PA#3:
Demand paging system을 위한 page replacement 기법 구현 및 검증
결과 보고서

2020년 가을학기
운영체제(SWE3004_41)
엄영익 교수님

소프트웨어학과
2018310520
김세란

<목차>

1. 개발 플랫폼 소개 및 컴파일/실행 방법 소개

운영체제: Linux 20.04 LTS
소스코드편집기: VSCode
프로그래밍 언어: C언어
컴파일러: GCC
컴파일 방법: 터미널에서 명령어 'gcc (소스코드명).c -o (실행파일명)' 사용
실행 방법: 터미널에 명령어 './(실행파일명)' 사용

2. 설계/구현 아이디어 및 그에 대한 구현 방법 및 기타 가정 설명

전체 설명)
MIN, FIFO, LRU, LFU, CLOCK, WS 각 기법을 함수로 나누어 실행하였다.
순서대로 각 기법에 대한 함수를 호출하기 전에, page frame 정보를 초기화하였다.
사용한 헤더파일은 <stdio.h>,<stdlib.h>이다.

****먼저 설명한 기법과 공통된 부분은 최대한 생략하고, 차이점 위주로 설명하겠다.**
1) MIN 기법

설계/구현 아이디어 및 기타 가정

각 string 하나에 대하여 (시간: 1초)
1) page fault 존재 여부 확인
2)-1. Page fault가 존재 하지 않는다면 3)으로
2)-2. 비어있는 frame이 존재하는 지 확인
2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다.
2)-2-2. MIN 기법에 따라 victim을 선정하고, 페이지를 교체한다.
MIN 기법: 존재한 프레임의 모든 페이지 번호에 대해, 종료되는 시각까지 가장 먼저 다시 참조되는 시각들을 비교한다. 종료되는 시각까지 참조되지 않는 것은 (가정)무한대(MAX_INT>STRING 최대길이)로 표시한다. (가정)이때, 가장 큰 다시 참조되는 시각을 가진 페이지 중 페이지 번호가 가장 작은 것을 victim으로 선정한다.
Victim으로 선정된 페이지가 있는 프레임에서, 페이지를 교체한다.
3) page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 출력한다.

구현 방법 – 데이터

```
typedef struct _frame{
    int page_number; // 로드된 페이지 번호
    int load_time; // 페이지가 해당 프레임에 로드된 시각
    int reference_time; // 페이지가 프레임에 있는 동안 참조된 가장 최근 시각
    int reference_frequecy; // 페이지가 "현재 시각 이전까지 모두" 참조된 횟수
    int reference_bit; // clock 기법에서 필요한 reference bit
}Frame;
```

Page frame 개수에 맞게 frame 구조체 배열을 선언하였다.
Frame 구조체는 page_number 라는 변수를 가지고 있다. 이는 각 시각 마다 page frame이 가지고 있는 page 번호를 의미한다.

```
Frame ** f = (Frame**) malloc(sizeof(Frame*)*M);
for(int i=0;i<M;i++){
    f[i]=(Frame*) malloc(sizeof(Frame));
    frame_init(f[i]);
}
```

구현 방법 – 알고리즘

```
for(int i=0;i<K;i++){
    for(int j=0;j<M;j++){ // Page Fault 여부 체크
        if(f[j]->page_number==RS[i]){
            PF=-1;
            break;
        }
        PF=0;
    }
    if(PF!=-1){ // Page Fault 가 존재
        PAGEFAULT++;
        if(NofE<M){ // 빈자리가 존재할 때
            PF=NofE; // 빈자리 중 가장 작은 번호의 페이지 프레임에 Page Fault 발생
            f[PF]->page_number=RS[i];
            NofE++;
            victim=-1;
        }
        else{ // 빈자리가 존재하지 않음, MIN기법에 맞게 페이지 교체
            //victim이 있는 frame 선택
            int max_fid,max_time,x;
            for(int j=0;j<M;j++){ //존재하는 모든 프레임에 대해
                for(int t=T;t<K;t++){ // 변수 x 에 이후에 다시 등장하는 시각을 대입
                    if(f[j]->page_number==RS[t]){
                        x=t;
                        break;
                    }
                    x=INT_MAX; // 이후에 다시 등장하지 않으면 무한대
                }

                if(j==0){
                    max_time=x;
                    max_fid=j;
                }
                else{
                    if(max_time<x){
                        max_time=x;
                        max_fid=j;
                    }
                }
            }
            //페이지 교체
            PF=max_fid;
            victim=f[PF]->page_number;
            f[PF]->page_number=RS[i];
        }
    }
```

