[Procjet-3] A new page replacement

소프트웨어학과 201720707 나용성
소프트웨어학과 201921085 곽수정

- LRU management에 대한 이해

Linux kernel은 memory page를 active, inactive lru list를 이용해 관리한다. Active lru list에는 자주 쓰인 page를 inactive lru list에는 잘 안쓰이는 page들이 저장되어 memory가 부족할 땐 inactive list의 tail에서부터 page를 free 또는 swap시켜 free memory를 확보하며 이를 eviction 시킨다고 한다. 그러나 reclaim할 땐 eviction시킬 수 없는 상태의 Inactive page는 inactive list로 되돌아 가거나 active list로 옮겨질 수도 있다. 그리고 inactive list의 page는 reference됨에 따라 active list로 promotion될 수 있다. 또 active list에 있던 page도 메모리가 부족하면 최근에 접근되지 않은 page들은 inactive list로 demotion될 수 있다. 요약하자면 LRU management는 memory page가 접근빈도에 따라 그리고 메모리가 부족해짐에 따라 active, inactive list사이를 이동하거나 inactive list에서 eviction되며 active list에서 inactive list로 가는것을 demotion, inactive list의 page가 자주 접근됨에 따라 active list로 이동하는 것을 promotion, 그리고 inactive list에서 memory가 부족할 때 page를 eviction 시키는 과정을 reclaim이라고 하며 이 과정들은 모두 page의 reference bit가 설정됐는지 여부를 참조한다. promotion, demotion, reclaim을 수행하게 되는 중요함수들은 다음과 같다.

- **page_referenced** : 입력된 page를 reference pte의 갯수를 return하며 page의 referenced bit를 clear한다.
- **shrink_page_list** : 입력된 inactive list에 reclaim을 수행하는 함수이다. shrink_page_list는 입력된 inactive list의 page들을 iterate하면서 referenced bit값을 포함한 다양한 조건에 따라 page를 eviction시킬 지 또는 inactive list에 keep할 지 또는 activate 시켜 active list에 추가되도록 할 지 판별하여 처리한다. eviction된 page의 개수는 nr_reclaimed에 저장되어 return되며 activate된 page의 개수는 로컬변수 pgactivate에 저장된다.
- **shrink_inactive_list** : 메모리가 부족할 때 inactive list에 실행되며 inactive list로 shrink_page_list를 실행하는 wrapper 함수이다.
- **reclaim_pages** : reclaim에 관여하는 함수로 shrink_page_list를 사용해 reclaim을 수행한다.

shrink_active_list : 메모리가 부족할 때 실행되어 active list의 page를 iterate하면서 page_referenced함수를 사용해 reference되지 않은 page들을 inactive list로 demotion시키는 demotion을 담당하는 함수이다. shrink_active_list는 demotion된 active page의 개수가 저장되는 로컬변수 nr_deactivate를 갖고있다.

- **mark_page_accessed** : 위에서 설명한 LRU management에서 promotion을 담당하는 함수로 page를 access할 때 access된 page에 대해 수행되어 referenced bit를 set하거나 이미 set 되어 있는 inactive page면 active list로 promotion 시키는 일을 수행하는 함수이다. mark_page_accessed가 수행되면 inactive, active page에 각각 다음과 같은 변화가 발생한다.

inactive, **unreferenced** page -> inactive, **referenced** page
inactive, referenced page -> **active, unreferenced** page (promotion된 page에 해당한다.)
active, **unreferenced** page -> active, **referenced** page

- Describe how to monitor memory management statistics

1, 2번 통계는 각각 anon, file page의 active, inactive list에 page가 몇개 있는지(1번 요구사항)와 그 중 reference bit가 set된 page는 몇개인지(2번 요구사항)이다. 이 두 통계를 내기 위해서는 각 lru list의 head를 이용해 iterate해보면 iterate되는 횟수만큼 page가 저장된 것으로 생각할 수 있으므로 iterate할 때 마다 page count를 1 increase하고 검색된 page entry의 reference bit를 확인하여 그 값을 reference bit count에 더하는 과정을 반복하여 각 list의 page개수와 reference bit가 set된 page의 개수를 구할 수 있을 것이다. 그래서 우리는 lru list가 시스템에서 어떤 곳에서 관리되고 있는지 확인해 보았다. 우리가 실습에 사용하는 linux kernel버전은 pglist_data라는 구조체에 lruvec 구조체 멤버를 갖고있고 이 lruvec 구조체에 각 lru list의 head의 배열을 갖고있다는 점을 확인했다. pglist_data는 NUMA machine에서 각 NUMA node마다 존재하여 NUMA node의 layout을 저장하고 있는 구조체라고 한다. mmzones.h에는 시스템에 존재하는 모든 pglist_data를 iterate할 수 있는 매크로인 for_each_online_pgdat를 제공하고 있다. 물론 우리가 사용하는 virtual machine은 멀티 NUMA를 구현하고 있지 않으므로 하나의 pglist_data만 사용하게 되지만 scalability를 위해 for_each_online_pgdat를 이용해 online상태의 pglist_data를 iterate해 현재 pglist_data의 lruvec에 접근해 각 list_head를 이용해 iterate를 돌며 위에서 설명한 방법으로 통계를 내어 각각의 pglist_data에 대해 lru list에서 page 개수와 referenced page의 개수를 구해 출력하는 과정을 시스템 콜이 호출됐을 때 수행하도록 하여 구현했다.

