곽민석 - 20183380
2023년 12월 4일 월요일

운영체제

페이지 교체 알고리즘 시뮬레이션 구현과 비교

1. 과제 설명

페이지 교체 알고리즘 Optimal, FIFO, LRU, Second-Chance(One-handed Clock)을 모두 구현하고, 각 알고리즘별로 분석하는것이 이 과제의 내용이다.

FIFO는 원형큐를 이용하여 가장 먼저 들어온 페이지를 교체 페이지로 이용하는 방식이다. 이 방식은 추후 Second-Chance에서 사용된다.

Second-Chance는 FIFO를 이용하여 One-handed Clock 방식을 이용하여 교체 프레임으로 지정되어도 기회를 한번 주도록 구현하였다.

LRU는 언제 참조가 되었는지 확인할수 있는 배열을 두어 최근 사용수가 가장 적은 페이지를 교체하는 방식으로 구현하였다.

마지막으로 Optimal은 들어올 모든 페이지를 계산하여 가장 쓰이지 않을 페이지를 교체하도록 구현하였다.

실행 후 옵션을 입력하게 되면 옵션에 맞게 결과가 출력되며, 출력된 내용은 파일로 저장된다. 만약 파일을 입력하지 않고, 생성하는 방식으로 실행하는 경우 “input.in" 파일로 저장된다.

2. 설치 및 실행 방법

A. Application Compile and Execution

1. 프로젝트 폴더에서 “make” 명령어를 이용하여 컴파일 한다.
2. “./final”를 실행한다.

4. 코드 설명

A. get_file_contents

```c
// 파일을 읽어 내용을 반환하는 함수.
void get_file_contents(char *path_file, int *arr_mem_acc) {
    FILE *file = fopen(path_file, "r");

    // 파일에서 줄 단위로 읽어와 배열에 저장
    char buffer[1024];

    for(int i = 0;i < MAX_INPUT;i++) {
        fgets(buffer, sizeof(buffer), file);
        arr_mem_acc[i] = atoi(buffer);
    }

    fclose(file);
}
```

해당 함수는 기존에 정의된 페이지 요구 파일을 가져와 배열에 할당하는 함수이다. 지정된 파일의 이름을 읽기 전용으로 가져와 배열에 정수로 저장한다.

B. get_random_input

```c
// MAX 값에 맞는 무작위 정수형 배열로 반환하는 함수.
void get_random_input(int arr_mem_acc[], int max) {
    FILE *file = fopen("input.in", "w");
    int tmp;
    int min = 0;
    // int min = max - (max / 16);
    printf("%d %d\n", min, max);

    for(int i = 0;i < MAX_INPUT;i++) {
        tmp = rand() % (max - min + 1) + min;
        arr_mem_acc[i] = tmp;
        fprintf(file, "%d\n", tmp);
    }

    fclose(file);
}
```

해당 함수는 페이지 요구를 랜덤함수를 이용하여 생성하는 함수이다. 주석처리된 변수 min을 통해 일정 구간 내에서만 요구하도록 만들수 있다. 또한 변수 max를 받아 메모리 주소로 표현 가능한 범위까지만 생성하도록 제작하였다.

C. save_log

```
// 배열의 내용을 저장하는 함수.
void save_log(char *name_file, char **contents, int op_head) {
    FILE *file = fopen(name_file, "w");

    // is_head가 1이면 배열의 첫 번째 항목을 파일에 쓰기.
    if (op_head == 1) {
        fprintf(file, "%-10s %-10s %-10s %-10s %-10s %-10s\n", "No.", "V.A", "Page No.",
"Frame No.", "P.A.", "Page Fault");
    }

    // 배열의 나머지 내용을 파일에 쓰기.
    for (int i = 0;i < MAX_INPUT + 1; i++) {
        fprintf(file, "%s", contents[i]);
    }

    fclose(file);
}
```

해당 함수는 지금까지 출력된 내용을 파일로 저장하는 함수이다. 파일의 이름은 옵션에 맞게 저장되도록 함수 main에서 정의하였다.