옆의 코드는
구현 아이디어 2번를 구현한 것이다.

2)-1. Page fault가 존재 하지 않는다면 3)으로
2)-2. 비어있는 frame이 존재하는 지 확인
2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다.
2)-2-2. MIN 기법에 따라 victim을 선정하고, 페이지를 교체한다.
MIN 기법:
존재한 프레임의 모든 페이지 번호에 대해, 종료되는 시각까지 가장 먼저 다시 참조되는 시각들을 비교한다. 종료되는 시각까지 참조되지 않는 것은 (가정)무한대(MAX_INT>STRING 최대길이)로 표시한다. (가정)이때, 가장 큰 다시 참조되는 시각을 가진 페이지 중 페이지 번호가 가장 작은 것을 victim으로 선정한다.
Victim으로 선정된 페이지가 있는 프레임에서, 페이지를 교체한다.

2) FIFO 기법

설계/구현 아이디어 및 기타 가정

Victim을 찾기 위해, 각 프레임에 page를 로드할 때 각 프레임에 page 로딩 시각을 기록한다. 나머지는 1.MIN과 비슷하다. 다만, 페이지를 교체할 victim 을 찾는 방법에 차이가 있다. (2-2번)
각 string 하나에 대하여 (시간: 1초)
1) page fault 존재 여부 확인
2)-1. Page fault가 존재 하지 않는다면 3)으로
2)-2. 비어있는 frame이 존재하는 지 확인
2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다. **+ 로딩되는 프레임에 로딩하는 페이지의 로딩 시각을 업데이트한다.**
2)-2-2. FIFO 기법에 따라 victim을 선정하고, 페이지를 교체한다.
각 페이지 프레임에 대하여, 가장 작은 로딩 시각 정보를 가진 페이지 프레임의 페이지가 victim으로 선정된다. Victim 선정 후에는 해당 페이지를 교체한다. **+ 새로 로딩하는 페이지의 로딩 시각을 프레임의 로딩 시각 정보에 업데이트한다.**
FIFO기법: 3) page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 출력한다.

구현 방법 – 데이터

```
typedef struct _frame{
    int page_number; // 로드된 페이지 번호
    int load_time; // 페이지가 해당 프레임에 로드된 시각
    int reference_time; // 페이지가 프레임에 있는 동안 참조된 가장 최근 시각
    int reference_frequecy; // 페이지가 "현재 시각 이전까지 모두" 참조된 횟수
    int reference_bit; // clock 기법에서 필요한 reference bit
}Frame;
```

나머지 설명은 1.MIN과 동일하다.
각 프레임 구조체는 새로 로딩되는 페이지의 로딩 시각 정보를 가지고 있는데, 이 데이터는 Load_time이라는 변수에 업데이트된다

구현 방법 – 알고리즘

```
//FIFO: 프레임에 페이지 새로 로드 로딩 시각 업데이트
f[PF]->load_time=T;
```

```
//교체할 프레임 찾기 (FIFO: 각 프레임 별 페이지의 로딩 시각 가장 작은 것)
int min = f[0]->load_time;
int min_fid;
min_fid=0;
for(int j=1;j<M;j++){
    if(min>f[j]->load_time){
        min=f[j]->load_time;
        min_fid=j;
    }
}
```

나머지 설명은 1.MIN과 동일하다.
위:
새로 로딩하는 페이지의 로딩 시각을 프레임의 로딩 시각 정보에 업데이트함

아래: FIFO에서 victim 을 선정하는 법
각 페이지 프레임에 대하여, 가장 작은 로딩 시각 정보를 가진 페이지 프레임의 페이지가 victim으로 선정된다. Victim 선정 후에는 해당 페이지를 교체

	한다.

