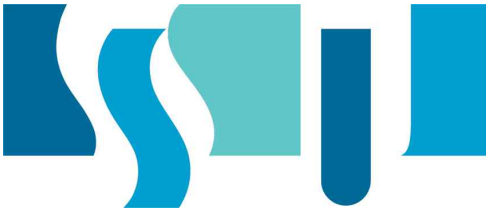

Assignment #5

- 페이지 교체 기법 구현 -

컴파일 옵션 및 Makefile은 없습니다.
제출하는 소스코드는 'os5.c' 하나입니다.
또한, Used method를 입력받을 때, 각 method의 문자열을 입력하지 않고 OPT은 1을, FIFO는 2를, LRU는 3을, Second-chance는 4를 입력하도록 설계하였습니다.

과목명	운영체제
교수명	김철홍
학 과	숭실대학 컴퓨터학부
학 번	20192958
이 름	김수은
제출일	2021.11.21.일요일

목차

1. 목적/배경설명

이 과제의 목적은 가상메모리 관리 기법의 하나인 페이지 교체 기법 중에서 optimal, FIFO(First-In-First-Out), LRU(Least Recently Used), Second-Chance Algorithm을 이해하고 이해를 바탕으로 실제로 각 알고리즘에 맞는 동작 과정을 시간 순서대로 보여주는 시뮬레이터를 구현하는 것이다. 각 알고리즘을 구현하며 각 기법 간의 page fault 발생 횟수를 비교, 분석하여 알고리즘에 대한 깊은 이해를 하는 것이 목적이자 목표이다.

optimal 알고리즘은 가장 오랫동안 안 불릴 페이지를 쫓아내는 방식이다. 이번 과제에서는 page reference string이 들어오기 때문에 미래에 어떤 페이지가 들어올지 예측 가능하여 optimal 알고리즘을 사용할 수 있다. optimal 알고리즘은 가장 페이지 폴트의 개수를 줄일 수 있는 효율적인 방법이지만 실제로는 구현이 어렵다. 미래에 참조될 page를 정확하게 예측할 수 없기 때문이다.

FIFO 알고리즘은 가장 오래된 페이지를 쫓아내는 방식이다. 이 알고리즘을 수행하기 위해서는 각 페이지가 얼마나 됐는지를 기록해주거나 페이지가 올라온 순서를 큐에 저장하여 사용하는 방식 등이 있다. 본인이 FIFO 알고리즘을 구현할 때는 첫 번째 방식인 각 페이지가 얼마나 됐는지를 기록해주었다. 이 방식을 구현하기 위해서 각 페이지는 시간이 지날 때마다 age라는 변수를 두어 그 변수 값을 증가시켜줬다. 그리고 변수 age의 값이 가장 큰 페이지가 교체되게 하였다. 추가적으로 FIFO 알고리즘을 기반으로 구현되는 Second-Chance 알고리즘은 첫 번째가 아닌 두 번째 방식으로 구현을 해주었다. (Second-Chance 알고리즘을 어떻게 구현하였는지는 뒤에서 마저 설명하도록 하겠다.) FIFO 방식은 가장 간단한 알고리즘으로 구현이 간단하다는 장점이 있지만 비효율적이다.

LRU 알고리즘은 가장 오래 사용되지 않은 페이지를 쫓아내는 알고리즘이다. 과거의 데이터를 바탕으로 페이지가 사용될 시간을 예측하여 교체하는 것이다. optimal 알고리즘은 실제 구현이 불가능하지만 optimal 알고리즘의 방식과 비슷한 효과를 낼 수 있는 방법이 바로 LRU 알고리즘이다. 대체적으로 FIFO 알고리즘보다 효율적이며 Optimal 알고리즘보다는 비효율적이다.