D. get_frame_offset

```
// 프레임 내의 오프셋을 구하는 함수.
int get_frame_offset(int size_frame) {
    int offset = 0;

    if(size_frame == 1) offset = 10;
    else if(size_frame == 2) offset = 11;
    else if(size_frame == 4) offset = 12;

    return offset;
}
```

해당 함수는 프레임 크기에 맞게 프레임 주소 크기를 받는 함수이다. 18bit의 경우 10자리, 19bit의 경우 11자리 그리고 20bit의 경우 12자리가 할당되도록 하였다.

E. get_page_number

```
// 가상 주소를 페이지 주소로 변환하는 함수.
int get_page_number(int size_frame, int va) {
    int offset = get_frame_offset(size_frame);

    return va >> offset;
}
```

해당 함수는 프레임 사이즈에 맞게 가상 주소에서 페이지 번호를 가져오는 함수이다. 해당 함수는 비트 이동 연산자를 이용하여 계산한다.

F. get_physical_addr

```
// 가상 주소를 실제 주소로 변환하는 함수.
int get_physical_addr(int size_frame, int num_frame, int va) {
    int offset = get_frame_offset(size_frame);
    int num_page_base = get_page_number(size_frame, va);

    return va - (num_page_base << offset) + (num_frame * size_frame * 1024);
}
```

해당 함수는 프레임 크기와 프레임 번호에 따른 가상 주소를 실제 주소로 변환하는 함수이다.

G. find_next_page

```
// 다음에 얼마나 뒤에 나오는지 구하는 함수.
void find_next_page(int array[], int next_occurrences[]) {
    const int NOT_FOUND = 99999;

    // 배열을 순회하면서 각 원소가 다음으로 나오는 인덱스를 찾기
    for (int i = 0; i < MAX_INPUT; i++) {
        int found_index = NOT_FOUND;

        // 현재 원소 이후에서 해당 원소를 찾기
        for (int j = i + 1; j < MAX_INPUT; j++) {
            if (array[j] == array[i]) {
                found_index = j;
                break; // 찾으면 반복 중단
            }
        }

        next_occurrences[i] = found_index;
    }
}
```

해당 함수는 Optimal 알고리즘 내의 기능 중 해당 프레임이 이후 언제 나오는지 계산하기 위한 함수이다. 만약 이후 사용되지 않는 경우 99999를 할당한다.

H. FIFO

```c
// FIFO 페이지 교체 알고리즘 시뮬레이션 함수.
char ** FIFO(int *arr_input, int len_va, int size_frame, int size_pm, int op_input) {
    // 기록 결과 배열.
    int cnt_page_fault = 0;
    char **result = (char **)malloc(sizeof(char **) * MAX_INPUT + 1);
    for(int i = 0;i < MAX_INPUT + 1;i++) result[i] = (char *)malloc(sizeof(char *) *
LEN_RESULT);

    // 현재 참조하고 있는 위치.
    int num_frame = 0;

    // 페이지 갯수 구하기.
    int max_pages = size_pm / size_frame;
    int *pages = (int *)malloc(sizeof(int) * max_pages);

    // page 초기화.
    for(int i = 0;i < max_pages;i++) {
        pages[i] = -1;
    }

    for(int i = 0;i < MAX_INPUT;i++) {
        int num_phyx;
        int is_hit = -1;
        int num_page = get_page_number(size_frame, arr_input[i]);

        // HIT 확인.
        for(int j = 0;j < max_pages;j++) {
            // printf("Page %d = %d\n", j + 1, pages[j]);
            if(pages[j] == num_page) {
                is_hit = j;
                break;
            }
        }

        if(is_hit != -1) num_phyx = get_physical_addr(size_frame, is_hit, arr_input[i]);
        else num_phyx = get_physical_addr(size_frame, num_frame % max_pages,
arr_input[i]);

        // HIT가 아닌 경우 페이지 교체.
        if(is_hit == -1) {
            is_hit = 'F';
            pages[num_frame % max_pages] = num_page;
            // printf("%-10d %-10d %-10d %-10d %-10d %-10d\n", i + 1, arr_input[i],
num_page, (num_frame % max_pages), num_phyx, is_hit);
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, (num_frame % max_pages),
                    num_phyx, (char)is_hit);
            num_frame++;
            cnt_page_fault++;
        }
```

```
        else {
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, is_hit,
                    num_phyx, 'H');
        }
    }
    sprintf(result[MAX_INPUT], "Page Fault count: %5d\n", cnt_page_fault);

    return result;
}
```