3) LRU 기법

설계/구현 아이디어 및 기타 가정
Victim을 찾기 위해, 각 프레임에 있는 page를 로드/참조할 때 각 프레임에 page 참조 시각 정보를 기록한다. **나머지는 1.MIN과 비슷하다. 다만, 페이지를 교체할 victim 을 찾는 방법에 차이가 있다. (2-2번)** 각 string 하나에 대하여 (시간: 1초) 1) page fault 존재 여부 확인 2)-1. Page fault가 존재 하지 않는다면 3)으로 2)-2. 비어있는 frame이 존재하는 지 확인 2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다. **2)-2-2.** LRU 기법에 따라 victim을 선정하고, 페이지를 교체한다. LRU기법: 각 페이지 프레임에 대하여, 가장 작은 참조 시각 정보를 가진 페이지 프레임의 페이지가 victim으로 선정된다. Victim 선정 후에는 해당 페이지를 교체한다. 3) **+ 로딩되는 프레임에 로딩하는 페이지의 참조 시각을 업데이트한다.** page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 출력한다.
구현 방법 – 데이터

`typedef struct _frame{`
`  int page_number; // 로드된 페이지 번호`
`  int load_time; // 페이지가 해당 프레임에 로드된 시각`
`  int reference_time; // 페이지가 프레임에 있는 동안 참조된 가장 최근 시각`
`  int reference_frequecy; // 페이지가 "현재 시각 이전까지 모두" 참조된 횟수`
`  int reference_bit; // clock 기법에서 필요한 reference bit`
`}Frame;`

나머지 설명은 1.MIN과 동일하다. 각 프레임 구조체는 새로 로딩되는 페이지의 참조 시각 정보를 가지고 있는데, 이 데이터는 reference_time이라는 변수에 업데이트된다
구현 방법 – 알고리즘
`//LRU: 프레임에 페이지 새로 로드 로딩/참조 시각 업데이트` `f[PF]->reference_time=T;`
나머지 설명은 1.MIN과 동일하다. **위:** **새로 로딩하는 페이지의 참조 시각을 프레임의 참조 시각 정보에 업데이트함**

`//교체할 프레임 찾기 (LRU: 각 프레임 별 페이지의 참조 시각 가장 작은 것)`
`int min = f[0]->reference_time;`
`int min_fid=0;`
`for(int j=1;j<M;j++){`
`  if(min>f[j]->reference_time){`
`  min=f[j]->reference_time;`
`  min_fid=j;`
`  }`
`}`

아래: LRU에서 victim 을 선정하는 법 각 페이지 프레임에 대하여, 가장 작은 참조 시각 정보를 가진 페이지 프레임의 페이지가 victim으로 선정된다. Victim 선정 후에는 해당 페이지를 교체한다

4) LFU 기법

설계/구현 아이디어 및 기타 가정
Victim을 찾기 위해, 각 string 의 하나씩 지남에 따라, 해당되는 페이지의 참조 횟수와, 가장 최근 참조 시각을 기록한다. (이것은 프레임에 대해 정보를 저장하지 않는다.) **나머지는 1.MIN과 비슷하다. 다만, 페이지를 교체할 victim 을 찾는 방법에 차이가 있다. (2-2번)** 각 string 하나에 대하여 (시간: 1초) 1) page fault 존재 여부 확인 2)-1. Page fault가 존재 하지 않는다면 3)으로 2)-2. 비어있는 frame이 존재하는 지 확인 2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다. **2)-2-2.** LFU 기법에 따라 victim을 선정하고, 페이지를 교체한다. LFU기법: 각 페이지 프레임에 대하여, 현재까지 가장 적게 참조 한 것 중(가정) 참조 시각이 가장 작은 page를 victim으로 선정된다. (locality), 참조 시각마저 같다면, 페이지 프레임 번호가 작은 것이 선택된다. Victim 선정 후에는 해당 페이지를 교체한다. 3) **+ 해당 페이지에 대해서 참조 횟수에 대한 변수를 증가시키고, 가장 최근 참조 시각도 현재로 업데이트한다.** page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 출력한다.
구현 방법 – 데이터

