과제 #3-1 : SSU Memory Allocation

○ 과제 목표

- 운영체제 커널의 메모리 할당 및 해제 방법 이해

○ 기본 지식

- xv6의 메모리 관리 이해
  - ✓ 페이지 테이블의 구조와 동작 방식 분석
  - ✓ vm.c, trap.c, proc.c, defs.h, mmu.h 등에서 동작하는 메모리 관리 방법(가상 메모리, 동적 메모리, PTE 등) 분석

○ 과제 내용

1-1. ssusbrk() 시스템 콜 구현

- (정의) 메모리를 지연 할당/해제하는 시스템 콜로 두 개의 매개변수를 입력받음, 첫 번째 매개변수는 페이지의 크기이고 두 번째 매개변수는 지연시킬 tick의 값임
- (기능 1) 첫 번째 매개변수가 양수인 경우, 입력된 페이지의 크기만큼의 메모리를 지연 할당함
  - ✓ 첫 번째 매개변수는 할당할 메모리의 크기(0은 안됨)이며, 페이지 크기의 배수임
    - 유효한 값을 입력받고, 가상 메모리를 성공적으로 할당했다면 할당한 메모리의 주소를 반환
      - ☞ 이 경우 두 번째 매개변수의 값이 무엇이든 무시됨
      - ☞ 유효하지 않은 값을 입력받았을 때는 -1을 반환함
  - ✓ ssusbrk() 시스템 콜이 호출됐을 때 가상 메모리만 할당하고 물리 메모리는 할당하지 않음
    - 할당된 가상 메모리에 사용자가 접근할 때 해당 가상 메모리에 대응하는 물리 메모리가 존재하지 않을 경우 특정 trap이 발생하며, 이때 물리 메모리를 페이지 단위로 할당하도록 구현
    - 둘 이상의 가상 메모리 페이지가 할당된 경우 접근된 페이지에 대해서만 물리 메모리 페이지를 할당함
- (기능 2) 첫 번째 매개변수가 음수인 경우, 입력된 페이지의 크기만큼의 메모리를 두 번째 매개변수의 tick만큼 지연시킨 후 해제함
  - ✓ 첫 번째 매개변수는 해제할 메모리의 크기(0은 안됨)이며, 페이지 크기의 배수이고,
    두 번째 매개변수는 메모리 해제 시 지연시킬 tick의 값이며, 양수만 가능함
    - 유효한 값을 입력받고, 가상 메모리를 성공적으로 해제했다면 해제한 메모리의 주소를 반환
      - ☞ 유효하지 않은 값을 입력받았을 때는 -1을 반환
  - ✓ ssusbrk() 시스템 콜이 호출되고 입력받은 tick값이 지난 후 메모리 페이지 해제 작업이 수행됨
    - 이전에 요청된 메모리 해제가 수행되지 않은 시점에 다시 ssusbrk()가 호출된 경우 이전에 해제 요청한 페이지에 새로 요청된 페이지가 더해지고, tick 값은 뒤에 호출된 ssusbrk()의 값을 기준으로 초기화됨
      - ☞ 시스템 콜이 호출된 시점과 해제 작업이 수행된 시점의 날짜 및 시간 정보를 화면에 출력하고, 해제된 메모리의 상태를 확인하기 위해 memstat()을 호출해야 함
      - ☞ 테스트 프로그램을 수행하기 위해 메모리 해제가 이뤄진 후 프로세스를 종료하도록 구현해야 함
- (Hint 1) 기존 xv6는 위 기능들에 대한 trap을 처리하지 않으며, 이를 처리하도록 수정해야 함
  해당 trap에 대한 질문은 받지 않으며, trap의 종류와 처리 방법에 대한 설명을 보고서에 포함
- (Hint 2) xv6에는 시간 정보를 저장하기 위한 구조체와 시간 정보를 읽을 수 있는 시스템 콜이 존재함
- 원할한 과제 진행을 위해 ssusbrk_test1/2/3 프로그램 제공(수정 불가), 제공하는 ssusbrk_1/2/3.c를 xv6에 추가하기 위해 Makefile을 수정할 것

1-2. memstat() 시스템 콜 구현

- (정의) 현재 할당된 가상 메모리와 물리 메모리의 정보를 출력하는 시스템 콜
- (기능 1) 호출한 프로세스의 가상 메모리 페이지와 물리 메모리 페이지의 개수를 화면에 출력
- (기능 2) 현재 물리 메모리로 할당되어 있는 페이지 디렉토리 엔트리(PDE)와 페이지 테이블 엔트리(PTE)의 값을 화면에 출력

