고려대학교
KOREA UNIVERSITY

KU-The Future

Page Replacement Algorithms 과제 해설

2022년 1학기 운영체제 수업 실습 03

조교 김명현 (freckie@korea.ac.kr)

2022.06.08

Section 1

“Assignment 3-1”

Assignment 3-1 (6점)

Stack을 이용한 LRU Replacement 시뮬레이션

- 파일명: 학번-이름-1.c

  1. **generate_ref_arr()** 함수 구현 (랜덤 reference string 생성하여 리턴)
  2. **lru()** 함수 구현 (page fault 발생 횟수 리턴)

- 보고서에 다음 내용 추가

  - 주어진 Reference String S에 대해, 페이지 참조마다 Stack 내용이 어떻게 변하는지 표로 작성

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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

stack	[0]	[1]	[2]	[3]
page	7			

head → bottom

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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

stack	[0]	[1]	[2]	[3]
page	0	7		

head → ← bottom

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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

head → | *stack* | [0] | [1] | [2] | [3] | ← bottom

page	1	0	7	

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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

stack	[0]	[1]	[2]	[3]
page	2	1	0	7

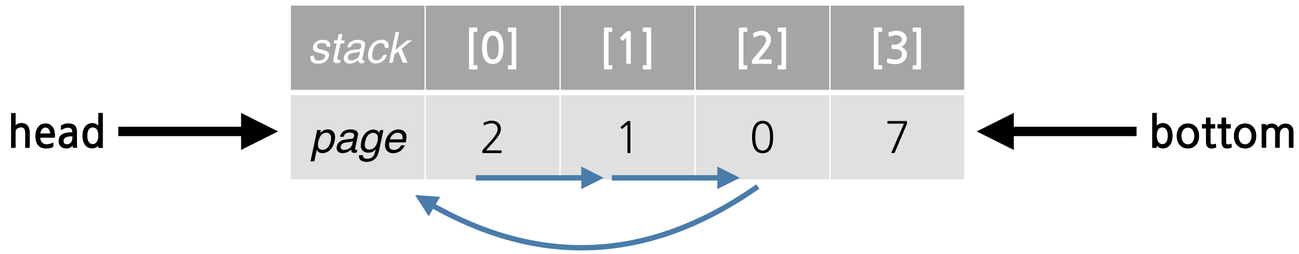
head → ← bottom

Stack을 이용한 LRU 시뮬레이션

hit

$S =$

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



Stack을 이용한 LRU 시뮬레이션

hit

$S =$

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

head → | *stack* | [0] | [1] | [2] | [3] | ← bottom

page	0	2	1	7

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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

stack	[0]	[1]	[2]	[3]
page	0	2	1	7

head → bottom

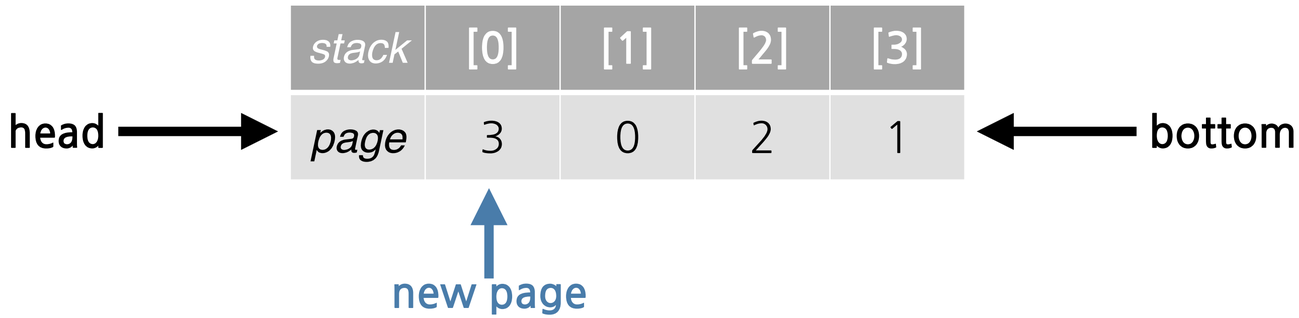
victim page

Stack을 이용한 LRU 시뮬레이션

fault

$S =$

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



Stack을 이용한 LRU 시뮬레이션

hit

$S =$

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

head → | *stack* | [0] | [1] | [2] | [3] | ← bottom

page	0	3	2	1

1. Assignment 3-1

Stack을 이용한 LRU 시뮬레이션

$S =$

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

stack	[0]	[1]	[2]	[3]
page	1	0	7	2

head → ← bottom

1. Assignment 3-

Stack을 이용한 LR 션

```
7 | 7 . . . (fault)
0 | 0 7 . . (fault)
1 | 1 0 7 . (fault)
2 | 2 1 0 7 (fault)
0 | 0 2 1 7
3 | 3 0 2 1 (fault)
0 | 0 3 2 1
4 | 4 0 3 2 (fault)
2 | 2 4 0 3
3 | 3 2 4 0
0 | 0 3 2 4
3 | 3 0 2 4
2 | 2 3 0 4
1 | 1 2 3 0 (fault)
2 | 2 1 3 0
0 | 0 2 1 3
1 | 1 0 2 3
7 | 7 1 0 2 (fault)
0 | 0 7 1 2
1 | 1 0 7 2
Page Faults : 8
```

head ➔ ➔ bottom

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
    // Initializing frames (based on stack)
    int size = 0;
    int* frames = (int*) malloc(sizeof(int) * frame_sz);
    for (i=0; i<frame_sz; i++) frames[i] = EMPTY_FRAME;

    // Iterating reference string
    for (i=0; i<ref_arr_sz; i++) {
        is_fault = _contains(frames, frame_sz, ref_arr[i]);
        target = is_fault;

        // Miss (page fault occurred)
        if (is_fault == -1) {

            // Shift each element to the right once
            if (size < frame_sz) size++; // if the frames has empty slot
            for (j=size-1; j>0; j--) frames[j] = frames[j-1];

            page_faults++;

        } else {
            for (j=target; j>0; j--) frames[j] = frames[j-1];
        }
        frames[0] = ref_arr[i];
    }
```

1. Assignment 3-

Stack을 이용한 LR 션

```
7 | 7 . . . (fault)
0 | 0 7 . . (fault)
1 | 1 0 7 . (fault)
2 | 2 1 0 7 (fault)
0 | 0 2 1 7
3 | 3 0 2 1 (fault)
0 | 0 3 2 1
4 | 4 0 3 2 (fault)
2 | 2 4 0 3
3 | 3 2 4 0
0 | 0 3 2 4
3 | 3 0 2 4
2 | 2 3 0 4
1 | 1 2 3 0 (fault)
2 | 2 1 3 0
0 | 0 2 1 3
1 | 1 0 2 3
7 | 7 1 0 2 (fault)
0 | 0 7 1 2
1 | 1 0 7 2
Page Faults : 8
```

head ➔ ⬅ bottom

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
52      // Initializing frames (based on stack)
53      int size = 0;
54      int* frames = (int*) malloc(sizeof(int) * frame_sz);
55      for (i=0; i<frame_sz; i++) frames[i] = EMPTY_FRAME;
56
57      // Iterating reference string
58      for (i=0; i<ref_arr_sz; i++) {
59          is_fault = _contains(frames, frame_sz, ref_arr[i]);
60          target = is_fault;
61
62          // Miss (page fault occurred)
63          if (is_fault == -1) {
64
65              // Shift each element to the right once
66              if (size < frame_sz) size++; // if the frames has empty slot
67              for (j=size-1; j>0; j--) frames[j] = frames[j-1];
68
69              page_faults++;
70
71          } else {
72              for (j=target; j>0; j--) frames[j] = frames[j-1];
73          }
74          frames[0] = ref_arr[i];
75      }
```

