컴퓨터학부 20202920 조민혁

1.개요

간단한 소프트웨어 MMU를 구현하여 보기 위해서, alloc\_page\_table함수와 mmu\_address\_translation함수를 구현하는 프로그램입니다. alloc\_page\_table함수는 address\_sapce\_bits를 입력받아 페이지 테이블의 크기를 계산하여, 페이지 테이블의 크기와 PTE의 크기를 곱한 만큼 malloc을 사용하여 동적 메모리를 할당하고, page\_table이 NULL이 아닌 경우에는 0으로 메모리를 초기화 시켜주고 NULL인 경우에는 에러를 출력한 후 프로세스를 종료시킵니다. mmu\_address\_translation함수 같은 경우에는 가상 주소를 입력받아 vpn마스크를 적용시켜서 vpn값을 얻어내고, 해당 vpn을 통해 pte의 값을 찾아 낸 후에 pte가 valid한지, access가능한지 판단하여, vpn,pfn,valid,access에 대한 값을 출력하고 각 상황에 대해 SUCCESS , NOT\_ACCESSIBLE, NOT\_VALID를 리턴하는 프로그램입니다.

2.상세설계

2-1.기 구현된 내용과 수정한 내용

우선 init\_page\_table함수의 경우, 페이지 테이블의 사이즈를 계산한 후, 해당 페이지 테이블의 절반만큼을 채워나갑니다. 이때 인덱스가 4로 나누어 떨어지는 경우에는 해당 PTE에는 ACCESS\_MASK를 부여하지 않아 접근할 수 없도록 합니다.

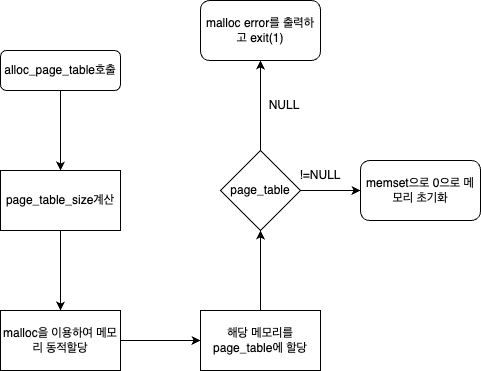
init\_mmu\_variables함수는 vpn\_mask , shift, offset\_mask값등 mmu\_address\_translation함수에서 사용할 수 있는 전역변수의 값들을 초기화해주는 역할을 하는 함수입니다.

alloc\_page\_table함수의 경우, 페이지 테이블의 크기를 계산하고 이에 PTE의 크기만큼을 곱하여 동적 메모리를 할당한 후 해당 값을 page\_table에 할당하여 줍니다. 또한 page\_table이 NULL이 아닌 경우에는 할당된 메모리를 0으로 초기화시켜줍니다. 이때 PTE의 크기를 unsigned int로 한 이유는 32비트와 64비트 운영체제에서 uint는 32비트 즉, 4바이트 인것이고, 본 과제에서의 PTE의 구조는 32비트를 사용하기 때문에 이를 활용하였습니다.

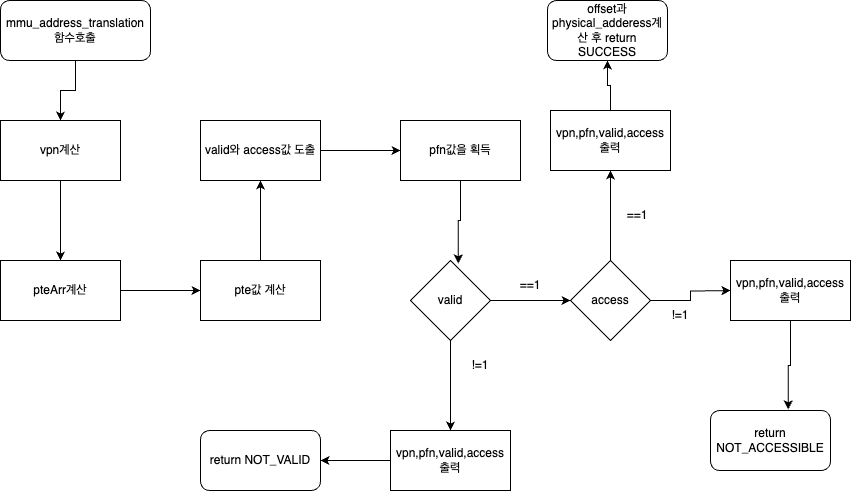
마지막으로 mmu\_address\_translation함수의 경우에는 가상 주소를 vpn\_mask를 통해 vpn을 구해내고 이를 통해서 pte의 주소와 pte의 값을 구해낼 수 있습니다. pte의 값에서 VALID\_MASK와 ACCESS\_MASK를 활용하여 valid와 access값을 구해내고 pfn도 PFN\_SHIFT를 통해서 구해냅니다. 그 후 valid와 access값에 따라서 SUCCESS, NOT\_ACESSIBLE, NOT\_VALID로 각각 다르게 리턴하며 그 이전에 vpn, pfn, valid, access값을 출력합니다.

