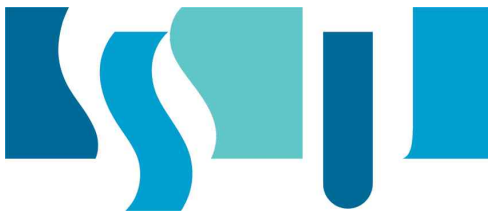


운영체제 과제 5

강의명	운영체제
교수명	김철홍 교수님
학 과	컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.11.21.

< 목차 >

1. 개요

1-0. 요약

제출 파일 목록 : 20172655-5.c, 테스트입력파일(test1, test2, test3)
컴파일 및 실행 : gcc 20172655-5.c, 동일 폴더 내 테스트 파일명 입력하여 실행

1-1. 페이지 교체란

운영체제는 프로세스들에게 독립적인 메모리를 공간을 제공하기 위해 가상 메모리를 사용한다. 프로세스들이 사용 중인 가상 메모리 공간의 총합은 시스템의 실제 메모리 공간보다 더 크므로, 공간적인 제약이 발생하게 된다. 따라서 운영체제는 프로세스의 모든 페이지를 프레임과 매핑시키는 것이 아닌, 요청에 따라 필요한 경우 해당 페이지에 프레임을 할당한다. 이때 사용 가능한 프레임이 존재하지 않을 경우, page replacement algorithm에 따라 교체될 페이지가 결정된다. 시뮬레이터로 구현된 페이지 교체 알고리즘은 Optimal, FIFO, LRU, Second-Chance 총 4가지이다.

1-2. 페이지 교체 알고리즘

OPT, FIFO, LRU, SC의 동작 구현은 다음과 같다.

1) Optimal Algorithm
- 현재 프레임 내 페이지들 중 다음 연산 요청까지의 거리(distance)가 가장 큰 페이지를 새로운 페이지와 교체.
- 사용 가능한 프레임 존재 시 바로 페이지에 할당.
- 다음 연산까지의 거리가 동일한 경우 인덱스가 작은 순으로 선택.

2) First In First Out Algorithm
- 페이지 교체 필요 시 프레임 할당 된 순서대로 새로운 페이지와 교체.
- 사용 가능한 프레임 존재 시 바로 페이지 할당.

3) Least Recently Used Algorithm
- 현 시점에서 가장 오랫동안 사용되지 않은 페이지를 새로운 페이지와 교체
- 사용 가능한 프레임 존재 시 바로 페이지에 할당.
- 사용 시점은 프레임 각각에 카운터 변수를 두어 판단.
- 현재 프레임들 내에 사용할 페이지 존재(히트) 시 카운터 0으로 설정, 페이지 교체 필요 시 카운터 값이 큰 순서로 새로운 페이지와 교체 .
-

4) Second Chance Algorithm
- 모든 프레임에 레퍼런스 비트를 두어 교체 시점에 한 번의 기회를 줌.
- 사용 가능한 프레임 존재 시 바로 페이지에 할당.
- 프레임 할당 시 레퍼런스 비트 값은 0, 이후 히트 시 레퍼런스 비트 값 1로 변경.
- next victim변수를 인덱스로 하여 다음 페이지 교체 대상 판단. 이 때 레퍼런스 비트의 값이 1이면 0으로 교체 후 다음 페이지로 넘어감. 레퍼런스 비트가 0인 페이지를 새로운 페이지와 교체.

