Project 3 Report

소프트웨어학과 201920739 박지윤
소프트웨어학과 201920784 강윤지

1. Describe how to monitor memory management statistics

리눅스에서 list를 이용하여 LRU를 구현한다는 점을 이용하여 INACTIVE_ANON, ACTIVE_ANON, INACTIVE_FILE, ACTIVE_FILE의 수를 구하였다. 우리가 사용하는 리눅스의 버전은 5.15.4인데, memory cgroup의 node별로 LRU 리스트를 관리한다. 따라서 root의 memory cgroup을 가져와서 mem_cgroup_lruvec를 통해 관리되고 있는 lruvec(LRU 리스트)를 얻어오고, 이 구조체의 lists에 접근하여 list_for_each_entry를 통해 list에 있는 page들을 순회하면서 각 page의 수를 구하였다.

Reference bit가 set된 active/inactive page의 수 또한 위와 같이 lruvec을 이용하였다. 이때도 위와 같이 list_for_each_entry를 통해 list에 있는 page들을 순회하는데, 순회할 때 page_referenced라는 macro를 사용하였다. 이 macro는 reference bit가 설정이 되지 않았으면 0을, 설정이 되었으면 1이상의 값을 return하기 때문에 해당 macro의 return값이 0보다 큰 경우 reference bit가 set된 page를 세는 counter의 값을 1 증가시켜주어 원하는 결과를 구할 수 있었다.

Demotion된 page 수와 promotion된 page의 수는 this_cpu_read에 원하는 argument를 pass해주어 구하였다. Page가 active에서 inactive로 변하거나, inactive에서 active로 변하는 경우, count_vm_counts 라는 함수를 이용하여 vm_event_states에서 인자로 넘겨받은 index에 demotion 혹은 promotion된 page의 수만큼 increment해준다. 이 점을 이용하여 vm_event_states에 원하는 인자에 접근하면 얼마만큼의 page가 demotion 혹은 promotion되었는지 알 수 있기 때문에, this_cpu_read()를 이용하여 원하는 값을 읽어올 수 있었다.

Demotion이나 promotion과 비슷하게, evict도 page가 reclaim되는 경우, count_vm_counts 라는 함수를 이용하여 vm_event_states에서 인자로 넘겨받은 index에 evict된 page의 수만큼 increment해준다. 이 점을 이용하여 vm_event_states에 원하는 인자(이 경우 PGSTEAL_ANON, PGSTEAL_FILE)에 접근하면 얼마만큼의 page가 demotion 혹은 promotion되었는지 알 수 있기 때문에, this_cpu_read()를 이용하여 원하는 결과를 얻을 수 있었다.

2. Describe how to implement your replacement algorithm

LRU가 가장 오랫동안 참조되지 않은 page를 replacement하는 것이라면, 이번에 구현한 counting-based clock algorithm은 가장 참조횟수가 적은 page를 replacement 하는 것이다. 이를 위해선 각 page마다 참조횟수를 counting 하는 변수가 필요했다. 각 page마다 필요한 것이기에 struct page에 counter이라는 변수를 추가해 주었으며, linux kernel 내부에서 선언된 각 변수는 선언된 순서를 지켜주어야 하기 때문에, 새로 추가될 counter 변수는 struct의 제일 마지막에 위치시켰다.

모든 page를 확인해야 하므로, list_for_eache_entry를 통해 각 list를 순회하며 위에서 reference bit를 확인하기 위해 사용한 page_referenced라는 macro를 이용하여 해당 page가 참조된 적이 있는지 확인하였다. 이 값이 양수가 나온다면 참조된 적이 있었다는 것을 나타내기에 참조 횟수를 나타내는 counter의 값을 page_referenced 함수의 return값만큼 증가시켜준다. 이후, 다음에 list를 순회할 때 잘못된 값이 들어오지 않도록 확인한 page의 reference bit를 초기화 해주는 작업을 ClearPageReferenced를 통해 수행하였다.

만약 page reclaim이 발생해야 하는 경우, try_to_free_pages()가 호출이 되는데 여기선 앞에서 설정한 각 page의 counter 값들을 list를 순회하며 확인한다. counter의 값이 참조가 된 횟수를 말하는 것이니, counter 가 0이라면 한번도 참조가 되지 않았음을 의미하며 앞으로 참조가 될 가능성이 적다고 판단되기에 이 page를 evict시키도록 한다. 여기서 2가지의 경우로 나뉘게 되는데, 만약 현재 page가 active list에 위치한다면 inactive list의 tail로 evict 시키게 되고, 현재 page가 inactive list에 위치한다면 free_pages의 tail로 evict 시키고 page를 free해주었다. 만약 counter가 0이 아니라면, 나중에도 참조가 될 가능성이 있다 판단해 list의 제일 마지막에 다시 넣어줌과 동시에 참조 횟수인 counter를 하나 감소시켜 주었다.

reference counter를 증가시키기 위해서는 주기적으로 page들을 scan하여 page가 참조가 되었는지 안되었는지를 계속 확인해야한다. 이를 위해 timer handler를 설정하여 3초마다 한번씩 reference bit를 확인하고, counter에 더하고, 다시 reference bit를 초기화 하는 과정이 일어나도록 하여 try_to_free_pages에서 확인하는 counter의 값을 주기적으로 변동시키도록 하였다.