frame 초기화

1. Assignment 3-

Stack을 이용한 LR 션

```
7 | 7 . . . (fault)
0 | 0 7 . . (fault)
1 | 1 0 7 . (fault)
2 | 2 1 0 7 (fault)
0 | 0 2 1 7
3 | 3 0 2 1 (fault)
0 | 0 3 2 1
4 | 4 0 3 2 (fault)
2 | 2 4 0 3
3 | 3 2 4 0
0 | 0 3 2 4
3 | 3 0 2 4
2 | 2 3 0 4
1 | 1 2 3 0 (fault)
2 | 2 1 3 0
0 | 0 2 1 3
1 | 1 0 2 3
7 | 7 1 0 2 (fault)
0 | 0 7 1 2
1 | 1 0 7 2
Page Faults : 8
```

head ➡ ⬅ bottom

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

있으면 index 반환

없으면 -1

```c
52      // Initializing frames (based on stack)
53      int size = 0;
54      int* frames = (int*) malloc(sizeof(int) * frame_sz);
55      for (i=0; i<frame_sz; i++) frames[i] = EMPTY_FRAME;
56
57      // Iterating reference string
58      for (i=0; i<ref_arr_sz; i++) {
59          is_fault = _contains(frames, frame_sz, ref_arr[i]);
60          target = is_fault;
61
62          // Miss (page fault occurred)
63          if (is_fault == -1) {
64
65              // Shift each element to the right once
66              if (size < frame_sz) size++; // if the frames has empty slot
67              for (j=size-1; j>0; j--) frames[j] = frames[j-1];
68
69              page_faults++;
70
71          } else {
72              for (j=target; j>0; j--) frames[j] = frames[j-1];
73          }
74          frames[0] = ref_arr[i];
75      }
```

1. Assignment 3-

Stack을 이용한 LR 션

```
7 | 7 . . . (fault)
0 | 0 7 . . (fault)
1 | 1 0 7 . (fault)
2 | 2 1 0 7 (fault)
0 | 0 2 1 7
3 | 3 0 2 1 (fault)
0 | 0 3 2 1
4 | 4 0 3 2 (fault)
2 | 2 4 0 3
3 | 3 2 4 0
0 | 0 3 2 4
3 | 3 0 2 4
2 | 2 3 0 4
1 | 1 2 3 0 (fault)
2 | 2 1 3 0
0 | 0 2 1 3
1 | 1 0 2 3
7 | 7 1 0 2 (fault)
0 | 0 7 1 2
1 | 1 0 7 2
Page Faults : 8
```

head ➔ ➔ bottom

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
52      // Initializing frames (based on stack)
53      int size = 0;
54      int* frames = (int*) malloc(sizeof(int) * frame_sz);
55      for (i=0; i<frame_sz; i++) frames[i] = EMPTY_FRAME;
56
57      // Iterating reference string
58      for (i=0; i<ref_arr_sz; i++) {
59          is_fault = _contains(frames,
60          target = is_fault;
61
62          // Miss (page fault occurred)
63          if (is_fault == -1) {
64
65              // Shift each element to the right once
66              if (size < frame_sz) size++; // if the frames has empty slot
67              for (j=size-1; j>0; j--) frames[j] = frames[j-1];
68
69              page_faults++;
70
71          } else {
72              for (j=target; j>0; j--) frames[j] = frames[j-1];
73          }
74          frames[0] = ref_arr[i];
75      }
```

stack	[0]	[1]	[2]	[3]
page	?	.	.	.

1. Assignment 3-

Stack을 이용한 LR 션

```
7 | 7 . . . (fault)
0 | 0 7 . . (fault)
1 | 1 0 7 . (fault)
2 | 2 1 0 7 (fault)
0 | 0 2 1 7
3 | 3 0 2 1 (fault)
0 | 0 3 2 1
4 | 4 0 3 2 (fault)
2 | 2 4 0 3
3 | 3 2 4 0
0 | 0 3 2 4
3 | 3 0 2 4
2 | 2 3 0 4
1 | 1 2 3 0 (fault)
2 | 2 1 3 0
0 | 0 2 1 3
1 | 1 0 2 3
7 | 7 1 0 2 (fault)
0 | 0 7 1 2
1 | 1 0 7 2
Page Faults : 8
```

head ➔ ➔ bottom

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
52      // Initializing frames (based on stack)
53      int size = 0;
54      int* frames = (int*) malloc(sizeof(int) * frame_sz);
55      for (i=0; i<frame_sz; i++) frames[i] = EMPTY_FRAME;
56
57      // Iterating reference string
58      for (i=0; i<ref_arr_sz; i++) {
59          is_fault = _contains(frames, frame_sz, ref_arr[i]);
60          target = is_fault;
61
62          // Miss (page fault occurred)
63          if (is_fault == -1) {
64
65              // Shift each element to the right once
66              if (size < frame_sz) size++; // if the frames has empty slot
67              for (j=size-1; j>0; j--) frames[j] = frames[j-1];
68
69              page_faults++;
70
71          } else {
72              for (j=target; j>0; j--) frames[j] = frames[j-1];
73          }
74          frames[0] = ref_arr[i];
75      }
```

Section 2

“ Assignment 3-2 ”

Assignment 3-2 (7점)

Clock Algorithm 시뮬레이션

- 파일명: 학번-이름-2.c

1. **generate_ref_arr()** 함수 구현 (랜덤 reference string 생성하여 리턴)
2. **lru()** 함수 구현 (page fault 발생 횟수 리턴)

2. Assignment 3-2

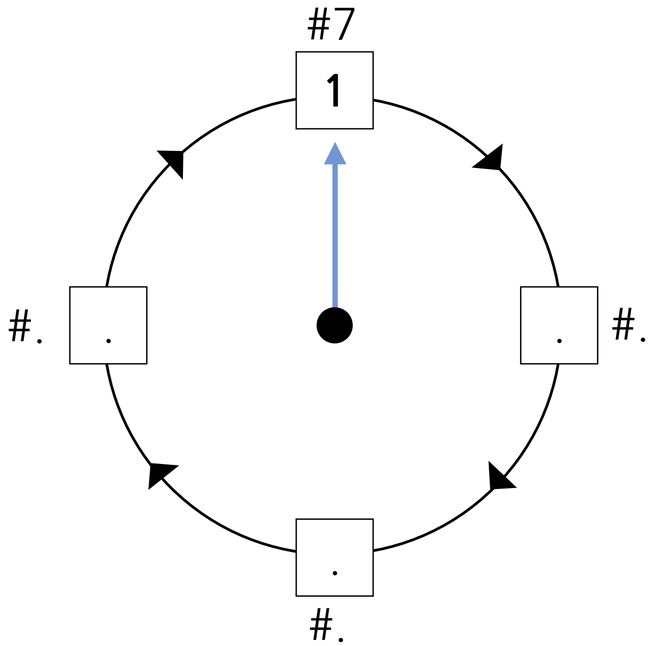
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ 빈 프레임이 존재하는 경우는 포인터가 순회하지 않음.