✓ memstat()호출 결과는 (예시 1-1), (예시 1-2), (예시 1-3) 참고

(예시 1-1). ssusbrk_test1 실행 결과

```
$ ssusbrk_test1
### Allocation test start
 vp: 3, pp: 3
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067
ok
ok
 vp: 4, pp: 3
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067
 vp: 4, pp: 4
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067 - 0xdee3067
ok
 vp: 4, pp: 4
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067 - 0xdee3067
 vp: 6, pp: 4
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067 - 0xdee3067
 vp: 6, pp: 5
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067 - 0xdee3067 - 0xdee1067
 vp: 6, pp: 6
 PDE - 0xdf2d027
 PTE - 0xdf2c027 - 0xdf74067 - 0xdee3067 - 0xdee1067 - 0xdf22067
ok
$
```

(예시 1-2). ssusbrk_test2 실행 결과

```
$ ssusbrk_test2
### Deallocation test1 start
 vp: 5, pp: 5
 PDE - 0xdee1027
 PTE - 0xdee2027 - 0xdedf067 - 0xdfbc067 - 0xdf76067
Memory deallocation request(300): 2024-11-14 15:43:13
Memory deallocation excute: 2024-11-14 15:43:16
 vp: 4, pp: 4
 PDE - 0xdee1027
 PTE - 0xdee2067 - 0xdedf067 - 0xdfbc067
$
```

(예시 1-3). ssusbrk_test3 실행 결과

```
$ ssusbrk_test3
### Deallocation test2 start
 vp: 5, pp: 5
 PDE - 0xdfc2027
 PTE - 0xdfc1027 - 0xdfc5067 - 0xdf2b067 - 0xdf2d067
Memory deallocation request(1000): 2024-11-14 15:43:42
Memory deallocation request(500): 2024-11-14 15:43:42
Memory deallocation excute: 2024-11-14 15:43:47
 vp: 3, pp: 3
 PDE - 0xdfc2027
 PTE - 0xdfc1067 - 0xdfc5067
$
```

과제 #3-2 : Simple Software MMU

○ 과제 목표

- 간단한 소프트웨어 MMU 구현

○ 기본 내용

- 컴파일 및 실행
  - ✓ gcc - o ssu_mmu ssu_mmu.c -lm
  - ✓ ./ssu_mmu [address_space_size_in_bits] [page_size_in_bytes]
  - ✓ 예) ./ssu_mmu 6 16
- 과제 진행 시 주의 사항
  - ✓ main() 는 수정하지 않아도 됨
  - ✓ 기본 ssu_mmu.c는 Linux 및 BSD에서 실행. 반드시 Linux 환경에서 실행
  - ✓ 기 구현된 함수 설명
    - void init_page_tabe(int address_space_bits, int Page_bytes);
      - ☞ (정의) 초기 PTE들을 페이지 테이블에 삽입하는 함수
      - ☞ VPN n 은 PFN*2에 매핑. • PTE의 50%만 채울 수 있음. VPN이 4 로 나눌 수 있으면 accessble 할 수 없는 PTE가 됨. 페이지 테이블에 학생들의 PTE들을 별도로 넣을 수 없음
    - void init_mmu_variables(int address_space_bits, int Page_bytes)
      - ☞ (정의) mmu_address_translateion()을 위한 전역변수 초기화하는 함수
      - ☞ vpn_mask, shift, offset_mask 의 기본 개념은 주어진 pseudo 코드에서 확인
  - ✓ 본 과제에서 PTE 구조

32 bit 사용, bit 12-31 : PFN, bit 0 : Valid 비트 (1 : Valid, 0: Invalid), Bit 1 : Access bit (1 : Can access, 0 : Cannot access), other bits are not used

○ 과제 내용

2-1. void alloc_page_table(int address_space_bits, int page_bytes); 구현

- (정의) 전역변수 page_table을 위해 메모리 공간 할당하는 함수
- (기능 1) address_space_bit와 page_bytes 로 페이지 테이블의 크기와 page를 계산. 페이지 테이블의 크기를 구하는 함수를 참고하기 바람.
- (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

2-2. int mmu_address_translation(unsigned int virtual_address, unsigned int *physical_address); 구현

- (정의) 가상주소를 물리주소로 변환하는 함수로 다음 슈도 코드 참고. 단 여기서, 마지막 라인, Register = AccessMemory(PhysAddr); 는 구현하지 않아도 됨.

```
//extract the VPN from the Virtual Address
VPN = (Virtual_Address & VPN_Mask)) >> SHIFT

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