해당 함수는 FIFO 알고리즘을 시뮬레이션 하기 위한 함수이다. 동적배열을 이용하여 환경에 맞게 크기를 할당한 후 이후 순환큐와 같이 동작하도록 고안하였다. HIT 여부에 따라 페이지 교체 필요 여부를 판단하며, 교체가 필요한 경우 순환큐의 동작과 같이 순번이 온 공간에 할당한다.

I. LRU

```
// LRU 페이지 교체 알고리즘 시뮬레이션 함수.
char ** LRU(int *arr_input, int len_va, int size_frame, int size_pm, int op_input) {
    // 기록 결과 배열.
    int cnt_page_fault = 0;
    char **result = (char **)malloc(sizeof(char **) * MAX_INPUT + 1);
    for(int i = 0;i < MAX_INPUT + 1;i++) result[i] = (char *)malloc(sizeof(char *) *
LEN_RESULT);

    // 페이지 갯수 구하기.
    int max_pages = size_pm / size_frame;
    int *pages = (int *)malloc(sizeof(int) * max_pages);
    int *ages = (int *)malloc(sizeof(int) * max_pages);

    // page 초기화.
    for(int i = 0;i < max_pages;i++) {
        pages[i] = -1;
        ages[i] = -9999;
    }

    for(int i = 0;i < MAX_INPUT;i++) {
        int num_page = get_page_number(size_frame, arr_input[i]);
        int pos_least_page = 0;
        int is_hit = -1;
        int num_phyx;
        char is_hit_c = 'H';

        // Hit 검사 및 가장
        for(int j = 0;j < max_pages;j++) {
            if(pages[j] == num_page) {
                ages[j] = 0;
                is_hit = j;
                break;
            }
            else {
                if(ages[pos_least_page] > ages[j]) pos_least_page = j;
            }
        }

        // Hit 실패인 경우 페이지 교체.
        if(is_hit == -1) {
            pages[pos_least_page] = num_page;
            ages[pos_least_page] = 0;
            is_hit = pos_least_page;
            cnt_page_fault++;
            is_hit_c = 'F';
        }
```

```
        // 물리 주소 계산.
        num_phyx = get_physical_addr(size_frame, is_hit, arr_input[i]);

        // 에이징 계산.
        for(int j = 0;j < max_pages;j++) ages[is_hit]--;

        // 출력.
        sprintf(result[i],
                "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                i + 1, arr_input[i],
                num_page, is_hit,
                num_phyx, is_hit_c);
    }
    sprintf(result[MAX_INPUT], "Page Fault count: %5d\n", cnt_page_fault);

    return result;
}
```

해당 함수는 LRU 알고리즘을 시뮬레이션 하기 위한 함수이다. 동적배열을 이용하여 환경에 맞게 프레임과 최근 사용 횟수를 저장하는 배열을 선언한다. 사용 횟수를 저장하는 배열은 ages이며 해당 배열의 숫자는 오래될수록 더 작은수를 가지도록 하였다. HIT되지 않은 경우 가장 오래된 프레임을 교체하도록 하였다.

