os_lab3_32220393_김경열

2025-06-14

1. 구현 코드 설명.

-impl.h 파일

```
class GreedyFTL : public FlashTranslationLayer {
protected:
    int current_block;
    int current_page;
    bool in_gc;
public:
    // 생성자: 총 블록 수와 블록당 페이지 수를 전달
    GreedyFTL(int total_blocks, int block_size);
    // 소멸자
    ~GreedyFTL();

    // Garbage Collection 수행
    void garbageCollect() override;
    // LPN 영역에 데이터 쓰기
    void writePage(int logicalPage, int data) override;
    // LPN 영역에서 데이터 읽기
    void readPage(int logicalPage) override;
};
```

```
class CostBenefitFTL : public FlashTranslationLayer {
protected:
    int current_block;
    int current_page;
    bool in_gc;
public:
    // 생성자: 총 블록 수와 블록당 페이지 수를 전달
    CostBenefitFTL(int total_blocks, int block_size);
    // 소멸자
    ~CostBenefitFTL();

    // Garbage Collection 수행 (Cost-Benefit 정책)
    void garbageCollect() override;
    // LPN 영역에 데이터 쓰기
    void writePage(int logicalPage, int data) override;
    // LPN 영역에서 데이터 읽기
    void readPage(int logicalPage) override;
};
```

.h파일에서 두 클래스 모두 lashTranslationLayer 클래스를 상속하고, .cpp 파일 구현에서 사용될 변수들과 생성자, 소멸자, 함수들을 정의했다.

Current_block은 현재 쓰기를 진행중인 활성화된 블록의 번호를 표현하고 current_page 변수는 활성화된 블록 내 페이지의 위치를 나타낸다. in_gc 변수는 GC중인지 여부를 확인 하기위해 사용된다.

-Greedy 생성자

user32220393@ubuntuSYSPR

```
/*
*    DKU Operating System Lab
*        Lab3 (Flash Translation Layer)
*        Student id : 32220393
*        Student name : 김경열
*        Date : 2025-06-12
*/


#include "ftl_impl.h"
#include <iostream>
#include <algorithm>
//  GreedyFTL 구현부
// 생성자 정의
// 생성자: 초기 상태 설정 및 메모리 구조 초기화
GreedyFTL::GreedyFTL(int total_blocks, int block_size)
    : FlashTranslationLayer(total_blocks, block_size) {
    name = "GreedyFTL"; // FTL 이름 설정
    current_block = 0;  // 첫 번째 블록부터 시작
    current_page = 0;   // 첫 페이지부터 시작
    in_gc = false;      // GC 상태 초기화

    blocks.resize(total_blocks); // 전체 블록 수 만큼 resize
    for (auto& block : blocks)
        block.pages.resize(block_size); // 각 블록에 페이지 구조 할당

    L2P.resize(total_blocks * block_size, -1); // 논리 -> 물리 매핑 테이블 초기화
}
```

원래 헤더 파일에 생성자 정의와 함께 초기화를 진행하려고 했으나, 프로그램이 계속 터져서 cpp파일에 생성자 초기화를 진행했다.

Greedy 생성자에서 먼저 이름과 각 변수들을 초기화한다. 그 다음 전체 블록 수만큼 blocks의 크기를 조정하고, L2P 테이블을 전체 페이지

수만큼 할당한다.

-garbageCollect

```
user32220393@ubuntuSYSPR
// GC 수행 함수: 가장 invalid page가 많은 victim 블록 선택 후 valid page 복사
void GreedyFTL::garbageCollect() {
    int victim = -1;            // victim 블록의 인덱스를 저장할 변수 (-1은 아직 선택되지 않음)
    int max_invalid = -1;       // 가장 많은 invalid 페이지 수를 저장할 변수

    // 모든 블록을 순회하며 INVALID 수가 가장 많은 블록 찾기
    for (int b = 0; b < total_blocks; ++b) {
        int inv_cnt = 0;        // 현재 블록의 invalid 페이지 수를 센다
        for (const auto& pg : blocks[b].pages)
            if (pg.state == INVALID) inv_cnt++; // 해당 페이지가 INVALID 상태이면 count 증가

        // invalid 수가 더 많거나, 같을 경우 인덱스가 더 작은 쪽을 victim으로 선택 (tie-break)
        if (inv_cnt > max_invalid || (inv_cnt == max_invalid && (victim == -1 || b < victim))) {
            max_invalid = inv_cnt;  // 현재까지의 최댓값 갱신
            victim = b;             // victim 후보 갱신
        }
    }
```