2. Assignment 3-2

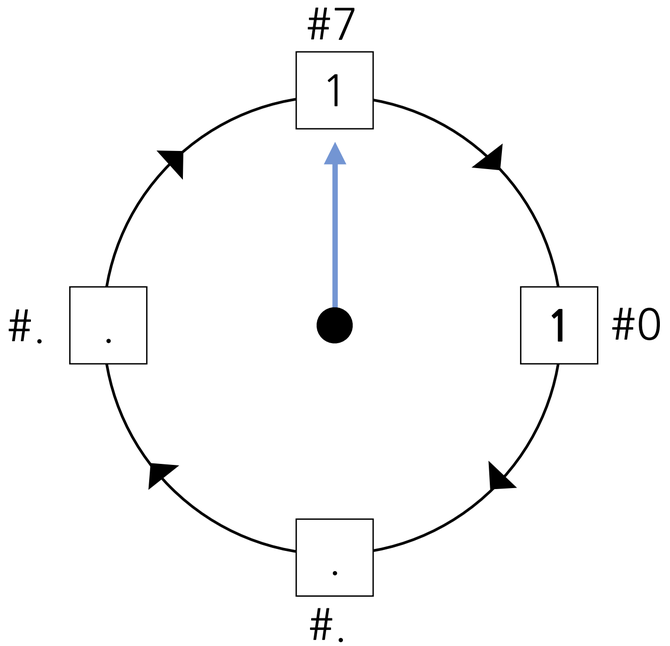
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ 빈 프레임이 존재하는 경우는 포인터가 순회하지 않음.



2. Assignment 3-2

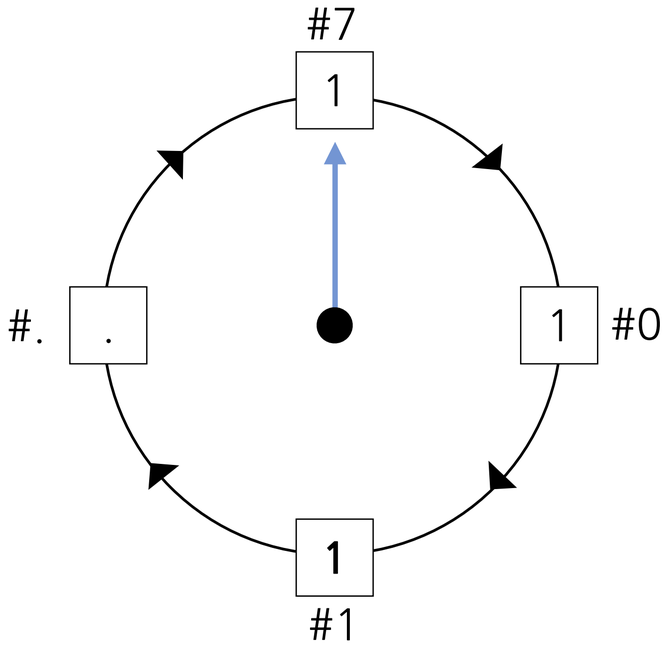
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ 빈 프레임이 존재하는 경우는 포인터가 순회하지 않음.



2. Assignment 3-2

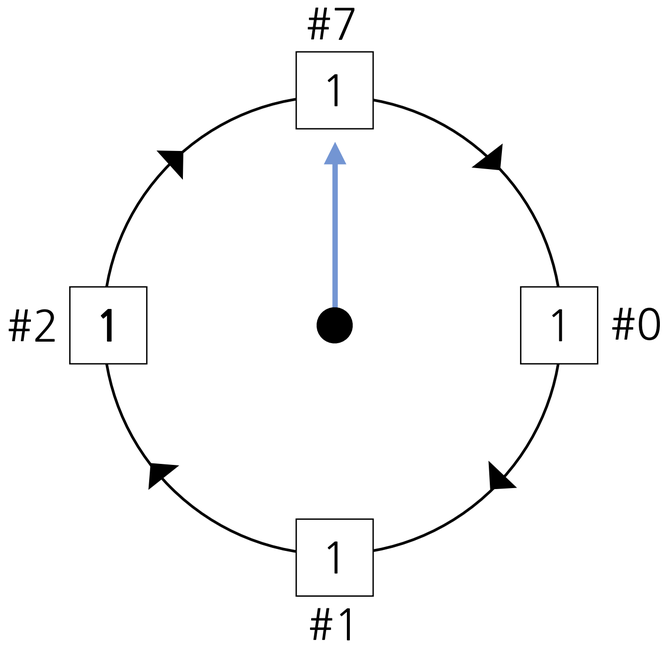
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ 빈 프레임이 존재하는 경우는 포인터가 순회하지 않음.



2. Assignment 3-2

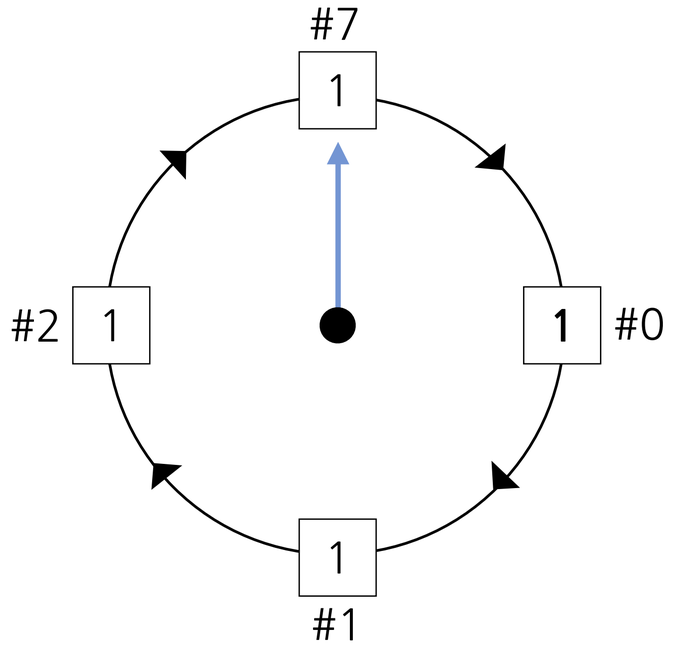
Clock Algorithm 시뮬레이션

hit

$S =$

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

→ hit인 경우 해당 페이지의 reference bit를 1로 세팅



2. Assignment 3-2

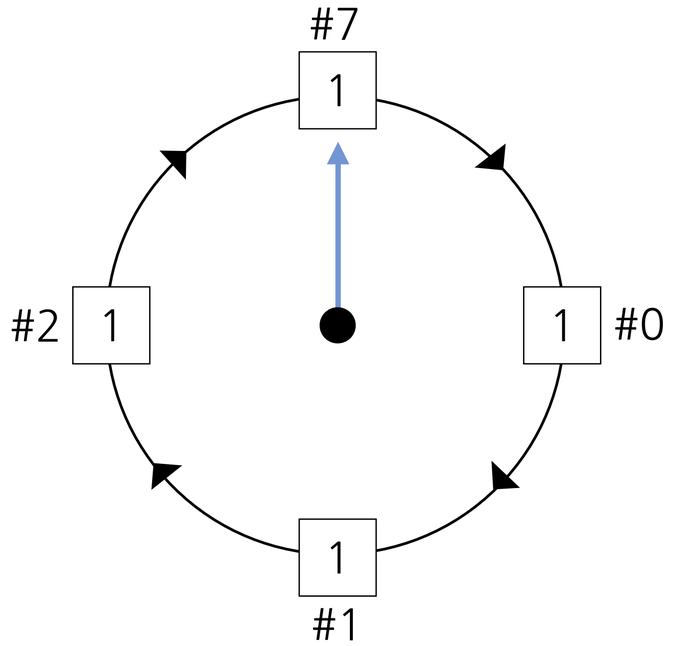
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ victim 페이지를 찾아야하므로 포인터가 큐를 순회