J. second_chance

```
// Second chance 페이지 교체 알고리즘 시뮬레이션 함수.
char ** second_chance(int *arr_input, int len_va, int size_frame, int size_pm, int
op_input) {
    // 기록 결과 배열.
    int cnt_page_fault = 0;
    char **result = (char **)malloc(sizeof(char **) * MAX_INPUT + 1);
    for(int i = 0;i < MAX_INPUT + 1;i++) result[i] = (char *)malloc(sizeof(char *) *
LEN_RESULT);

    // 현재 참조하고 있는 위치.
    int num_frame = 0;

    // 페이지 갯수 구하기.
    int max_pages = size_pm / size_frame;
    int *pages = (int *)malloc(sizeof(int) * max_pages);
    int *hit = (int *)malloc(sizeof(int) * max_pages);

    // page, hit 여부 초기화.
    for(int i = 0;i < max_pages;i++) {
        pages[i] = -1;
        hit[i] = 0;
    }

    printf("%d %d %d\n", len_va, size_frame, size_pm);

    for(int i = 0;i < MAX_INPUT;i++) {
        int num_phyx;
        int is_hit = -1;
        int num_page = get_page_number(size_frame, arr_input[i]);

        // HIT 확인.
        for(int j = 0;j < max_pages;j++) {
            // printf("Page %d = %d\n", j + 1, pages[j]);
            if(pages[j] == num_page) {
                is_hit = j;
                hit[j] = 1;
                break;
            }
        }

        // HIT이 아닌 경우 페이지 교체.
        if(is_hit == -1) {
            int is_replaced = 0;

            // 현재 순번의 HIT 확인.
            for(int j = num_frame % max_pages;j < max_pages;j++) {
                if(hit[j] == 0) {
                    pages[j] = num_page;
                    hit[j] = 0;
                    is_replaced = 1;
                    break;
                }
                else hit[j] = 0;

                num_frame++;
            }
```

```
            // 페이지 교체 실패시 앞부터 다시 순환.
            if(!is_replaced) {
                for(int j = 0;j < (num_frame % max_pages) + 1;j++) {
                    if(hit[j] == 0) {
                        pages[j] = num_page;
                        hit[j] = 0;
                        break;
                    }
                    else hit[j] = 0;

                    num_frame++;
                }
            }
        }

        if(is_hit == -1) {
            num_phyx = get_physical_addr(size_frame, (num_frame % max_pages),
arr_input[i]);
            // printf("%-10d %-10d %-10d %-10d %-10d %-10d\n", i + 1, arr_input[i],
num_page, (num_frame % max_pages), num_phyx, is_hit);
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, (num_frame % max_pages),
                    num_phyx, 'F');
            num_frame++;
            cnt_page_fault++;
        }
        else {
            num_phyx = get_physical_addr(size_frame, is_hit, arr_input[i]);
            // printf("%-10d %-10d %-10d %-10d %-10d %-10d\n", i + 1, arr_input[i],
num_page, is_hit, num_phyx, is_hit);
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, is_hit,
                    num_phyx, 'H');
        }
    }
    sprintf(result[MAX_INPUT], "Page Fault count: %5d\n", cnt_page_fault);

    return result;
}
```

해당 함수는 Second-chance 알고리즘을 시뮬레이션 하기 위한 함수이다. 동적배열을 이용하여 환경에 맞게 크기를 할당한 후 FIFO와 동일하게 순환큐와 같이 동작하도록 고안하였다. 여기서 추가로 기회 부여를 위해 최근 HIT 여부를 저장하는 배열을 선언하였다. HIT가 되지 않는 경우 최근 사용 여부를 보고, 사용된 경우 순환하며 사용되지 않은 프레임을 찾는다. 이후 검색된 프레임을 이용하여 할당한다.

