컴퓨터학부 20221867 이채은

**1. 개요**

- 가상 주소를 물리 주소로 변환하는 간단한 소프트웨어 MMU(Memory Management Unit)를 구현하는 것을 목표로 한다. 32비트 페이지 테이블 엔트리(PTE)를 사용하며, 각 PTE는 물리 프레임 번호(PFN, 12-31비트), 접근 권한(Access 비트, 1비트), 유효성 검사(Valid 비트, 0비트)로 구성된다.

주요 구현 내용은 페이지 테이블 할당(alloc\_page\_table)과 주소 변환(mmu\_address\_translation) 두 가지이다. 페이지 테이블은 전체의 50%만 매핑되며, VPN n은 PFN n\*2에 매핑되는 규칙을 따른다. 또한 VPN이 4로 나누어 떨어지는 경우 해당 페이지는 접근이 불가능하도록 설계되었다.

주소 변환 과정에서 유효하지 않은 페이지 접근 시 Segmentation Fault가, 접근 권한이 없는 페이지 접근 시 Protection Fault가 발생한다. 이를 통해 기본적인 메모리 보호 메커니즘을 구현하였다.

**2. 상세 설계 & 3. 실행 결과 & 4. 주석 달린 소스코드**

**- 2-1. void alloc\_page\_table(int address\_space\_bits, int page\_bytes); 구현**

- (기능 1) address\_space\_bit와 page\_bytes로 페이지 테이블의 크기와 page를 계산. 페이지 테이블의 크기를 구하는 함수 참고

- (기능 2) 페이지 테이블 크기 \* PTE 크기로 페이지 테이블을 위한 malloc()을 이용한 동적 메모리 할당

- (기능 3) 전역변수 page\_table에 malloc()의 리턴 값 할당

- (기능 4) 할당된 메모리를 0으로 초기화하고 리턴

- 주석을 상세하게 달아서 설명

|  |
| --- |
| void alloc\_page\_table(int address\_space\_bits, int page\_bytes); |
| void alloc\_page\_table(int address\_space\_bits, int page\_bytes)  {  // 1. 페이지 테이블의 크기 계산  // 2^(address\_space\_bits) / 2^(log2(page\_bytes)) = 전체 페이지 테이블 엔트리 개수  int page\_table\_size = pow(2, address\_space\_bits - (int)log2(page\_bytes));    // 2. PTE 크기(sizeof(unsigned int))를 곱해 전체 메모리 크기 계산 후 할당  page\_table = (unsigned int \*)malloc(page\_table\_size \* sizeof(unsigned int));    // 3. 할당된 메모리를 0으로 초기화  if (page\_table != NULL) {  memset(page\_table, 0, page\_table\_size \* sizeof(unsigned int));  }  } |

**- 2-2. int mmu\_address\_translation(unsigned int virtual\_address, unsigned int \*physical\_address); 구현**

- 과제 요구 사항을 참고하여, 마지막 라인인 Register = AccessMemory(PhysAddr);는 구현하지 않았음

- 최대한 슈도 코드에서 요구하는 대로 형식에 맞게 노력하며 구현하였음

- 주석을 상세하게 달아서 설명

|  |
| --- |
| int mmu\_address\_translation(unsigned int virtual\_address, unsigned int \*physical\_address); |
| int mmu\_address\_translation(unsigned int virtual\_address, unsigned int \*physical\_address)  {  // 1. 페이지 크기와 최대 VPN 계산  int page\_size = 1 << shift; // 페이지 크기 = 2^shift  unsigned int max\_vpn = (vpn\_mask >> shift); // 최대 가능한 VPN 값    // 2. 가상 주소에서 VPN 추출  unsigned int vpn = (virtual\_address & vpn\_mask) >> shift;    // 3. VPN 범위 검사 (잘못된 접근 방지)  if (vpn > max\_vpn) {  printf(" (vpn:%08x, pfn: %08x, valid: 0, access: 0)", vpn, 0);  return NOT\_VALID;  }    // 4. 페이지 테이블에서 PTE 가져오기  unsigned int pte = page\_table[vpn];    // 5. PTE에서 필요한 정보 추출  unsigned int pfn = pte >> PFN\_SHIFT; // PFN은 12-31 비트  int valid = pte & VALID\_MASK; // Valid 비트는 0번 비트  int access = (pte & ACCESS\_MASK) >> 1; // Access 비트는 1번 비트    // 6. 디버그 정보 출력 (8자리 16진수)  printf(" (vpn:%08x, pfn: %08x, valid: %d, access: %d)",  vpn, pfn, valid, access);    // 7. Valid 비트 검사  if (!valid) {  return NOT\_VALID; // Segmentation Fault 발생  }    // 8. Access 비트 검사  if (!access) {  return NOT\_ACCESSIBLE; // Protection Fault 발생  }    // 9. 물리 주소 계산  // offset은 하위 비트 추출  unsigned int offset = virtual\_address & offset\_mask;  // 물리 주소 = (PFN << shift) | offset  \*physical\_address = (pfn << shift) | offset;    return SUCCESS;  } |

✓ 실행 결과

|  |
| --- |
| ./ssu\_mmu 32 4096 |
| A screenshot of a computer program  Description automatically generated |

- 결과가 정확히 일치함