2-2. 함수간의 호출 그래프

alloc\_page\_table함수의 호출 그래프



mmu\_address\_translation함수 호출 그래프



main함수의 호출 그래프

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명

3.실행결과

실행예제로 되어있는 것과 동일한 상태로 실행하였습니다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

4.소스코드

…기존코드

/\*

alloc\_page\_table();

전역 변수인 page\_table을 위한 메모리 공간을 할당.

페이지 테이블의 크기를 계산 -> 페이지 테이블의 크기와 PTE의 크기를 곱한 만큼 malloc()을 사용하여 동적 메모리를 할당

-> 세 번째로, malloc()의 반환값을 page\_table에 할당 -> 마지막으로, 할당된 메모리를 0으로 초기화

\*/

void alloc\_page\_table(int address\_space\_bits, int page\_bytes)

{

/\* 프로그램 직접 작성\*/

//페이지 테이블의 크기 계산

int page\_table\_size = pow(2, address\_space\_bits - (int)log2(page\_bytes));

//PTE = 32bits => 4byte

//동적 메모리를 할당함

unsigned int \*temp = malloc(sizeof(unsigned int)\*page\_table\_size);

//page\_table에 반환값을 할당

page\_table = temp;

if(page\_table!=NULL){

//페이지를 0으로 초기화시켜줍니다.

memset(page\_table,0,sizeof(unsigned int) \* page\_table\_size);

}else{

printf("malloc error\n");

exit(1);

}

}

…기존코드

/\*

mmu\_address\_translatio();

가상 주소를 물리적 주소로 변환. 변환에 성공하면, 변환된 주소를 physical\_address 변수에 복사하고 SUCCESS를 반환.

PTE가 유효하지 않으면, 이 함수는 NOT\_VALID를 반환.

PTE에 접근할 수 없으면, 이 함수는 NOT\_ACCESSIBLE을 반환.

\*/

int mmu\_address\_translation(unsigned int virtual\_address, unsigned int \*physical\_address)

{

/\* 프로그램 직접 작성하되 아래 변수 가능하면 사용\*/

unsigned int vpn;

unsigned int pfn;

int valid;

int access;

unsigned int offset;

unsigned int \*pteArr; //pte의 주소를 나타냅니다.

unsigned int pte; //pte를 담기 위함입니다.

//extract the VPN from the Virtual Address

//shift는 4096 기준 12비트입니다.

vpn = (virtual\_address & vpn\_mask) >> 12;

//form the address of the Page Table Entry (PTE) PTEAddr = PTBR + (VPN + sizeof(PTE)

pteArr = page\_table + vpn;

//pte에 access합니다.

pte = \*pteArr;

//valid와 access를 뽑아냅니다.

valid = pte & VALID\_MASK;

access = (pte & ACCESS\_MASK) >> 1;

//pfn을 구해냅니다.

pfn = pte >> PFN\_SHIFT;

//접근할 수 없는 경우에는 NOT\_ACCESSIBLE

//페이지 테이블의 인덱스가 4로 나눠 떨어지는 경우에는 접근할 수 없다고 나타내주면 된다.

if(valid == 1){

//valid 가능한 것

if(access == 1){

//access 가능

//12비트를 이동하여 pfn 20비트 확보

offset = virtual\_address & offset\_mask;

\*physical\_address = (pfn << PFN\_SHIFT) | offset;

printf(" (vpn:%08x, pfn: %08x, valid: %d, access: %d) ", vpn, pfn, valid, access);

return SUCCESS;

}else{

//access 가능하지 않은 경우

printf(" (vpn:%08x, pfn: %08x, valid: %d, access: %d) ", vpn, pfn, valid, access);

return NOT\_ACCESSIBLE;

}

}else{

printf(" (vpn:%08x, pfn: %08x, valid: %d, access: %d) ", vpn, pfn, valid, access);

return NOT\_VALID;

}

}

…기존코드