OS project3 vitual memory simulator report

학번 : 201520908

이름 : 유성민

학과 : 소프트웨어

학년 : 3

1. translate() 구현

outer page table의 INDEX 16개와 inner page table INDEX 16개로 각각 구성이 되어,

VPN은 총 0-255 (256개 INDEX)을 갖게 됩니다. 순서가 VPN -> outer -> inner -> PFN이 되어 VPN 이 16개로 나뉘어야 되기 때문에, OUTER는 VPN / 16 해서 총 16개 INDEX를 갖고, INNER는 256개가 16개 단위(OUTER 단위)마다 순환 되야 되기 때문에 VPN % 16이 됩니다. 그래서 translate()에서는 해당 index를 참조하는데 그 값이 NULL이면 page fault handler를 호출하는 기능, 해당 index를 참조하는데 그 값이 유효하면 구조체를 타고가서 참조하고 그 값을 파라미터로 넘어온 PFN변수에 집어넣어 주기만 하면 됩니다. 총 4가지의 경우가 있습니다.

여기서 outer_index = vpn / 16, inner_index = vpn % 16

사용한 변수들은 pros는 struct process 구조체 배열로 프로세스들이 생성될 때 마다 이 배열에 담깁니다. posp는 pros배열을 왔다 갔다 거리면서 현재 참조중인 프로세스를 가리키는 포인터입니다. 편의 상 outer pagetable은 outer로, inner pagetable는 inner라고 각각 줄이겠습니다.

다음과 같은 4가지 경우가 있습니다. 1) !pros[current>pid].pagetable.outer_ptes[outer_index] 는 현재 참조하고자 하는 프로세스의 outer가 없는 경우를 지칭합니다. => false 리턴

2) !pros[current->pid].pagetable.outer_ptes[outer_index]->ptes[inner_index].valid 는 현재 참조하고자 하는 프로세스의 outer는 있지만 inner가 참조된 적이 없는 경우 valid bit이 꺼져 있을 것입니다. => false 리턴

3) rw && cow && !pros[current->pid].pagetable.outer_ptes[outer_index]>ptes[inner_index].writable 이 경우는 cow를 처리하는 메커니즘으로 rw는 write = 1, cow는 fork된 프로세스의 페이지를 참조하는 경우를 나타내는 flag 3) 그리고 이 페이지 테이블에서 writable bit가 현재 fork되었으니 꺼졌을 것이므로 이런 페이지를 참조하는 경우를 처리하는 메커니즘입니다. => false 리턴

4) 위 세가지 모두 해당 안되는 경우 단순히 구조체를 타고 들어가서 저장된 값을 파라미터로 넘어온 pfn에 저장시킨 뒤 true를 리턴 합니다. => page fault가 아닌 유일한 경우

2. handle_page_fault() 구현

여기서는 tanslate()에서 false를 리턴한 3가지 조건을 각각에 맞게 처리해줘야 합니다.

1) outer 가 없는 경우

prosp라는 프로세스 구조체를 가리키는 포인터 변수를 선언하고 여기에 pros[current->pid]를 map시키고 이 포인터 변수를 사용하여 malloc()으로 공간을 pte_directory(outer의 크기)만큼 할당합니다. 그리고 할당된 공간에 가서 inner를 참조하고 inner의 각 변수 pfn, valid, writable 값을 차례대로 집어 넣습니다.

1) pfn은 alloc_page()함수를 사용하여 저장

2) valid bit = true로 갱신

3) writable = true

그리고 생성된 pros[current->pid]를 current와 map시킵니다. 그 후에 리턴 시킵니다

2) outer는 있으나 inner로 갔을 때 이전에 참조된 적 없는 페이지를 참조하는 경우

inner를 참조해서 inner의 각 변수 pfn, valid, writable 값을 차례대로 집어 넣습니다.

1) pfn은 alloc_page()함수를 사용하여 저장

2) valid bit = true로 갱신

3) writable = true 그 후에 리턴 시킵니다.

3) cow 처리 메커니즘

pros배열에 각각의 프로세스가 들어가 있으므로 현재 참조되고 있는 프로세스로 가서 페이지 테이블을 갱신 시킵니다. 참조 중인 프로세스의 outer로 가서 inner를 참조해서 inner의 각 변수 pfn, valid, writable 값을 차례대로 집어 넣습니다.

1) pfn은 alloc_page()함수를 사용하여 저장

2) valid bit = true로 갱신

3) writable = true 그 후에 리턴 시킵니다.

3. switch_process()

시작은 process 0를 list에 집어넣는 과정으로 시작합니다. 그 후, isforked라는 bool형 배열을 이용하여 각각의 프로세스들이 fork되었음을 표시하는 flag배열을 만듭니다. 그리고 2가지 case로 나눠지는데, 1) 현재 switch 하려는 프로세스가 없어서 current로부터 fork되는 경우, 2) switch 하려는 프로세스가 이미 존재하여 단순히 바꿔 주기만 하면 되는 경우로 나눠지게 됩니다.

1) 현재 switch 하려는 프로세스가 없어서 current로부터 fork되는 경우

과정별로 말씀드리겠습니다.

현재 current process의 pagetable로 가서 cow를 위해서 writable bit를 모두 끕니다.

그 다음으로 pros[pid](프로세스 배열) prosp(프로세스 가리키는 포인터)로 가리켜 놓고 loop를 돌면서, current process의 page table의 내용을 복사합니다.

당연하게도 current가 i번째 outer가 없으면 복사되는 것도 i번째 outer가 없어야 하므로, 루프를 돌면서 current->pagetable.outer[i]가 null이면 continue를 하여 loop를 제어합니다.

그 다음으로, fork된 새로운 프로세스 pros[pid]를 list_add()를 사용하여 리스트에 집어넣고

context switch를 하기 위해 current에 pros[pid]를 map시킵니다. 그리고 리턴 시킵니다.

2) switch 하려는 프로세스가 이미 존재하여 단순히 바꿔 주기만 하면 되는 경우

list를 traversal하면서 해당 프로세스를 pid를 이용해서 찾고, 찾은 경우 current랑 map 시키고, 찾은 경우 더 이상 못 돌게 하기 위해서 goto문을 사용하여 list_for_each_entry loop를 빠져나가서 리턴 시킵니다. 이 때, context switch된 프로세스들은 cow의 가능성이 있는 프로세스들이기에 이를 지칭하기 위해 cow변수 값을 true로 바꾸고 리턴 합니다.

4. Lesson learned

제일 힘들었던 거는 프로세스 각각의 페이지 테이블을 구현하는 것이었는데,

처음에 잘못 생각해서 모든 프로세스가 한 개의 페이지테이블을 동시에 가리켜서 cow가 구현이 잘 안되다가 다시 갈아엎고 새로 시작했는데도 다시 똑 같은 문제 발생하여 다시 갈아엎고 다시 해서 7전8기 정신으로 해냈습니다. 이번학기 동안 악명높은 운체 과제 4개를 모두 해냈다는게 매우 뿌듯하고, 내주신 교수님께 감사인사 드리고 싶습니다. 많이 배웠습니다.

제가 malloc을 자주 안 써서 그런지 애를 많이 먹었는데 이번 과제 하면서 malloc에 대해 확실한 개념이 잡혔습니다. 또 kernel의 list등등 여러모로 알게 되는게 많았습니다 감사합니다(_ _)