Second-Chance 알고리즘은 한 번의 기회를 더 주는 알고리즘이다. 각 페이지에는 1비트 짜리 참조 비트가 존재하고 초기값은 모두 0이며 access가 되면 1로 바꿔준다. 그러나 FIFO 알고리즘이 기본 알고리즘이기 때문에 참조 비트가 1이라도 선택이 될 수 있는데 이때 만일 참조 비트가 1이라면 최근에 access되었다는 것이라고 간주하고 0으로 바꾼 후 한 번의 기회를 더 주고 이 페이지가 아닌 다른 victim 페이지를 찾는다. 본인은 순환큐를 이용하여 구현을 하였고 순환 큐를 가리키는 포인터를 이동시켜주는 방식으로 구현을 하였다.

2. 설계/구현 (+소스코드)

실행 초기화면에서 맨 처음 input 파일의 이름을 입력받는다. 그리고 이 입력받은 인풋 파일을 읽어온다. 첫 번째 줄에 나온 문자열은 atoi 함수를 이용하여 정수형으로 바꾼 후 그 값은 pageframe의 수인 paeframenum 변수에 저장하여 사용한다. 그리고 인풋 파일에서 두 번째 줄에 나오는 prs값을 입력받고 빈칸을 기준으로 자르고 각각을 atoi 함수를 이용하여 인트로 바꾼 후 prs배열에 그 값을 순서대로 저장해준다. prs에 있는 값의 개수를 prsnum이라는 변수에 저장하여 사용한다. 그리고 출력해야 할 문장을 양식에 맞게 출력해주고 1~4 중 하나의 값을 입력하는데 OPT은 1을, FIFO는 2를, LRU는 3을, Second-chance는 4을 의미한다. 선택된 값에 따라서 각 번호에 해당하는 알고리즘으로 출력을 해주었다.

(밑에서 계속)

1) OPT

```
if(usedmethod == 1){
    int pagefaultOPT=0;
    int check=0;
    int checkEmptyOPT=0;

    for (int i =0; i<prsnum; i++){
        printf("%d\t\t",i+1);

        check = 0;
        checkEmptyOPT = 0;

        //pageframe배열에서 같은 값이 있는지 찾아봄
        for(int j=0; j<pageframenum; j++){ //pageframe배열에서 같은 값이 있는지 찾아봄

            if(prs[i] == pageframe[j]){ //있으면 page fault 아니고 출력 아무것도 안함
                check = 1;

                //프린트해주기
                for(int c=0; c<pageframenum; c++){
                    if(pageframe[c] != -1){
                        printf("%d", pageframe[c]);
                    }
                    printf("\t");
                }
            }
        }

        //check가 되지 않은 pagefault가 발생하는 경우가 여기로 옴
        if(!check){ //check가 안된 즉, pagefault가 난 친구들이 여기로 옴
            pagefaultOPT++;

            //-1인 자리가 있는지 확인하고 있으면 그 자리에 바로 넣어주고
            //프린트문 찍어주고 바로 나오면 됨
            for(int j=0; j<pageframenum; j++){ //-1인 자리가 있는지 확인
                if(pageframe[j] == -1){ //-1인 자리가 있으면 그 자리에 바로 넣어주고
                    pageframe[j] = prs[i];
                    checkEmptyOPT = 1; //-1인 비어있는 자리가 있어서 이미 처리했음을 의미

                    //값을 출력해줌
                    for(int k=0; k<pageframenum; k++){
                        if(pageframe[k] != -1){ //-1이 아닌 값만 출력해줌
                            printf("%d", pageframe[k]);
                        }
                        printf("\t"); //-1이던 아니던 탭은 한번씩 해줘야함
                    }
                    //빠져나감
```

```
                    break;
                }
            }

            //-1(비어있는 자리)가 없어서 원래 있던 놈들을 쫓아내고 넣어야 하는 경우
            if(!checkEmptyOPT){
                int count=0;
                int checkcount[pageframenum];
                int replaceIndex = -1;

                memset(checkcount,0,sizeof(int)*pageframenum);

                for(int j=i+1; j<prsnum; j++){
                    for(int k=0;k<pageframenum;k++){
                        if(prs[j] == pageframe[k]){
                            if(checkcount[k]==0){
                                checkcount[k]=1;
                                count++;

                                if(count==pageframenum){
                                    replaceIndex = k;
                                    break;
                                }
                            }
                        }
                    }
                }

                if(replaceIndex == -1){
                    for(int j=0; j<pageframenum; j++){
                        if(checkcount[j] ==0){
                            replaceIndex = j;
                            break;
                        }
                    }
                }

                pageframe[replaceIndex] = prs[i];

                //다 바꿔줬으니깐 출력해줌
                for(int j=0; j<pageframenum; j++){
                    printf("%d\t", pageframe[j]);
                }
            }
```

```
            printf("F");
        }

        printf("\n");
    }

    printf("Number of page faults : %d times\n", pagefaultOPT);
```