`int reference[100]; //page 번호 별 참조된 횟수`
`int r_time[100]; // page 번호 별 참조된 가장 최근 시각`
`for(int i=0;i<100;i++){`
`  reference[i]=0;`
`  r_time[i]=0;`
`}`

나머지 설명은 1.MIN과 동일하다. 하지만 이번에는 프레임에 victim 선정 시 필요한 정보를 저장하지 않는다. LFU 실행 함수 안에, Page 별 현재까지의 참조 횟수와, 참조된 가장 최근 시각에 대한 배열을 지역 변수로 선언한다.
구현 방법 – 알고리즘

`//LFU: 프레임에 페이지 참조 횟수 및 시각 증가`
`reference[RS[i]]++;`
`r_time[RS[i]]=T;`

나머지 설명은 1.MIN과 동일하다. **위:** **현재 참조하는 페이지에 대해서, 현재 횟수를 증가시키고, 최근 참조 시각도 업데이트한다.**

`//교체할 프레임 찾기 (LFU: 각 프레임 별 페이지의 참조 횟수 가장 작은 것,`
`// 횟수가 같은 것 중에는 참조 시각이 가장 이른 것)`
`int min = reference[f[0]->page_number];`
`int min_fid=0;`
`int min_time=r_time[f[0]->page_number];`
 	
`for(int j=1;j<M;j++){`
`  if(min>reference[f[j]->page_number]){`
`  min=reference[f[j]->page_number];`
`  min_time=r_time[f[j]->page_number];`
`  min_fid=j;`
`  }`
`  else if (min==reference[f[j]->page_number]){`
`  if (min_time > r_time[f[j]->page_number]){`
`  min_time=r_time[f[j]->page_number];`
`  min_fid=j;`
`  }`
`  }`
`}`
	아래: LFU에서 victim 을 선정하는 법
	Victim 페이지를 기준은 두 가지이다.
	페이지 프레임에 존재하는 모든 페이지 번호 중
	1. 우선 참조 횟수가 가장 작아야 한다.
	2. 이후 참조 횟수가 같은 것이 있다면, 참조 시각이 가장 작은 것을 선택한다.
	3. 참조 시각마저 같다면, 페이지 프레임 번호가 작은 페이지가 선택된다.

5) CLOCK 기법

설계/구현 아이디어 및 기타 가정
Victim을 찾기 위해, 각 프레임에 있는 page를 로드/참조할 때 각 프레임에 reference bit 을 업데이트한다. **나머지는 1.MIN과 비슷하다. 다만, 페이지를 교체할 victim 을 찾는 방법에 차이가 있다. (2-2번)** (가정) 초기 reference bit은 모두 0으로 초기화. (새로 load하면 1로 업데이트) , pointer는 프레임 번호 0에서 시작함. 각 string 하나에 대하여 (시간: 1초) 1) page fault 존재 여부 확인 2)-1. Page fault가 존재 하지 않는다면 **+ 해당 프레임의 참조 비트를 1로 업데이트.** 3)으로 2)-2. 비어있는 frame이 존재하는 지 확인 2)-2-1. 비어있는 frame이 존재한다면, 빈자리 중 가장 작은 번호의 페이지 프레임에 page fault가 발생한다고 보고, 페이지를 로드 한다. **2)-2-2.** CLOCK 기법에 따라 victim을 선정하고, 페이지를 교체한다. CLOCK기법: pointer가 0부터 프레임 번호를 돌면서, reference bit가 1인 것은 0으로 바꾸고 포인터를 증가시킨다, reference bit이 0인 프레임이 나올 때까지 지속한다. 가장 먼저 나오는 reference bit이 0인 프레임에 있는 페이지가 바로 교체할 victim 이다. **+ 해당 프레임의 참조 비트를 1로 업데이트** 3) page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 출력한다.
구현 방법 – 데이터