K. OPT

```
// Optimal 페이지 교체 알고리즘 시뮬레이션 함수.
char ** OPT(int *arr_input, int len_va, int size_frame, int size_pm, int op_input) {
    // 기록 결과 배열.
    char **result = (char **)malloc(sizeof(char **) * MAX_INPUT + 1);
    for(int i = 0;i < MAX_INPUT + 1;i++) result[i] = (char *)malloc(sizeof(char *) *
LEN_RESULT);

    // 페이지 갯수 구하기.
    int max_pages = size_pm / size_frame;
    int *pages = (int *)malloc(sizeof(int) * max_pages);
    int *next_hit = (int *)malloc(sizeof(int) * max_pages);
    int page_num[MAX_INPUT];
    int cnt_page_fault = 0;
    int num_frame = 0;

    // page, hit 여부 초기화.
    for(int i = 0;i < max_pages;i++) {
        pages[i] = -1;
        next_hit[i] = 99999;
    }

    // 페이지 넘버 선 계산.
    for(int i = 0;i < MAX_INPUT;i++) {
        page_num[i] = get_page_number(size_frame, arr_input[i]);
    }
    find_next_page(page_num, page_num);
    for(int i = 0;i < MAX_INPUT;i++) {
        printf("%d\n", page_num[i]);
    }

    for(int i = 0;i < MAX_INPUT;i++) {
        int num_phyx;
        int is_hit = -1;
        int num_page = get_page_number(size_frame, arr_input[i]);

        // HIT 확인.
        for(int j = 0;j < max_pages;j++) {
            // printf("Page %d = %d\n", j + 1, pages[j]);
            if(pages[j] == num_page) {
                is_hit = j;
                next_hit[j] = page_num[i];
                break;
            }
        }

        // HIT이 아닌 경우 페이지 교체 후 출력.
        if(is_hit == -1) {
            int idx_victim = 0;

            // 가장 오래 쓰이지 않을 페이지 찾기.
            for(int j = 1;j < max_pages;j++) {
                if(next_hit[idx_victim] < next_hit[j]) idx_victim = j;
            }
```

```
            // 찾은 페이지 교체.
            pages[idx_victim] = num_page;
            next_hit[idx_victim] = page_num[i];

            num_phyx = get_physical_addr(size_frame, (num_frame % max_pages),
arr_input[i]);
            // printf("%-10d %-10d %-10d %-10d %-10d %-10d\n", i + 1, arr_input[i],
num_page, (num_frame % max_pages), num_phyx, is_hit);
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, (num_frame % max_pages),
                    num_phyx, 'F');
            num_frame++;
            cnt_page_fault++;
        }
        else {
            num_phyx = get_physical_addr(size_frame, is_hit, arr_input[i]);
            // printf("%-10d %-10d %-10d %-10d %-10d %-10d\n", i + 1, arr_input[i],
num_page, is_hit, num_phyx, is_hit);
            sprintf(result[i],
                    "%-10d %-10d %-10d %-10d %-10d %-10c\n",
                    i + 1, arr_input[i],
                    num_page, is_hit,
                    num_phyx, 'H');
        }
    }
    sprintf(result[MAX_INPUT], "Page Fault count: %5d\n", cnt_page_fault);

    return result;
}
```

해당 함수는 Optimal 알고리즘을 시뮬레이션 하기 위한 함수이다. 동적배열을 이용하여 환경에 맞게 크기를 할당한 후 요청 배열 전체를 보고 언제 다시 사용되는지 계산한다. 이후 계산된 결과에 따라 가장 오래동안 사용되지 않을 프레임을 HIT가 되지 않을때 사용하도록 하였다.

L. main

```c
int main() {
    int arr_input[MAX_INPUT];
    int len_va, size_frame, size_pm, op_algo, op_input;
    char path_input_file[1024];
    char **result;

    printf("A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): ");
    scanf("%d", &len_va);
    printf("\n");

    printf("B. Simulation에 사용할 페이지(프레임)의 크기를 선택하기오 (1. 1KB 2. 2KB 3. 4KB): ");
    scanf("%d", &size_frame);
    printf("\n");

    printf("C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): ");
    scanf("%d", &size_pm);
    printf("\n");

    printf("D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오\n");
    printf("(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): ");
    scanf("%d", &op_algo);
    printf("\n");

    printf("E. 가상주소 스트링 입력방식을 선택하시오\n");
    printf("(1. input.in 자동 생성 2. 기존 파일 사용): ");
    scanf("%d", &op_input);
    printf("\n");

    if(op_input == 2) {
        printf("F. 입력 파일 이름을 입력하시오: ");
        scanf("%s", path_input_file);
        printf("%s\n", path_input_file);
        // fgets(path_input_file, sizeof(path_input_file), stdin);
    }

    // 랜덤 함수 초기화.
    srand(time(NULL));

    // 옵션에 따른 파일 입력.
    if(op_input == 1) {
        if(size_frame == 1)
            get_random_input(arr_input, 262144);
        else if(size_frame == 2)
            get_random_input(arr_input, 524288);
        else if(size_frame == 3)
            get_random_input(arr_input, 1048576);
    }
    else if(op_input == 2) {
        get_file_contents(path_input_file, arr_input);
    }

    // 실제 크기로 변환.
    len_va += 17;
    if(size_frame == 3) size_frame = 4;
    size_pm *= 32;
```

```
    switch(op_algo) {
        case 1:
            result = OPT(arr_input, len_va, size_frame, size_pm, op_input);
            break;
        case 2:
            result = FIFO(arr_input, len_va, size_frame, size_pm, op_input);
            break;
        case 3:
            result = LRU(arr_input, len_va, size_frame, size_pm, op_input);
            break;
        case 4:
            result = second_chance(arr_input, len_va, size_frame, size_pm, op_input);
            break;
        default:
            printf("올바른 옵션을 입력해주세요. \n");
            exit(0);
    }

    printf("%-10s %-10s %-10s %-10s %-10s %-10s\n", "No.", "V.A", "Page No.", "Frame No.",
"P.A.", "Page Fault");
    for(int i = 0;i < MAX_INPUT + 1;i++) {
        printf("%s", result[i]);
    }

    switch(op_algo) {
        case 1:
            save_log("output.opt", result, 1);
            break;
        case 2:
            save_log("output.fifo", result, 1);
            break;
        case 3:
            save_log("output.lru", result, 1);
            break;
        case 4:
            save_log("output.sc", result, 1);
            break;
    }

    return 0;
}
```

