Operating Systems

Assignment #1 KU_MMU

컴퓨터공학과

201811259

배수빈

■ Basic Design

-MMU 란 ?

CPU가 메모리에 접근하는 것을 관리하는 하드웨어로 가상 메모리 주소를 실제 메모리 주소로 변화하여 메모리 보호, 캐시관리, 버스 중재 등의 역할을 담당한다.

-ku_mmu의 구현

```
typedef struct ku_pte {
        char pte;
}ku_pte;

typedef struct ku_mmu_Node { //페이지의 정보를 가지는 노드
        struct ku_mmu_Node *prev;
        struct ku_mmu_Node *next;
        char pid;
        int level;
        int index;
        int pfn;
        struct ku_pte *pte;
}ku_mmu_Node;

typedef struct ku_mmu_LIST {
        struct ku_mmu_Node *head;
        struct ku_mmu_Node *tail;
}ku_mmu_List;

typedef struct ku_mmu_Page { //4바이트의 페이지
        char level, index, pid, a;
}ku_mmu_Page;


typedef ku_mmu_Page* ku_mmu_Space;
ku_mmu_Space *ku_mmu_Memory, *ku_mmu_Swap; //페이지배열로 활용하는 메모리 공간
ku_mmu_List *ku_mmu_list, *ku_mmu_swapList;
unsigned int ku_mmu_memorysize = 0, ku_mmu_swapsize = 0;//메모리배열 사이즈
```

위는 구현한 코드에서 활용하는 변수들과 구조체들이다.

ku_mmu_List와 ku_mmu_Node를 활용해서 linkedlist자료구조를 구현한다. 해당 리스트의 변수로는 ku_mmu_list와 ku_mmu_swapList가 존재하며 각각 메모리에 저장 되어있는 페이지의 정보를 저장하는 노드를 가지고, swap공간에 저장 되어있는 페이지의 정보를 저장하는 노드를 가진다.

ku_mmu_Page는 4바이트 사이즈를 가진다. ku_mmu_Space타입은 ku_mmu_Page 배열로 선언했다. 해당 타입의 변수로 ku_mmu_Memory와 ku_mmu_Swap이 있고 각각 메모리공간, 스왑공간을 의미한다. 예를 들어 ku_mmu_Memory[2]는 메모리의 시작주소+12에 위치하는 페이지를 의미한다.

ku_mmu_memorysize와 ku_mmu_swapsize는 각 메모리공간에 저장될 수 있는 페이지의 개수, 즉

ku_mmu_Page배열의 크기이다.

-table과 node

table을 생성하면 해당 테이블에 대한 노드가 생성되어 ku_mmu_list에 추가가 되는데 이때 노드에 저장되는 정보들에 관해서 설명하겠다. 노드 구조체의 멤버로는 pid, level, index, pfn, pte가 있다. 이따 pid는 해당 테이블이 어떤 프로세스를 위한 테이블인지를 나타내고 pfn은 리스트의 몇 번 째에 저장이 되었는지, pte는 해당 페이지의 다음level의 페이지의 pfn과 present bit을 나타내는 변수이다. level에 대해 설명하자면 해당 노드가 page directory에 대한 노드이면 level이 0, page middle directory에 대한 노드이면 level이 1, page table에 대한 노드이면 2, 최종 페이지에 대한 노드이면 3이다. index는 같은 level의 페이지들 중 몇번 째에 해당하는 페이지인지를 말한다. 이는 virtual address의 vpn정보들로 결정이 된다. 만약 virtual address가 10 11 00 01 이라고 하면 page middle directory는 상위 두 비트의 10 에 해당하는 것이므로 index가 2 인 것이고 세 번째인 것이다. 또 page table은 11에 따라서 index가 3이고 네번째 인 것이다.