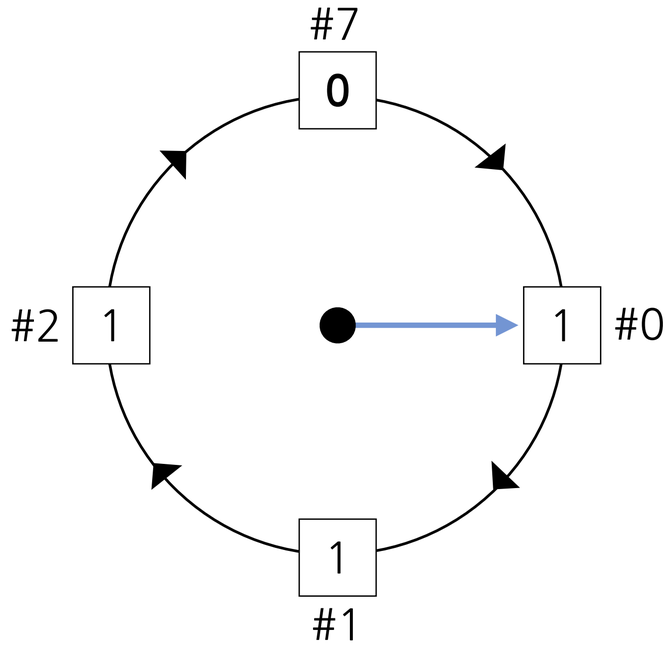
2. Assignment 3-2

Clock Algorithm 시뮬레이션

fault

$S =$

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



→ victim 페이지를 찾아야하므로 포인터가 큐를 순회,

reference bit가 1인 경우 0으로 세팅

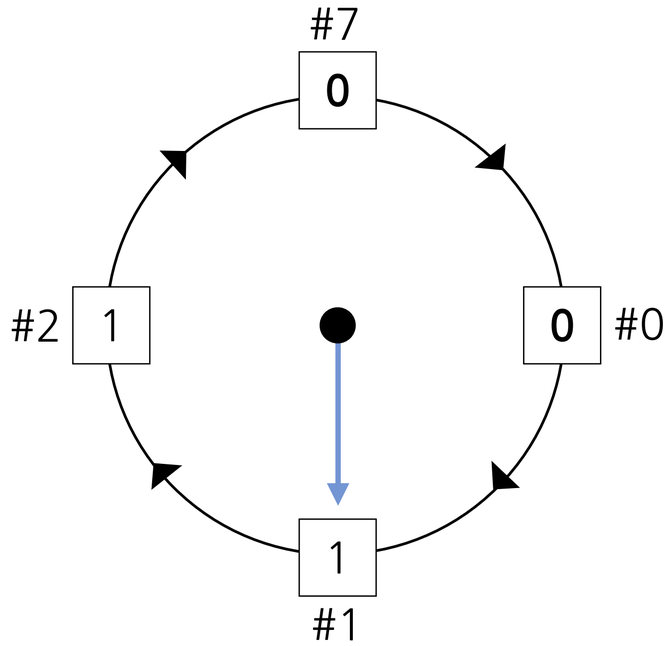
2. Assignment 3-2

Clock Algorithm 시뮬레이션

fault

$S =$

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



→ victim 페이지를 찾아야하므로 포인터가 큐를 순회,

reference bit가 1인 경우 0으로 세팅

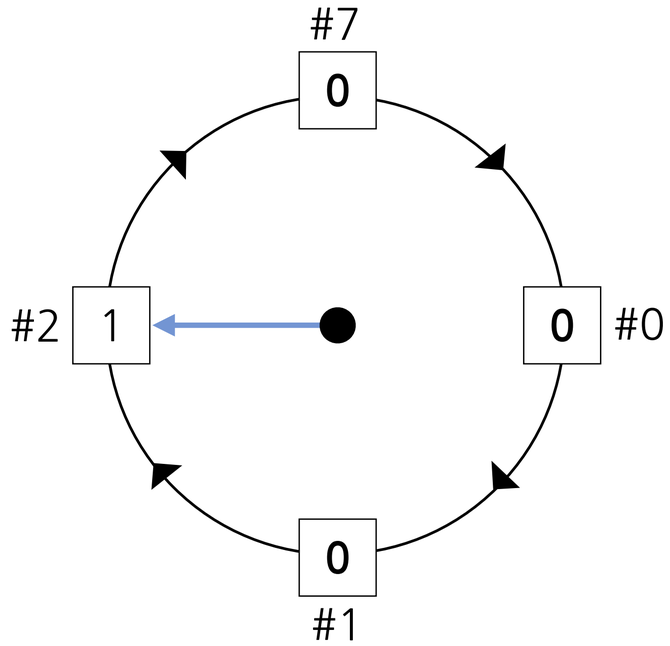
2. Assignment 3-2

Clock Algorithm 시뮬레이션

fault

$S =$

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



→ victim 페이지를 찾아야하므로 포인터가 큐를 순회,

reference bit가 1인 경우 0으로 세팅

2. Assignment 3-2