```
//fetch the PTE
PTE = AcessMemory(PTEAddr)

//check if process can access the page
if (PTE.Valid == False)
  RaiseException(Segmentation_Fault)
else if (CanAcess(PTE.Protectbits)== Faulse)
  RaiseExecption(Protjection_Fault)
else // Access is OK : form physical address and fetch it
  ofset = Virtual_Address * Offset_Mask
  PhysAddr = (PTE.PFN << PFN_SHIFT) | offeset
  Register = AccessMemory(PhysAddr)
```

- vpn_mask, shift, and offset_mask 등 MMU pseudo-code 정보는 init_mmu_variables()에서 구현되어 있음
- mmu_address_translation()의 호출이 성공하면, 변환된 주소를 physical_address 변수에 복사하고, "SUCCESS" 표준 출력, 만약에 PTE가 Valid 하지 않으면 "NOT_VALID" 리턴, PTE에 접근할 수 없으면, "NOT_ACCESSIBLE" 리턴

(참고) 실행결과

(예시 2-1)./ssu_mmu 32 4096

```
./ssu_mmu 32 4096
SSU_MMU Simulator
Input a virtual address of hexadecimal value without "0x" (-1 to exit): abc
Virtual address: 0xabc (vpn:00000000, pfn: 00000000, valid: 1, access: 0) -> Protection Fault.
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 1abc
Virtual address: 0x1abc (vpn:00000001, pfn: 00000002, valid: 1, access: 1) -> Physical address: 0x2abc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 10001abc
Virtual address: 0x10001abc (vpn:00010001, pfn: 00020002, valid: 1, access: 1) -> Physical address: 0x20002abc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 20002abc
Virtual address: 0x20002abc (vpn:00020002, pfn: 00040004, valid: 1, access: 1) -> Physical address: 0x40004abc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 30003abc
Virtual address: 0x30003abc (vpn:00030003, pfn: 00060006, valid: 1, access: 1) -> Physical address: 0x60006abc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 40004abc
Virtual address: 0x40004abc (vpn:00040004, pfn: 00080008, valid: 1, access: 0) -> Protection Fault.
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 50005abc
Virtual address: 0x50005abc (vpn:00050005, pfn: 000a000a, valid: 1, access: 1) -> Physical address: 0xa000aabc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 60006abc
Virtual address: 0x60006abc (vpn:00060006, pfn: 000c000c, valid: 1, access: 1) -> Physical address: 0xc000cabc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 70007abc