`typedef struct _frame{`
`  int page_number; // 로드된 페이지 번호`
`  int load_time; // 페이지가 해당 프레임에 로드된 시각`
`  int reference_time; // 페이지가 프레임에 있는 동안 참조된 가장 최근 시각`
`  int reference_frequecy; // 페이지가 "현재 시각 이전까지 모두" 참조된 횟수`
`  int reference_bit; // clock 기법에서 필요한 reference bit`
`}Frame;`

나머지 설명은 1.MIN과 동일하다. 각 프레임 구조체는 reference_bit을 가지고 있다.

`int pointer=0;`	아래는 clock기법에 필요한 포인터에 대한 변수이다. 0에서 시작하여, Frame 전체를 계속 돈다.
구현 방법 – 알고리즘	

`//CLOCK: reference bit 1`
`f[j]->reference_bit=1;`
`//CLOCK: reference bit 1`
`f[PF]->reference_bit=1;`
	나머지 설명은 1.MIN과 동일하다.
	위:
	참조된 페이지의 페이지 프레임의 reference bit을 1로 업데이트한다.

`//교체할 프레임 찾기 (CLOCK: reference bit이 0인 것)`
`while(1){`
`  if(f[pointer]->reference_bit==0){`
`  PF=pointer;`
`  pointer=(pointer+1)%M;`
`  break;`
`  }`
`  f[pointer]->reference_bit=0;`
`  pointer=(pointer+1)%M;`
`}`
	아래: LFU에서 victim 을 선정하는 법
	Reference bit이 1이면 0으로 업데이트 후 pointer를 증가시킨다.
	Reference bit이 0인 것을 찾을 때 까지 pointer를 증가시킨다.

6) WS 기법

설계/구현 아이디어 및 기타 가정
각 string 하나에 대하여 (시간: 1초) 1) page 삭제 : 현재 mem에 존재하는 페이지 중 (1) 현재 페이지 번호 아닌 것. (2) 현재 이전의 Window Size의 스트링 만큼에 존재하지 않는 것 → 이렇게 삭제를 먼저 하면, 메모리에 존재하는 페이지 프레임 개수가 window size +1 개로 유지된다. 따라서 메모리에 존재하는 페이지를 window size + 1만큼의 배열에 저장한다. 2) page 추가 : PAGE FAULT가 발생하여 page 로드가 필요한 지 파악 후, page fault가 발생하였다면, page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 나타내는 배열 중 빈자리에 해당 page를 로드하고, 메모리 상태를 출력한다.
구현 방법 – 데이터

`int mem[W+1];`
`for(int i=0;i<W+1;i++){`
`  mem[i]=-1;`
`}`

위 설명 처럼 삭제를 먼저 하면, 메모리에 존재하는 페이지 프레임 개수가 window size +1 개로 유지된다. 따라서 메모리에 존재하는 페이지를 window size + 1만큼의 배열에 저장한다. 가장 처음에는 -1로 초기화한다. -

	1은 아무 페이지가 아니라는 의미이다.

구현 방법 – 알고리즘

```
//삭제 : 1) 현재 페이지 번호 아닌 것, 2) 앞의 W 스트링 만큼에 존재하지 않는 것
for(int j=0;j<W+1;j++){
    if(mem[j]==-1){
        continue;
    }
    else{
        int d=0;
        for(int k=0;k<=W;k++){
            if(mem[j]==RS[i-k]){
                d=1;
                break;
            }
        }
        if(d==0){
            out=mem[j];
            mem[j]=-1;
            NofFA--;
        }
    }
}
```