2. 구현

2-1. 소스코드

20172655-5.c

main 함수 부분

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const char* const methods[4]={"OPT", "FIFO", "LRU", "Second-Chance"}; //페이지 교체 기법 스트링상수
int frame_cnt=0; //피지컬 프레임 개수(파일)
int *frame; //피지컬 프레임 상태 저장, 출력용
int ref_cnt=0; //페이지 레퍼런스 스트링 개수(파일)
int ref_arr[30]; //페이지 레퍼런스 스트링 저장 배열(파일)

void print_table(int); //테이블 출력용 함수
void print_frame_stat(void); //매 시점 프레임 상태 출력
void func_OPT(void);
void func_FIFO(void);
void func_LRU(void);
void func_Second_Chance(void);

int main(void) {
        char file_name[100]; //파일 명 입력 받는 배열
        FILE *fp; //파일포인터
        char temp[100], *tok; //파일 내용 임시 저장, 토크나이징에 사용
```

```
	printf("파일 이름 입력 : ");
	scanf("%[^\n]", file_name);
	if(NULL==(fp=fopen(file_name, "r"))) {
		printf("Wrong file name.\n");
		exit(1);
	}
	printf("=====================================\n");
	fgets(temp, 99, fp);
	frame_cnt=atoi(temp); //피지컬 프레임 개수(파일입력)
	fgets(temp, 99, fp); //페이지 레퍼런스 스트링(파일입력)
	temp[strlen(temp)-1]='\0';

	tok=strtok(temp,  " "); //페이지 레퍼런스 스트링 토큰 단위 분리
	while(tok!=NULL) {
		ref_arr[ref_cnt++]=atoi(tok); //ref_arr 배열에 저장
		tok=strtok(NULL, " ");
	}
	for(int i=0; i<4; i++)
	{
		print_table(i);
		switch(i) {
			case 0://OPT
				func_OPT();
				break;
			case 1://FIFO
				func_FIFO();
				break;
			case 2://LRU
				func_LRU();
				break;
			case 3://Second_Chance
				func_Second_Chance();
				break;
		}
	}
	return 0;
}
```

출력 관련 함수

```
void print_table(int i) {
        printf("Used method : %s\n", methods[i]);
        printf("page reference string : ");
        for(int i=0; i<ref_cnt; i++)
                printf("%d ", ref_arr[i]);
        printf("\n");
        printf("\n%13s    ", "frame");
        for(int i=1; i<=frame_cnt; i++)
                printf("%-8d", i);
        printf("page fault\n");
        printf("time\n");
        return;
}

void print_frame_stat(void) {
        for(int i=0; i<frame_cnt; i++) {
                if(frame[i]==-1) //프레임에 -1값 -> 사용x 프레임, 페이지 없음
                        printf("%-8c", ' ');
                else
                        printf("%-8d", frame[i]);
        }
        return;
}
```

1) Optimal Algorithm

```
void func_OPT(void) {
        int hit=0, fault_cnt=0, stack_cnt=0, come_late=0;
        int *distance=(int*)calloc(frame_cnt, sizeof(int));
        //현재 프레임 내 페이지들의 다음 요청까지의 거리
        frame=(int*)calloc(frame_cnt, sizeof(int));
        //피지컬 프레임, 페이지 번호 저장 됨
        for(int i=0; i<frame_cnt; i++) frame[i]=-1; //프레임 배열 초기화
        for(int i=0; i<ref_cnt; i++) { // i : 현재 reference
                come_late=0;
                printf("%-16d", i+1);
                //계산
                for(int j=0; j<frame_cnt; j++) { // hit 여부 판단
```

```
                        if(frame[j]==ref_arr[i]) { // hit
                                hit=1;
                        }
                }
                if(hit==0) { //miss
                        if(stack_cnt==frame_cnt) { // 남은 frame 없음
                                for(int j=1; j<frame_cnt; j++) {
                                        if(distance[come_late]<distance[j])
                                                come_late=j;
                                } //현재 프레임들 다음 요청까지 거리 비교
                                frame[come_late]=ref_arr[i];
                                // come_late번째 프레임에 새 페이지 저장
                        }
                        else { // 빈 frame 있음
                                for(int j=0; j<frame_cnt; j++) {
                                        if(frame[j]==-1) { //사용x 프레임에 할당
                                                frame[j]=ref_arr[i];
                                                stack_cnt++;
                                                break;
                                        }
                                }
                        }
                }
                //매 page replacement 후 distance 계산(i+1번 레퍼런스 기준)
                for(int j=0; j<frame_cnt; j++) {
                        if(frame[j]!=-1) { //계산 시 할당x 프레임은 제외
                                distance[j]=100; //매우 먼 거리, 변동 X시 앞으로
등장하지 않는 페이지 의미 -> 다음 page replacement에 사용된다
                                for(int k=i+1; k<ref_cnt; k++) {
                                        if(frame[j]==ref_arr[k]) {
                                                distance[j]=k-(i+1);
                                                break;
                                        }
                                }
                        }
                }
```

