Operating System

Assignment 3

2013147513 컴퓨터과학과
조영재

1. 작성한 프로그램의 동작 과정과 구현 방법.

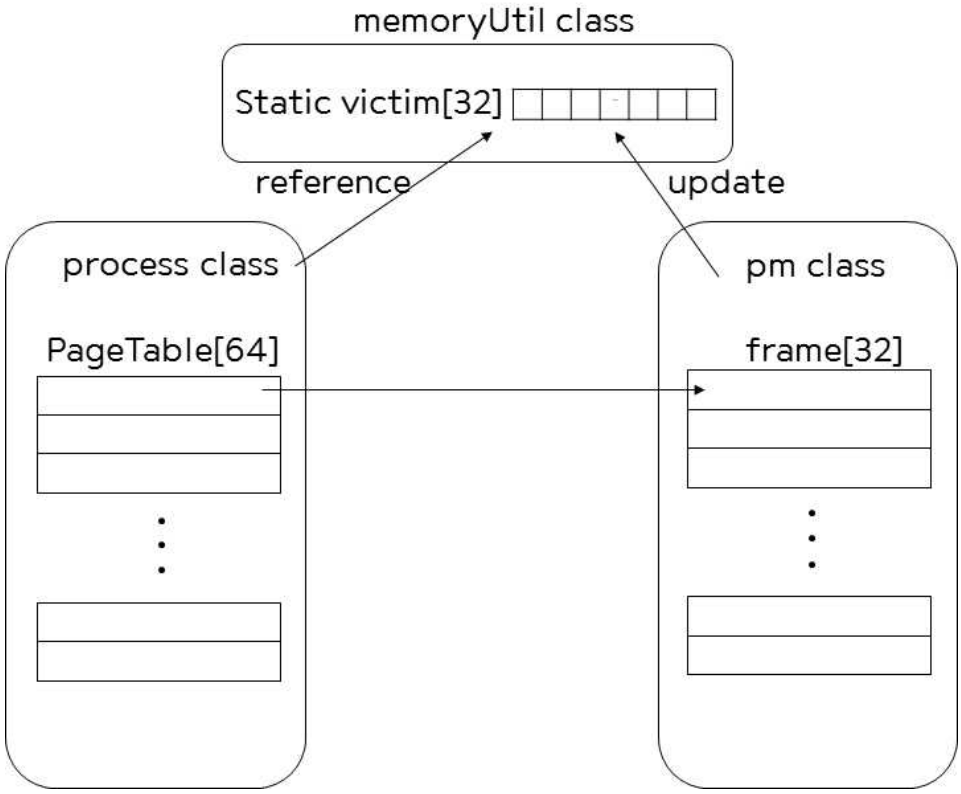
전반적인 프로그램은 다음과 같은 파일들로 구성된다.
① main.cpp : 파일을 입력으로 받아 각각의 명령어들을 파싱하고, 그에 따라 명령어들을 처리하게 하는 메인함수

②memoryUtil.cpp : process클래스와 pm(physicalMemory)클래스의 부모클래스. 두 개의 하위 클래스가 공통으로 LRU교체정책과 관련된 자료구조를 사용할 수 있도록 하기 위해 victim이라는 리스트를 스태틱 변수로 가지고 있다.

③ process.cpp : 프로세스에 관련된 정보를 가지고 있는 클래스. 가지고 있는 정보로는, 해당 페이지의 allocation ID, Valid Bit, pid, Page table을 가지고 있으며, 그들을 활용할 수 있는 함수들을 포함하고 있다.

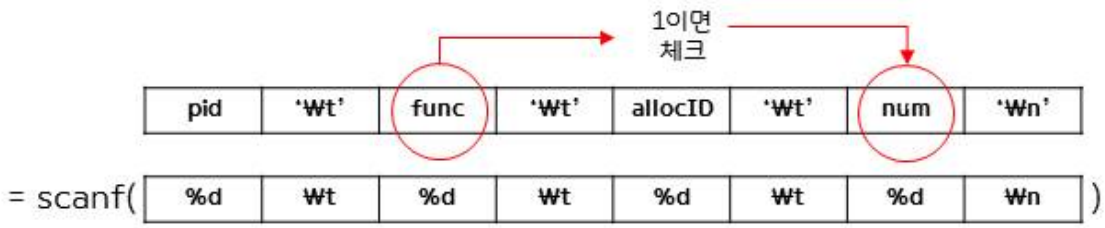
④ physicalMemory.cpp : 프로세스에서 메모리로 메모리 할당을 해주기 위한 변수들과 함수들을 가지고 있다. 구체적으로는 프레임 배열, 페이지폴트 카운터 및 메모리 할당 함수와 LRU를 업데이트 해주는 함수들이 있다.