Virtual address: 0x70007abc (vpn:00070007, pfn: 000e000e, valid: 1, access: 1) -> Physical address: 0xe000eabc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 7ffffabc
Virtual address: 0x7ffffabc (vpn:0007ffff, pfn: 000ffffe, valid: 1, access: 1) -> Physical address: 0xfffffeabc
Input a virtual address of hexadecimal value without "0x" (-1 to exit): 80000abc
Virtual address: 0x80000abc (vpn:00080000, pfn: 00000000, valid: 0, access: 0) -> Segmentation Fault.
Input a virtual address of hexadecimal value without "0x" (-1 to exit): -1
```

과제 #3-3 : Multi-level File System and Page Cache

○ 과제 목표

- xv6 메모리 및 파일시스템 구조 이해 및 응용

○ 기본 지식

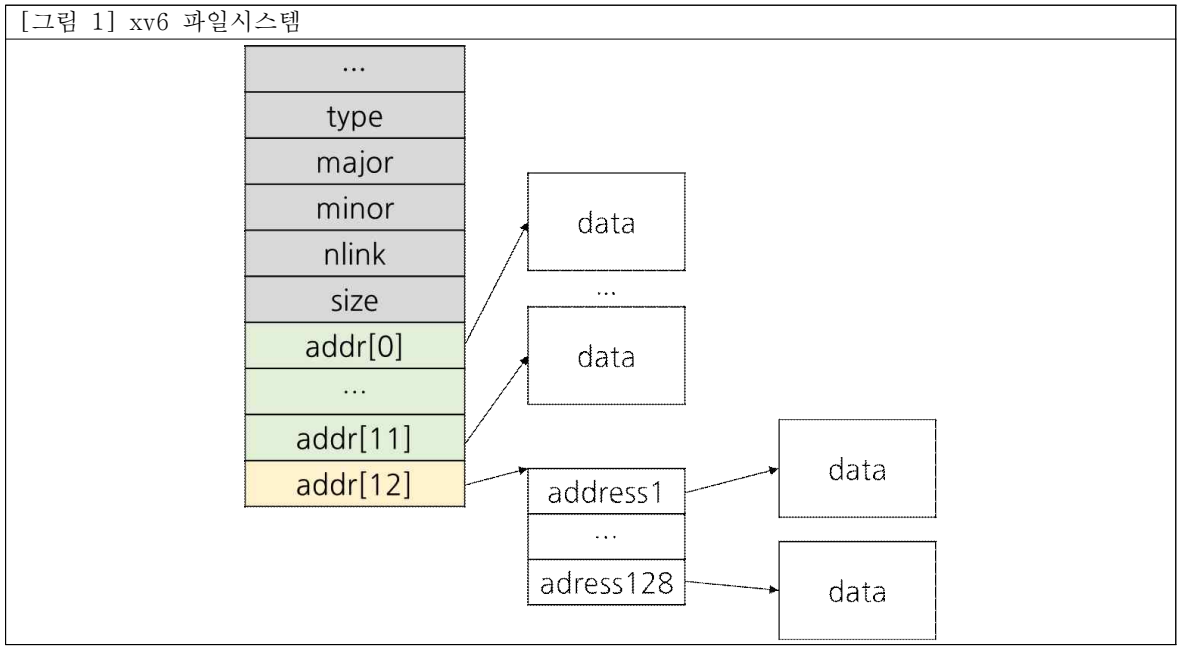
- xv6의 파일시스템 이해
  - ✓ fs.c, fs.h, file.c, file.h, defs.h, params.h 등에서 동작하는 파일시스템 구조 분석(read, write, close, link 등)
- 트리 자료구조에 대한 이해
  - ✓ 레드 블랙 트리의 기능 분석 (탐색, 삽입, 삭제, 순회)
- 메모리 할당과 해제에 대한 이해
  - ✓ kalloc.c, umalloc.c 등에서 동작하는 메모리 할당 및 해제 구조 분석 (kalloc, kfree, malloc, free)

○ 과제 내용

3-1. multi-level 파일시스템 구현

- (정의) 3-level 파일시스템 구현
- Hint 1, [그림 1]은 xv6 파일시스템이 사용하는 inode 구조체의 일부를 나타냄.

[그림 1] xv6 파일시스템



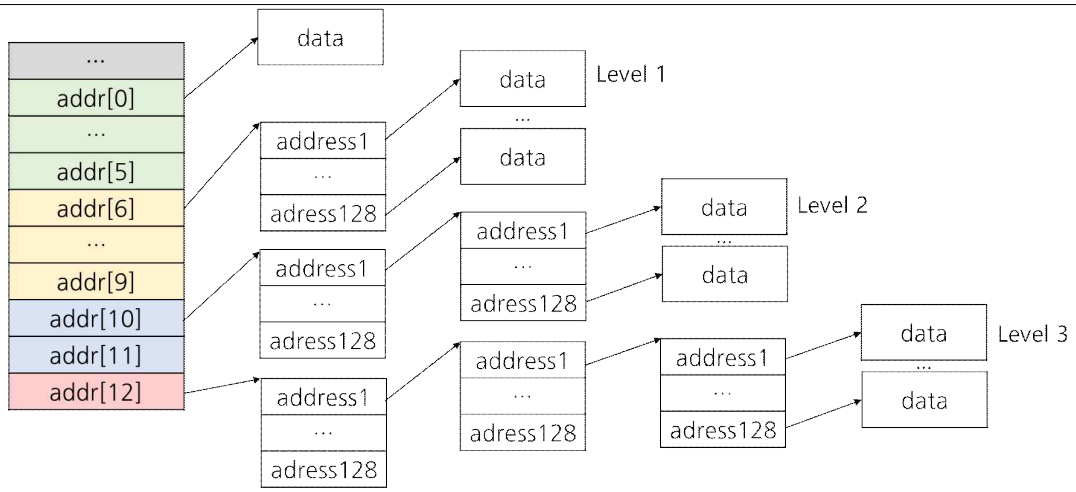
- ✓ addrs 배열이 가리키는 블록 번호는 디스크에 파일이 실제로 저장된 데이터 블록의 번호를 의미함
- ✓ 기존 xv6의 파일시스템 구조는 12개의 direct 블록, 1개의 indirect 블록으로 구성되어있음.
- ✓ direct 블록인 addrs[0] ~ addrs[11]에는 파일의 내용이 저장된 디스크의 데이터 블록을 가리키는 포인터 변수가 저장됨
- ✓ indirect 블록인 addrs[12]는 데이터 블록을 직접 가리키는 것이 아닌, 또 다른 인덱스 블록(데이터 블록 테이블)을 가리키며 인덱스 블록 내부에는 실제 데이터 블록을 가리키는 포인터 변수들이 저장됨
- ✓ xv6는 하나의 파일에 140개(12+128)의 데이터 블록을 저장 공간으로 사용할 수 있으며, 이에 따라 한 파일의 크기는 140 * 512B(Bytes) = 71,680B, 약 70KB임

- (기능 1) 6개(addrs[0] ~ addrs[5])의 direct 블록, 4개(addrs[6] ~ addrs[9])의 indirect 블록, 2개

(addrs[10] ~ addrs[11])의 2-level indirect 블록, 1개(addrs[12])의 3-level indirect 블록을 가지는 파일 시스템을 구현

✓ 대략적인 구조는 [그림 2]과 같음

[그림 2] xv6 multi-level 파일시스템



- 따라서 본 과제에서 구현하는 파일시스템은 2,130,438(6+128*4+128*128*2+128*128*128)개의 데이터 블록을 저장 공간으로 사용할 수 있으며 약 1GB의 파일 크기를 지원함

✓ 원활한 과제 진행을 위해 ssufs_test 프로그램 제공 예정

- 제공하는 ssufs_test를 실행하기 위해 Makefile을 수정할 것

-(기능 2) 대용량 파일 생성을 위해 params.h의 FSSIZE를 변경해야함 (1000 -> 2500000)

✓ 대용량 multi-level 파일시스템을 구현한 후, ssufs_test의 실행 결과는 (예시 3-1)과 같음

(예시 3-1). ssufs_test 실행 결과 (mode : MODE_DEFALUT)