-pte

pte는 다음 level의 페이지를 가리키는 변수이다. 만약 page middle directory가 생성이 되면 해당 va가 필요로 하는 page directory는 생성이 되어있는 상태일 것이고, page directory의 pte를 생성된 page middle directory가 적재된 ku_mmu_Memory에서의 인덱스와 present bit을 가지도록 한다. 만약 ku_mmu_Memory[3]에 page middle directory가 생성이 되어 저장되었다면, page directory의 pte는 3을 2만큼 left shift한 값과 1인 present bit(메모리에 저장되어 있으므로) 을 or 연산한 값으로 설정될 것이다. 즉 pfn인 3은 0000 0011 의 값인데 이를 2만큼 left shift한 0000 1100과 0000 0001를 or연산한 0000 1101 즉 13이 저장될 것이다. swap in이 되는 경우도 마찬가지로 page table의 pte값을 위와 같이 계산해서 설정한다.

만약 메모리 공간이 부족하여 swap out될 경우 해당 페이지를 가리키는 page table의 pte값이 변경되어야 한다. 해당 페이지가 메모리에 존재하지 않게 되는 것이기 때문에 present bit은 0이 되고 pfn은 swap공간에서의 인덱스로 설정된다. 이때 swap 공간에서 pfn을 표시할 수 있는 비트는 7비트 이다. 따라서 만약 swap out되는 페이지가 ku_mmu_Swap[12]에 적재가 된다고 하면 12, 즉 0000 1100를 1만큼 left shift한 0001 1000과 0인 present bit을 or연산한 0001 1000이 저장되는 것이다.

-구현

구현에 대한 설명은 밑의 함수 표에 자세히 설명되어 있다.

■ Description for important functions

Ku_mmu_init	Funtionality	***메모리공간, swap공간과 각 공간이 저장하는 페이지를 관리하는 리스트에 할당을 하고 메모리공간의 첫번째 공간에 os가 차지하는 페이지를 할당하는 함수** -ku_mmu_Memory, ku_mmu_Swap 공간에 인자로 받아오는 mem_size, swap_size만큼 할당해준다. -ku_mmu_setList()함수를 활용해 각 페이지를 관리하는 ku_mmu_list, ku_mmu_swapList를 생성한다. -각 메모리에 저장될수있는 페이지의 개수 즉 mem_size / 4, swap_size/4를 전역변수인 ku_mmu_memorysize, ku_mmu_swapsize 에 저장한다. (4로 나누는 이유는 각 메모리에 저장되는 페이지의 크기가 4이기 때문이다.)
	Parameters	unsigned int mem_size, unsigned int swap_size
	Return Value	Void *
ku_run_proc	Funtionality	***** 인자로 받아오는 pid의 페이지 디렉토리가 있으면 ku_cr3에 해당 페이지의 pte(시작주소값) 를 저장하고 없으면 생성해서 지정하는 함수이다.** -ku_mmu_checkPD(pid, 0 , 0, ku_mmu_list) 를 통해 pid에 해당하는 페이지 디렉토리가 있는지 확인해서 있으면 그 페이지를 관리하는 노드를 node에 저장한다. -없는 경우 node에 level과 index가 모두 0 인 페이지 (페이지 디렉토리) 를 생성해서 저장한다. - *ku_cr3에 node->pte를 저장한다.
	Parameters	char pid, struct ku_pte **ku_cr3
	Return Value	int
ku_page_fault	Funtionality	***** 인자로 받아오는 va값에 따라서 pid에 해당하는 프로세스가 실행될 수 있도록 네종류의 페이지들을 메모리에 적재하는 함수** -ku_mmu_vitOp함수를 통해 va에서 각 페이지에 대한 index 값을 저장한다. - checkpage(pmdIndex, ptIndex, finIndex) 를 통해 어떤 레벨의 페이지가 없는 지 확인한다. - checkpage의 리턴값이 0인 경우 (pmd, pt, 최종페이지가 없는 경우) 각 ku_mmu_generatePage()를 통해 세 레벨의 페