```
                print_frame_stat(); //i번째 페이지 사용 후 프레임배열 상태 출력
                if(hit==0) { fault_cnt++; printf("F\n"); }
                 //히트 못하면 page fault 횟수 1 증가
                else {hit=0; printf("\n"); }
        }
        printf("Number of page faults : %d times\n", fault_cnt);
        printf("=====================================\n");
        free(distance);
        free(frame);
        return;
}
```

2) First In First Out Algorithm

```
void func_FIFO(void) {
        int hit=0, fault_cnt=0, stack_cnt=0, mod=0;
        frame=(int*)calloc(frame_cnt, sizeof(int)); //피지컬 프레임 상태 저장, 출력용
        for(int i=0; i<frame_cnt; i++) frame[i]=-1; //frame 배열 초기화
        for(int i=0; i<ref_cnt; i++) { // i : 현재 reference
                printf("%-16d", i+1);
                //계산
                for(int j=0; j<frame_cnt; j++) { // hit 여부 판단
                        if(frame[j]==ref_arr[i]) {hit=1;} // hit시 값 1로 설정
                }
                if(hit==0) { //miss
                        if(stack_cnt==frame_cnt) { // 남은 frame 없음
                                frame[mod]=ref_arr[i]; //가장 오래된 프레임
                                mod=(mod+1)%frame_cnt; //다음 교체 대상
                        }
                        else { // 빈 frame 있음
                                for(int j=0; j<frame_cnt; j++) {
                                        if(frame[j]==-1) {
                                                frame[j]=ref_arr[i];
                                                stack_cnt++;
                                                break;
                                        }
                                }
                        }
```

```
                }
                print_frame_stat(); //i번째 페이지 사용 후 프레임배열 상태 출력
                if(hit==0) { fault_cnt++; printf("F\n"); }
                else {hit=0; printf("\n"); }
        }
        printf("Number of page faults : %d times\n", fault_cnt);
        printf("=====================================\n");
        free(frame);
        return;
}
```

3) Least Recently Used Algorithm

```
void func_LRU(void) {
        int hit=0, fault_cnt=0, stack_cnt=0, index=0;
        int *counter=(int*)calloc(frame_cnt, sizeof(int));
        //피지컬 프레임의 카운터, LRU 판단
        frame=(int*)calloc(frame_cnt, sizeof(int));
        //피지컬 프레임 상태 저장, 출력용
        for(int i=0; i<frame_cnt; i++) frame[i]=-1;
        for(int i=0; i<ref_cnt; i++) { // i : 현재 reference
                printf("%-16d", i+1);
                //계산
                index=0; //counter에서 가장 카운터 값 큰 프레임의 인덱스 저장
                for(int j=0; j<frame_cnt; j++) { // hit 여부 판단
                        if(frame[j]==ref_arr[i]) {
                                counter[j]=0; //히트 시 카운터 값 0으로 설정
                                hit=1;
                                break;
                        } // hit
                }
                if(hit==0) { //miss
                        if(stack_cnt==frame_cnt) { // 남은 frame 없음
                                for(int j=1; j<frame_cnt; j++) {
                                        if(counter[index]<counter[j])
                                                index=j;
                                } //카운터 값 가장 큰 페이지를 교체
                                frame[index]=ref_arr[i];
                                counter[index]=0; //교체 후 카운터 0으로 설정
```