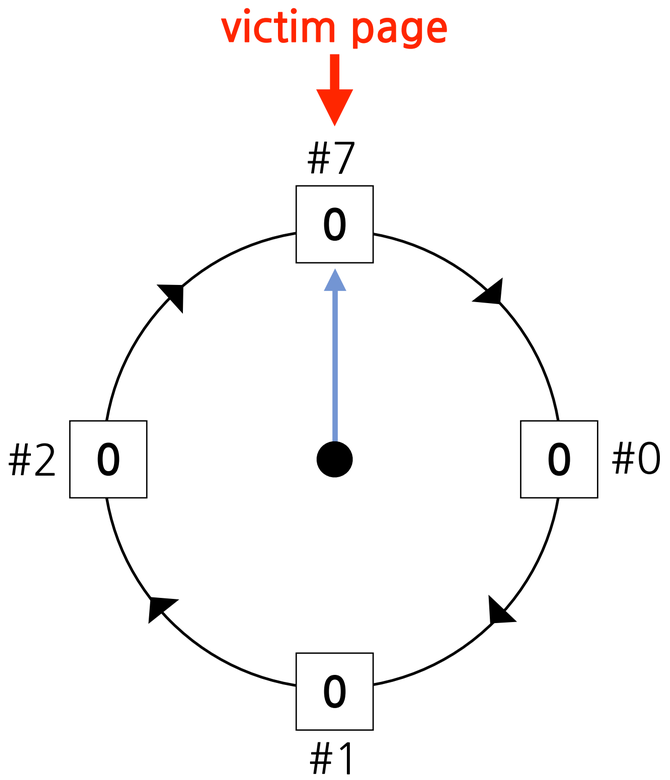
Clock Algorithm 시뮬레이션

fault

$S =$

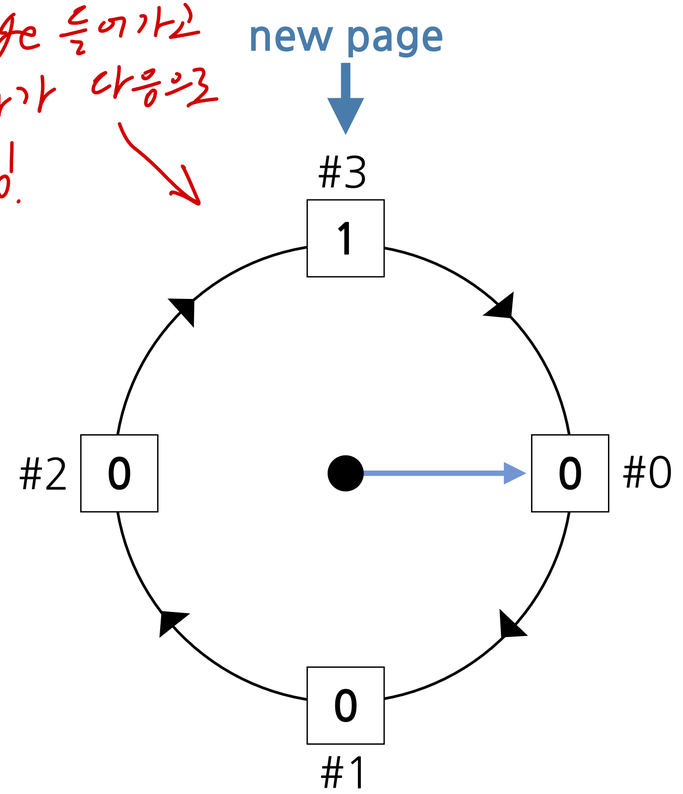
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7	7															
	0	0	0	0															
		1	1	1															
			2	2															

→ reference bit가 0인 경우 해당 페이지를 교체함



2. Assignment 3-2

Clock Algorithm 시뮬레이션



fault

$S =$

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

→ reference bit가 0인 경우 해당 페이지를 교체함

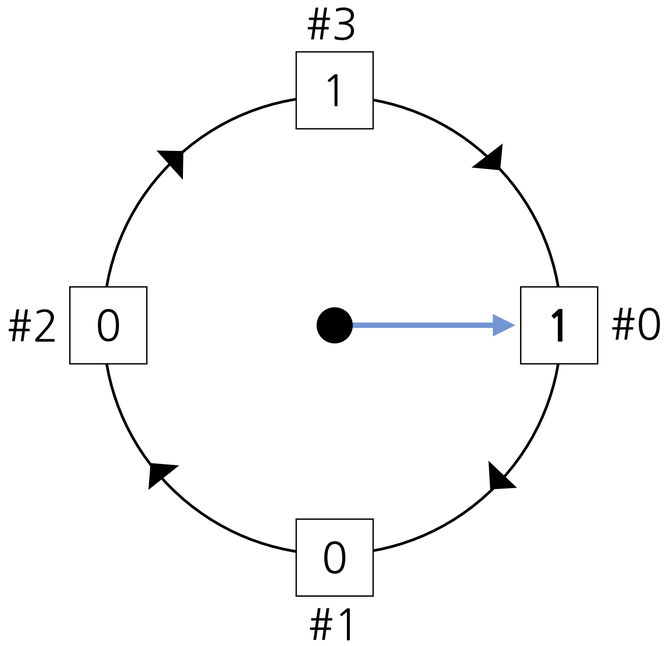
2. Assignment 3-2

Clock Algorithm 시뮬레이션

hit

$S =$

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



→ hit인 경우 해당 페이지의 reference bit를 1로 세팅

(포인터는 움직이지 않음)