pageframe에 해당하는 pageframe 배열은 -1로 초기화해준다. 검색하고자 하는 prs값이 pageframe 배열에서 같은 값이 있는지 찾아본다. 같은 값이 있으면 페이지 폴트가 아니고 hit인 경우다. hit했다는 것을 체크하기 위해서 check 변수에 1을 저장하고 pageframe 배열의 값이 -1인 경우를 제외해주고 값을 출력해준다. 그리고 check 변수의 값이 0인 경우는 페이지 폴트가 발생한 경우이다. 그래서 페이지 폴트의 횟수를 저장하는 변수의 값을 하나 증가시켜준다. 페이지 폴트가 발생하는 경우는 크게 두 가지로 나뉘는데 pageframe 배열의 값이 -1인 자리가 있어서 그 자리에 넣어줄 수 있는 경우와 pageframe 배열이 꽉 차서 원래 들어있던 값을 쫓아내고 그 자리에 넣어야 하는 경우이다.
첫 번째 경우는 pageframe 배열의 값이 -1인 자리가 있으면 가장 먼저 만난 -1인 자리에 해당 순서의 prs값을 넣어준다. 두 번째 경우는 원래 있던 값을 쫓아내야 하는데 그 쫓아내는 인덱스를 찾기 위해서 다음과 같은 과정을 수행해준다. 현재 순번에 해당하는 prs값 이후의 prs값을 하나씩 체크하면서 그 prs 값(미래에 나올 prs 값)와 동일한 페이지가 pageframe에 있다면 checkcount배열의 값이 0인지 검사하고 그렇다면 count변수 값을 하나 증가시켜준다. checkcount배열은 pageframe개수만큼 생성하고 초기화를 0으로 해준 배열이다. 이 배열을 생성한 이유는 이미 어떤 페이지에 대해서 검사가 이뤄졌다면 두 번 이상 검사가 이뤄지지 않게 하기 위해서이다. 그래서 checkcount배열의 값이 0인지 검사하는 것이고 0이라면 해당 pageframe 인덱스에 해당하는 checkcount배열의 값을 1로 설정해주고 count 값을 하나 증가시킨다. 그리고 count 값이 pageframe의 개수와 같아지게 되는 시점이 있다면 그 때의 pageframe 인덱스 번호를 replaceIndex라는 변수에 저장하고 이것을 쫓아내게 한다. 만약에 replaceIndex의 값이 -1이라는 것은 미래에 한 번도 쓰이지 않는 페이지가 있다는 소리이고 그 페이지를 쫓아내준다. 그러한 페이지가 여러 개가 있을 경우, 포문을 돌리면서 가장 먼저 만나는 페이지를 replaceIndex로 설정하고 그것을 쫓아낸다.