```
                        }
                        else { // 빈 frame 있음
                                for(int j=0; j<frame_cnt; j++) {
                                        if(frame[j]==-1) {
                                                frame[j]=ref_arr[i];
                                                stack_cnt++;
                                                break;
                                        }
                                }
                        }
                }
                for(int j=0; j<frame_cnt; j++) { //페이지 들어있는 프레임의 카운터 증가
                                        if(frame[j]!=-1) counter[j]++;
                }
                print_frame_stat();

                if(hit==0) { fault_cnt++; printf("F\n"); }
                else {hit=0; printf("\n"); }
        }
        printf("Number of page faults : %d times\n", fault_cnt);
        printf("=======================================\n");
        free(frame);
        free(counter);
        return;
}
```

4) Second Chance Algorithm

```
void func_Second_Chance(void) {
        int hit=0, fault_cnt=0, stack_cnt=0, next_victim=0;
        int *label=(int*)calloc(frame_cnt, sizeof(int)); //히트 레이블
        frame=(int*)calloc(frame_cnt, sizeof(int)); //피지컬 프레임 상태 저장, 출력용
        for(int i=0; i<frame_cnt; i++) frame[i]=-1; //프레임 배열 -1로 초기화
        for(int i=0; i<ref_cnt; i++) { // i : 현재 reference
                printf("%-16d", i+1);
                //계산
                for(int j=0; j<frame_cnt; j++) { // hit 여부 판단
```

```
                        if(frame[j]==ref_arr[i]) { // hit
                                hit=1;
                                label[j]=1;
                                //히트 시 해당 페이지 레이블 1로 설정
                        }
                }
                if(hit==0) { //miss
                        if(stack_cnt==frame_cnt) { // 남은 frame 없음
                                while(1) { //next_victim으로 교체 대상 지정
                                        if(label[next_victim]==1) {
                                                label[next_victim]=0;

next_victim=(next_victim+1)%frame_cnt;
//레이블이 1일 시 0으로 변경 후 다음으로 넘어간다
                                        }
                                        else {
                                                frame[next_victim]=ref_arr[i];

next_victim=(next_victim+1)%frame_cnt;
//레이블이 0일 시 바로 교체
                                                break;
                                        }
                                }
                        }
                        else { // 빈 frame 있음
                                for(int j=0; j<frame_cnt; j++) {
                                        if(frame[j]==-1) {
                                                frame[j]=ref_arr[i];
                                                label[j]=0;
                                                stack_cnt++;
                                                break;
                                        }
                                }
                        }
                }
                print_frame_stat();
                if(hit==0) { fault_cnt++; printf("F\n"); }
```