해당 함수는 main 함수로 입력을 받아 옵션에 맞게 동작하는 함수이다. 입력이 모두 받아진 상황에서 파일 사용 여부에 따라 파일을 읽어온 후 입력에 따라 알고리즘을 선택해 시뮬레이션 한다.

알고리즘 Optimal, FIFO, LRU, Second-Chance(One-handed Clock)은 각각 실행 후 결과를 output.opt, output.fifo, output.lru 그리고 output.sc로 저장된다.

5. 실행 결과 및 비교 평가

아래의 실행은 모두 18bit 주소 길이, 4KB의 페이지 크기 그리고 32KB의 물리 메모리 크기를 가졌을 때를 가정한 실행이다.

A. FIFO

```
No.         V.A         Page No.    Frame No.   P.A.        Page Fault
1           129754      31          0           2778        F
2           761778      185         1           8114        F
3           346228      84          2           10356       F
4           848066      207         3           12482       F
5           345675      84          2           9803        H
6           132343      32          4           17655       F
7           561605      137         5           20933       F
8           895322      218         6           26970       F
9           156459      38          7           29483       F
10          274357      66          0           4021        F
-------------------------------------------------------------------
5000        826869      201         2           11765       F
Page Fault count:  4851
```

Fig 1. FIFO 결과.

B. LRU

```
No.         V.A         Page No.    Frame No.   P.A.        Page Fault
1           129754      31          0           2778        F
2           761778      185         1           8114        F
3           346228      84          2           10356       F
4           848066      207         3           12482       F
5           345675      84          2           9803        H
6           132343      32          4           17655       F
7           561605      137         5           20933       F
8           895322      218         6           26970       F
9           156459      38          7           29483       F
10          274357      66          0           4021        F
-------------------------------------------------------------------
5000        826869      201         0           3573        F
Page Fault count:  4833
```

Fig 2. LRU 결과.

C. Optimal

```
No.         V.A         Page No.    Frame No.   P.A.        Page Fault
1           129754      31          0           2778        F
2           761778      185         1           8114        F
3           346228      84          2           10356       F
4           848066      207         3           12482       F
5           345675      84          2           9803        H
6           132343      32          4           17655       F
7           561605      137         5           20933       F
8           895322      218         6           26970       F
9           156459      38          7           29483       F
10          274357      66          0           4021        F
-------------------------------------------------------------------
5000        826869      201         7           32245       F
Page Fault count:  4064
```

Fig 3. Optimal 결과.