2) FIFO

```
else if(usedmethod == 2){
    int pagefaultFIFO=0;
    int check;
    int age[pageframenum];
    int checkEmptyFIFO=0;

    memset(age,0,sizeof(int)*pageframenum);

    for(int i=0; i<prsnum; i++){
        printf("%d\t\t", i+1);

        check = 0;
        checkEmptyFIFO =0;

        //일단은 page fault인지 아닌지 체크해야 함
        //아래는 페이지폴트가 아닌 경우임
        for(int j=0; j<pageframenum; j++){
            if(prs[i] == pageframe[j]){
                check=1;

                //프린트해주고 age도 증가시켜 줘야함
                for(int k=0; k<pageframenum; k++){
                    if(pageframe[k] != -1){
                        age[k]++;
                        printf("%d", pageframe[k]);
                    }
                    printf("\t");
                }
            }
        }

        //페이지폴트이면
        //경우는 두가지임
        //1. pageframe이 -1로 돼있어서 거기에다가 넣어주는 것,, 이때도 age++해야함
        //2. -1이 없으면 즉 비어있는 페이지없으면 age가 가장 큰 것으로 교체해주고 나머지는 age++

        if(!check){ //페이지폴트인 경우
            pagefaultFIFO++;

            //1번 경우
            for(int j=0; j<pageframenum; j++){
                if(pageframe[j] == -1){
                    pageframe[j] = prs[i];
                    checkEmptyFIFO=1;
```

```
                for(int k=0; k<pageframenum; k++){
                    if(pageframe[k] != -1){
                        age[k]++;
                        printf("%d", pageframe[k]);
                    }
                    printf("\t");
                }
                break;
            }
        }

        //2번 경우
        if(!checkEmptyFIFO){
            //가장 큰 age를 찾아줌
            int max = age[0];
            int maxindex = 0; //맨 처음 인덱스를 가리키고 있음

            for(int j=0; j<pageframenum; j++){
                if(age[j] > max){
                    max = age[j];
                    maxindex = j;
                }
            }

            //가장 큰 age값을 가진 maxindex로 교체해줌
            //교체를 해준 애를 포함해서 모든 값을 age++해줌
            //교체를 해준 애는 age값을 초기화해줌
            pageframe[maxindex] = prs[i];
            age[maxindex] =0;

            for(int j=0; j<pageframenum; j++){
                age[j]++;
                printf("%d\t", pageframe[j]);
            }
        }

        printf("F");
    }
    printf("\n");
}
printf("Number of page faults : %d times\n", pagefaultFIFO);
```

각 페이지가 얼마나 오래 됐는지를 저장할 age를 이름으로 하는 배열을 추가로 생성하게 되고 페이지 폴트가 일어나지 않는 경우는 페이지들의 age가 하나씩 늘어나게 된다. 페이지 폴트가 일어나지 않는 경우는 남아있는 pageframe 공간이 있는 경우와 그렇지 않는 경우로 나눌 수 있다. 남아있는 pageframe가 있는 경우 비어 있는 공간에 페이지가 들어오고 들어있는 페이지들의 age를 하나씩 증가시켜준다. 페이지 폴트가 일어났는데 남아있는 pageframe이 없는 경우는 저장해놨던 age값을 이용하여 쫓아낼 페이지를 정하게 된다. age값이 가장 오래된 페이지를 쫓아내고 그 자리에 새로운 페이지를 넣게 된다. 그리고 페이지가 들어올 자리는 age가 0으로 초기화된다. 그리고나서 마지막에 페이지가 들어있는 모든 페이지의 값을 증가시켜준다.
피포를 구현하는 방식에는 각 페이지가 얼마나 됐는지를 기록해주거나 페이지가 올라온 순서를 큐에 저장하여 사용하는 방식이 있는데 본인이 피포를 구현할 때 이용한 방식은 각 페이지가 얼마나 됐는지를 age 배열에 저장해준 방식이다. age 값을 늘려가면서 페이지를 쫓아내야 할 상황이 오면 이 age 값을 이용하여 가장 큰 age 값을 가진 페이지를 쫓아내는 것이다. 가장 큰 age가 가장 오래 전에 들어왔다는 것을 의미하기 때문이다.
그런데 이 첫 번째 방식 말고도 두 번째 방식인 페이지가 올라온 순서를 큐에 저장하여 사용하는 방식도 있다. 이 방식을 구현할 때에는 포인터를 하나 생성을 한다. 그리고 페이지가 올라온 순서대로 배열에 저장을 하고 이 배열의 인덱스를 가리킬 포인터 변수 하나를 만든다. 맨 처음에 포인터는 배열의 가장 처음에 들어온 인덱스 0에 해당하는 값을 가리키고 있다. 그러다가 쫓아내야 할 페이지가 생기게 되면 이 포인터가 가리키는 0 인덱스 자리에 해당하는 페이지를 쫓아낸다. 그리고 포인터는 다음 인덱스를 가리키게 된다. 그러기 위해서 포인터값을 +1 해주고 그 값을 pageframe의 개수만큼 나머지 연산을 한다. 그렇게 순환 큐를 구현하게 된다. FIFO를 기반으로 돌아가는 Second-Chance의 경우는 앞서 말한 두 번째 방식으로 구현을 해주었다. 피포를 이렇게 두 가지 방식으로 구현해봄으로써 다양한 방식을 이용하여 더 깊은 이해를 할 수 있었으며 다양한 구현 방법을 생각해보게 되었다.

