Project3 Final report

20210084 김지민, 20210216 양준영

1. Frame Table
   A. Solution: frame.c, frame.h 파일 새로 생성.
      i. struct frame
         1. void* kvaddr: 페이지 주소와 매핑될 가상주소
         2. void* upage: 이 프레임에 할당될 페이지 주소
         3. struct list_elem f_elem: frame_list에 들어갈 리스트 요소
         4. struct thread* f_t: 해당 프레임을 할당한 thread.
         5. bool not_evict: true면 해당 프레임이 evict 알고리즘에 의해 쫓겨나지 않도록 함.
      ii. void init_frame () -> init.c의 메인함수에서 호출한다.
         1. frame.c의 함수들은 모두 critical section에서 시행되어야 하므로 여기에 사용될 lock 'f_lock'을 초기화해준다.
         2. frame_list를 초기화
         3. clock_algo에서 사용할 clock 포인터를 NULL로 설정
      iii. struct frame* allocate_frame
         1. palloc_get_page 함수를 호출해 새로운 프레임 주소를 할당받는다.
         2. 만약 이 값이 NULL이라면 clock_algo 함수를 호출하여 프레임 하나를 쫓아낸 후 다시 할당받는다.
         3. 이후 frame structure 하나를 동적할당 받고 이 프레임의 멤버에 각각 알맞은 값을 할당한다. (kvaddr 멤버에 새로 할당받은 프레임 주소를 저장하고 f_t 멤버에 현재 쓰레드를 저장한다든지 등등)
         4. 마지막으로 frame_list에 이 프레임을 추가하고 해당 프레임을 반환

한다.

iv. void deallocate_frame_with_lock

1. deallocate_frame 함수 호출 앞뒤로 lock_acquire와 lock_release를 하는 함수.

v. void deallocate_frame

1. deallocate_frame_with_lock과 따로 선언하는 이유는 frame.c에서 이미 lock을 acquire한 상태에서 frame을 deallocate해야하는 경우가 있기 때문.
2. 인수로 받은 frame을 frame_list에서 삭제, 이후 이 frame의 kvaddr을 palloc_free_page 함수에 넣어 호출함으로써 완전히 해제.
3. 마지막으로 frame구조체까지 free.

vi. void clock_algo

1. clock algorithm을 이용하여 evict할 frame을 고른다.
2. 우선 clock이 가리키는 frame부터 시작하여 frame_list의 끝 부분까지 돈다. 이때 현재 frame의 not_evict가 true라면 고르면 안되므로 다음 frame으로 넘어간다.
3. 현재 frame의 pagedir_is_accessed(pagedir, e->upage)가 true라면, pagedir_set_accessed(pagedir, e->upage, false)를 호출함으로써 access여부를 false로 바꾸고 넘어간다.
4. 만약 pagedir_is_accessed(pagedir, e->upage)가 false라면 access 여부가 false라는 것이므로 이 frame을 victim으로 고른다.
5. Victim이 안 나온다면 이번엔 list_begin부터 list_end까지 같은 과정을 반복한다.
6. 이후 이 victim frame에 대한 마무리 과정을, evict_frame 함수로 처리한다.

vii. void evict_frame (struct frame* f)

1. 인수로 받은 frame를 내쫓는 과정을 담당.

2. 우선 frame에서 내쫓기고 swap 상태로 넘어가는 것이므로 vm_swap_out함수와 supp_set_swap함수를 호출한다.
3. 이후 pagedir_is_dirty함수를 호출함으로써 만약 true면 해당 페이지가 수정되었다는 의미이므로 supp_set_dirty 함수를 호출하여 이 프레임을 할당했던 쓰레드의 supplemental page table에도 이 사실을 반영한다.
4. pagedir_clear_page을 호출하여 해당 쓰레드의 페이지 디렉토리에 해당 페이지를 not present로 mark한다.
5. 마지막으로 deallocate_frame함수를 호출하여 해당 프레임을 해제한다.

viii. void frame_unset_not_evict

1. 인수로 받은 frame의 not_evict을 false로 할당.

ix. void frame_set_not_evict

1. 인수로 받은 frame의 not_evict을 true로 할당.