```
$ ssufs_test
### test1 start
create and write 5 blocks...
ok
close file descriptor...
ok
open and read file...
ok
unlink file1...
ok
open file1 again...
failed
### test1 passed...

### test2 start
create and write 500 blocks...
ok
close file descriptor...
ok
open and read file...
ok
unlink file2...
ok
open file2 again...
failed
### test2 passed...
```

```
### test3 start
create and write 5000 blocks...
ok
close file descriptor...
ok
open and read file...
ok
unlink file3...
ok
open file3 again...
failed
### test3 passed...

### test4 start
create and write 50000 blocks...
ok
close file descriptor...
ok
open and read file...
ok
unlink file4...
ok
open file4 again...
failed
### test4 passed...
```

3-2. 레드블랙트리로 이루어진 페이지 캐시 구현

-(정의) 레드블랙트리를 이용해 최근 참조 블록의 디스크 접근을 최소화하는 페이지 캐시 구현

✓ (예시 3-2)처럼 file.h 내부의 struct inode에 레드블랙트리에 해당하는 포인터 선언

(예시 3-2). xv6 file.h의 inode구조체

```
// in-memory copy of an inode
struct inode {
    //레드 블랙 트리를 가리키는 포인터 선언
};
```

-(기능 1) 레드블랙트리는 최대 100개의 노드를 가짐

-(기능 2) 하나의 레드블랙트리가 사용할 수 있는 메모리는 최대 4KB

✓ 한번의 kalloc으로 레드블랙트리와 모든 노드가 구현 되어야함

✓ 레드블랙트리의 각 노드들을 할당받는 코드는 직접 하나의 페이지를 분리하는 코드를 작성하여 구현

-(기능 3) 레드블랙트리 구조 페이지 캐시의 cache hit 처리

✓ 찾고자 하는 블록 번호가 레드블랙트리에 이미 존재한다면 cache hit이며 바로 레드블랙트리에서 그에 해당하는 주소를 가져옴

✓ 기존에 존재했던 키를 찾은 경우 해당 키가 가장 최근에 접근한 키가 됨

-(기능 4) 레드블랙트리 구조 페이지 캐시의 cache miss 처리

✓ 찾고자 하는 블록 번호가 레드블랙트리에 없다면 새로운 (블록 번호, 주소) 쌍을 기존의 bmap에서 구하는 방식과 동일하게 계산한 후 레드블랙트리에 추가

✓ 레드블랙트리의 최대 크기를 넘는 경우, 레드블랙트리에서 가장 오래된 노드 삭제 후 (블록 번호, 주소)쌍

추가

✓ 결과적으로 bmap으로 들어온 (블록 번호, 주소)쌍이 cache hit, cache miss 경우 모두 가장 최근에 접근한 키가 됨

3-3. int lseek(int fd, int offset, int whence); 구현

- (정의) 테스트 프로그램의 파일 오프셋 임의 접근을 위한 lseek 시스템콜 구현

✓ (예시 3-3)에서 sys_lseek 코드를 제공함, 나머지 코드 및 매크로는 직접 추가 해야함.

(예시 3-3). sys_lseek 코드