3) LRU

```
else if(usedmethod == 3){
    int pagefaultLRU =0;
    int check=0;
    int checkEmptyLRU=0;

    for (int i =0; i<prsnum; i++){
        printf("%d\t\t",i+1);

        check = 0;
        checkEmptyLRU =0;

        //pageframe배열에서 같은 값이 있는지 찾아봄
        for(int j=0; j<pageframenum; j++){ //pageframe배열에서 같은 값이 있는지 찾아봄

            if(prs[i] == pageframe[j]){ //있으면 page fault 아니고 출력 아무것도 안함
                check = 1;
                //printf("[%d]이미 있는 경우\n", i);

                //프린트해주기
                for(int c=0; c<pageframenum; c++){
                    if(pageframe[c] != -1){
                        printf("%d", pageframe[c]);
                    }
                    printf("\t");
                }
            }
        }

        //check가 되지 않은 pagefault가 발생하는 경우가 여기로 옴
        if(!check){ //check가 안된 즉, pagefault가 난 친구들이 여기로 옴
            pagefaultLRU++;

            //-1인 자리가 있는지 확인하고 있으면 그 자리에 바로 넣어주고
            //프린트문 찍어주고 바로 나오면 됨
            for(int j=0; j<pageframenum; j++){ //-1인 자리가 있는지 확인
                if(pageframe[j] == -1){ //-1인 자리가 있으면 그 자리에 바로 넣어주고
                    pageframe[j] = prs[i];
                    checkEmptyLRU = 1; //-1인 비어있는 자리가 있어서 이미 처리했음을 의미

                    //값을 출력해줌
                    for(int k=0; k<pageframenum; k++){
                        if(pageframe[k] != -1){ //-1이 아닌 값만 출력해줌
                            printf("%d", pageframe[k]);
                        }
                        printf("\t"); //-1이던 아니던 탭은 한번씩 해줘야함
                    }
                    //빠져나감
```

```
                    break;
                }
            }

            //비어있는 자리가 없어서 원래 있던 놈을 쫓아내고 넣는 경우
            if(!checkEmptyLRU){
                int count=0;
                int checkcount[pageframenum];
                int replaceIndex=-1;

                memset(checkcount, 0, sizeof(int)*pageframenum);

                //앞에 페이지들을 확인해야함
                //확인을 하다가 마지막놈을 count를 하다가 conut-1이랑 같은 값이 되면
                //그 놈을 replaceindex로 결정함

                for(int j=i-1;j>-1;j--){
                    for(int k=0;k<pageframenum;k++){
                        if(pageframe[k]==prs[j]) {
                            if(checkcount[k]==0){ //체크가 안됐을 때만 count를 해줌
                                checkcount[k]=1; //확인이 됐음을 체크함
                                count++;

                                if(count==pageframenum){
                                    replaceIndex = k;
                                    break;
                                }
                            }
                        }
                    }
                }
                pageframe[replaceIndex] = prs[i];

                for(int j=0; j<pageframenum; j++){
                    printf("%d\t", pageframe[j]);
                }
            }
            printf("F");
        }
        printf("\n");
    }
    printf("Number of page faults : %d times\n", pagefaultLRU);
```