		이지들을 모두 생성한다. - checkpage의 리턴값이 1인 경우(pt, 최종페이지가 없는 경우) 두 레벨의 페이지들을 모두 생성한다. - checkpage의 리턴값이 2인 경우 (최종페이지가 없는 경우) 페이지를 생성하거나 swapIn 한다.
	Parameters	char pid, char va
	Return Value	int
Ku_mmu_ setPte	Funtionality	*** pfn값에 따라 인자로 받아오는 ku_pte 포인터 변수의 pte 값을 세팅해주는 함수** - present bit 가 0 일 경우 pte로 접근해야하는 페이지가 swap공간에 적재되는 경우 이므로 pte의 8비트 중 상위 7비트가 offset에 관한정보(pfn으로 활용됨) 이고 하위 1비트만 present bit으로 활용된다. 따라서 pfn을 1비트 left shift한 값과 인자값으로 받아오는 present 값을 or 연산해서 ku_pte의 pte에 저장한다. - present bit가 1일 경우 pte로 접근해야하는 페이지가 메모리공간에 적재되어있는 경우 이므로 pe의 8비트 중 상위 6비트가 pfn에 관한 정보이고 하위 1비트만 present bit으로 활용된다. 따라서 pfn을 2비트 left shift한 값과 인자값으로 받아오는 present값을 or연산해서 ku_pte의 pte에 저장한다.
	Parameters	ku_pte *ku_pte, int pfn, int present
	Return Value	void
Ku_mmu_bitOp	Funtionality	*** 인자로 받아오는 level에 따라서 각 페이지에서 몇번째 인덱스의 pte를 참조해야 하는지에 대한 index를 리턴하는 함수** -va는 8비트로 되어있고 각 두비트 씩 다음 pte를 가지는 index정보를 가진다. -level이 0일 경우 상위 두비트에 해당하는 값을 리턴한다. 즉 0xc0과 va를 &연산한 뒤 6비트 right shift한 값을 리턴한다.
	Parameters	int level, char va
	Return Value	int
Ku_mmu_ checkPD	Funtionality	*** 인자로 받아오는 list구조에 인자로 받아오는 pid, index, level에 해당하는 페이지가 있는지 확인하고 있으면 그 페이지에 대한 정보를 저장하는 Node를 리턴하는 함수**

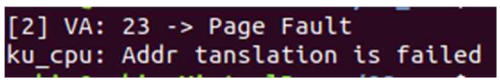
		-인자로 받아오는 level이란 pagedirectory : 0 page middle directory : 1 page table : 2 최종 page : 3 을 말한다. - index란 해당pid에 따른 해당 page level중 몇번 째 페이지인지 (예- level 1, index 2 이면 page middle directory중 두번째 / ku_mmu_bitOp(0, va) 한 결과값이 2 인 것) -인자로 받아오는 리스트의 head->next 노드부터 tail->prev 노드까지의 노드마다 pid, level, index를 비교한다.
	Parameters	char pid, int level, int index, ku_mmu_List* list
	Return Value	ku_mmu_Node *
Ku_mmu_ swapOut	Funtionality	*** 메모리공간에 가장 먼저 add되었던 level 3에 해당하는 페이지 즉, 최종페이지를 메모리공간과, ku_mmu_list에서 삭제하는 함수** ***삭제한 페이지의 메모리공간에서의 pfn(인덱스)을 리턴하는 함수** - ku_mmu_list (메모리공간에 저장되어 있는 페이지들을 관리하는 리스트) 의 head->next노드부터 가장 앞에 있는 3의 level값을 가지는 노드를 temp라고하자. -temp노드가 가지는 pfn이 해당 페이지가 메모리에 저장되어 있는 인덱스 이기 때문에 ku_mmu_Memory[temp->pfn] 에 해당하는 페이지를 pagetodelete라고하자. 1. 메모리공간에서의 삭제 -temp노드는 ku_mmu_list에서 ku_mmu_delegefromList()함수를 통해 삭제한다. -pagetodelete페이지가 저장되어있던 공간에 NULL값을 저장한다. -pagetodelete를 가리키던 page table의 pte값을 swap공간에서 페이지가 저장되어있는 index값과 0인 present bit으로 설정한다. 2. swap공간으로의 추가 - temp 노드를 ku_mmu_swapList에 addNodetoList함수를 통해 추가한다.