```
int
sys_lseek(void){
  struct file *f;
  int n, whence;
  uint off;

  if(argfd(0, 0, &f) < 0 || argint(1, &n) || argint(2, &whence))
    return -1;

  switch (whence){
  case SEEK_SET:
    off = 0;
    break;
  case SEEK_CUR:
    off = f->off;
    break;
  case SEEK_END:
    off = f->ip->size;
    break;

  default:
    return -1;
    break;
  }

  off += n;

  if(off < 0 || off > f->ip->size)
    return -1;

  f->off = off;
  return off;
}
```

3-4. int rb_print(int fd); 구현

- (정의) 현재 페이지 캐시의 레드블랙트리 노드 상태를 in-order 방식으로 출력하는 시스템콜 구현
- (기능 1) 첫째 줄부터 노드 개수만큼 in-order 방식으로 레드블랙트리를 탐색하여 key, val, depth, color, parent key 출력

✓ root node의 depth는 1, parent key는 -1

✓ (예시 3-4)의 rb_print system call 이후 메시지 참고

3-5. int rb_count(int fd); 구현

- (정의) 현재 페이지 캐시의 정보를 출력하는 시스템콜 구현

- (기능 1) bmap count, cache hit count, disk acess count 출력

✓ bmap함수의 호출 횟수를 나타내는 bmap count, cache hit 횟수를 나타내는 cache hit count, disk 접근 횟수를 나타내는 disk access count 출력

✓ (예시 3-4)의 rb_count system call 이후 메시지 참고

(예시 3-4). ssufs_test 실행 결과 (test1 : MODE_PRINT|MODE_COUNT, test2 : MODE_PRINT, test3 : MODE_COUNT, test4 : 생략)

```
$ ssufs_test
### test1 start
create and write 5 blocks...
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_print system call start...
ok
rb_count system call start...
bmap access count : 5, cache hit count : 0, disk access count : 0
ok
rb_print system call start...
key : 1, value : 1290, depth : 1, color : B, parent key : -1
key : 0, value : 1289, depth : 2, color : B, parent key : 1
key : 3, value : 1292, depth : 2, color : B, parent key : 1
key : 2, value : 1291, depth : 3, color : R, parent key : 3
key : 4, value : 1293, depth : 3, color : R, parent key : 3
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_print system call start...
ok
rb_count system call start...
bmap access count : 5, cache hit count : 2, disk access count : 0
ok
rb_print system call start...
key : 2, value : 1291, depth : 1, color : B, parent key : -1
key : 0, value : 1289, depth : 2, color : R, parent key : 2
key : 4, value : 1293, depth : 2, color : R, parent key : 2
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_print system call start...
ok
rb_count system call start...
bmap access count : 5, cache hit count : 3, disk access count : 0
```

```
ok
rb_print system call start...
key : 0, value : 1289, depth : 1, color : B, parent key : -1
key : 3, value : 1292, depth : 2, color : R, parent key : 0
ok
close file descriptor...
ok
unlink file1...
ok
open file1 again...
failed
### test1 passed...

### test2 start
create and write 500 blocks...
ok
close file descriptor...
ok
open and read file...
rb_print system call start...
ok
rb_print system call start...
key : 431, value : 1724, depth : 1, color : B, parent key : -1
key : 415, value : 1708, depth : 2, color : B, parent key : 431
key : 407, value : 1700, depth : 3, color : B, parent key : 415
key : 403, value : 1696, depth : 4, color : B, parent key : 407
key : 401, value : 1694, depth : 5, color : B, parent key : 403
key : 400, value : 1693, depth : 6, color : B, parent key : 401
key : 402, value : 1695, depth : 6, color : B, parent key : 401
key : 405, value : 1698, depth : 5, color : B, parent key : 403
key : 404, value : 1697, depth : 6, color : B, parent key : 405
key : 406, value : 1699, depth : 6, color : B, parent key : 405
key : 411, value : 1704, depth : 4, color : B, parent key : 407
key : 409, value : 1702, depth : 5, color : B, parent key : 411
key : 408, value : 1701, depth : 6, color : B, parent key : 409
key : 410, value : 1703, depth : 6, color : B, parent key : 409
key : 413, value : 1706, depth : 5, color : B, parent key : 411
key : 412, value : 1705, depth : 6, color : B, parent key : 413
key : 414, value : 1707, depth : 6, color : B, parent key : 413
key : 423, value : 1716, depth : 3, color : B, parent key : 415
key : 419, value : 1712, depth : 4, color : B, parent key : 423
key : 417, value : 1710, depth : 5, color : B, parent key : 419
key : 416, value : 1709, depth : 6, color : B, parent key : 417
key : 418, value : 1711, depth : 6, color : B, parent key : 417
key : 421, value : 1714, depth : 5, color : B, parent key : 419
key : 420, value : 1713, depth : 6, color : B, parent key : 421
key : 422, value : 1715, depth : 6, color : B, parent key : 421
key : 427, value : 1720, depth : 4, color : B, parent key : 423
key : 425, value : 1718, depth : 5, color : B, parent key : 427
key : 424, value : 1717, depth : 6, color : B, parent key : 425
key : 426, value : 1719, depth : 6, color : B, parent key : 425
key : 429, value : 1722, depth : 5, color : B, parent key : 427
key : 428, value : 1721, depth : 6, color : B, parent key : 429
key : 430, value : 1723, depth : 6, color : B, parent key : 429
```

