<운영체제 Assignment 2>

컴퓨터공학과 201811250 마지영

1. Basic design

- ku_cpu.c 의 전체적인 흐름 파악

: 제일 먼저 구현해야 할 각 함수의 역할을 파악하기 위해 제공해주신 ku_cpu.c의 흐름을 파악해보았다.

실행 시 입력 값은 pid와 va가 들어있는 텍스트 파일, physical memory 사이즈와 swap space 사이즈이다. 그 후에, 입력 받은 텍스트 파일에 해당하는 파일을 오픈하고, physical memory와 swap space에 각각 입력받은 크기를 할당하기 위해 ku_mmu_init 함수를 실행한다. 따라서 이를 통해 ku_mmu_init 함수는 전체적으로 사용할 변수들에게 일정 크기를 할당해 주는 등, 프로그램 시작을 위한 배경 설정을 하는 함수라는 것을 알 수 있다.

그 후에는 오픈한 파일에 있는 pid와 va값을 읽어오면서 만약에 읽어온 pid값인 fpid가 현재 ku_cpu.c가 가지고 있는 pid 값과 같지 않다면, ku_run_proc 함수를 실행한다. 이를 통해, ku_run_proc 함수는 현재 실행하고 있는 프로세스와 다른 프로세스가 실행을 시작하려고 할 때, context switch를 해주는 함수라는 것을 알 수 있다.

또한, ku_traverse 함수에 현재 실행중인 pid의 page directory base register에 해당하는 ku_cr3, virtual address, 그리고 임의로 정해준 physical memory인 pmem 포인터 변수를 넘겨주고, 리턴값으로 pa, 즉 physical address를 받아오는 것을 보면 virtual address와 page directory base register값을 이용해서 physical address를 계산하는 함수라는 것을 알 수 있었다. 그리고 만약에 현재 physical address에 mapping이 되어있지 않다면 리턴 값이 0이 되고, ku_page_fault 함수를 실행하게 된다.

ku_page_fault 함수는 Mapping이 되어있지 않을 때 실행하는 함수이기 때문에 단순하게 말하자면 mapping이 되어있지 않은 문제를 해결해주는 함수라고 말할 수 있다.

Ku_page_fault 함수를 처음으로 실행하고 난 후, 다시 한 번 ku_traverse함수를 실행하여 physical address를 받아오게 되는데, 만약에 ku_page_fault 함수가 제대로 작동을 했다면, 해당 주소에 mapping이 되어있을 것이고, 따라서 pa가 0이 되는 경우는 없을 것이다. 하지만 memory 공간 부족으로 mapping이 불가능했다면 여전히 va값에 해당하는 주소는 mapping이 되어있지 않을 것이고, 따라서 pa값은 여전히 0이 나오게 될 것이다. 그리고 이 경우에는 ku_cpu.c를 종료하게 된다.

- 구현을 위해 필요한 조건들 분석

: 먼저 함수들이 해줘야 하는 일을 자세하게 정리하면서, 이에 필요한 변수들을 정리해보았다.

1. ku_run_proc(fpid, &ku_cr3)

   : 해당 함수는 fpid가 한 번도 실행된 적 없는 프로세스의 pid에 해당을 하는지, 아니면 실행 된 적이 있어서, 프로세스에 해당하는 page directory가 존재하는지, 이 두 가지로 나눌 수 있다. 그런데, 이 두 가지를 구분을 하기 위해서는 pid와 page directory base register값을 매칭한 PCB가 필요하다는 것을 알 수 있다.

   ① Fpid가 한 번도 실행이 된 적 없는 경우

      : fpid를 위한 page directory를 생성해야 한다. 이는 freelist로부터 받아온 physical memory의 free한 공간에 할당을 해야 한다. 그리고 fpid값과 생성한 page directory의 base register 값을 PCB에 추가해주면 된다.

   ② Fpid가 예전에 실행된 적이 있는 경우

      : PCB를 조사해서 fpid에 해당하는 page directory base register값을 받아온다.

   두 경우에 대해 위와 같이 실행을 한 후에는 ku_cr3에 page directory base register값을 할당해야 하는데, 이 부분에 애를 많이 먹었다. Ku_cr3에 PTE값을 넣으라고 하셔서 처음에는 page directory base register의 PFN과 present bit를 2진수로 나타내고, 이를 다시 10진수로 바꿔서 ku_cr3에게 넘겨주는 방식을 취했다. 하지만, 계속해서 다른 값이 넘겨져서 결국, ku_traverse 함수의 assembly까지 분석을 하였고, 그 결과, page directory base register의 주소 그 자체를 ku_cr3에게 할당해야 한다는 것을 알게 되었다.

2. Ku_page_fault(pid,va)

   : 해당 함수는 세 가지 경우로 나눠서 각각 구현을 해주어야 했다. Va에 해당하는 주소를 따라가면서 조사하던 중, 현재 조사하는 physical address에 할당 되어있는 PDE/PTE의 값이 0인 경우, present bit가 1인 경우, present bit가 0인 경우로 나눌 수 있다.

   ① physical address에 할당되어 있는 값이 0인 경우

      : 해당 경우는 해당 주소가 아직 mapping이 되어있지 않다는 뜻이다. 따라서 va에 해당하는 주소를 위해 page middle directory, page table, page frame 중 필요한 부분을 physical memory의 free한 부분에 생성해야 한다. 그래서 va를 따라서

필요한 부분을 만들고, 해당 부분에 다음 page table에 해당하는 PTE 값을 할당해 주었다. 또한, swap되는 경우를 위해, page frame인 경우에는 FIFO의 순서를 기록하기 위한 자료구조인 p_list에 해당 page frame의 시작주소를 추가해주었다.