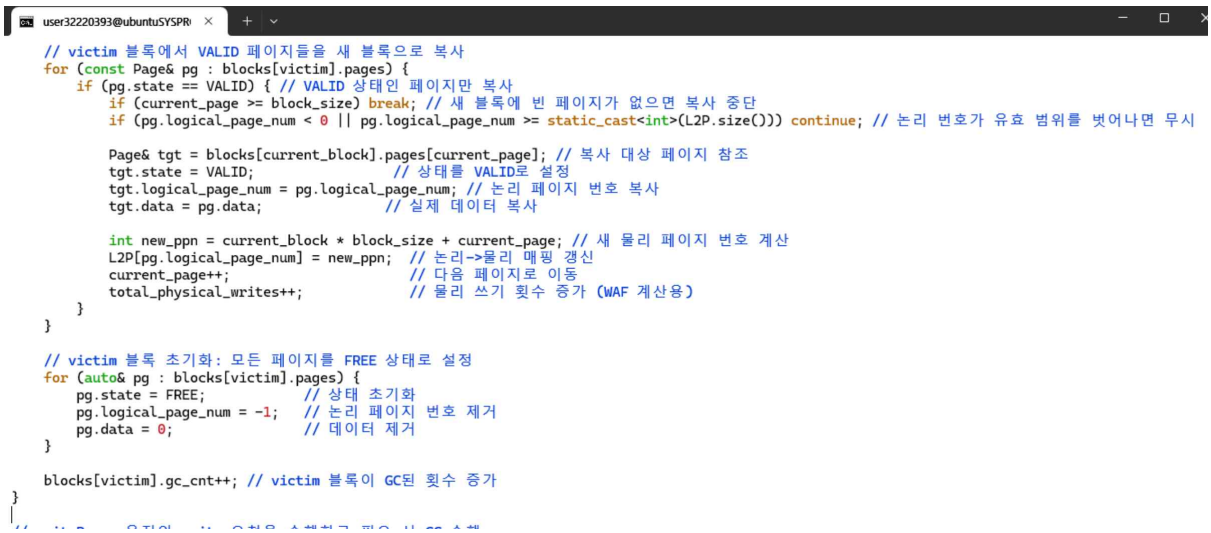
먼저 victim 대상의 인덱스를 저장할 변수와 invalid 페이지 수를 저장할 변수를 초기화 한다. 이 후 모든 블록을 순회해서 invalid 페이지의 개수를 계산한다.

가장 invalid 페이지가 많은 블록을 희생할 블록으로 선정하는데, 동일한 개수일 경우 인덱스가 더 낮은 쪽을 선정한다.

```
    if (victim == -1) return; // 적절한 victim 블록이 없으면 GC를 수행하지 않고 종료

    // 새로운 free 블록을 찾기 (victim 블록은 제외)
    int new_block = -1;  // 새로 할당할 블록 번호 (-1은 아직 없음)
    for (int b = 0; b < total_blocks; ++b) {
        if (b == victim) continue; // victim 블록은 제외

        // 해당 블록의 모든 페이지가 FREE 상태인 경우 선택
        if (std::all_of(blocks[b].pages.begin(), blocks[b].pages.end(),
                        [](const Page& pg) { return pg.state == FREE; })) {
            new_block = b;
            break;
        }
    }

    if (new_block == -1) return; // 복사 대상 블록이 없으면 GC 종료

    current_block = new_block;  // 새 블록을 활성 블록으로 설정
    current_page = 0;           // 페이지 인덱스도 초기화
```

유효한 victim 블록이 없으면 수행을 종료한다.

Victim을 제외한 블록들 중에서 모든 페이지가 free인 블록을 찾아서 new_block으로 저장 및 활성 블록으로 설정해준다. 또한 write를 시작할 페이지에 대한 정보도 0으로 초기화 한다.



```
    // victim 블록에서 VALID 페이지들을 새 블록으로 복사
    for (const Page& pg : blocks[victim].pages) {
        if (pg.state == VALID) { // VALID 상태인 페이지만 복사
            if (current_page >= block_size) break; // 새 블록에 빈 페이지가 없으면 복사 중단
            if (pg.logical_page_num < 0 || pg.logical_page_num >= static_cast<int>(L2P.size())) continue; // 논리 번호가 유효 범위를 벗어나면 무시

            Page& tgt = blocks[current_block].pages[current_page]; // 복사 대상 페이지 참조
            tgt.state = VALID;                // 상태를 VALID로 설정
            tgt.logical_page_num = pg.logical_page_num; // 논리 페이지 번호 복사
            tgt.data = pg.data;               // 실제 데이터 복사

            int new_ppn = current_block * block_size + current_page; // 새 물리 페이지 번호 계산
            L2P[pg.logical_page_num] = new_ppn;  // 논리->물리 매핑 갱신
            current_page++;                      // 다음 페이지로 이동
            total_physical_writes++;             // 물리 쓰기 횟수 증가 (WAF 계산용)
        }
    }

    // victim 블록 초기화: 모든 페이지를 FREE 상태로 설정
    for (auto& pg : blocks[victim].pages) {
        pg.state = FREE;            // 상태 초기화
        pg.logical_page_num = -1;   // 논리 페이지 번호 제거
        pg.data = 0;                // 데이터 제거
    }

    blocks[victim].gc_cnt++; // victim 블록이 GC된 횟수 증가
}
```