구현한 자료구조는 다음과 같다.



input.txt를 받는 방법

input.txt를 리다이렉션으로 주는 경우는 입력이 input.txt파일과 자동으로 연결된다. 따라서 scanf로 파일을 읽을 수 있다. 각각의 입력 형식에 맞추어 첫 번째와 두 번째 줄은 한 개의 정수값만 받았으며, 세 번재 줄 부터는 본격적인 명령어 이므로, 한줄의 입력을 받을 때 우선 앞의 세 개의 정수만 입력을 받아, 두 번째 정수의 0,1 여부에 따라 4번째 정수를 받거나 다음줄로 넘어가게 만들었다. 추가적으로 명령어 뒤에 주석이 뒤에 붙을수도 있기 때문에, 개행문자가 나올 때 까지 문자를 계속 읽게 만들었다.

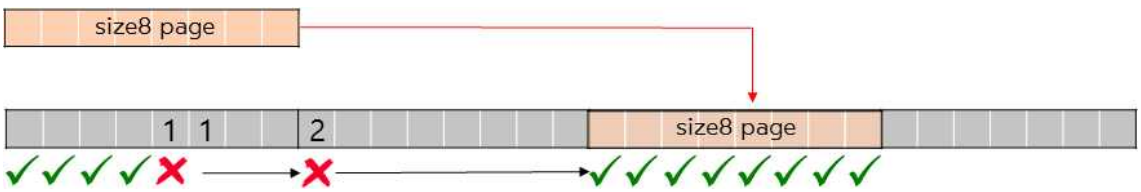


Buddy system

프로세스의 페이지테이블에서 물리메모리로 할당하는데에는 buddy system을 이용하였다. 제일 처음, 할당하려는 allocID가 페이지테이블에서 차지하고 있는 개수보다 크거나 같은 것 중 가장 작은 2의 지수승을 찾아야 하고, 이것이 필요한 frame수 이다. 이는 다음과 같은 공식으로 구할 수 있다.

$$^{eil(\log n)}, \ n = \text{페이지테이블 사이즈}$$

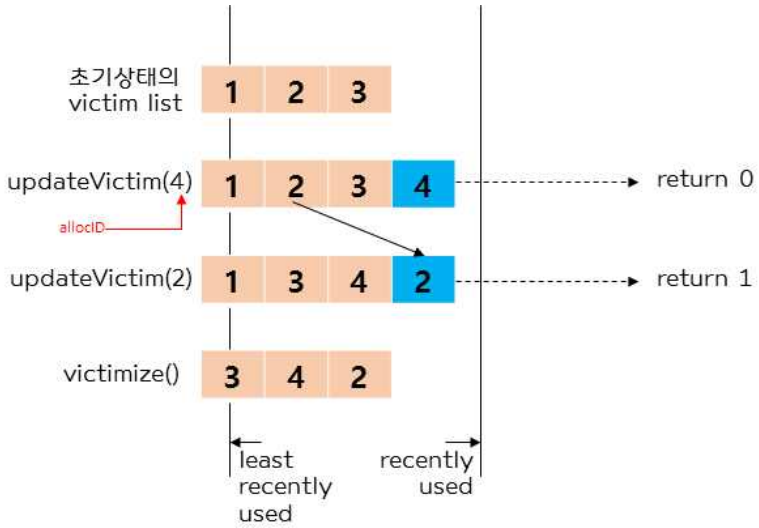
예를들어 차지하고 있는 페이지테이블이 13이라면, 필요한 frame은 16이 된다. 이렇게 필요한 frame수를 f라 한다면, 물리메모리의 frame들을 각각 f개의 프레임이 한세트를 이루도록 나눈다. 이후 물리메모리의 낮은 주소부터 시작하여, 프레임이 차 있는지를 확인하여 해당 세트 내의 f개의 프레임이 모두 비어있는 세트에 페이지를 할당하고, 하나라도 다른 페이지가 할당되어 있으면, 다음 세트로 넘어가서 비어있는지 체크를 시작한다.