먼저 3, 4번 통계에서 필요한 값들은 num_active_to_inactive, num_inactive_to_active, num_evicted_from_inactive 이렇게 세 가지 인데 이 세 값은 시스템이 부팅될 때 0으로 초기화되어야 하므로 이를 수행하는 함수 initialize_lru_statistics함수를 정의하여 start_kernel 함수에서 실행하도록 하여 시스템 부팅 시 위의 세 값을 초기화 하도록 하였다.

3. cumulative number of pages moved from active list to inactive list

=> active list에서 inactive list로 page가 이동하는 경우는 shrink_active_list에 의해 demotion될 page로 분류되는 경우이며 이 때 shrink_active_list에서 사용하는 로컬변수 nr_deactivate에 demotion된 page의 개수가 저장됨을 위에서 설명했다.

그러므로 shrink_active_list가 호출될 때 마다 nr_deactivate의 값을 num_active_to_inactive에 더해서 cumulative number of pages moved from active list to inactive list의 통계를 내고 시스템 콜이 호출될 때 num_active_to_inactive의 값을 출력하도록 했다.

cumulative number of pages moved from inactive list to active list

=> inactive list에서 active list로 page가 이동하는 경우는 inactive_list에 reclaim이 발생했을 때 page가 active list에 이동해야 된다고 분류되는 경우와 referenced bit가 set된 inactive page에 mark_page_accessed가 호출돼 promotion되어 active list로 이동하는 경우가 있다. shrink_page_list는 pgactivate에 activate시킬 page의 개수를 저장함을 위에서 설명했으므로 shrink_page_list가 종료되기 전에 num_inactive_to_active에 pgactivate의 값을 더하고 referenced bit가 set된 inactive page에 mark_page_accessed가 호출되어 active list로 promotion이 일어난 경우에도 num_inactive_to_active의 값을 1 increase시켜 cumulative number of pages moved from inactive list to active list의 통계를 내고 시스템 콜이 호출될 때 num_inactive_to_active의 값을 출력하도록 했다.

4. cumulative number of pages evicted from inactive list

=> inactive list에서 page가 eviction되는 경우는 inactive_list에 reclaim이 발생해 shrink_page_list 함수에 의해 eviction 될 page로 분류되는 경우이며 shrink_page_list는 eviction된 page의 개수를 nr_reclaimed에 저장해 return시킴을 위에서 설명했다. 그러므로 shrink_page_list가 nr_reclaimed를 return하기 직전에 num_evicted_from_inactive값에 nr_reclaimed의 값을 더해서 cumulative number of pages moved from inactive list to active list의 통계를 내고 시스템 콜이 호출될 때 num_evicted_from_inactive의 값을 출력하도록 했다.

위의 모든 통계는 하나의 시스템 콜에서 동시에 출력되도록 하여 통계간의 상관관계가 일치하는지를 확인할 수 있도록 했다.

- Describe how to implement your replacement algorithm

먼저 part 2: Implementation의 1번 요구사항을 충족하기 위해 mm_types.h파일에 struct page가 정의되어 있는데 이 구조체의 마지막 위치에 ref_counter라는 atomic_t변수를 추가했다. 마지막 위치에 추가한 이유는 struct page의 멤버를 인덱스를 이용해 접근하는 함수도 있기 때문에 중간에 기존의 멤버들 사이에 위치하게 되면 에러가 발생할 수 있기 때문이다. atomic_t로 선언한 이유는 ref_counter를 Increase, decrease시킬 때 race condition이 발생하지 않도록 하기위해 atomic_t변수를 사용했다.