****삭제를 먼저 진행하고, 추가를 진행한다.**

1) page 삭제 : 현재 mem에 존재하는 페이지 중 (1) 현재 페이지 번호 아닌 것. (2) 현재 이전의 Window Size의 스트링 만큼에 존재하지 않는 것을 삭제,
현재 메모리 상태에 있는 페이지 개수(NofFA) 감소, 메모리 상태에 존재하는 페이지 제거 (mem 배열에 -1 대입)

```
//추가
int x=0;
for(int j=0;j<W+1;j++){
    if(mem[j]==RS[i]){
        x=1;
        break;
    }
}
if(x==0){//PAGE FAULT, 추가
    PAGEFAULT++;
    int j;

    for( j=0;j<W+1;j++){
        if(mem[j]==-1){
            break;
        }
    }
    in=RS[i];
    mem[j]=RS[i];
    NofFA++;
}
```

2) page 추가 : PAGE FAULT가 발생하여 page 로드가 필요한 지 파악 후, page fault가 발생하였다면, page fault 여부에 따라 page fault 횟수에 대한 변수를 증가시키고, 메모리 상태를 나타내는 배열 중 빈자리에 해당 page를 로드하고, 메모리 상태를 출력한다.
현재 메모리 상태에 있는 페이지 개수 증가(NofFA), 메모리 상태에 존재하는 페이지 추가 (mem 배열에 빈자리: -1)

예외처리)

```
int rt=getInput(N,M,W,K,RS);
// 예외처리, 비정상적인 input을 받으면 에러 종류를 출력하고 프로그램 종료
if(rt==-1){
return 0;
}
```

3. 실행 결과 출력물 및 출력물에 대한 설명

공통 설명) 시간 단위를 [초]로 한다. 1초에서 시작하고, string 하나 지날 때 마다 1 초씩 증가한다고 가정한다.

교안에 있는 예시)

1) MIN

Input.txt	실행 결과
6 4 3 14 1 2 6 1 4 5 1 2 1 4 5 6 4 5	MIN FRAME NUM: 0 1 2 3 T= 1: (1) — — — PAGE FAULT: (— -> 1) AT [#0] T= 2: 1 (2) — — PAGE FAULT: (— -> 2) AT [#1] T= 3: 1 2 (6) — PAGE FAULT: (— -> 6) AT [#2] T= 4: 1 2 6 — NO PAGE FAULT T= 5: 1 2 6 (4) PAGE FAULT: (— -> 4) AT [#3] T= 6: 1 2 (5) 4 PAGE FAULT: (6 -> 5) AT [#2] T= 7: 1 2 5 4 NO PAGE FAULT T= 8: 1 2 5 4 NO PAGE FAULT T= 9: 1 2 5 4 NO PAGE FAULT T= 10: 1 2 5 4 NO PAGE FAULT T= 11: 1 2 5 4 NO PAGE FAULT T= 12: (6) 2 5 4 PAGE FAULT: (1 -> 6) AT [#0] T= 13: 6 2 5 4 NO PAGE FAULT T= 14: 6 2 5 4 NO PAGE FAULT PAGEFAULT: 6
출력물 설명	
가장 위 줄: Page Frame 번호 중간: T가 1초씩 증가하면서 메모리 상태(페이지가 존재하면 페이지 번호, 존재하지 않으면 --) 메모리 상태: 괄호가 있는 것: PAGE FAULT 발생 위치 옆의 설명: PAGE FAULT 여부, 교체된 페이지 번호, 교체된 프레임 위치 번호 가장 아래 줄: PAGE FAULT 횟수	