B. Discussion

i. 기존 디자인과의 비교: 인수로 upage를 받으면 해당되는 frame을 찾는 search_frame 함수를 구현하여 deallocate_frame 함수에서 사용하려 했으나 deallocate_frame의 구조 변경으로 인해 search_frame이 필요 없어졌다.

ii. 어려웠던 점: 처음에 deallocate_frame 함수 같은 경우 인수로 void* kvaddr를 받고, search_frame 함수로 해당 frame을 찾으려 했다. 그러나 오류가 많이 뜨는 바람에 search_frame 함수를 삭제하고 deallocate_frame 함수가 인수로 struct frame을 받도록 함으로써 frame을 찾는 과정을 없앴다.

2. Lazy loading

A. Solution

i. 기존 process.c의 load_segment 함수에서는 while문 안에서 페이지를 할당 받아 파일을 읽고 메모리에 바로 올렸었는데, 이 과정을 삭제한다.

ii. 대신 supp_insert_page함수를 호출하여 프레임 할당은 하지 않고 해당 페이지에 대한 정보만 supplemental page table에 저장한다.

iii. 이후 lazy loading에 의한 페이지 폴트가 발생하면 exception.c의 page_fault함수에서 load_page함수를 호출하여 프레임을 할당하고 다시 리턴한다.

iv. setup_stack 함수에서 할당 받은 페이지를 supplemental page table에 넣는 작업이 필요하다. Setup_stack함수에서 호출하는 intall_page 함수를 수정한다. Install_page에서 supp_insert_page함수를 호출함으로써 해당 페이지를 supplemental page table에 추가한다. 추가될 때, 엔트리의 status가 1('프레임에 할당'이라는 뜻)이 되도록 한다.

B. Discussion.

i. 기존 디자인과의 비교: 기존 디자인 보고서에서는 setup_stack에서 할당한 페이지를 supplemental page table에 추가하는 과정이 생략되어 있었다. 이를 구현하기 위해 install_page 함수에 supp_insert_page 함수를 호출하는 과정을 추가하였다. 또한 frame의 not_evict 값을 set, unset하는 함수를 구현하였다.

ii. 어려웠던 점: 바로 위에서 말했듯이 setup_stack에서 할당한 페이지 또한 supt에 추가했어야 했는데 이를 계속 놓쳤었다.

3. Supplemental Page table

A. Solution: page.c, page.h 새로 생성

i. struct supplemental_page_table

1. struct hash h_supp: hash 테이블을 멤버로 갖는다.

ii. struct supp_entry: supplemental_page_table의 각 엔트리

1. upage, kvaddr: 페이지, 프레임 주소

2. struct frame*f: 해당 페이지와 연결된 frame 구조체.

3. Struct hash_elem elem: 해시 테이블을 이용하므로 hash 함수에서 사용할 수 있도록 해시 요소 추가.

4. Dirty: 수정되었는지 여부를 나타냄.
5. Swap_idx: swap table에서의 index
6. File, offset, read_bytes, zero_bytes, write는 file로부터 온 페이지인 경우 필요한 멤버들.

iii. struct supplemental_page_table* supp_create

1. malloc으로 supplemental_page_table 하나 동적할당한 후, hash_init 함수로 h_supp 멤버를 초기화한다. 이때, hash_init에 들어갈 function 2개가 필요하다.
2. unsigned hash_function: 페이지 주소 즉, upage를 기준으로 해시.
3. bool less_function: 두 element의 upage 값을 비교하여 그 결과를 반환한다.

iv. void supp_destroy

1. 인수로 받은 supp page table을 destroy한다. hash_destroy함수로 supt의 h_supp를 free시킨 후 마지막으로 supt도 free시킨다.
2. 이때 hash_destroy함수에 넣을 destroy_function을 구현한다.
3. void destroy_function: 인수로 받은 hash_elem *elem을 가진 supp_entry를 hash_entry 함수로 찾은 후 이 엔트리를 free시킨다.