이후 victim 블록의 모든 valid한 페이지를 새 블록에 복사 준비를 하는데 새로운 블록에 오류가 있는 경우는 멈추거나 무시해준다.

새로운 블록의 현재 페이지에 앞에서 말한 valid한 페이지의 데이터를 복사해주고, 복사된 데이터에 대한 새로운 PPN을 계산해서 L2P를 갱신과 동시에 physical write 횟수와 페이지 인덱스를 증가시킨다.

마지막에 victim 블록의 모든 페이지를 초기화하고, victim 블록의 GC 횟수를 증가시킨다.

-greedyWrite

```
// writePage: 유저의 write 요청을 수행하고 필요 시 GC 수행
void GreedyFTL::writePage(int lpn, int data) {
    if (lpn < 0 || lpn >= static_cast<int>(L2P.size())) return; // 논리 주소가 유효한지 확인

    total_logical_writes++; // 논리 쓰기 횟수 증가 (WAF 계산용)

    if (L2P[lpn] != -1) { // 이미 매핑된 물리 페이지가 있다면
        int old_ppn = L2P[lpn]; // 기존 물리 페이지 번호 가져오기
        if (old_ppn >= 0 && old_ppn < total_blocks * block_size) { // 유효한 물리 페이지인지 확인
            blocks[old_ppn / block_size].pages[old_ppn % block_size].state = INVALID; // 기존 물리 페이지를 INVALID 처리
        }
    }

    int free_blocks = 0; // free 블록 수 초기화
    for (const auto& blk : blocks) { // 모든 블록을 순회하며
        if (std::all_of(blk.pages.begin(), blk.pages.end(), [](const Page& pg) { return pg.state == FREE; })) {
            free_blocks++; // 해당 블록이 전부 FREE 상태이면 free 블록 수 증가
        }
    }

    if (free_blocks <= 2 && current_page >= block_size && !in_gc) { // free 블록이 2개 이하이고 현재 블록이 꽉 찼다면
        in_gc = true; // GC 재귀 방지 플래그 설정
        garbageCollect(); // GC 수행
        in_gc = false; // 플래그 해제
    }
```

주소가 유효한지 먼저 확인을 하고, 확인 후 logical write 횟수를 증가시킨다.

L2P[lpn] 값 확인을 통해, 이미 물리 페이지에 작성된 것인지 확인을 하고, 그 페이지를 invalid 처리를 한다. 또한 모든 블록들을 확인해서, 페이지들이 모두 free 상태인 블록의 개수를 저장한다.

이후 남은 free 블록이 2개 이하 혹은 현재 블록이 꽉 차면 GC 함수를 호출해서 수행시킨다. (In_gc를 통해서 재귀 호출을 방지한다.)

```
    if (current_page >= block_size || // 현재 블록이 가득 찼거나
        current_block < 0 || current_block >= total_blocks || // 현재 블록 인덱스가 유효하지 않거나
        blocks[current_block].pages[current_page].state != FREE) { // 현재 페이지가 FREE 상태가 아니면
        for (int b = 0; b < total_blocks; ++b) { // 블록을 순회하면서
            if (std::all_of(blocks[b].pages.begin(), blocks[b].pages.end(), [](const Page& pg) { return pg.state == FREE; })) {
                current_block = b; // 새로운 블록으로 설정
                current_page = 0; // 페이지 인덱스 초기화
                break;
            }
        }
    }

    if (current_block >= total_blocks || current_page >= block_size) return; // 여전히 유효한 블록/페이지가 없으면 종료

    Page& pg = blocks[current_block].pages[current_page]; // 현재 페이지 참조
    pg.state = VALID; // 상태를 VALID로 설정
    pg.logical_page_num = lpn; // 논리 페이지 번호 저장
    pg.data = data; // 데이터 저장

    int ppn = current_block * block_size + current_page; // 물리 페이지 번호 계산
    L2P[lpn] = ppn; // 논리 -> 물리 매핑 갱신
    total_physical_writes++; // 물리 쓰기 횟수 증가 (WAF 측정용)
    current_page++; // 다음 페이지로 이동
}
```