LRU의 경우에도 페이지 폴트가 나지 않은 경우와 나는 경우가 있다. 페이지 폴트가 나지 않는 경우는 pageframe에 해당하는 prs값의 페이지가 있는 경우로, 들어와 있는 페이지를 출력해준다. 그리고 페이지 폴트가 나는 경우도 두 가지 경우로 나뉜다. 첫 번째는 비어있는 pageframe이 경우 그 빈 자리에 페이지를 넣어주고 들어가 있는 페이지를 모두 출력해주면 된다. 두 번째는 비어있는 자리가 없어서 원래 있는 페이지를 쫓아내야 하는 경우이다. 이 경우에는 과거에 나온 페이지들을 확인해줘야 한다. 과거를 확인하여 가장 오래 전에 사용된 페이지를 쫓아낸다. 그리고 그 자리에 해당 순서의 prs값을 넣어주고 들어있는 페이지들을 출력해준다.

4) Second-Chance

```
else if(usedmethod == 4){
    int pagefaultSC=0;
    int checkEmptySC=0;
    int check=0;
    int hit[pageframenum];
    int victim=-1;

    // -1페이지가 있을 때 victim을 검사함
    // victim이 -1이었으면 가장 먼저 들어온 놈임을 알 수 있음
    // 그럼 그 값으로 victim이 가리키게 함
    // 페이지 폴트가 난 경우 victim이 가리키는 놈을 방출시켜야함
    // 이때 hit bit이 1이면 그 다음놈이 가리키게 함

    memset(hit,0,sizeof(int)*pageframenum);
    memset(hit,0,sizeof(int)*pageframenum);

    for(int i=0; i<prsnum; i++){
        printf("%d\t\t", i+1);

        check=0;
        checkEmptySC=0;

        //페이지폴트가 아닌 경우임
        //hit일때 hit bit 1로 해줘야함
        for(int j=0; j<pageframenum; j++){
            if(prs[i] == pageframe[j]){
                check=1;
                hit[j]=1;

                //프린트해주고 hit bit를 1로 해줌
                for(int k=0; k<pageframenum; k++){
                    if(pageframe[k] != -1){
                        printf("%d", pageframe[k]);
                    }
                    printf("\t");
                }
            }
        }

        //페이지폴트이면
        //경우는 두가지임
        //1. pageframe이 -1로 돼있어서 거기에다가 넣어줌 그리고 victim이 -1이었으면
        //현재 인덱스 값을 넣어줘야 함
        //2. -1이 없으면 즉 비어있는 프레임이 없으면 victim이 가리키는 곳에다가 넣음
        //근데 hit검사를 해서 1이 아니어야지 넣고 1이면 0인 값이 나올 때까지
        //1->0으로 바꿔줘야함
```

```
if(!check){ //페이지폴트인 경우
    pagefaultSC++;

    //1번 경우
    for(int j=0; j<pageframenum; j++){
        if(pageframe[j] == -1){
            pageframe[j] = prs[i];
            checkEmptySC=1;
            if(victim==-1){
                victim = j;
            }

            for(int k=0; k<pageframenum; k++){
                if(pageframe[k] != -1){
                    printf("%d", pageframe[k]);
                }
                printf("\t");
            }
            break;
        }
    }

    //2번의 경우
    if(!checkEmptySC){
        //printf("여기는오니?");
        if(hit[victim]){
            while(1){ //hit bit가 0일 때까지 찾음
                if(!hit[victim]) break;
                hit[victim] =0;
                //printf("여기안와?");
                victim = (victim+1) % pageframenum;
            }
        }
        //hit bit 0인 것을 찾고 나서
        pageframe[victim] = prs[i];

        //그리고 그 다음 순서로 넘겨줌
        victim = (victim+1)%pageframenum;

        for(int j=0;j<pageframenum;j++){
            printf("%d\t", pageframe[j]);
        }
    }
    printf("F");
}
```

```
    printf("\n");
}
printf("Number of page faults : %d times\n", pagefaultSC);
```