위의 과정을 처리해 주는 함수가 checkAllocSpace()함수이고, 이 함수는 메모리 할당에 성공하면 1을, 할당할 메모리 공간이 없으면 0을 반환하도록 구현하였다. 만약 반환값이 0이라면 LRU정책에 의해 가용 공간이 생길때까지 메모리를 비워나간 후, 공간이 생기면 메모리를 할당하도록 구현하였다.

LRU replacement policy

물리메모리가 가득 찼을 경우, 교체해야할 페이지를 정하기 위해 LRU 교체정책을 사용하였다. LRU교체정책을 사용하기 위해 victim이라는 리스트를 만들었다. victim이라는 리스트에는, memory access가 수행될 때 해당하는 allocID가 리스트의 끝에 추가되게 되어있다. 또한, 접근하려는 allocID가 이미 리스트안에 존재할 경우, 해당 allocID를 꺼내서 리스트의 끝으로 보낸다. 이렇게 구현하면, 리스트의 맨 앞쪽에는 사용된지 가장 오래된 allocID부터 뒤로갈수록 최근에 사용된 allocID로 정렬되게 된다. 물리메모리가 가득 차서 frame에서 페이지를 제거해야할 상황이 생기면, 이 victim리스트의 맨 처음에 있는 allocID를 가진 페이지부터 제거하면 된다.



이와같이 victim 리스트를 업데이트 해주는 함수가 updateVictim()함수이다. 이 함수는 물리메모리에 이미 allocID가 있으면, 1을 반환하고 없으면 0을 반환해 준다. 그리고 페이지를 교체해야할 때, 리스트 맨앞의 원소를 물리메모리에서 제거해주는 함수가 victimize()이다.
이 함수들을 모아 alloc()함수를 만들었다. 이 함수가 실질적으로 메모리할당부터 LRU정책에 의한 교체까지 해주는 함수이다.

```
void pm::alloc(int allocID, int pageNum)
{
    if(pageNum > 32 || pageNum == 0)
    {
        //printf("invalid size of physical memo
        //printf("allocID:%d, pageNum:%d\n", al
        //exit(0);
        return;
    }
    //if allocID is already allocated, end
    if(updateVictim(allocID)) return;
    while(!checkAllocSpace(allocID, pageNum)){
        victimize();
    }
}
```

예외상황 처리

victim리스트를 업데이트함. 만약 물리메모리에 이미 allocID가 존재한다면, 메모리할당을 시도하지 않고 함수를 종료한다

물리메모리에 할당이 불가능하면 할당 가능할때까지 메모리를 계속 비워나감

화면에 결과로 표시되는 LRU는 victim리스트의 순서대로 물리메모리에서 몇 개의 frame을 차지하고 있는지 계산하여 출력하도록 하였다.