여기서는 현재 사용하려는 블록이 가득 찼거나, 현재 블록이 유효하지 않는 인덱스를 가지거나 혹은 현재 페이지가 free 상태가 아니면 전체 블록을 다시 순회하면서 비어 있는 블록을 찾아 변수를 재설정한다.

그 다음은 페이지에 실제로 데이터를 작성하는 것인데 현재 블록의 현재 페이지를 선택한 후, valid 상태로 설정하고, 어떤 논리 주소의 데이터인지를 저장하고 실제 데이터를 저장해준다.

마지막으로 물리 페이지에 대한 정보를 계산하고 L2P에 저장해준다. 그리고 physical write 횟수와 현재 페이지 인덱스를 증가시킨다.

-greedyRead

```
// readPage: 논리 페이지를 물리 위치에서 읽기
void GreedyFTL::readPage(int lpn) {
    if (lpn < 0 || lpn >= static_cast<int>(L2P.size())) return; // 논리 주소 유효성 확인

    int ppn = L2P[lpn]; // 물리 페이지 번호 조회
    if (ppn < 0 || ppn >= total_blocks * block_size) return; // 물리 주소가 유효한지 확인

    Page& pg = blocks[ppn / block_size].pages[ppn % block_size]; // 해당 물리 페이지 접근
    if (pg.state == VALID) // VALID 상태일 경우
        std::cout << "Read LPN " << lpn << ": " << pg.data << std::endl; // 데이터 출력
}
```

먼저 유효성 검사를 진행한다. 그런 다음에 L2P 배열을 통해서 물리 페이지 번호를 조회하고, 유효성을 검사해준다. 마지막으로 해당 페이지에 접근해서, valid 여부를 확인하고 데이터를 출력한다.

-CostBenefit

생성자 구현.

```
// CostBenefitFTL 구현부
CostBenefitFTL::CostBenefitFTL(int total_blocks, int block_size)
    : FlashTranslationLayer(total_blocks, block_size), // FTL 기본 초기화
      current_block(0), // 현재 활성 블록 초기값
      current_page(0), // 현재 페이지 인덱스 초기값
      in_gc(false) // GC 중 여부 초기화
{
    name = "CostBenefitFTL"; // FTL 이름 지정
    blocks.clear(); // 블록 벡터 초기화
    blocks.resize(total_blocks); // 총 블록 수만큼 크기 지정
    for (auto& blk : blocks) blk.pages.resize(block_size); // 각 블록에 페이지 벡터 크기 지
    L2P.clear(); // 논리->물리 매핑 테이블 초기화
    L2P.resize(total_blocks * block_size, -1); // 매핑 테이블 크기 지정 및 초기값 -1
}
```

Greedy와 마찬가지로 헤더파일내에서 구현하는 것을 실패하여, cpp파일에 구현하였다.

위에 생성자 초기화와 똑같이 현재 상태를 나타내는 변수들을 초기화해준다. 그리고 전체 블록 수만큼 blocks의 크기를 조정하고, L2P 테이블을 전체 페이지 수만큼 할당한다.

-garbageCollect


```
void CostBenefitFTL::garbageCollect() {
    int victim = -1; // victim 블록 인덱스 초기값
    double best_score = -1.0; // 최고 점수 초기값
    for (int b = 0; b < total_blocks; ++b) { // 모든 블록을 순회하며
        int vcnt = 0, icnt = 0; // VALID, INVALID 페이지 수 카운터 초기화
        for (const auto& pg : blocks[b].pages) {
            if (pg.state == VALID)   vcnt++; // VALID일 경우 증가
            if (pg.state == INVALID) icnt++; // INVALID일 경우 증가
        }
        if (icnt == 0) continue; // INVALID 페이지가 없으면 스킵
        double util = static_cast<double>(vcnt) / block_size; // utilization 계산
        double score = icnt / (1.0 + util); // cost-benefit 점수 계산
        if (score > best_score) { // 더 나은 점수면 victim 업데이트
            best_score = score; // 최고 점수 갱신
            victim = b; // victim 블록 갱신
        }
    }
    if (victim < 0) return; // victim을 못 찾으면 종료
```

먼저 초기 변수를 설정하는데, victim은 gc대상 블록 번호를 말하고

best_score는 지금까지 가장 높은 Cost_benefit 점수를 말하는 변수다. 먼저 모든 블록 순회를 통해서 각 블록에 대해 Valid와 Invalid 페이지 수를 계산한다.