2. Assignment 3-2

Clock Algorithm 시뮬레이션

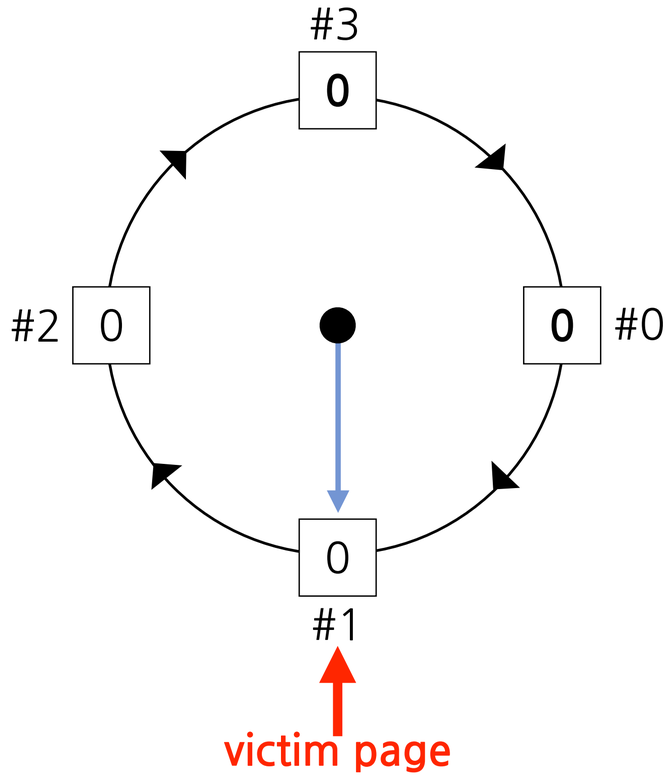
fault

$S =$

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

→ victim 페이지를 찾아야하므로 포인터가 큐를 순회,

reference bit가 1인 경우 0으로 세팅



2. Assignment 3-2

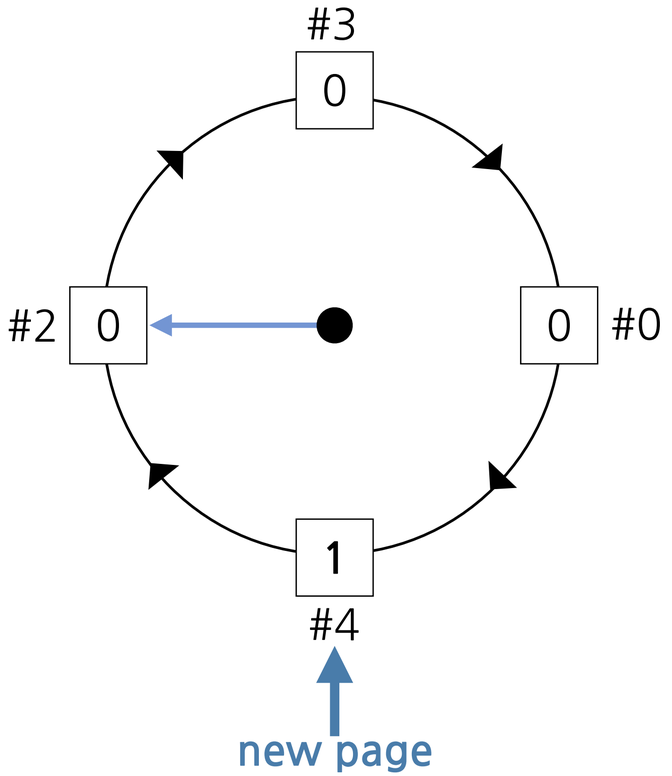
Clock Algorithm 시뮬레이션

fault

$S =$

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

→ reference bit가 0인 경우 해당 페이지를 교체함



2. Assignment 3-2

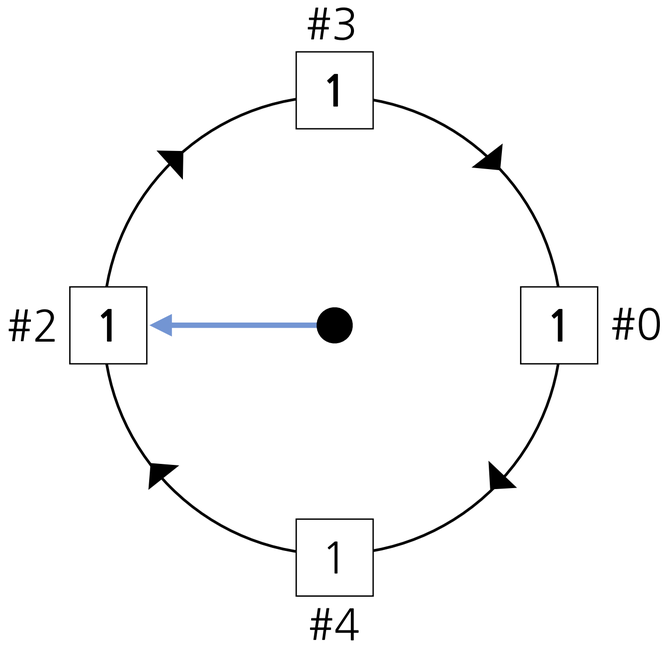
Clock Algorithm 시뮬레이션

hit hit hit hit hit

$S =$

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

→ hit인 경우 해당 페이지의 reference bit를 1로 세팅



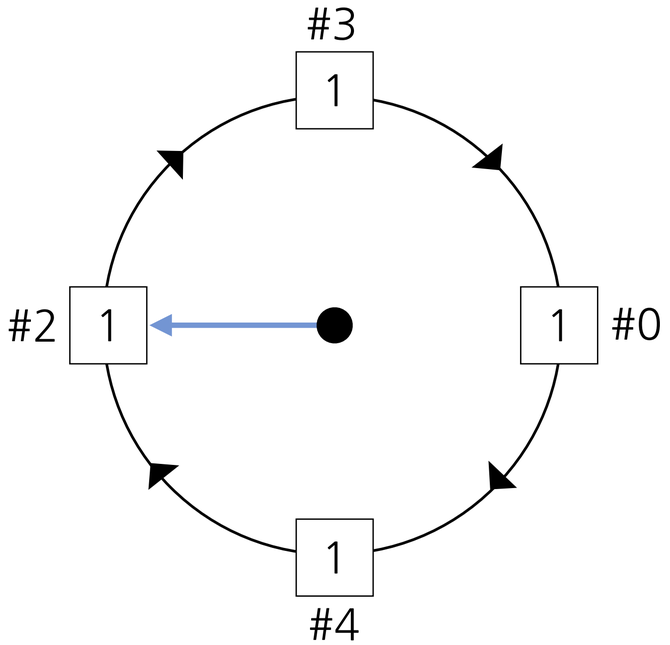
2. Assignment 3-2

Clock Algorithm 시뮬레이션

fault ★

$S =$

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



→ victim 페이지를 찾아야하므로 포인터가 큐를 순회,

reference bit가 1인 경우 0으로 세팅

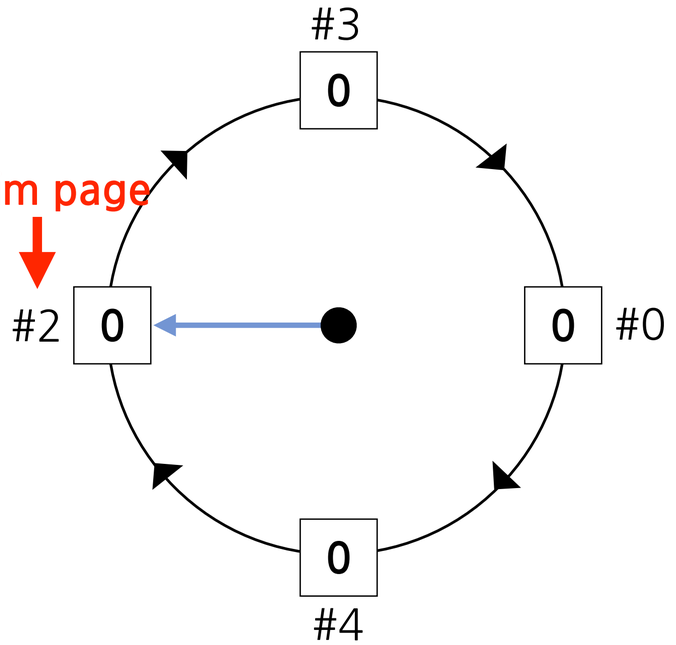
2. Assignment 3-2

Clock Algorithm 시뮬레이션

fault ★

$S =$

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



→ reference bit가 0인 경우 해당 페이지를 교체함

2. Assignment 3-2

Clock Algorithm 시뮬레이션

↓

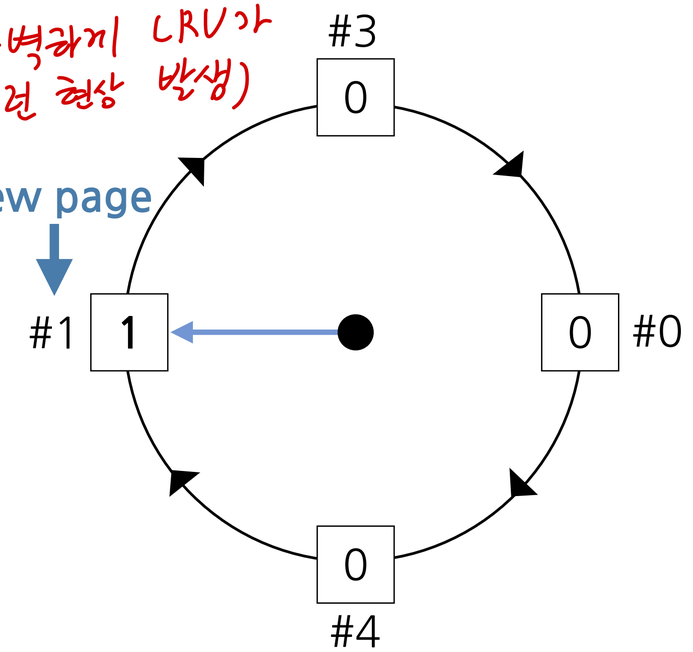
LRU "Approximation" Algorithm

교체된 2가 LRU가 아님 (clock 알고리즘이 완벽하게 LRU가 아니기 때문에 이런 현상 발생)