```
key : 447, value : 1740, depth : 2, color : B, parent key : 431
key : 439, value : 1732, depth : 3, color : B, parent key : 447
key : 435, value : 1728, depth : 4, color : B, parent key : 439
key : 433, value : 1726, depth : 5, color : B, parent key : 435
key : 432, value : 1725, depth : 6, color : B, parent key : 433
key : 434, value : 1727, depth : 6, color : B, parent key : 433
key : 437, value : 1730, depth : 5, color : B, parent key : 435
key : 436, value : 1729, depth : 6, color : B, parent key : 437
key : 438, value : 1731, depth : 6, color : B, parent key : 437
key : 443, value : 1736, depth : 4, color : B, parent key : 439
key : 441, value : 1734, depth : 5, color : B, parent key : 443
key : 440, value : 1733, depth : 6, color : B, parent key : 441
key : 442, value : 1735, depth : 6, color : B, parent key : 441
key : 445, value : 1738, depth : 5, color : B, parent key : 443
key : 444, value : 1737, depth : 6, color : B, parent key : 445
key : 446, value : 1739, depth : 6, color : B, parent key : 445
key : 463, value : 1756, depth : 3, color : R, parent key : 447
key : 455, value : 1748, depth : 4, color : B, parent key : 463
key : 451, value : 1744, depth : 5, color : B, parent key : 455
key : 449, value : 1742, depth : 6, color : B, parent key : 451
key : 448, value : 1741, depth : 7, color : B, parent key : 449
key : 450, value : 1743, depth : 7, color : B, parent key : 449
key : 453, value : 1746, depth : 6, color : B, parent key : 451
key : 452, value : 1745, depth : 7, color : B, parent key : 453
key : 454, value : 1747, depth : 7, color : B, parent key : 453
key : 459, value : 1752, depth : 5, color : B, parent key : 455
key : 457, value : 1750, depth : 6, color : B, parent key : 459
key : 456, value : 1749, depth : 7, color : B, parent key : 457
key : 458, value : 1751, depth : 7, color : B, parent key : 457
key : 461, value : 1754, depth : 6, color : B, parent key : 459
key : 460, value : 1753, depth : 7, color : B, parent key : 461
key : 462, value : 1755, depth : 7, color : B, parent key : 461
key : 471, value : 1764, depth : 4, color : B, parent key : 463
key : 467, value : 1760, depth : 5, color : B, parent key : 471
key : 465, value : 1758, depth : 6, color : B, parent key : 467
key : 464, value : 1757, depth : 7, color : B, parent key : 465
key : 466, value : 1759, depth : 7, color : B, parent key : 465
key : 469, value : 1762, depth : 6, color : B, parent key : 467
key : 468, value : 1761, depth : 7, color : B, parent key : 469
key : 470, value : 1763, depth : 7, color : B, parent key : 469
key : 479, value : 1772, depth : 5, color : R, parent key : 471
key : 475, value : 1768, depth : 6, color : B, parent key : 479
key : 473, value : 1766, depth : 7, color : B, parent key : 475
key : 472, value : 1765, depth : 8, color : B, parent key : 473
key : 474, value : 1767, depth : 8, color : B, parent key : 473
key : 477, value : 1770, depth : 7, color : B, parent key : 475
key : 476, value : 1769, depth : 8, color : B, parent key : 477
key : 478, value : 1771, depth : 8, color : B, parent key : 477
key : 487, value : 1780, depth : 6, color : B, parent key : 479
key : 483, value : 1776, depth : 7, color : R, parent key : 487
key : 481, value : 1774, depth : 8, color : B, parent key : 483
key : 480, value : 1773, depth : 9, color : B, parent key : 481
key : 482, value : 1775, depth : 9, color : B, parent key : 481
key : 485, value : 1778, depth : 8, color : B, parent key : 483
```