구한 페이지 수들을 사용하여 블록 내 valid 페이지 비율을 구하고, benefit 점수를 구한다. 가장 높은 점수를 가진 블록을 victim으로 선정한다.

```
    }
    if (victim < 0) return; // victim을 못 찾으면 종료

    for (const auto& pg : blocks[victim].pages) { // victim 블록에서
        if (pg.state == VALID) { // VALID 페이지만
            writePage(pg.logical_page_num, pg.data); // 다시 쓰기
        }
    }
    for (auto& pg : blocks[victim].pages) { // victim 블록을 초기화
        pg.state = FREE; // 상태를 FREE로 설정
        pg.logical_page_num = -1; // 논리 페이지 번호 초기화
        pg.data = 0; // 데이터 초기화
    }
    blocks[victim].gc_cnt++; // GC 카운터 증가
    current_block = victim; // victim을 새로운 활성 블록으로 지정
    current_page = 0; // 페이지 인덱스 초기화
}
```

Victim 블록의 valid 페이지에 대한 write를 실행시켜서 새로운 위치에 복사를 시키고, victim 블록을 free상태로 초기화 시킨다.

그런 다음, gc 횟수를 증가시키고 victim 블록을 새로운 활성 블록으로 설정한다.

-costWrite

```
user32220393@ubuntuSYSPR
```

```
void CostBenefitFTL::writePage(int lpn, int data) {
    if (lpn < 0 || lpn >= static_cast<int>(L2P.size())) return; // 논리 주소 유효성 검사
    total_logical_writes++; // 논리 쓰기 횟수 증가
    if (L2P[lpn] != -1) { // 기존 매핑 존재 시
        auto& old = blocks[L2P[lpn] / block_size] // 이전 페이지 접근
                          .pages[L2P[lpn] % block_size]; // 해당 페이지 위치 계산
        old.state = INVALID; // INVALID 처리
    }
    int free_blocks = 0; // free 블록 수 초기화
    for (const auto& blk : blocks) {
        if (std::all_of(blk.pages.begin(), blk.pages.end(), [](const Page& p){ return p.state == FREE; }))
            free_blocks++; // 완전히 비어있는 블록 수 계산
    }
    if (free_blocks <= 2 && !in_gc) { // GC 조건 확인
        in_gc = true; // GC 시작 표시
        garbageCollect(); // GC 실행
        in_gc = false; // GC 종료
    }
```

먼저 lpn에 대해 유효성 검사를 진행하고 logical write 횟수를 증가시킨다. 이 후, 해당 LPN이 이미 할당된 경우, 해당 물리 페이지를 invalid하게 상태를 변경시킨다.

GC 조건 확인을 위해서 모든 페이지가 free 상태인 블록의 수를 계산하고, 그 수가 2개 이하인 경우에 GC함수를 호출한다. In_gc를 통해서 재귀 호출을 방지한다.

```
    }
    bool found = false; // 유효한 위치 찾음 여부
    for (int i = 0; i < total_blocks * block_size; ++i) { // 전체 블록을 원형 탐색
        if (blocks[current_block].pages[current_page].state == FREE) { // 빈 페이지 찾기
            found = true; // 찾음 표시
            break; // 반복 종료
        }
        current_page = (current_page + 1) % block_size; // 다음 페이지로 이동
        if (current_page == 0)
            current_block = (current_block + 1) % total_blocks; // 다음 블록으로 이동
    }
    if (!found) return; // 빈 공간이 없으면 종료

    Page& pg = blocks[current_block].pages[current_page]; // 현재 페이지 참조
    pg.state = VALID; // VALID 상태로 설정
    pg.logical_page_num = lpn; // 논리 페이지 번호 기록
    pg.data = data; // 데이터 저장
    L2P[lpn] = current_block * block_size + current_page; // L2P 매핑 갱신
    total_physical_writes++; // 물리 쓰기 횟수 증가
    current_page++; // 페이지 포인터 증가
}
```

새로운 빈 페이지를 전체 공간을 돌며 탐색해서, 다음 쓸 수 있는 위치를 찾는다. 만약 빈공간을 찾지 못하면 write를 종료를 시킨다.

찾았을 경우, write를 수행한다. 페이지 상태를 valid로 변경하고 데이터 저장 및 논리 페이지 번호를 기록한다.