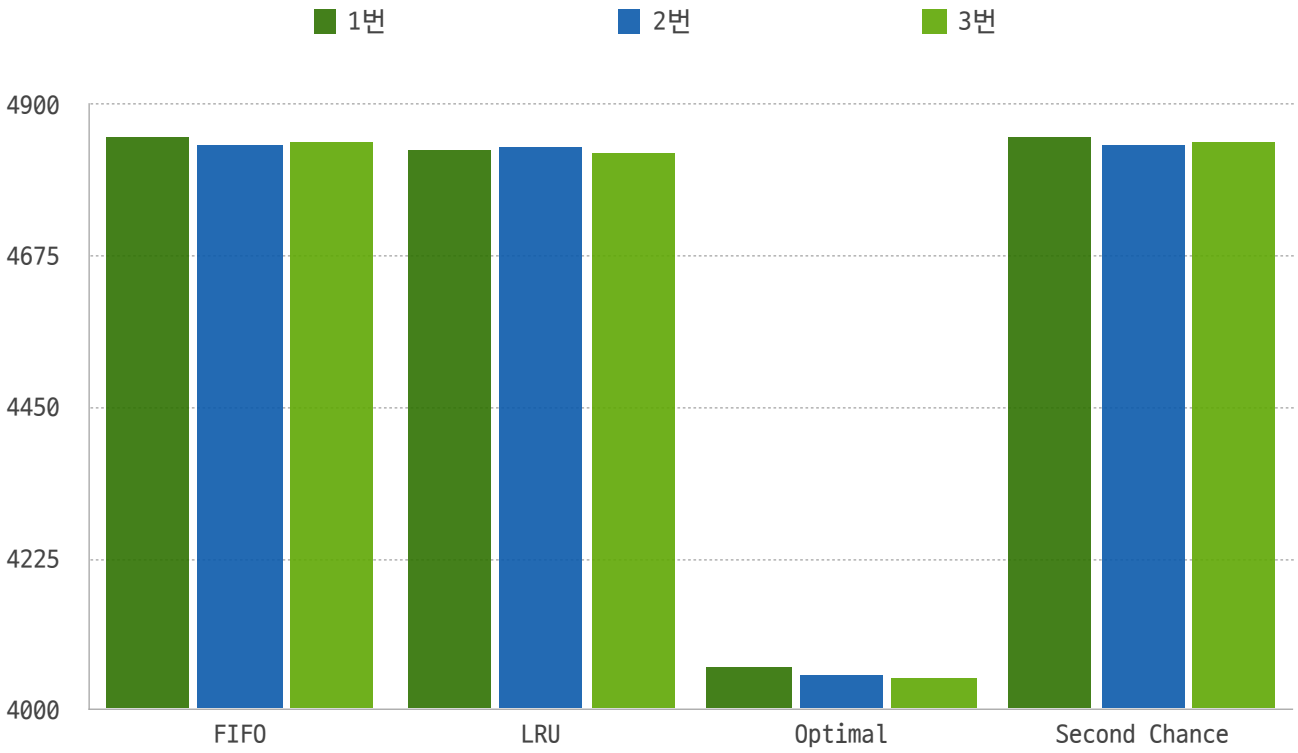
D. Second Chance

```
No.         V.A         Page No.    Frame No.   P.A.        Page Fault
1           129754      31          0           2778        F
2           761778      185         1           8114        F
3           346228      84          2           10356       F
4           848066      207         3           12482       F
5           345675      84          2           9803        H
6           132343      32          4           17655       F
7           561605      137         5           20933       F
8           895322      218         6           26970       F
9           156459      38          7           29483       F
10          274357      66          0           4021        F
-------------------------------------------------------------------
5000        826869      201         5           24053       F
Page Fault count:  4852
```

Fig 4. Second Chance 결과.

E. 알고리즘 비교

각 알고리즘을 5,000개의 요청에 대해 처리했을 때 3번의 시행에서 다음과 같은 Page Fault 횟수가 나왔다.



위의 테스트는 18bit 주소 길이, 4KB의 페이지 크기 그리고 32KB의 물리 메모리 크기를 가졌을 때를 가정한 실험이다. 5,000의 페이지 요청에 대해 Page Fault가 FIFO는 평균 약 4845.7회, LRU는 평균 약 4833회, Optimal은 평균 약 4054.3회 그리고 Second Chance는 평균 약 4845.7회를 기록하였다.

이를 바탕으로 본 결과 Optimal의 성능이 가장 뛰어나고 LRU 그리고 FIFO와 Second Chance 순이였다.