```
key : 484, value : 1777, depth : 9, color : B, parent key : 485
key : 486, value : 1779, depth : 9, color : B, parent key : 485
key : 491, value : 1784, depth : 7, color : R, parent key : 487
key : 489, value : 1782, depth : 8, color : B, parent key : 491
key : 488, value : 1781, depth : 9, color : B, parent key : 489
key : 490, value : 1783, depth : 9, color : B, parent key : 489
key : 495, value : 1788, depth : 8, color : B, parent key : 491
key : 493, value : 1786, depth : 9, color : R, parent key : 495
key : 492, value : 1785, depth : 10, color : B, parent key : 493
key : 494, value : 1787, depth : 10, color : B, parent key : 493
key : 497, value : 1790, depth : 9, color : R, parent key : 495
key : 496, value : 1789, depth : 10, color : B, parent key : 497
key : 498, value : 1791, depth : 10, color : B, parent key : 497
key : 499, value : 1792, depth : 11, color : R, parent key : 498
ok
close file descriptor...
ok
open and read file...
rb_print system call start...
ok
rb_print system call start...
key : 382, value : 1674, depth : 1, color : B, parent key : -1
key : 350, value : 1642, depth : 2, color : B, parent key : 382
key : 318, value : 1610, depth : 3, color : B, parent key : 350

... 중략

key : 492, value : 1785, depth : 9, color : B, parent key : 494
key : 496, value : 1789, depth : 9, color : B, parent key : 494
key : 498, value : 1791, depth : 10, color : R, parent key : 496
ok
close file descriptor...
ok
open and read file...
rb_print system call start...
ok
rb_print system call start...
key : 333, value : 1625, depth : 1, color : B, parent key : -1
key : 285, value : 1577, depth : 2, color : B, parent key : 333
key : 237, value : 1528, depth : 3, color : B, parent key : 285

... 중략

key : 495, value : 1788, depth : 9, color : B, parent key : 489
key : 492, value : 1785, depth : 10, color : R, parent key : 495
key : 498, value : 1791, depth : 10, color : R, parent key : 495
ok
close file descriptor...
ok
unlink file2...
ok
open file2 again...
failed
### test2 passed...
```

```
### test3 start
create and write 5000 blocks...
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_count system call start...
bmap access count : 5000, cache hit count : 0, disk access count : 9476
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_count system call start...
bmap access count : 5000, cache hit count : 2500, disk access count : 4738
ok
close file descriptor...
ok
open and read file...
rb_count system call start...
bmap access count : 0, cache hit count : 0, disk access count : 0
ok
rb_count system call start...
bmap access count : 5000, cache hit count : 3333, disk access count : 3159
ok
close file descriptor...
ok
unlink file3...
ok
open file3 again...
failed
### test3 passed...
```

과제 #3의 제출 마감 및 배점 기준

○ 과제 제출 마감

- 2024년 11월 25일 (화) 23시 59분 59초까지 구글클래스룸으로 제출
- 보고서 폴더 내 3-1, 3-2, 3-3을 별도로 만들고 3-1, 3-2, 3-3 관련 보고서(hwp, doc, docx 등으로 작성)가 별도로 존재함.
- 소스코드 폴더 내 3-1, 3-2, 3-3을 별도로 만들어야 함
  - ✓ 3-1 및 3-3의 경우 xv6에서 변경한 소스코드(Makefile 포함 필수), 변경의 기준은 xv6-public 원본이며, 첨부한 소스코드를 복사하여 채점함.
  - ✓ 3-2의 경우 ssu_mmu.c 만 포함
- 3-1, 3-2, 3-3 중 구현한 것만 폴더를 만들어야 함. 예를 들어 3-2만 구현한 경우 보고서 폴더와 소스코드 폴더에는 3-2 폴더만 있어야 함
- 마감시간 이후 24시간까지 지연 제출 가능. 그 이후 제출은 0점 처리. 설계과제 마감시간 이후 지연 제출은 30% 감점.

○ 필수 구현

- 3-1, 3-2, 3-3 전체

○ 배점 기준 (100점 만점 기준)

- 3-1 : 30점 (부분 점수 없음)

- 3-2 : 20점 (부분 점수 없음)

- 3-3 :50점
  - ✓ 3-1, 10점
  - ✓ 3-2, 25점 (필수 구현)
  - ✓ 3-3, 3점
  - ✓ 3-4, 6점
  - ✓ 3-5, 6점