fault ★

$S =$

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



→ reference bit가 0인 경우 해당 페이지를 교체함

2. Assignment 3-2

Clock Algorithm 시뮬레이션

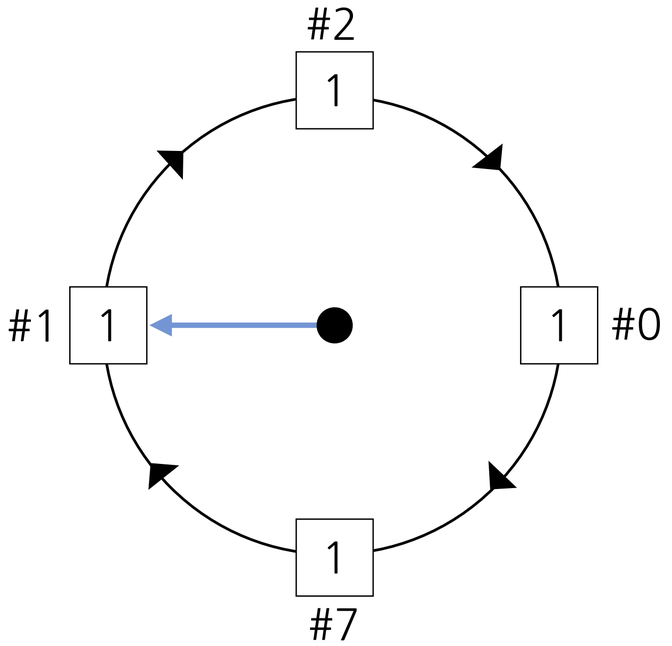
↓

LRU "Approximation" Algorithm

$S =$

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

Page Faults → 9



2. Assignment 3-2

Clock Algorithm 시

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 4 1 (fault)
2 | 2 0 4 1 (fault)
0 | 2 0 4 1
1 | 2 0 4 1
7 | 2 0 7 1 (fault)
0 | 2 0 7 1
1 | 2 0 7 1
Page Faults : 9
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
    // Initializing additional reference bits
    int ptr = 0; // clock pointer
    int* clock = (int*) malloc(sizeof(int) * frame_sz);
    for (i=0; i<frame_sz; i++) clock[i] = EMPTY_FRAME;

    // Iterating reference string
    for (i=0; i<ref_arr_sz; i++) {
        is_fault = _contains(frames, frame_sz, ref_arr[i]);

        // Miss (page fault occurred)
        if (is_fault == -1) {
            int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

            // Checking for empty frame slots
            if (empty_idx != EMPTY_FRAME) {
                target = empty_idx;
                clock[empty_idx] = 1;
            } else {
                while(clock[ptr] != 0) {
                    clock[ptr] = 0;
                    ptr = (ptr + 1) % frame_sz;
                }
                target = ptr;
            }

            // Page replacement
            frames[target] = ref_arr[i];
            clock[ptr] = 1;
            ptr = (ptr + 1) % frame_sz;
            page_faults++;

        } else {
            clock[is_fault] = 1;
        }
    }
```

→ Clock queue 선언

2. Assignment 3-2

Clock Algorithm 시

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 4 1 (fault)
2 | 2 0 4 1 (fault)
0 | 2 0 4 1
1 | 2 0 4 1
7 | 2 0 7 1 (fault)
0 | 2 0 7 1
1 | 2 0 7 1
Page Faults : 9
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
int ptr = 0; // clock pointer
int* clock = (int*) malloc(sizeof(int) * frame_sz);
for (i=0; i<frame_sz; i++) clock[i] = EMPTY_FRAME;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
            clock[empty_idx] = 1;
        } else {
            while(clock[ptr] != 0) {
                clock[ptr] = 0;
                ptr = (ptr + 1) % frame_sz;
            }
            target = ptr;
        }

        // Page replacement
        frames[target] = ref_arr[i];
        clock[ptr] = 1;
        ptr = (ptr + 1) % frame_sz;
        page_faults++;

    } else {
        clock[is_fault] = 1;
    }
}
```

→ 빈 frame이 있는 경우

2. Assignment 3-2

Clock Algorithm 시

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 4 1 (fault)
2 | 2 0 4 1 (fault)
0 | 2 0 4 1
1 | 2 0 4 1
7 | 2 0 7 1 (fault)
0 | 2 0 7 1
1 | 2 0 7 1
Page Faults : 9
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
int ptr = 0; // clock pointer
int* clock = (int*) malloc(sizeof(int) * frame_sz);
for (i=0; i<frame_sz; i++) clock[i] = EMPTY_FRAME;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
            clock[empty_idx] = 1;
        } else {
            while(clock[ptr] != 0) {
                clock[ptr] = 0;
                ptr = (ptr + 1) % frame_sz;
            }
            target = ptr;
        }

        // Page replacement
        frames[target] = ref_arr[i];
        clock[ptr] = 1;
        ptr = (ptr + 1) % frame_sz;
        page_faults++;

    } else {
        clock[is_fault] = 1;
    }
}
```

→ frame 내에 존재하는 경우

Section 3

“ Assignment 3-3 ”

Assignment 3-3 (7점)

Additional Reference Bits Algorithm 시뮬레이션

- 파일명: 학번-이름-3.c

  1. **generate_ref_arr()** 함수 구현 (랜덤 reference string 생성하여 리턴)
  2. **lru()** 함수 구현 (page fault 발생 횟수 리턴)

- 보고서에 다음 내용 추가

  - 주어진 Reference String S에 대해, 페이지 참조마다 0번 페이지의 Reference Bits이 어떻게 변하는지 표로 작성

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7			
ref bits	1000 0000			

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7	#0		
ref bits	0100 0000	1000 0000		

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7	#0	#1	
ref bits	0010 0000	0100 0000	1000 0000	

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

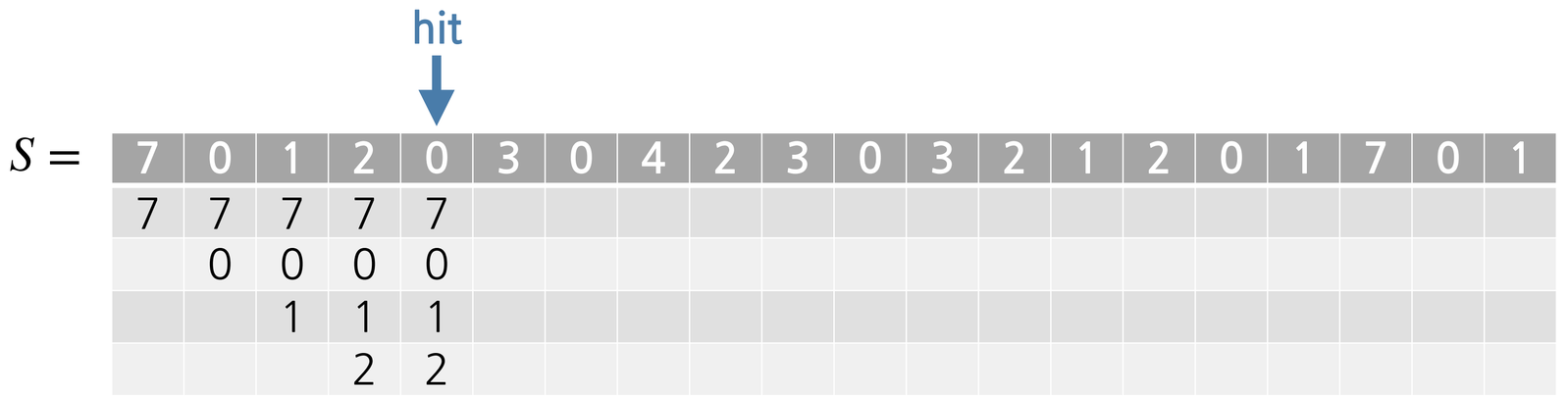
$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7	#0	#1	#2
ref bits	0001 0000	0010 0000	0100 0000	1000 0000

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션



frames	[0]	[1]	[2]	[3]
page no.	#7	#0	#1	#2
ref bits	0000 1000	1001 0000	0010 0000	0100 0000

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7	#0	#1	#2
ref bits	0000 1000	1001 0000	0010 0000	0100 0000
	8	*144*	*32*	*64*

victim page

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#3	#0	#1	#2
ref bits	1000 0000	0100 1000	0001 0000	0010 0000

new page

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

hit

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#3	#0	#1	#2
ref bits	0100 0000	1010 0100	0000 1000	0001 0000

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#3	#0	#1	#2
ref bits	0100 0000	1010 0100	0000 1000	0001 0000

victim page

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

fault

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#3	#0	#4	#2
ref bits	0010 0000	0101 0010	1000 0000	0000 1000

new page

3. Assignment 3-3

Additional Reference Bits Algorithm 시뮬레이션

$S =$

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

frames	[0]	[1]	[2]	[3]
page no.	#7	#0	#1	#2
ref bits	0100 0000	1001 0000	0010 0100	0000 1010

3. Assignment 3-2

Additional Refer

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
    // Initializing additional reference bits
    uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
    for (i=0; i<frame_sz; i++) refbits[i] = 0;

    // Iterating reference string
    for (i=0; i<ref_arr_sz; i++) {
        is_fault = _contains(frames, frame_sz, ref_arr[i]);
        target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

        // Miss (page fault occurred)
        if (is_fault == -1) {
            int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

            // Checking for empty frame slots
            if (empty_idx != EMPTY_FRAME) {
                target = empty_idx;
            } else {
                // Searching for the victim page that has the minimum refbits
                uint8_t target_refbits = 0b11111111; // max of unsigned int8
                for (j=0; j<frame_sz; j++) {
                    if (refbits[j] < target_refbits) {
                        target = j;
                        target_refbits = refbits[j];
                    }
                }
            }

            // Page Replacement
            frames[target] = ref_arr[i];
            refbits[target] = 0;
            page_faults++;
        }

        // Shifting refbits
        for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
        refbits[target] |= 0b10000000; // setting the first bit to 1
    }
```