```
                else {hit=0; printf("\n"); }
        }
        printf("Number of page faults : %d times\n", fault_cnt);
        printf("=======================================\n");
        free(label);
        free(frame);
        return;
}
```

2-2. 동작 결과

test1

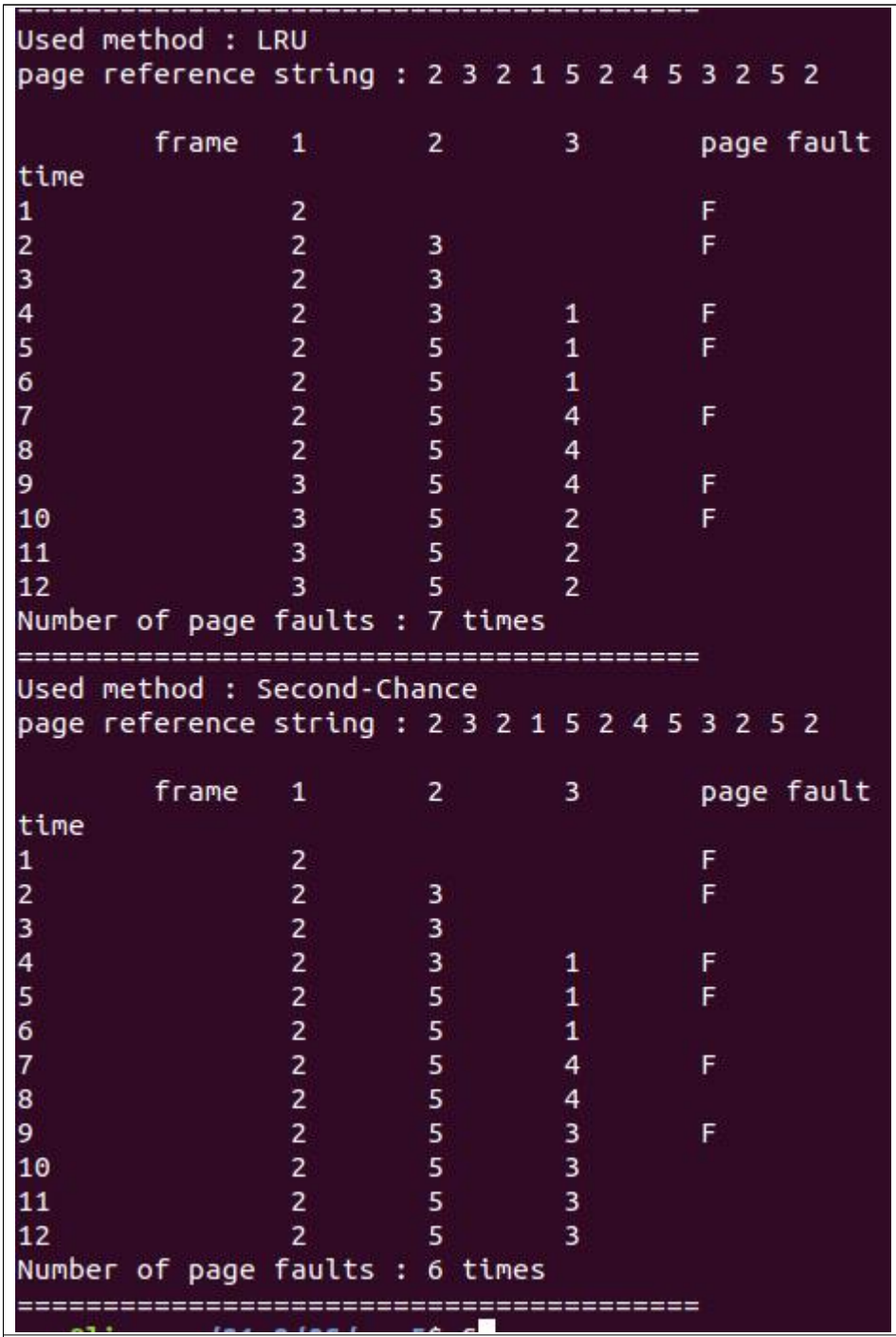
3

2 3 2 1 5 2 4 5 3 2 5 2

```
san@linux:~/21-2/OS/os-5$ ./a.out
파일 이름 입력 : test1
========================================
Used method : OPT
page reference string : 2 3 2 1 5 2 4 5 3 2 5 2

        frame   1       2       3       page fault
time
1               2                       F
2               2       3               F
3               2       3
4               2       3       1       F
5               2       3       5       F
6               2       3       5
7               4       3       5       F
8               4       3       5
9               4       3       5
10              2       3       5       F
11              2       3       5
12              2       3       5
Number of page faults : 6 times
========================================
Used method : FIFO
page reference string : 2 3 2 1 5 2 4 5 3 2 5 2

        frame   1       2       3       page fault
time
1               2                       F
2               2       3               F
3               2       3
4               2       3       1       F
5               5       3       1       F
6               5       2       1       F
7               5       2       4       F
8               5       2       4
9               3       2       4       F
10              3       2       4
11              3       5       4       F
12              3       5       2       F
Number of page faults : 9 times
========================================
```



page reference string 빈도 : 1(1회), 2(5회), 3(2회), 4(1회), 5(3회)

page fault 발생 횟수 : OPT(6), FIFO(9), LRU(7), SC(6)

test2

3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