이 알고리즘의 경우에도 페이지 폴트가 나지 않는 경우와 나는 경우로 나뉜다. 페이지 폴트가 나지 않는 경우는 hit인지 아닌지를 저장하기 위한 hit배열의 값을 1로 해준다. 그리고 들어와 있는 페이지들을 출력해준다. 페이지 폴트가 나는 경우는 크게 두 가지로 나뉜다. 남아있는 pageframe 자리가 있어서 그 자리에 들어가는 경우와 남아있는 pageframe이 없는 경우이다. 첫 번째의 경우, 페이지들은 차례로 pageframe에 들어가게 된다. 그래서 맨 처음 들어오는 페이지일 경우 이 페이지가 들어오는 첫 번째 pageframe을 포인터가 가리키고 있다. 포인터는 victim이라는 이름의 변수로 구현하였으며 해당 pageframe의 인덱스를 저장한다. 그리고 들어있는 페이지를 출력해준다. 그리고 두 번째의 경우인 남아있는 pageframe이 없는 경우, 구해두었던 victim이 가리키는 자리에 있는 페이지를 쫓아내고 그 자리에 해당 prs 값을 넣게 된다. 그런데 이 과정을 수행하기 이전에 저장해두었던 hit 배열을 살펴보아야 한다. 그 pageframe 인덱스에 해당하는 hit배열의 값이 1인 경우, hit의 값이 0인 페이지가 나올 때까지 탐색을 해야한다. hit값이 1인 페이지는 hit값을 0으로 바꾸어주고 victim이 가리키는 인덱스를 다음 인덱스로 한칸 이동시켜준다. victim의 값에 +1을 해준 다음 pageframe의 개수로 나머지 연산을 하면 순환 큐를 구현한 것이 된다. 그렇게 pageframe에 있는 페이지의 hit를 확인한다. hit가 0인 페이지가 나올 때까지 victim은 한 칸씩 이동을 한다. 그리고 hit의 값이 0인 페이지가 victim이 된다. 그렇게 victim을 결정해주었으면 그 자리에 해당 prs 값을 넣어주고 victim을 또 +1하고 pageframe 개수로 나머지 연산을 진행하여 다음 인덱스로 victim을 넘겨준다. 그리고 들어있는 페이지를 출력해준다.

3. 수행 결과

앞서 살펴본 네 가지의 알고리즘의 효율을 알아보기 위해서 page fault의 발생 횟수를 알아보도록 하겠다. 서로 다른 세 가지 인풋 파일 세 개를 준비하였다.

```
hgstudio@hgstudio-virtual-machine:~/os5$ cat in1.txt
3
1 3 2 4 3 5 1 3 5 4 7 2 4 1 2
```

1) OPT인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 1
page reference string : 1 3 2 4 3 5 1 3 5 4 7 2 4 1 2

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

2) FIFO인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 2
page reference string : 1 3 2 4 3 5 1 3 5 4 7 2 4 1 2

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

3) LRU인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 3
page reference string : 1 3 2 4 3 5 1 3 5 4 7 2 4 1 2

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

4) Second-Chance인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 4
page reference string : 1 3 2 4 3 5 1 3 5 4 7 2 4 1 2

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

네 개의 알고리즘 중에서 페이지 폴트의 발생 횟수를 비교해보면,
OPT는 8번, FIFO는 11번, LRU는 10번, Second-Chance는 10번인 것을 알 수 있다.
이번 인풋 파일의 경우, 모든 알고리즘 중에서 FIFO가 페이지 폴트가 가장 많이 발생하여 가장 비효율적이며 OPT 알고리즘이 페이지 폴트가 가장 적게 발생하여 가장 효율적이라고 볼 수 있다.
또한, OPT 알고리즘을 실질적으로 사용하기 힘들어서 나온 LRU의 경우에는 OPT보다는 많은 페이지 폴트가 발생하는 것을 알 수 있다.