		-ku_mmu_findNullspace2를 통해 swap공간에서 가장 앞부분에 있는 빈공간을 찾아 해당 공간에 pagetodelete를 추가한다.
	Parameters	Char va
	Return Value	int
Ku_mmu_ generatePage	Funtionality	*** va에 따라 인자로 받아오는 level, index에 해당하는 페이지를 생성한다. (페이지에 대한 정보를 저장하는 노드도 생성한다.)** - 생성할 페이지를 할당하고 해당 페이지의 구조체 멤버들을 받아오는 인자들을 활용해서 초기화해준다. - ku_mmu_findNullspace를 활용해서 메모리공간에서 가장 앞부분의 빈 공간을 찾아 해당 공간의 인덱스정보를 pfn으로 저장한다. -생성한 페이지를 가리키는 페이지(한단계 하위 level의 페이지)의 pte값을 생성한 페이지의 pfn과 1의 present bit을 갖도록 setPte함수를 활용해서 설정한다. -ku_mmu_Memory[pfn]에 생성한 페이지를 저장한다. -ku_mmu_list에 생성한 노드를 addNodetoList함수를 통해 저장한다.
	Parameters	int level, int index, char pid, char va
	Return Value	Ku_mmu_Node*
Ku_mmu_ findNullspace	Funtionality	*** va에 따라 인자로 받아오는 level, index에 해당하는 페이지를 생성한다. (페이지에 대한 정보를 저장하는 노드도 생성한다.)** - ku_mmu_Memory[0] 부터 끝까지 NULL인 공간이 있는지 확인해서 있으면 해당 공간의 index값을 리턴 한다. - 빈 공간이 없으면 swapOut()함수를 통해 swaping을 해주고 생긴 빈 공간의 index값을 리턴한다.
	Parameters	char va
	Return Value	int
Ku_mmu_ swapIn	Funtionality	*** 인자로 받아오는 노드를 ku_mmu_list에 추가하고, 그 노드에 해당하는 페이지를 메모리에 추가하는 함수** 1. swap공간에서의 삭제

		- 해당노드를 deletefromList를 통해 리스트에서 삭제한다. - swap공간에서 해당 노드가 속해 있던 인덱스의 공간을 NULL값으로 지정한다. 2. 메모리공간으로의 추가 - 해당 노드를 addNodetoList를 통해 ku_mmu_list에 추가한다. - findNullspace()를 통해 찾은 빈공간에 해당하는 메모리에서의 인덱스를 pfn으로 설정한다. - setPte()를 통해 해당 페이지를 가리키는 페이지에 해당하는 노드의 pte 값을 pfn값과 1인 present bit값으로 설정한다. - ku_mmu_Memory[pfn]에 해당 페이지를 저장한다.
	Parameters	ku_mmu_Node* node, char va
	Return Value	void
Ku_mmu_ checkPage	Funtionality	*** 메모리공간에 어느 페이지까지 존재하는지 확인하는 함수 (page directory까지 있으면 0리턴, page middle directory까지 있으면 1리턴, page table까지 있으면 2리턴)** -페이지 폴트가 발생했을 때 페이지 폴트 핸들러가 호출하는 함수이다. - 어느 페이지가 존재하지 않는 것인지 확인하는 함수이므로 ku_mmu_list의 모든 노드를 다 확인한다.
	Parameters	int pmdIndex, int ptIndex, int finIndex
	Return Value	int

■ 느낀 점

최선을 다해서 구현을 완성할 수 있도록 노력했으나



가 출력되는 실패의 결과로 마무리하게 되었습니다. 지금까지 시스템 프로그래밍, 운영체제 첫 번째 과제 모두 스스로 구현에 성공하여 기쁘고 성취감도 생겼으며 개념에 대한 이해도 완벽하게 할 수 있었습니다. 이처럼 이번 과제도 어렵지만 완성할 수 있을 거라고 생각하며 며칠동안 매달려서 구현하였으나 실패했습니다. 결과에 대해서는 매우 아쉽지만 개념을 숙지할 수 있는 좋은 기회였다고 생각합니다. 과제를 하면서 addressing과 multi level paging, swapping 과 같이 수업을 들으면서 어느정도 이해는 하였으나 완벽히는 이해하지 못하던 개념들을 완벽하게 정리할 수

있었습니다. 과제 제출 기한이 완료되고 구현에 성공한 학생이 있다면 참고하거나 도움을 받아 어느 부분이 잘못되었는지 확인하여 완벽하게 구현하고 복습하도록 하겠습니다.