2) FIFO

Input.txt	실행 결과
6 4 3 14 1 2 6 1 4 5 1 2 1 4 5 6 4 5	FIFO FRAME NUM: 0 1 2 3 T= 1: (1) — — — PAGE FAULT: (— -> 1) AT [#0] T= 2: 1 (2) — — PAGE FAULT: (— -> 2) AT [#1] T= 3: 1 2 (6) — PAGE FAULT: (— -> 6) AT [#2] T= 4: 1 2 6 — NO PAGE FAULT T= 5: 1 2 6 (4) PAGE FAULT: (— -> 4) AT [#3] T= 6: (5) 2 6 4 PAGE FAULT: (1 -> 5) AT [#0] T= 7: 5 (1) 6 4 PAGE FAULT: (2 -> 1) AT [#1] T= 8: 5 1 (2) 4 PAGE FAULT: (6 -> 2) AT [#2] T= 9: 5 1 2 4 NO PAGE FAULT T= 10: 5 1 2 4 NO PAGE FAULT T= 11: 5 1 2 4 NO PAGE FAULT T= 12: 5 1 2 (6) PAGE FAULT: (4 -> 6) AT [#3] T= 13: (4) 1 2 6 PAGE FAULT: (5 -> 4) AT [#0] T= 14: 4 (5) 2 6 PAGE FAULT: (1 -> 5) AT [#1] PAGEFAULT: 10
출력물 설명	
MIN과 동일	

3) LRU

Input.txt	실행 결과
6 4 3 14 1 2 6 1 4 5 1 2 1 4 5 6 4 5	LRU FRAME NUM: 0 1 2 3 T= 1: (1) — — — PAGE FAULT: (— -> 1) AT [#0] T= 2: 1 (2) — — PAGE FAULT: (— -> 2) AT [#1] T= 3: 1 2 (6) — PAGE FAULT: (— -> 6) AT [#2] T= 4: 1 2 6 — NO PAGE FAULT T= 5: 1 2 6 (4) PAGE FAULT: (— -> 4) AT [#3] T= 6: 1 (5) 6 4 PAGE FAULT: (2 -> 5) AT [#1] T= 7: 1 5 6 4 NO PAGE FAULT T= 8: 1 5 (2) 4 PAGE FAULT: (6 -> 2) AT [#2] T= 9: 1 5 2 4 NO PAGE FAULT T= 10: 1 5 2 4 NO PAGE FAULT T= 11: 1 5 2 4 NO PAGE FAULT T= 12: 1 5 (6) 4 PAGE FAULT: (2 -> 6) AT [#2] T= 13: 1 5 6 4 NO PAGE FAULT T= 14: 1 5 6 4 NO PAGE FAULT PAGEFAULT: 7
출력물 설명	
MIN과 동일	

4) LFU

Input.txt	실행 결과
6 4 3 14 1 2 6 1 4 5 1 2 1 4 5 6 4 5	LFU FRAME NUM: 0 1 2 3 T= 1: (1) — — — PAGE FAULT: (— -> 1) AT [#0] T= 2: 1 (2) — — PAGE FAULT: (— -> 2) AT [#1] T= 3: 1 2 (6) — PAGE FAULT: (— -> 6) AT [#2] T= 4: 1 2 6 — NO PAGE FAULT T= 5: 1 2 6 (4) PAGE FAULT: (— -> 4) AT [#3] T= 6: 1 (5) 6 4 PAGE FAULT: (2 -> 5) AT [#1] T= 7: 1 5 6 4 NO PAGE FAULT T= 8: 1 5 (2) 4 PAGE FAULT: (6 -> 2) AT [#2] T= 9: 1 5 2 4 NO PAGE FAULT T= 10: 1 5 2 4 NO PAGE FAULT T= 11: 1 5 2 4 NO PAGE FAULT T= 12: 1 5 (6) 4 PAGE FAULT: (2 -> 6) AT [#2] T= 13: 1 5 6 4 NO PAGE FAULT T= 14: 1 5 6 4 NO PAGE FAULT PAGEFAULT: 7
출력물 설명	
MIN과 동일	