Valid Bit 및 Page fault 처리

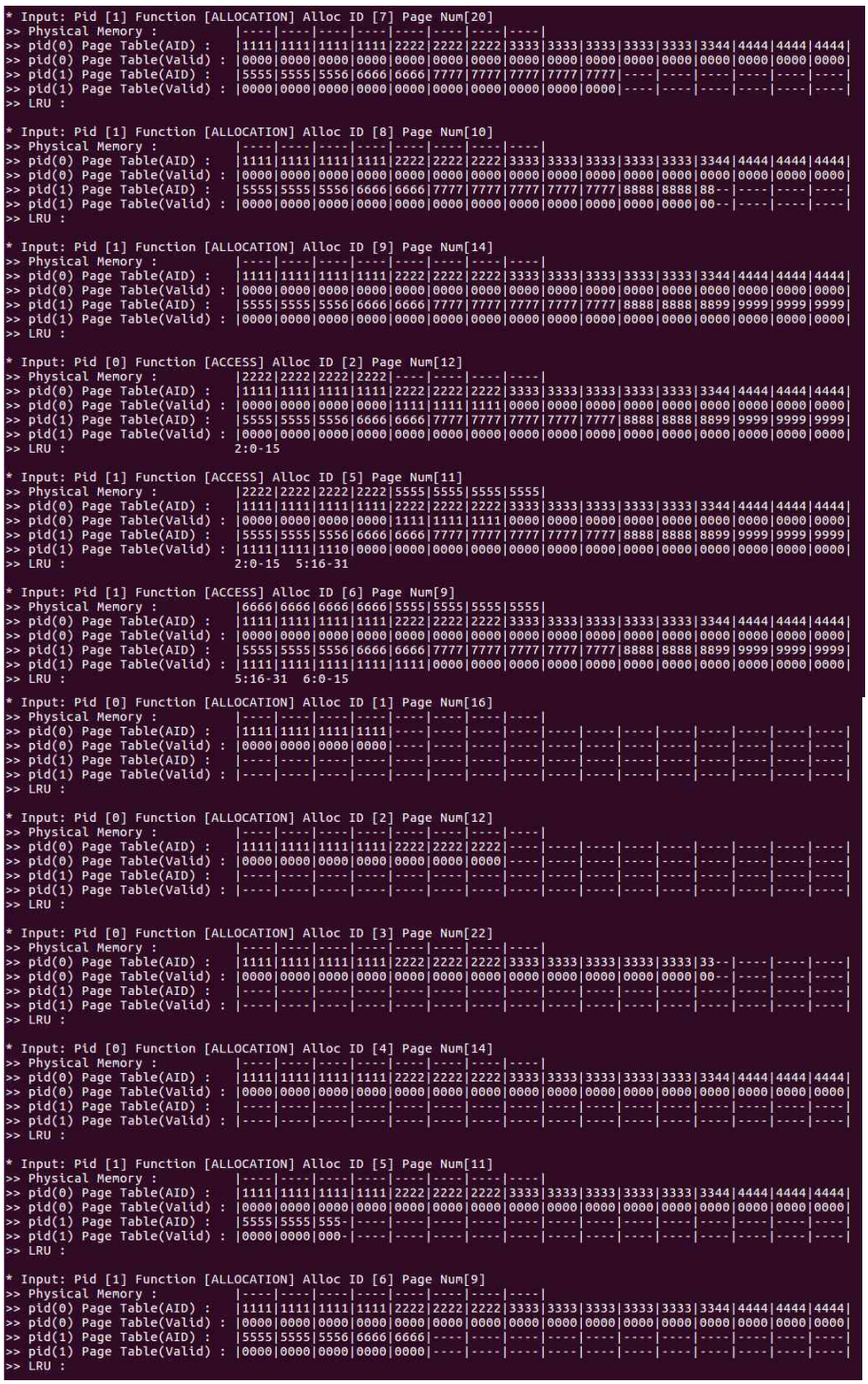
Victim리스트에 있는 allocID를 가진 페이지는 항상 물리메모리에 할당되어있다는 성질을 이용하여서 valid bit와 page fault를 구하였다.

Valid Bit는 victim리스트를 조사하여 그 안에 있는 allocID가 페이지테이블에 존재하면, allocID에 해당하는 페이지는 물리메모리에 존재한다는 뜻이므로, 그 allocID에 해당하는 valid bit를 모두 1로 바꾸도록 구현하였다.
Page fault는 victim리스트에 없는 allocID를 물리메모리에 할당할 경우 1씩 증가하도록 설계하였다. victim리스트에 없다는 말은 즉 물리메모리에 페이지가 없다는 뜻 이므로, page fault가 발생한 경우라고 볼 수 있다.