그 다음 2번 요구사항을 충족하기 위해 reference bit의 값에 따라 evict to destination이 수행되던 것을 ref_counter의 값에 따라 수행되는 것으로 변경해야 하는데 먼저 try_to_free_pages가 실행되면

try_to_free_pages -> do_try_to_free_pages -> shrink_zones -> shrink_node -> shrink_node_memcgs -> shrink_lruvec->shrink_list 순서로 함수가 호출되며 shrink_list는 list_head가 active list이냐 inactive list에 따라 shrink_active_list 또는 shrink_inactive_list를 호출하며 shrink_inactive_list는 shrink_page_list를 호출한다. 그러므로 try_to_free_pages가 실행됐을 때 scan pages가 실제로 수행되는 함수는 shrink_active_list와 shrink_page_list이다. 다른 순서로 위의 두 함수에 도달하기도 하지만 중요한 것은 결국 scan pages를 수행하고 demotion 또는 reclaim시키는 것은 위의 두 함수라는 사실이다. 그러므로 위의 두 함수에서 reference bit를 이용해 demotion또는 reclaim여부를 판별하던 것을 ref_counter의 값에 따라 판별하도록 변경하면 될것이라 생각했다.

먼저 shrink_active_list에서는 기존에는 active list를 iterate하면서 각 page에 page_referenced함수를 사용해 return 값이 양수일 땐 reference중인 pte가 존재하는 것이므로 active에 남기기를 시도하는 방식으로 동작했다. 이 때 page_referenced함수의 return값에 따라 branch되던것을 return값을 page의 ref_counter에 더하고 ref_counter의 값이 0이 아니면 active에 남기기를 시도하고 ref_counter를 1 감소시키는 방식으로 변경했다. 만약 ref_counter의 값이 0이라면 최근에 이 page에 대한 접근이 없었던 것이므로 inactive list로 demotion 시키게 된다.

shrink_page_list에서도 referenced bit에 의한 판별이 존재한다. referenced bit에 의한 판별은 page_referenced함수를 사용하는 page_check_references에서 일어나 판별 결과를 return하고 shrink_page_list는 return되는 enum값 네가지 PAGEREF_ACTIVATE, PAGEREF_KEEP, PAGEREF_RECLAIM, PAGEREF_RECLAIM_CLEAN에 따라 PAGEREF_RECLAIM, PAGEREF_RECLAIM_CLEAN일 땐 eviction을 더 시도하게 되고 PAGEREF_ACTIVATE일 땐 active list에 추가하고 PAGEREF_KEEP일 땐 inactive list에 남기게 된다. 이 때 page_chek_references함수는 shrink_active_list에서와 유사하게 page_referenced함수를 page에 사용해 return값이 양수일 땐 page를 reference중인 pte가 존재하는 것이므로 eviction되지 않도록 판별하고 0일 땐 eviction되도록 판별결과를 return한다. 이 때 shrink_active_list에서의 변경점과 유사하게 page_referenced함수의 return값을 page의 ref_counter에 add하고 page_referenced함수의 return값에 따라 branch되던

것을 page의 ref_counter의 값을 읽어 양수일 땐 eviction되지 않도록 판별하고 ref_counter를 1 감소시키고 0일 땐 eviction되도록 판별하도록 변경해 referenced bit에 의한 판별을 ref_counter에 의한 판별로 replace시켰다.
그 다음 3번 요구사항을 충족시키기 위해 vmscan.c에 timer handler함수인 ref_counter_timer_handler를 정의했다. ref_counter_timer_handler는 part1의 1, 2번 통계를 구하기 위해 모든 page를 iterate하던 것과 같은 방식으로 page를 iterate하고 각 iterate에서 page의 referenced bit을 test_and_clear_bit을 이용해 0으로 초기화 시키면서 기존값을 읽어서 기존값이 1일 땐 ref_counter를 1 increase시키도록 했다. 그리고 모든 page를 iterate한 뒤에 mod_timer로 다시 timer를 scheduling해서 ref_counter_timer_handler가 주기적으로 호출될 수 있게 했다. 그리고 vmscan.c에 ref_counter의 최댓값을 define을 이용해 정의하고 모든 increase될 수 있는 경우에 최댓값을 넘지 않도록 확인하는 과정을 추가하여 4번 요구사항까지 충족하여 기존의 second chance LRU-approximation algorithm을 counter-based clock algorithm으로 완전히 replace시켰다.