v. bool supp_insert_page

1. 인수로 받은 page에 대한 정보를 새로운 supp_entry에 저장한 후 해시 테이블에 추가하는 함수다.
2. 우선 supp_entry* spte 하나를 동적할당 받는다.
3. spte의 upage, kvaddr, status는 각각 인수로 받은 것으로 할당, dirty는 false로 설정한다.
4. 이때, spte->status가 1이면 해당 페이지는 프레임에 매핑되어있다는 의미이므로 swap_idx는 -1로, f (frame 구조체)도 인수로 받은 것으로 할당한다(1이 아닌 경우엔 f가 할당되어 있지 않을 것).
5. 만약 status가 3이라면 페이지가 file로부터 왔다는 의미이므로 file

과 관련된 멤버들 file, offset, read_bytes, zero_bytes, write를 할당한다.

6. 멤버 저장이 끝나면 hash_insert로 해당 엔트리를 추가한다.

vi. struct supp_entry* find_supp_entry

1. 인수로 supt과 upage를 받으면 upage를 멤버로 갖는 supp_entry를 반환하는 함수다.
2. 우선 hash_find 함수를 호출하여 supt의 h_supp에서 upage를 요소로 갖는 hash_elem h을 찾는다.
3. hash_entry 함수를 호출함으로써 해당 h를 멤버로 갖는 supp_entry를 찾고 이를 반환한다.

vii. void supp_set_swap

1. 인수로 받은 page에 대해 status를 swap 상태로 바꾸고 swap_idx를 할당하는 함수다.
2. 우선 find_supp_entry함수로 해당 page를 가지는 supp_entry* spte를 찾는다.
3. 이후, 해당 spte의 status를 2(swapped 상태를 의미)로, addr값은 NULL로(이제 프레임에 할당되어 있지 않으므로), swap_idx은 넘겨받은 인수 값으로 바꾼다.

viii. void supp_set_dirty

1. fine_supp_entry 함수를 호출하여 인수로 받은 page를 가진 spte를 찾는다.
2. 해당 spte의 dirty를 true로 바꾼다.

ix. bool load_page

1. 인수로 받은 upage를 frame에 매핑하여 load하는 함수다.
2. Find_supp_entry 함수로 해당 page를 가지는 spte를 찾는다.
3. Spte의 status가 1인 경우엔 이미 load되어 있을 테니 그대로 return true;하고 나머지 경우에 대해 처리해야 한다.

4. 우선 새로운 frame을 allocate_frame 함수로 할당한다.
5. spte -> status가 0인 경우, zero page를 의미하므로 새로운 frame을 PGSIZE만큼 0으로 채운다.
6. Status가 2인 경우 swapped 상태를 의미하므로 vm_swap_in 함수를 호출한다.
7. 마지막으로 3인 경우 file로부터 읽어와야 한다.
8. File_seek와 file_read함수로 원하는 위치에서 원하는 크기만큼 읽은 다음 나머지 공간은 0으로 채운다.
9. 새롭게 frame에 할당한 것이므로 pagedir_set_dirty 함수를 호출하여 dirty bit를 false로 설정한다.
10. Spte의 kvaddr은 새롭게 할당받은 프레임의 주소, status는 1로 (frame 상태를 의미), f는 새롭게 할당한 frame 구조체로 저장한다.