2. uname -a 실행결과화면과 개발 환경 명시

3. 결과 화면 스크린샷과 그것에 대한 토의 내용

input.txt는 과제 스펙에서 사용한 예제를 텍스트파일로 만들어서 사용하였다.
결과는 다음과 같다.



```
* Input: Pid [1] Function [ACCESS] Alloc ID [5] Page Num[11]
>> Physical Memory :          |6666|6666|6666|6666|5555|5555|5555|5555|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |1111|1111|1111|1111|1111|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      6:0-15  5:16-31

* Input: Pid [0] Function [ACCESS] Alloc ID [2] Page Num[12]
>> Physical Memory :          |2222|2222|2222|2222|5555|5555|5555|5555|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|1111|1111|1111|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |1111|1111|1110|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      5:16-31  2:0-15

* Input: Pid [0] Function [ACCESS] Alloc ID [3] Page Num[22]
>> Physical Memory :          |3333|3333|3333|3333|3333|3333|3333|3333|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1100|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      3:0-31

* Input: Pid [0] Function [ACCESS] Alloc ID [4] Page Num[14]
>> Physical Memory :          |4444|4444|4444|4444|----|----|----|----|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      4:0-15

* Input: Pid [1] Function [ACCESS] Alloc ID [6] Page Num[9]
>> Physical Memory :          |4444|4444|4444|4444|6666|6666|6666|6666|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0001|1111|1111|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      4:0-15  6:16-31

* Input: Pid [1] Function [ACCESS] Alloc ID [8] Page Num[10]
>> Physical Memory :          |8888|8888|8888|8888|6666|6666|6666|6666|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0001|1111|1111|0000|0000|0000|0000|0000|1111|1111|1100|0000|0000|0000|
>> LRU :                      6:16-31  8:0-15

* Input: Pid [0] Function [ACCESS] Alloc ID [3] Page Num[22]
>> Physical Memory :          |3333|3333|3333|3333|3333|3333|3333|3333|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1100|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      3:0-31

* Input: Pid [0] Function [ACCESS] Alloc ID [4] Page Num[14]
>> Physical Memory :          |4444|4444|4444|4444|----|----|----|----|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      4:0-15

* Input: Pid [1] Function [ACCESS] Alloc ID [6] Page Num[9]
>> Physical Memory :          |4444|4444|4444|4444|6666|6666|6666|6666|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0001|1111|1111|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      4:0-15  6:16-31

* Input: Pid [1] Function [ACCESS] Alloc ID [8] Page Num[10]
>> Physical Memory :          |8888|8888|8888|8888|6666|6666|6666|6666|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0001|1111|1111|0000|0000|0000|0000|0000|1111|1111|1100|0000|0000|0000|
>> LRU :                      6:16-31  8:0-15

* Input: Pid [0] Function [ACCESS] Alloc ID [4] Page Num[14]
>> Physical Memory :          |8888|8888|8888|8888|4444|4444|4444|4444|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1100|0000|0000|0000|
>> LRU :                      8:0-15  4:16-31

* Input: Pid [0] Function [ACCESS] Alloc ID [2] Page Num[12]
>> Physical Memory :          |2222|2222|2222|2222|4444|4444|4444|4444|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1111|2222|2222|2222|3333|3333|3333|3333|3333|3344|4444|4444|4444|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|1111|1111|1111|0000|0000|0000|0000|0000|0011|1111|1111|1111|
>> pid(1) Page Table(AID) :   |5555|5555|5556|6666|6666|7777|7777|7777|7777|7777|8888|8888|8899|9999|9999|9999|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> LRU :                      4:16-31  2:0-15

page fault = 10
```

1~9번 instruction까지 각각 pid0번과 1번 프로세스 페이지테이블에 잘 할당되는 것을 볼 수 있다.
10~11번 instruction에서는 physical memory에 원하는 allocID를 가진 페이지가 할당된 것을 볼 수 있다.
12번 instruction에서는 LRU교체정책에 의해 사용된지 더 오래된 2번 allocID를 가진 frame이 6번 allocID로 교체되는 것을 볼 수 있다.
13번 instruction에서는 5번 allocID를 access하려 하는데, 이미 존재하므로 physical memory에는 변화가 없다. 다만 LRU순서가 5번 allocID를 가진 frame이 더 최근에 사용된 것으로 6번과 순서가 바뀌게 된다.
14번 instruction에서는 physical memory에서 2번 allocID가 6번 allocID자리에 들어가게 된다.
15번 instruction에서는 22개의 페이지가 들어갈 공간이 필요하므로, 32개의 모든 physical memory가 비워지고 그 자리에 allocID 3이 할당된다.
16번 instruction에서는 allocID4인 페이지가 들어가야 하므로 allocID가 3이었던 frame들이 모두 비워지고 allocID 4로 채워지는 것을 관찰할 수 있다.
17번 instruction에서는 physical memory에 들어갈 자리가 남아있으므로 바로 할당되는 것을 볼 수 있다.
18~20번 instruction에서는 모두 LRU교체정책에 의해 요구된 페이지가 physical memory에 정상적으로 할당되는 것을 관찰할 수 있다.

11번의 memory access instruction에서 13번째 instruction을 제외하고는 모두 physical memory에 page가 존재하지 않는 page fault이므로 page fault는 10번이 발생하게 된다.

4. 과제 수행 시 겪었던 어려움

valid bit를 구현하는 것이 어려웠다. 어떤식으로 구해야 할지는 알았지만, victim리스트를 process클래스 또는 pm클래스 한쪽에 두면, 다른쪽 클래스에서는 접근하기가 어려우므로 이를 이용하지 못해 valid bit를 구하지 못하거나, replacement policy를 제대로 실현하기가 힘들었다. 그래서 이 victim리스트를 process클래스와 pm클래스의 상위클래스의 static변수로 만들어 양쪽 클래스에서 모두 접근 가능하도록 구현하였다. 이를 통해 process클래스에서는 victim리스트를 통해 valid bit를 구할 수 있게 되었고, pm클래스에서는 victim리스트를 통해 LRU replacement policy를 구현할 수 있었다.