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

1) The current number of pages in the current LRU page lists

[표1] LRU를 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
LRU_ACTIVE_FILE	4835	4811	-24
LRU_INACTIVE_FILE	25918	9646	-16,272
LRU_ACTIVE_ANON	49	49	0
LRU_INACTIVE_ANON	1797	99279	+97,482

[표2] Counter-based clock algorithm을 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
LRU_ACTIVE_FILE	8692	7934	-758
LRU_INACTIVE_FILE	22084	7644	-14,440
LRU_ACTIVE_ANON	47	48	+1
LRU_INACTIVE_ANON	1788	98395	+96,607

같은 test 파일로 test를 진행했기 때문에, memory를 allocation하기 전과 memory를 allocation을 한 후 각 page list의 변화량에는 차이가 크지 않다. 그러나, LRU_ACTIVE_FILE의 경우 LRU를 page replacement algorithm으로 사용했을 때에 비해 Counter-based clock algorithm이 약 4000만큼 크고, LRU_INACTIVE_FILE은 약 4000만큼 작은 것을 확인할 수 있는데, 이 차이는 리눅스 커널이 부팅하면서 기본적으로 사용하는 메모리들이 있을텐데, page replacement algorithm이 달라지면서 page가 하나라도 reference가 되어있으면 inactive로 이동하지 않기 때문에 이러한 차이를 낳은 것으로 보인다.

2) **The current number of pages those reference bits are set in active/inactive list**

[표3] LRU를 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
LRU_ACTIVE_FILE	1598	148	-1,450
LRU_INACTIVE_FILE	1207	2	-1,205
LRU_ACTIVE_ANON	20	2	-18
LRU_INACTIVE_ANON	1457	97709	+96,252

[표4] Counter-based clock algorithm을 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
LRU_ACTIVE_FILE	361	14	-347

LRU_INACTIVE_FILE	0	40	+40
LRU_ACTIVE_ANON	2	16	+14
LRU_INACTIVE_ANON	104	19264	+19,160

Counter-based clock algorithm이 LRU algorithm에 비해 reference bit가 set된 page의 수가 확연하게 적은 것을 확인할 수 있는데, 이는 Counter-based clock algorithm이 timer를 통해 주기적으로 scan함수를 부르면서 reference bit에 설정된 수만큼 reference counter에 더해주고, 원래의 reference bit에 setting된 값은 0으로 설정해주기 때문인 것으로 보인다.

3) The cumulative number of pages moved from active list to inactive list and vice versa

[표5] LRU를 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
Active to inactive	0	100	+100
Inactive to active	3580	3580	0

[표6] Counter-based clock algorithm을 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
Active to inactive	0	853	+853
Inactive to active	7455	7456	+1

Counter-based clock algorithm이 LRU algorithm에 비해 active에서 inactive list로 이동한 page의 수가 많다. 이는 counter-based clock algorithm에서, try_to_free_pages() 함수가 호출될 때, reference counter가 0이면 active list에서 inactive list로 가도록 구현되어있기 때문인 것으로 보인다. 또한, Counter-based clock algorithm의 경우 memory를 allocation하기 전에 LRU algorithm에 비해 Inactive에서 active로 간 page의 수도 많은 것을 확인할 수 있는데, linux kernel이 부팅하면서 여러 page를 memory에 올리는데, 처음에 inactive list에 올리게 된다. 그러나 LRU algorithm과 달리 counter-based clock algorithm은 try_to_free_pages()에서 LRU_ACTIVE_FILE, LRU_INACTIVE_FILE, LRU_ACTIVE_ANON, LRU_INACTIVE_ANON에 접근하기 때문에 짧은 시간 내에 refault가 나서 active list로 이동한 page가 많았던 것으로 보인다.

4) The cumulative number of pages evicted from inactive list

[표7] LRU를 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
Evicted pages (ANON)	0	169	+169
Evicted pages (FILE)	0	16,372	+16,372

[표8] Counter-based clock algorithm을 page replacement algorithm으로 사용한 경우

	Before memory allocation	After memory allocation	변화량
Evicted pages (ANON)	0	1134	+1134
Evicted pages (FILE)	0	15,289	+15,289

Counter-based clock algorithm을 사용한 경우, ANON page에서 LRU algorithm에 비해 변화량이 큰 것을 확인할 수 있는데, 이는 user level application에서 메모리를 동적할당하였기 때문에 주로 file list page보다는 anon list page를 사용하여서 그런 것으로 보인다.