마지막으로 물리 페이지에 대한 정보를 계산하고 L2P에 저장해준다. 그리고 physical write 횟수와 현재 페이지 인덱스를 증가시킨다.

-costRead

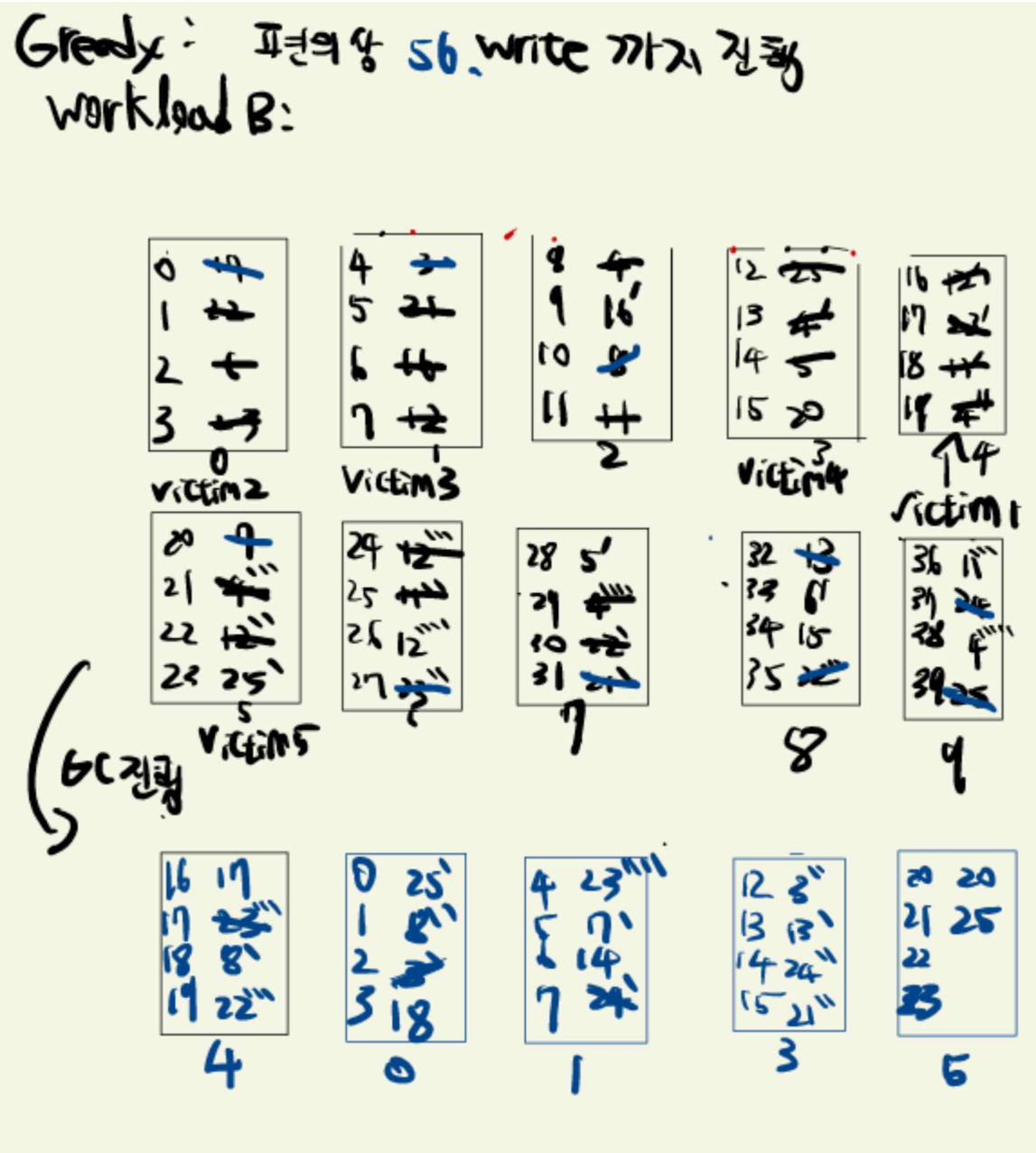
```
void CostBenefitFTL::readPage(int lpn) {
    if (lpn < 0 || lpn >= static_cast<int>(L2P.size())) return; // 유효성 검사
    int ppn = L2P[lpn]; // 물리 페이지 번호 조회
    if (ppn < 0) return; // 매핑이 없으면 종료
    auto& pg = blocks[ppn / block_size].pages[ppn % block_size]; // 페이지 접근
    if (pg.state == VALID) // VALID 상태면
        std::cout << "Read LPN " << lpn << ": " << pg.data << std::endl; // 출력
}
```

Greedy와 비슷하게 유효성 검사를 먼저 진행한다. Lpn과 ppn에 대해서 유효성 확인을 하고 난 후, 페이지 접근을 해서 valid 인지 확인을 한다.