```
파일 이름 입력 : test2
========================================
Used method : OPT
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       0       3       F
7               2       0       3
8               2       4       3       F
9               2       4       3
10              2       4       3
11              2       0       3       F
12              2       0       3
13              2       0       3
14              2       0       1       F
15              2       0       1
16              2       0       1
17              2       0       1
18              7       0       1       F
19              7       0       1
20              7       0       1
Number of page faults : 9 times
========================================
Used method : FIFO
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       3       1       F
7               2       3       0       F
8               4       3       0       F
9               4       2       0       F
10              4       2       3       F
11              0       2       3       F
12              0       2       3
13              0       2       3
14              0       1       3       F
15              0       1       2       F
16              0       1       2
17              0       1       2
18              7       1       2       F
19              7       0       2       F
20              7       0       1       F
Number of page faults : 15 times
```

```
Used method : LRU
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       0       3       F
7               2       0       3
8               4       0       3       F
9               4       0       2       F
10              4       3       2       F
11              0       3       2       F
12              0       3       2
13              0       3       2
14              1       3       2       F
15              1       3       2
16              1       0       2       F
17              1       0       2
18              1       0       7       F
19              1       0       7
20              1       0       7
Number of page faults : 12 times
========================================
Used method : Second-Chance
page reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

        frame   1       2       3       page fault
time
1               7                       F
2               7       0               F
3               7       0       1       F
4               2       0       1       F
5               2       0       1
6               2       0       3       F
7               2       0       3
8               4       0       3       F
9               4       0       2       F
10              3       0       2       F
11              3       0       2
12              3       0       2
13              3       0       2
14              3       1       2       F
15              3       1       2
16              0       1       2       F
17              0       1       2
18              0       1       7       F
19              0       1       7
20              0       1       7
Number of page faults : 11 times
```

page reference string 빈도 : 0(6회), 1(4회), 2(4회), 3(3회), 4(1회), 7(2회)

page fault 발생 횟수 : OPT(9), FIFO(15), LRU(12), SC(11)

test3

4

0 3 2 1 5 5 4 1 0 2 3

```
san@linux:~/21-2/OS/os-5$ ./a.out
파일 이름 입력 : test3
=======================================
Used method : OPT
page reference string : 0 3 2 1 5 5 4 1 0 2 3

        frame   1       2       3       4       page fault
time
1               0                               F
2               0       3                       F
3               0       3       2               F
4               0       3       2       1       F
5               0       5       2       1       F
6               0       5       2       1
7               0       4       2       1       F
8               0       4       2       1
9               0       4       2       1
10              0       4       2       1
11              3       4       2       1       F
Number of page faults : 7 times
=======================================
Used method : FIFO
page reference string : 0 3 2 1 5 5 4 1 0 2 3

        frame   1       2       3       4       page fault
time
1               0                               F
2               0       3                       F
3               0       3       2               F
4               0       3       2       1       F
5               5       3       2       1       F
6               5       3       2       1
7               5       4       2       1       F
8               5       4       2       1
9               5       4       0       1       F
10              5       4       0       2       F
11              3       4       0       2       F
Number of page faults : 9 times
---------------------------------------
```

```
========================================
Used method : LRU
page reference string : 0 3 2 1 5 5 4 1 0 2 3

        frame   1       2       3       4       page fault
time
1               0                               F
2               0       3                       F
3               0       3       2               F
4               0       3       2       1       F
5               5       3       2       1       F
6               5       3       2       1
7               5       4       2       1       F
8               5       4       2       1
9               5       4       0       1       F
10              2       4       0       1       F
11              2       3       0       1       F
Number of page faults : 9 times
========================================
Used method : Second-Chance
page reference string : 0 3 2 1 5 5 4 1 0 2 3