→ frame 별로 unsigned 8 bit 를 줌

signed 면 첫번째 bit 1일 때 음수로 인식됨

3. Assignment 3-2

Additional Refer g

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
    // Initializing additional reference bits
    uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
    for (i=0; i<frame_sz; i++) refbits[i] = 0;

    // Iterating reference string
    for (i=0; i<ref_arr_sz; i++) {
        is_fault = _contains(frames, frame_sz, ref_arr[i]);
        target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

        // Miss (page fault occurred)
        if (is_fault == -1) {
            int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

            // Checking for empty frame slots
            if (empty_idx != EMPTY_FRAME) {
                target = empty_idx;
            } else {
                // Searching for the victim page that has the minimum refbits
                uint8_t target_refbits = 0b11111111; // max of unsigned int8
                for (j=0; j<frame_sz; j++) {
                    if (refbits[j] < target_refbits) {
                        target = j;
                        target_refbits = refbits[j];
                    }
                }
            }

            // Page Replacement
            frames[target] = ref_arr[i];
            refbits[target] = 0;
            page_faults++;
        }

        // Shifting refbits
        for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
        refbits[target] |= 0b10000000; // setting the first bit to 1
    }
```

3. Assignment 3-2

Additional Refer

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
for (i=0; i<frame_sz; i++) refbits[i] = 0;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);
    target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
        } else {
            // Searching for the victim page that has the minimum refbits
            uint8_t target_refbits = 0b11111111; // max of unsigned int8
            for (j=0; j<frame_sz; j++) {
                if (refbits[j] < target_refbits) {
                    target = j;
                    target_refbits = refbits[j];
                }
            }
        }

        // Page Replacement
        frames[target] = ref_arr[i];
        refbits[target] = 0;
        page_faults++;
    }

    // Shifting refbits
    for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
    refbits[target] |= 0b10000000; // setting the first bit to 1
}
```

→ 빈 frame이 있는 경우

3. Assignment 3-2

Additional Refer

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
for (i=0; i<frame_sz; i++) refbits[i] = 0;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);
    target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
        } else {
            // Searching for the victim page that has the minimum refbits
            uint8_t target_refbits = 0b11111111; // max of unsigned int8
            for (j=0; j<frame_sz; j++) {
                if (refbits[j] <= target_refbits) {
                    target = j;
                    target_refbits = refbits[j];
                }
            }
        }

        // Page Replacement
        frames[target] = ref_arr[i];
        refbits[target] = 0;
        page_faults++;
    }

    // Shifting refbits
    for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
    refbits[target] |= 0b10000000; // setting the first bit to 1
}
```

target 보다 작으면 바꿈

3. Assignment 3-2

Additional Refer

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
for (i=0; i<frame_sz; i++) refbits[i] = 0;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);
    target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
        } else {
            // Searching for the victim page that has the minimum refbits
            uint8_t target_refbits = 0b11111111; // max of unsigned int8
            for (j=0; j<frame_sz; j++) {
                if (refbits[j] <= target_refbits) {
                    target = j;
                    target_refbits = refbits[j];
                }
            }
        }

        // Page Replacement
        frames[target] = ref_arr[i];
        refbits[target] = 0;
        page_faults++;
    }

    // Shifting refbits
    for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
    refbits[target] |= 0b10000000; // setting the first bit to 1
}
```

3. Assignment 3-2

Additional Refer

```
7 | 7 . . . (fault)
0 | 7 0 . . (fault)
1 | 7 0 1 . (fault)
2 | 7 0 1 2 (fault)
0 | 7 0 1 2
3 | 3 0 1 2 (fault)
0 | 3 0 1 2
4 | 3 0 4 2 (fault)
2 | 3 0 4 2
3 | 3 0 4 2
0 | 3 0 4 2
3 | 3 0 4 2
2 | 3 0 4 2
1 | 3 0 1 2 (fault)
2 | 3 0 1 2
0 | 3 0 1 2
1 | 3 0 1 2
7 | 7 0 1 2 (fault)
0 | 7 0 1 2
1 | 7 0 1 2
Page Faults : 8
```

```c
int _contains(int* arr, size_t sz, int target) {
    int i = 0;
    for (i=0; i<sz; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

```c
// Initializing additional reference bits
uint8_t* refbits = (uint8_t*) malloc(sizeof(uint8_t) * frame_sz);
for (i=0; i<frame_sz; i++) refbits[i] = 0;

// Iterating reference string
for (i=0; i<ref_arr_sz; i++) {
    is_fault = _contains(frames, frame_sz, ref_arr[i]);
    target = is_fault; // _contains returns the index of ref_arr[i] if it exists in the frames

    // Miss (page fault occurred)
    if (is_fault == -1) {
        int empty_idx = _contains(frames, frame_sz, EMPTY_FRAME);

        // Checking for empty frame slots
        if (empty_idx != EMPTY_FRAME) {
            target = empty_idx;
        } else {
            // Searching for the victim page that has the minimum refbits
            uint8_t target_refbits = 0b11111111; // max of unsigned int8
            for (j=0; j
                if (refb
                    targ
                    targ
                }
            }
        }

        // Page Replacement
        frames[target] = ref_arr[i];
        refbits[target] = 0;
        page_faults++;
    }

    // Shifting refbits
    for (j=0; j<frame_sz; j++) refbits[j] >>= 1;
    refbits[target] |= 0b10000000; // setting the first bit to 1
}
```

실제로는
reference될 때마다가 아닌
일정 시간 마다

*만약 refbits[target]이
" 0010 0010 " 이라면,

```
  0010 0010
or) 1000 0000
-------------
  1010 0010
```

더하는 느낌으로

예시 답안 코드)

- https://github.com/ku-cloud/22spring-os-lab03 리포지토리 assignment 폴더에 모두 업로드 되어 있습니다.

End of Document