② present bit가 1인 경우

: 해당 경우는 현재 조사하는 physical address가 physical memory에 맵핑되어있다는 것을 의미한다. 따라서, 이 경우에는 PTE값을 통해 그냥 다음 physical address로 넘어가면 된다.

③ Present bit가 0인 경우

: 해당 경우는 현재 조사하는 physical address가 swap space에 위치한다는 것을 의미한다. 따라서 swap space offset을 통해서 해당 부분을 physical memory에 다시 맵핑 시켜주면 된다.

3. 추가 구현한 함수 : swapping

: 현재 physical memory에 free한 부분이 없는 경우를 위해 swap in을 해야 하는 경우를 위해 구현을 하였다. 이 경우에는 swap space의 free한 부분이 필요하기 때문에, 이를 위해 swap space의 freelist를 위한 전역 변수가 필요할 것이다.

따라서 swap space를 위한 freelist에서 현재 free한 위치를 받아오고, p_list에서 가장 먼저 physical memory에 생성된 page frame을 받아오고 이를 swap space로 옮겨주고, physical memory에서는 없애주면 된다. 그리고 해당하는 부분을 physical memory를 위한 freelist에 넣어주면 된다.

4. ku_mmu_init 과 전역 변수 선언

: 위에 함수들을 분석한 결과 다음과 같이 필요한 변수, 자료구조를 정리하였다.

• char* ku_mmu_physical_memory : ku_mmu에서 physical memory 포인터로 사용

• char* ku_mmu_swap_space : ku_mmu에서 swap space 포인터로 사용

• Linkedlist ku_mmu_PCB : Process Control Block으로 사용할 링크드 리스트, 노드를 통해 pid와 pdbr을 저장

• Linkedlist ku_mmu_freelist : ku_mmu에서 freelist로 사용할 링크드 리스트

• Linkedlist ku_mmu_s_free : swap space의 free한 공간을 관리하기 위함

• struct ku_ctr

• struct Node : 링크드 리스크 구현 위함

• struct linkedlist : 여러 자료구조 구현 위함

• Linkedlist ku_mmu_p_list : 현재 사용 중인 physical memory 파악 위함

따라서, 주어진 값들로 physical memory와 swap space의 크기를 할당하고, 각각의 freelist에 모든 page frame을 넣어 주었다.

2. Description for important functions

searchPCB	Functionality	Parameter로 받은 pid를 이용하여 해당 노드를 리턴 (노드에는 pdbr등 관련 정보가 있음)
	Parameters	Int pid
	Return Value	Node*

Ku_mmu_init	Functionality	Physical memory, swap space를 할당하고, freelist들을 초기화하고 physical memory의 포인터를 리턴
	Parameters	Unsigned int mem_size Unsigned int swap_size
	Return Value	Void*

Swapping	Functionality	Physical memory에 free한 공간이 없을 때, ku_mmu_p_list를 통해 swap space로 넘겨도 되는 부분을 swap space로 넘기고, 이 때, ku_mmu_s_free를 통해 swap space의 free한 공간을 받아와서 그 곳에 저장을 한다. 그리고 이 과정을 통해 확보된 physical memory의 free한 공간은 다시 ku_mmu_freelist에 넣어준다. 그리고 성공 여부를 리턴
	Parameters	
	Return Value	int

Ku_run_proc	Functionality	파라미터로 받은 pid의 page directory에 해당하는 부분이 맵핑 되어있는지 seachPCB 함수를 이용하여 조사하고, 맵핑이 되어있지 않다면, ku_mmu_freelist에서 free한 부분을 받아와서 그 부분에 page directory를 생성해준다. 그리고 pid와 해당 pdbr을 ku_mmu_PCB에 넣어준다. 마지막으로 해당 pdbr을 ku_cr3에 넣어준다. 만약 PCB에 해당 pid의 page directory가 존재 한다면, 해당 pdbr을 ku_cr3에 넣어준다. 성공 여부를 리턴한다.
	Parameters	Char pid Ku_pte **ku_cr3
	Return Value	int

Ku_page_fault	Functionality	Va를 따라서 조사하던 physical memory에 들어있는 값이 00000000, _____0, _____1의 경우를 나누어서 진행 1. 00000000의 경우 : 맵핑이 되어있지 않은 경우이기 때문에, ku_mmu_freelist를 통해 physical memory의 free한 부분을 받아서 이 부분에 해당 부분을 맵핑한다. 2. _____0의 경우 : 현재 swap space에 있기 때문에, _____의 부분에 있는 부분을 가져와서 physical memory의 free한 부분에 맵핑해준다. 3 _____1의 경우 : 현재 맵핑 되어있는 경우이기 때문에, va를 따라간다.
	Parameters	Char pid, char va
	Return Value	int

3. 실행해본 결과

```
jiyoung@jiyoung-VirtualBox:~$ ./k first.txt 56 40
[1] VA: 2 -> Page Fault
[1] VA: 2 -> PA: 18
[2] VA: 2 -> Page Fault
[2] VA: 2 -> PA: 34
[3] VA: 1 -> Page Fault
[3] VA: 1 -> PA: 49
[4] VA: 3 -> Page Fault
[4] VA: 3 -> PA: 51
jiyoung@jiyoung-VirtualBox:~$ ./k first.txt 40 40
[1] VA: 2 -> Page Fault
[1] VA: 2 -> PA: 18
[2] VA: 2 -> Page Fault
[2] VA: 2 -> PA: 34
ku_cpu: Fault handler is failed
```