- Evaluate comparison between default and your new replacement memory management by testing user level application on your kernel

counter based clock algorithm을 사용하는 kernel의 테스트 결과이다.
테스트는 다음의 순서대로 수행하였다.

```
root@sce394_vm:~/project/user# ./test
[   87.172377] ============================================================
[   87.176414] anon inactive count : 1795, anon inactive ref bits 0
[   87.176752] anon active count : 45, anon active ref bits 1
[   87.177128] file inactive count : 11380, file inactive ref bits 0
[   87.177330] file active count : 2981, file active ref bits 194
[   87.177520] num_active_to_inactive : 0, num_inactive_to_active : 3584
[   87.177714] num_evicted_from_inactive : 0
[   87.177931] ============================================================
INIT: Id "S1" respawning too fast: disabled for 5 minutes
[  114.287936] ============================================================
[  114.288237] anon inactive count : 1742, anon inactive ref bits 0
[  114.288395] anon active count : 44, anon active ref bits 0
[  114.288532] file inactive count : 84154, file inactive ref bits 242
[  114.288682] file active count : 2985, file active ref bits 0
[  114.288924] num_active_to_inactive : 0, num_inactive_to_active : 3596
[  114.289091] num_evicted_from_inactive : 0
[  114.289196] ============================================================
```

먼저 400mb가량의 파일을 fopen하는 경우를 실행해보니 file inactive count가 매우 많이 늘었다.

```
root@sce394_vm:~/project/user# ./test2
[  335.963994] ============================================================
[  335.964282] anon inactive count : 1737, anon inactive ref bits 0
[  335.964467] anon active count : 45, anon active ref bits 1
[  335.964719] file inactive count : 105111, file inactive ref bits 0
[  335.965189] file active count : 5629, file active ref bits 194
[  335.965385] num_active_to_inactive : 0, num_inactive_to_active : 7115
[  335.965680] num_evicted_from_inactive : 55604
[  335.966007] ============================================================
[  336.180807] ============================================================
[  336.181108] anon inactive count : 8134, anon inactive ref bits 0
[  336.181293] anon active count : 46, anon active ref bits 0
[  336.181502] file inactive count : 99986, file inactive ref bits 26338
[  336.181781] file active count : 5629, file active ref bits 0
[  336.182133] num_active_to_inactive : 0, num_inactive_to_active : 7116
[  336.182431] num_evicted_from_inactive : 60729
[  336.182705] ============================================================
```

그 다음 malloc을 이용해 anon page를 인위적으로 생성했더니 anon inactive count가 늘어나고 num_evicted_from_inactive가 증가하면서 file inactive count가 감소하였다.

```
root@sce394_vm:~/project/user# ./test2 20
[  349.790656] ============================================================
[  349.791010] anon inactive count : 1737, anon inactive ref bits 0
[  349.791203] anon active count : 46, anon active ref bits 1
[  349.791363] file inactive count : 99986, file inactive ref bits 0
[  349.791534] file active count : 5631, file active ref bits 193
[  349.791700] num_active_to_inactive : 0, num_inactive_to_active : 7122
[  349.791970] num_evicted_from_inactive : 60729
[  349.792119] ============================================================
[  350.123168] ============================================================
[  350.123443] anon inactive count : 27331, anon inactive ref bits 0
[  350.123633] anon active count : 46, anon active ref bits 0
[  350.123818] file inactive count : 81275, file inactive ref bits 0
[  350.124090] file active count : 5631, file active ref bits 0
[  350.124257] num_active_to_inactive : 0, num_inactive_to_active : 7122
[  350.124413] num_evicted_from_inactive : 79440
[  350.124522] ============================================================
```

더 많은 malloc을 수행하자 더 많은 양의 file inactive count가 줄었고 num_evicted_from_inactive역시 같이 증가했다.

```
root@sce394_vm:~/project/user# ./test2 30
[  357.594718] ============================================================
[  357.595081] anon inactive count : 1737, anon inactive ref bits 0
[  357.595265] anon active count : 46, anon active ref bits 1
[  357.595430] file inactive count : 80980, file inactive ref bits 0
[  357.595613] file active count : 5639, file active ref bits 193
[  357.595887] num_active_to_inactive : 0, num_inactive_to_active : 7127
[  357.596087] num_evicted_from_inactive : 79735
[  357.596216] ============================================================
[  358.604163] ============================================================
[  358.604463] anon inactive count : 40141, anon inactive ref bits 0
[  358.604632] anon active count : 46, anon active ref bits 0
[  358.604763] file inactive count : 628, file inactive ref bits 0
[  358.605049] file active count : 4302, file active ref bits 0
[  358.605209] num_active_to_inactive : 18, num_inactive_to_active : 7321
[  358.605403] num_evicted_from_inactive : 82242
[  358.605531] ============================================================
```

한번 더 더욱 많은 양의 malloc을 수행하자 이번엔 사용한 메모리의 양이 총 메모리를 초과하여 거의 모든 file inactive list의 page들이 free되었다.

알고리즘이 정상적으로 작동되는 사실은 referenced bit을 주기적으로 ref_counter에 추가하는 timer가 정상적으로 작동해 reference bit count가 거의 항상 0이고 이 때문에 mark_page_accessed에 의해 active list로 넘어가는 page가 거의 없다는 점을 통해 알 수 있었다.