tc_malloc()의 경우 입력을 받고서 init여부를 먼저 판단한다. 그 후 크기에 맞게 alloc_small들을 호출하거나, alloc_pages를 통해 large object를 할당해준다.

2-5 init함수
초기화를 하는 함수들이다. central init의 경우 meta_addr를 초기화 하는 부분이 있는데, 미리 100MB가량을 meta data, 이 Malloc 구조를 유지하기 위한 저장소로 미리 정의를 해둔다.

3-0 성능

gdb환경에서는 돌지만, 일반 환경에서는 작동을 하지 않는다.
이는 race condition으로 인한 error로 보이는데 그 이유는 아래와 같다.

1. stdout과 fflush로 일부러 속도를 늦출 경우 segfault가 나는 빈도가 줄었다.
2. 이전 과제에서 gdb에서 작동하지만, 그렇지 않았던 이유는 header file에 정의되지 않은 함수가 있어서였는데, 해당 문제의 경우 문제되는 함수가 호출될때 seg fault가 났지만, 지금의 경우 thread 700개가 동작하는 동안도 오류가 없던 적도 있기에 해당 문제는 아니다.

일단 해당 부분은 해결하려 하지만, 시간이 부족해 제출한다. 내일 중으로 개선을 시도해서 추가 제출할 예정이다.

4-0 컴파일
헤더파일과 spin lock의 사용을 위해서 -lpthread와 -pthread 옵션을 넣고서 컴파일을 해야한다.