x. bool supp_unmap

1. 파일이 매핑된 메모리 영역이 해제될 때 그 영역에 있던 page 내용을 다시 file에 반영하는 함수다. Status가 1, 2인 경우만 보면 된다.
2. Status가 1인 경우(frame에 매핑된 경우): frame_set_not_evict 함수를 호출함으로써 작업이 마무리되기 전까지는 해당 frame이 evict 되지 않도록 한다. 이후 dirty 값을 조사하여 해당되는 경우에만 file_write_at 함수를 호출한다. 그 다음 deallocate_frame_with_lock 함수와 pagedir_clear_page 함수를 호출함으로써 frame을 해제하고 해당 페이지를 not_present로 마크한다.
3. 2인 경우(swapped): 만약 dirty가 true라면, palloc_get_page(0)를 호출하여 임시로 페이지를 할당받은 후 vm_swap_in 함수를 호출하여 그 공간에 해당 swap_idx에 있는 내용을 저장한다. 이후 file_write_at 함수를 호출함으로써 그 내용을 file에 반영한 후, palloc_free_page 함수를 호출하여 임시로 할당받은 페이지를 해제한다.
4. 작업이 끝나면 hash_delete 함수로 해당 엔트리를 hash 테이블에서

삭제한다.

xi. void set_supp_entry

1. find_supp_entry함수로 인수로 받은 page를 멤버로 가지는 spte를 찾는다.

2. 이후 해당 spte의 f에 대해 frame_set_not_evict를 호출함으로써 해당 프레임이 evict되지 않도록 mark한다.

xii. void unset_supp_entry

1. 위 함수와 반대로 frame_unset_not_evict함수를 호출한다.

B. Discussion

i. 기존 디자인과의 비교: 해당 페이지의 dirty 정보나 status를 바꾸는 함수를 추가함으로써 frame 상태에 있던 페이지가 쫓겨났을 때를 대비한 기능을 구현하였다. 또한 set_supp_entry, unset_supp_entry 함수를 추가함으로써 해당 supp_entry의 frame의 not_evict 값을 변경하는 기능을 구현하였다.

ii. 어려웠던 점: supp_unmap함수에서 해당 페이지가 프레임에 있는 경우, file_write_at 함수 도중에 이 프레임이 evict되지 않도록 구현하는 것이 까다로웠다. 나중에 frame_set_not_evict 함수를 추가 구현하고 이를 file_write_at 함수 호출 이전에 먼저 호출함으로써 해결하였다.

4. Stack Growth

A. Solution

i. 우선 syscall.c의 syscall_handler에 thread_current() -> current_esp = f->esp;를 추가함으로써 esp 값을 저장한다.

ii. exception.c의 page_fault 함수에서 stack growth를 해결한다.

1. 만약 !not_present가 true라면 이는 stack growth가 아닌 실제 잘못된 경우이므로 먼저 처리한다.

2. Stack growth인 경우, find_supp_entry를 호출하여 fault_page가 supt에 있는지 조사하고 없으면 supp_insert_page를 호출하여 추가

한다. 이때, zero page로 추가한다.

3. 이후 load_page를 호출하여 fault_page에 대해 frame을 할당한다. 실패하면 예외처리해준다.

B. Discussion

i. 기존 디자인과의 비교: 기존 디자인 보고서에서는 stack_growth 함수를 따로 구현하여 이 기능을 수행하려 했으나 exception.c의 page_fault 함수를 수정하는 것만으로 충분하다는 생각이 들어 따로 새로운 함수를 구현하지는 않았다. 대신 thread 구조체에 current_esp 멤버를 추가하고 syscall.c에서 current_esp을 update한 후 이를 exception.c의 page_fault에서 활용하는 방향으로 바뀌었다.

ii. 어려웠던 점: lazy loading 구현하면서 page_fault를 어느 정도 구현해 놓았기 때문에 stack growth에서는 크게 어려운 부분은 없었다.

5. File Memory Mapping

A. Solution

i. Mmap

1. 해당 주소가 괜찮은 지 확인하기 위해 upage가 null이거나 pg_ofs(upage)가 0이 아니면 -1을 반환하며, fd 값이 정상적인지 확인하기 위해 1이하면 -1을 반환한다.