모든 과정을 거친 후에 유효한 데이터만 출력시킨다.

문제풀이

Greedy 경우:



B보다 C가 WAF가 낮게 1.0에 가깝게 나오는 걸 알 수 있었다.

Discussion

-work load B/ Greedy

```
[ RUN      ] Default/FTLTest.Greedy/1

[ FTL Type: Greedy, Workload: B ]


[ Flash Memory Status ]
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 | Block 9 |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  22 V   |  21 V   |  24 V   |   6 V   |         |   1 V   |  12 V   |   7 V   |  15 V   |         |
|  14 V   |  18 V   |   8 V   |   5 V   |         |  11 V   |  10 V   |  25 V   |  11 I   |         |
|  13 I   |  17 V   |  23 V   |   9 V   |         |   0 V   |   4 V   |  23 I   |   2 V   |         |
|  19 V   |  16 V   |   9 I   |         |         |  13 V   |  22 I   |  20 V   |   3 V   |         |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  GC:25  |  GC:21  |  GC:21  |  GC:16  |  GC:21  |  GC:16  |  GC:14  |  GC:19  |  GC:16  |  GC:0   |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+

[ LBA to PPN Mapping Table ]
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|LBA:|   0|   1|   2|   3|   4|   5|   6|   7|   8|   9|  10|  11|  12|  13|  14|  15|  16|  17|  18|  19|  20|  21|  22|  23|  24|  25|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|PPN:|  22|  20|  34|  35|  26|  13|  12|  28|   9|  14|  25|  21|  24|  23|   1|  32|   7|   6|   5|   3|  31|   4|   0|  10|   8|  29|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

[ WAF Statistics ]
Total Logical Writes: 400.000
Total Physical Writes: 707.000
Write Amplification Factor (WAF): 1.768
```

Total Logical Writes: 400

Total Physical Writes: 707

WAF=1.768(707/400)

총 GC 횟수: 169회

GC 분산 정도: Block 0을 제외하고 전반적으로 고르게 GC가 진행되었지만, WAF가 2에 가까울정도로 비효율적이다.

-work load C/ Greedy

```
[       OK ] Default/FTLTest.Greedy/1 (0 ms)
[ RUN      ] Default/FTLTest.Greedy/2

[ FTL Type: Greedy, Workload: C ]


[ Flash Memory Status ]
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 | Block 9 |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  16 V   |  18 V   |  15 V   |   6 V   |  19 V   |   8 V   |         |  10 V   |  14 V   |         |
|  24 V   |  23 V   |   5 V   |  22 V   |   3 I   |  25 V   |         |   3 V   |  13 V   |         |
|  21 V   |   0 I   |   1 I   |  20 V   |   9 V   |   0 V   |         |   2 V   |   2 I   |         |
|   1 I   |   1 V   |  17 V   |   1 I   |   4 V   |  12 V   |         |         |   3 I   |         |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  GC:25  |  GC:31  |  GC:9   |  GC:13  |  GC:17  |  GC:12  |  GC:14  |  GC:10  |  GC:12  |  GC:0   |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+


[ LBA to PPN Mapping Table ]
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|LBA:|   0|   1|   2|   3|   4|   5|   6|   7|   8|   9|  10|  11|  12|  13|  14|  15|  16|  17|  18|  19|  20|  21|  22|  23|  24|  25|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|PPN:|  22|   7|  30|  29|  19|   9|  12|   -|  20|  18|  28|   -|  23|  33|  32|   8|   0|  11|   4|  16|  14|   2|  13|   5|   1|  21|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

[ WAF Statistics ]
Total Logical Writes: 400.000
Total Physical Writes: 603.000
Write Amplification Factor (WAF): 1.508
```

Total Logical Writes: 400

Total Physical Writes: 603

WAF=1.508(603/400)

총 GC 횟수: 143회

WAF는 workload B보다 낮아서 어느정도 효율적으로 진행되었지만, Block 0번이랑 1번이 과하게 재사용 되어서 형평성 문제가 발생한다. (hot: 0,1)