5) CLOCK

Input.txt	실행 결과
4)LFU와 동일	CLOCK FRAME NUM: 0 1 2 3 T= 5: --> 1/1 2/1 6/1 (4/1) PAGE FAULT: displaced: —, loaded: 4 AT [#3] T= 6: (5/1) -> 2/0 6/0 4/0 PAGE FAULT: displaced :1, loaded: 5 AT [#0] T= 7: 5/1 (1/1) -> 6/0 4/0 PAGE FAULT: displaced :2, loaded: 1 AT [#1] T= 8: 5/1 1/1 (2/1) -> 4/0 PAGE FAULT: displaced :6, loaded: 2 AT [#2] T= 9: 5/1 1/1 2/1 -> 4/0 NO PAGE FAULT T= 10: 5/1 1/1 2/1 -> 4/1 NO PAGE FAULT T= 11: 5/1 1/1 2/1 -> 4/1 NO PAGE FAULT T= 12: --> 5/0 1/0 2/0 (6/1) PAGE FAULT: displaced :4, loaded: 6 AT [#3] T= 13: (4/1) -> 1/0 2/0 6/1 PAGE FAULT: displaced :5, loaded: 4 AT [#0] T= 14: 4/1 (5/1) -> 2/0 6/1 PAGE FAULT: displaced :1, loaded: 5 AT [#1] PAGEFAULT: 10
출력물 설명	
페이지가 모두 로드 된 이후를 출력 (그 전의 모든 숫자는 PAGE FAULT 발생) 가장 위 줄: Page Frame 번호 중간: T가 1초씩 증가하면서 메모리 상태(페이지가 존재하면 페이지 번호, 존재하지 않으면 --) 메모리 상태: 괄호가 있는 것: PAGE FAULT 발생 위치, →가 있는 것: clock pointer가 가리키는 위치. 옆의 설명: PAGE FAULT 여부, 교체된 페이지 번호, 교체된 프레임 위치 번호	

가장 아래 줄: PAGE FAULT 횟수

6) WS

Input.txt	실행 결과
5 4 3 13 4 3 0 2 2 3 1 2 4 2 4 0 3	WS FRAME NUM: MEMORY STATE (4) — — — PAGE FAULT IN: 4 OUT: — 4 (3) — — PAGE FAULT IN: 3 OUT: — 4 3 (0) — PAGE FAULT IN: 0 OUT: — 4 3 0 (2) PAGE FAULT IN: 2 OUT: — — 3 0 (2) NO PAGE FAULT IN: — OUT: 4 — (3) 0 2 NO PAGE FAULT IN: — OUT: — (1) 3 — 2 PAGE FAULT IN: 1 OUT: 0 1 3 — (2) NO PAGE FAULT IN: — OUT: — 1 3 (4) 2 PAGE FAULT IN: 4 OUT: — 1 — 4 (2) NO PAGE FAULT IN: — OUT: 3 — — (4) 2 NO PAGE FAULT IN: — OUT: 1 (0) — 4 2 PAGE FAULT IN: 0 OUT: — 0 (3) 4 2 PAGE FAULT IN: 3 OUT: — PAGEFAULT: 8 MEAN NUM OF FRAMES ALLOCATED:2.71
출력물 설명	
페이지가 모두 로드 된 이후를 출력 (그 전의 모든 숫자는 PAGE FAULT 발생) 가장 위 줄: Page Frame 번호 중간: T가 1초씩 증가하면서 메모리 상태(페이지가 존재하면 페이지 번호, 존재하지 않으면 --) 메모리 상태: 괄호가 있는 것: PAGE FAULT 발생 위치, 옆의 설명: PAGE FAULT 여부, 추가된, 삭제된 페이지 번호 PAGEFAULT 횟수, MEAN NUMBER OF FRAMES ALLOCATED: (소숫점 두자리까지 표기)	

다양한 입력에 대한 결과 (같은 폴더에 입력을 첨부했습니다.)

1) input1.txt
40 10 10 1000
....
PAGE FAULT 횟수(기법 별 순서대로): 657 905 907 906 906 898

2) input2.txt ← input1.txt에 비해 페이지 번호들이 덜 분산 되어있는 케이스.
10 10 10 1000
....
PAGE FAULT 횟수(기법 별 순서대로): 452 751 746 761 745 754

3) input3.txt
100 20 100 100000
.....
PAGE FAULT 횟수(기법 별 순서대로): 47963 79819 79691 79678 79742 35883