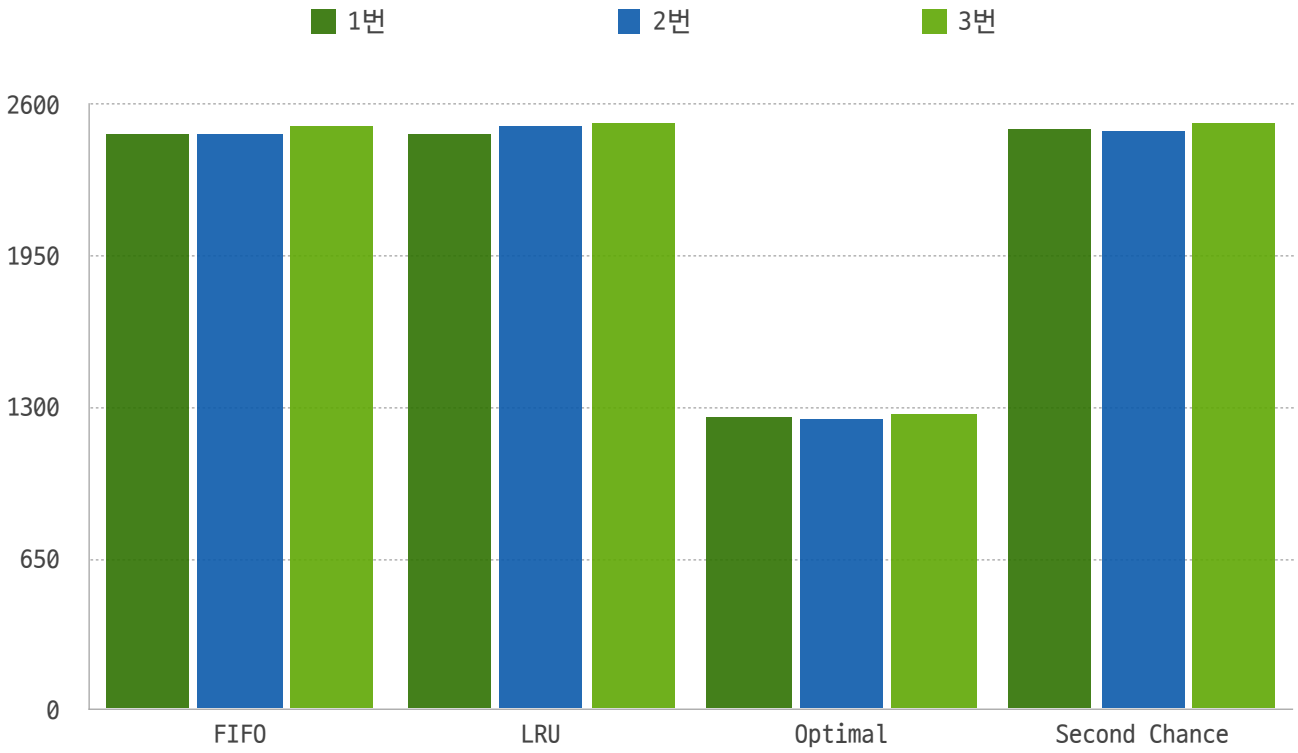
F. 알고리즘 추가 비교

추가로, 위의 실험 환경에서 페이지 요청이 일부 구간에서만 요청이 들어온다고 가정했을 때 어떤 변경점이 있는지 확인해보았다. 테스트에서 입력 옵션은 동일하게 고정한 상태로, 다음의 구간 내에서만 요청을 생성하도록 제한해보았다.

$$MIN = MAX - (MAX \div 16)$$

식 1. 최소 범위를 구하는 식.

위의 식에서 MAX는 메모리 주소의 최대 값이다. 위의 식을 통해 3번 실행한 결과는 다음과 같다.



5,000의 페이지 요청에 대해 Page Fault가 FIFO는 평균 약 2483.3회, LRU는 평균 약 2500.7회, Optimal은 평균 약 1254.3회 그리고 Second Chance는 평균 약 2500회를 기록하였다.

이를 바탕으로 본 결과 Optimal의 성능이 가장 뛰어나고 FIFO 그리고 Second Chance와 LRU 순이였다.

G. 종합

모든 알고리즘 중 Optimal이 모든 환경에서 가장 좋은 성능을 보여주었고 나머지의 알고리즘은 때에따라 다른 결과를 보여주었다.