        frame   1       2       3       4       page fault
time
1               0                               F
2               0       3                       F
3               0       3       2               F
4               0       3       2       1       F
5               5       3       2       1       F
6               5       3       2       1
7               5       4       2       1       F
8               5       4       2       1
9               5       4       0       1       F
10              5       2       0       1       F
11              5       2       3       1       F
Number of page faults : 9 times
========================================
```

page reference string 빈도 : 0(2회), 1(2회), 2(2회), 3(2회), 4(1회), 5(2회)
page fault 발생 횟수 : OPT(7), FIFO(9), LRU(9), SC(9)

예시를 통해 Optimal algorithm가 가장 적은 page fault가 발생시킴을 알 수 있다. 그러나 실제로는 다음 요청이 어느 페이지의 데이터가 필요한 지 알 수 없으므로 완벽한 Optimal algorithm의 구현은 불가능하다.

그 다음으로 준수한 성능을 보여주는 LRU, SC는 과거의 데이터를 바탕으로 교체할 페이지를 선택한다. 예시1, 2에서 과거의 사용 정도를 고려하지 않는 FIFO보다 뛰어난 성능을 보여준다.

예시3은 의도적으로 page reference string 내 동일 페이지 간 거리를 최대한 멀리 배치하여 FIFO가 LRU, SC와 비슷한 결과를 보여주도록 하였다. 미래를 알고 있는 Optimal의 경우 다른 세 방법보다 뛰어난 결과를 나타낸다.

3. 과제를 통해 배운 점

운영체제는 프로세스들이 독립된 충분한 크기의 메모리 공간을 갖고 동작한다고 인식하도록 가상 메모리를 지원한다. 프로세스들이 사용하는 가상메모리 공간의 총합은 실제 물리적 메모리 공간의 크기보다 커지므로 1대 1 매핑이 불가능하다. 이를 해결하기 위해 운영체제는 각 프로세스들의 사용 빈도가 낮은 페이지들은 디스크에 보존하다가 필요에 의한 요청 시 메모리에 올리며 이를 Demand paging이라 한다.

물리적 메모리의 크기는 한정적이므로 경우에 따라 free frame이 존재하지 않을 수 있다. 이때 어떤 page를 메모리에서 잠시 swap out 시킬지 선택하기 위해 Page replacement algorithm을 사용하다. 대표적으로 FIFO, Optimal, LRU, Second-Chance 같은 알고리즘이 있으며 이들의 목적은 page fault 횟수를 최대한 줄이는 것이다.

FIFO algorithm은 프레임에 들어온 페이지 순서대로 새로운 페이지와 교체된다. FIFO queue를 이용하여 간단하게 구현할 수 있으나 성능이 reference string 순서에 영향을 받고, 프레임 개수가 늘어나도 page fault 횟수가 늘어나는 경우가 존재한다. (Belady's Anomaly)

Optimal algorithm은 현 시점에서 가장 오랫동안 사용되지 않을 것 같은 페이지를 새로운 페이지와 교체한다. 가장 이상적인 방법이지만, reference string의 순서를 아는 것은 불가능하므로 완전한 구현은 불가능하다.

LRU algorithm은 가장 많이 사용되는 방법으로 현 시점에서 가장 오랫동안 사용되지 않은 페이지를 새로운 페이지와 교체한다. 프레임마다 counter변수를 두거나 더블링크드리스트로 구현할 수 있지만 전자의 경우 page fault 시마다 카운터 값이 가장 큰 프레임 탐색에 대한 오버헤드가, 후자의 경우 업데이트 비용에 대한 오버헤드가 발생한다.

Second Chance algorithm은 FIFO방식에 reference bit를 두어 page fault 시 현재 교체 대상 페이지가 최근에 참조되었으면 한 번의 기회를 더 주는 방식이다. 현재 next victim이 가리키고 있는 페이지의 reference bit의 값이 1이면 0으로 변경 후 다음 페이지를 확인하며, 0일 경우 새로운 페이지와 교체한다. 더 발전시켜서 reference bit에 주기를 주어 오래 전에 참조한 페이지를 메모리에 유지시키는 것을 방지하기도 한다.