또 다른 경우를 살펴보면,

```
hgstudio@hgstudio-virtual-machine:~/os5$ cat in2.txt
4
3 2 5 1 2 5 6 2 10 4 5 2 3 4 1 5 4 3 2 1
```

1) OPT인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 1
page reference string : 3 2 5 1 2 5 6 2 10 4 5 2 3 4 1 5 4 3 2 1

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

2) FIFO인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 2
page reference string : 3 2 5 1 2 5 6 2 10 4 5 2 3 4 1 5 4 3 2 1

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

3) LRU인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 3
page reference string : 3 2 5 1 2 5 6 2 10 4 5 2 3 4 1 5 4 3 2 1

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

4) Second-chance인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 4
page reference string : 3 2 5 1 2 5 6 2 10 4 5 2 3 4 1 5 4 3 2 1

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

네 개의 알고리즘 중에서 페이지 폴트의 발생 횟수를 비교해보면,
OPT는 9번, FIFO는 12번, LRU는 13번, Second-Chance는 11번인 것을 알 수 있다.
경우에 따라서 FIFO가 가장 낮은 효율이 아닌 경우도 물론 존재한다.
그러나 OPT는 이번에도 가장 적은 페이지 폴트인 것을 알 수 있다.

마지막으로,

```
hgstudio@hgstudio-virtual-machine:~/os5$ cat in5.txt
4
1 2 3 1 2 4 3 5 2 1 2 4 7 4 2 8 10 2 5 1
```

1) OPT인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 1
page reference string : 1 2 3 1 2 4 3 5 2 1 2 4 7 4 2 8 10 2 5 1

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

2) FIFO인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 2
page reference string : 1 2 3 1 2 4 3 5 2 1 2 4 7 4 2 8 10 2 5 1

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

3) LRU인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 3
page reference string : 1 2 3 1 2 4 3 5 2 1 2 4 7 4 2 8 10 2 5 1

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

4) second-Chance인 경우

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 4
page reference string : 1 2 3 1 2 4 3 5 2 1 2 4 7 4 2 8 10 2 5 1

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

네 개의 알고리즘의 페이지 폴트의 발생 횟수를 비교해보면,
OPT는 9번, FIFO는 14번, LRU는 12번, Second-Chance는 11번인 것을 알 수 있다.
이번에도 OPT의 경우 페이지 폴트 개수가 가장 적은 것을 알 수 있다.

추가적으로 FIFO의 문제이자 이상 현상인 Belady's Anomaly을 직접 확인해보겠다.

1) 3개의 프레임인 FIFO인 경우

```
hgstudio@hgstudio-virtual-machine:~/os5$ cat in3.txt
3
1 2 3 4 1 2 5 1 2 3 4 5
```

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 2
page reference string : 1 2 3 4 1 2 5 1 2 3 4 5

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

2) 4개의 프레임인 FIFO인 경우

```
hgstudio@hgstudio-virtual-machine:~/os5$ cat in4.txt
4
1 2 3 4 1 2 5 1 2 3 4 5
```

```
Used method (OPT(1) FIFO(2) LRU(3) Second-Chance(4)) : 2
page reference string : 1 2 3 4 1 2 5 1 2 3 4 5

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

프레임의 개수는 더 늘었는데 오히려 페이지 폴트의 횟수가 더 늘어나는 현상을 확인할 수 있다. 일반적으로 더 많은 프레임을 보유하면 그만큼 더 성능이 향상될 것으로 기대하지만 FIFO에서는 그 반대의 결과가 나오기도 한다.