2. 파일을 reopen을 통해 fd로부터 받아오며, 이 과정에서 파일을 받아오지 못하거나, 파일의 길이가 0 이하라면 -1을 반환한다. (이 과정에서부터 파일을 다루므로 synch를 통해 lock을 해둔다)

3. while 문을 돌면서 파일을 페이지 단위로 나누어 파일 정보를 저장한다. 이 과정에서 해시 테이블을 사용할 수 있을 것을 생각된다. page.c의 hash table에 supplemental page table의 정보를 저장한다.

4. 이후 추가한, mmap_list에 mmap한 파일의 정보들을 저장하고 mmap의 id를 반환한다.

ii. Munmap

1. 제거할 mmap 파일을 mmap id를 통해 찾는다.
2. 반복문을 탐방하면서 해당 supplemental page를 사용했음을 pin을 설정하여 알린다.
3. Frame에 저장된 경우에 dirty bit가 1이라면 file_write_at을 이용하여 주소에 값을 적고 할당을 해제한다.
4. Swap disk에 저장된 경우에 dirty bit가 1이라면 swap_in을 한 후에 file write를 하고 free를 해준다. Dirty bit가 1이 아닌 경우엔 bitmap을 1로 세팅한다.
5. Hash table에 저장한 파일을 제거하도록 한다.
6. 이후, mmap list, file, 할당한 pagedir을 해제한다.

B. Discussion

i. 기존 디자인과의 비교

1. 일단 mmap list가 추가되어 mmap한 값을 mmap list에 추가하거나 읽고 삭제하는 부분이 추가되었다.
2. Mmap의 경우 계획과 상당히 유사하게 구현하였지만, munmap의 경우, 저장 위치 케이스 별로 나누어 처리를 하도록 구현하였다.

ii. 어려웠던 점

1. 해당 부분이 page의 구현에 상당한 영향을 받기 때문에 팀원의 구현과 내가 생각한 구현이 달라서 적용하는 것이 어려웠었다.

6. Swap table

A. Solution

i. Swap_init

1. Swap할 block을 block_get_role(BLOCK_SWAP)을 통해 받아온다.

2. 받아온 block을 바탕으로 block 크기를 페이지 크기로 나눈 만큼의 크기로 bitmap을 만든다.
3. 이후 bitmap의 값을 모두 참으로 설정한다.

ii. Swap in 함수

1. 반복문을 통해 PGSIZE를 BLOCK당 섹터 크기로 나눈 sectors 만큼 swap block로부터 block_read()를 통해 블록을 가져오고 bitmap_set 함수를 사용하여 가져온 블록들을 바꿔준다.

iii. Swap out 함수

1. swap할 때, bitmap_scan을 사용하여 바꿀 위치를 얻는다.
2. 반복문을 통해 sectors만큼 swap index에 페이지를 write하고 swap한 위치를 반환한다.

B. Discussion

i. 기존 디자인 비교

1. Swap table은 design report에서 정의했던 슈도 코드의 내용의 거의 따랐기 따랐다. Bitmap_flip 함수를 구현하려고 하였으나 해당 함수가 간단하여 그냥 한 함수 내에 구현하였다.

ii. 어려웠던 점

1. 계획대로 구현되어 구현상에 큰 어려움은 없었던 것 같다.

7. On process termination

A. Solution

i. Process exit에서 mmap을 저장했던 mmap_list를 전부 닫는다. 또한, 자식들에게 할당했던 page를 전부 free해준다.

ii. 새로 추가한 supplemental page table을 destroy해서 정상적으로 termination이 될 수 있도록 하였다.

B. Discussion

i. 기존 디자인과의 비교

1. 계획대로 process_exit에서 supplemental page table을 destroy하도록 구현하였다.

2. 그 과정에서 file 구조체의 변경으로 인한 코드 변경, mmap을 한 것들 것 munmap 과정에 필요하여 mmap list를 만들고 해당 리스트의 값들을 비우는 코드를 추가하였다.

ii. 어려웠던 점

1. 파일 구조체가 변경됨에 따라 종료 과정에서 처리해야하는 구조체들이 변경되었고 해당 값들을 처리하는 것이 어려웠다.