-work load B/ Cost_Benefit

```
[ RUN      ] Default/FTLTest.Cost_benefit/1

[ FTL Type: Cost-Benefit, Workload: B ]


[ Flash Memory Status ]
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 | Block 9 |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|   9 V   |  20 V   |   0 V   |  21 V   |  13 V   |  11 V   |  22 V   |  23 V   |         |         |
|         |   5 V   |  14 V   |  25 V   |  18 V   |         |  10 V   |  19 V   |         |         |
|         |   1 V   |   8 V   |   3 V   |   4 V   |         |  24 V   |  16 V   |         |         |
|         |  15 V   |  12 V   |   6 V   |   2 V   |         |   7 V   |  17 V   |         |         |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  GC:52  |  GC:53  |  GC:45  |  GC:42  |  GC:48  |  GC:52  |  GC:40  |  GC:31  |  GC:0   |  GC:0   |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+


[ LBA to PPN Mapping Table ]
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|LBA:|  0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23| 24| 25|
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|PPN:|  8|  6| 19| 14| 18|  5| 15| 27| 10|  0| 25| 20| 11| 16|  9|  7| 30| 31| 17| 29|  4| 12| 24| 28| 26| 13|
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

[ WAF Statistics ]
Total Logical Writes: 1348.000
Total Physical Writes: 1348.000
Write Amplification Factor (WAF): 1.000
[       OK ] Default/FTLTest.Cost_benefit/1 (0 ms)
```

Total Logical Writes: 1348

Total Physical Writes: 1348

WAF=1.00(1348/1348)

총 GC 횟수: 363회

WAF=1 로 매우 효율적으로 수행되었으며, Gc 횟수는 363회로 Greedy보다 2배이상 많고, 전체적으로 GC가 고르게 실행되었다.

-work load C/ Cost_Benefit

```
[ RUN      ] Default/FTLTest.Cost_benefit/2

[ FTL Type: Cost-Benefit, Workload: C ]


[ Flash Memory Status ]
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 | Block 9 |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|   5 V   |   0 V   |   2 V   |   3 V   |  19 V   |  25 V   |  20 V   |   1 V   |  10 V   |         |
|         |  12 V   |         |  24 V   |  14 V   |         |   9 V   |  13 V   |         |         |
|         |  23 V   |         |  16 V   |   8 V   |         |   4 V   |   6 V   |         |         |
|         |  21 V   |         |  18 V   |  15 V   |         |  17 V   |  22 V   |         |         |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|  GC:20  | GC:105  |  GC:73  |  GC:95  |  GC:5   |  GC:4   |  GC:5   |  GC:49  |  GC:0   |  GC:0   |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+

[ LBA to PPN Mapping Table ]
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|LBA:|  0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23| 24| 25|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|PPN:|  4| 28|  8| 12| 26|  0| 30|  -| 18| 25| 32|  -|  5| 29| 17| 19| 14| 27| 15| 16| 24|  7| 31|  6| 13| 20|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

[ WAF Statistics ]
Total Logical Writes: 1013.000
Total Physical Writes: 1013.000
Write Amplification Factor (WAF): 1.000
[       OK ] Default/FTLTest.Cost_benefit/2 (0 ms)
```

Total Logical Writes: 1013

Total Physical Writes: 1013

WAF=1.00(1013/1013)

총 GC 횟수: 356회

B와 마찬가지로 WAF=1이 나와 매우 효율적으로 수행되었으며, Gc 횟수는 356회로 Greedy보다 2배이상 많다. 또한 Greedy/C와 비슷하게 몇개의 block에 GC가 집중된 것을 볼 수 있다. 이 때문에 Wear-leveling 관점에서 성능이 안좋다. (hot: 1,2,3,7)

총 정리 및 느낀점

WAF 측면에서 보면 Cost_Benefit 이 절대적인 우세를 가지는 걸 볼 수 있지만, GC가 Greedy에 비해 훨씬 자주 일어나므로 GC면에서는 Greedy가 성능이 좋다.

Wear-Leveling 측면에서 보면, Workload B가 C보다 고르게 GC가 진행되는 것을 확인 할 수 있다.

이번 학기 운영체제 수업을 통해서 Scheduling과제랑 FTL 과제가 수업 내용을 이해하는데 크게 도움이 되었다고 생각한다.

특히, FTL 과제는 SSD에 대한 구조를 이해하는데 크게 도움이 되었고, Scheduling과제에서도 마찬가지로 각각의 정책들을 이해하는데 많은 도움을 받았다.

개인적으로 평소에 궁금하던 파일 시스템과 디바이스 드라이버에 대해 많은 것을 배울수 있어서 뜻깊었다.