운영체제(SCSC) Project-2

Virtual Memory Management ¶

팀 명단: 한대룡(2014311688), 류지원(2015312436)

과제 목표

- Demand paging system을 가정하고 이를 위한 replacement 기법 구현

과제 내용

- 주어진 page reference string을 입력 받아 아래의 각 replacement 기법으로 처리했을 경우의 memory residence set 변화 과정 및 page fault 발생 과정 추적/출력
- MIN, FIFO, LRU, LFU, WS Memory Management
- Page frame 할당량 및 window size 등은 입력으로 결정하며, 초기 할당된 page frame들은 모두 비어 있는 것으로 가정
- 각 기법의 실행 결과에 대한 출력 방법은 팀 별로 design하여 진행
- 기타 필요한 가정은 각 팀 별로 설정하고 보고서에 명시

Abstract

- 본 과제는 MS 플랫폼에서 Python을 이용하여 진행
- MIN, FIFO, LRU, LFU, WS Memory Management를 각각 function으로 define하여 진행
- 각각의 함수 모두 N, M, W, K page_string을 input으로 받되 WS 기법만 W를 사용

Import Package & Define Function

- 필요한 package를 import하고 text파일을 읽는 함수, 뒤의 memory management 함수를 구현하는데 필요한 함수를 추가적으로 정의

In [12]:
```
import numpy as np
```

In [13]:
```
def read_input(input_file):
    with open(input_file) as f:
        index = 0
        for line in f:
            array = line.split()
            if(index == 0):
                N = int(array[0])
                M = int(array[1])
                W = int(array[2])
                K = int(array[3])
                index = index + 1

            else:
                page_string = array
                page_string = [int (i) for i in page_string]

    return (N, M, W, K, page_string)
```

In [16]:
```
N, M, W, K, page_string = read_input("input1.txt")
print(N, M, W, K, page_string)
```

```
6 4 3 14 [1, 2, 6, 1, 4, 5, 1, 2, 1, 4, 5, 6, 4, 5]
```

In [49]:
```
class Queue():
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()


    ## LRU의 경우 이미 메모리에 참조된 페이지가 후에 다시 들어올 경우 queue에서 위치
를 바꿔줘야 하므로 이를 제거하는 remove를 추가로 설정
    def remove(self, item):
        self.items.remove(item)
```

In [56]:
```
## LFU는 각 page 마다 참조 횟수가 필요하므로 이를 class로 정의하여 추적
class LFU:
    def __init__(self, id):
        self.id = id
        self.reference_times = 0

    def increase(self):
        self.reference_times += 1
```

In [57]:
```
## WS는 각 page마다 memory 상태 정보가 필요하고 memory에 존재하는 경우와 memory에 존재하지 않는 경우를 알 수 있도록 class로 정의
class WS:
    def __init__(self, id):
        self.id = id
        self.state = "-"

    def memory_on(self):
        self.state = str(self.id)

    def memory_off(self):
        self.state = "-"
```

MIN

- 새로 들어오는 페이지가 기존의 메모리를 할당받고 있으면 그대로 진행한다.
- memory에 자리가 있는 경우 memory를 할당하고 자리가 없는 경우는 남은 reference string을 확인하여 현재 memory에 존재하는 페이지 중 가장 나중에 필요한 페이지를 빼고 새로운 페이지에 메모리를 할당한다.

In [48]:
```
def MIN_replacement(N, M, W, K, page_string):
    print("MIN")
    print("메모리 상태 변화 과정")

    memory_state = []
    page_fault = 0

    for i in range(0, K):
        exist = False

        for j in range(0,len(memory_state)):
            if(page_string[i] == memory_state[j]):
                exist = True

        if(not exist):
            if(len(memory_state) < M):
                memory_state.append(page_string[i])
                page_fault = page_fault + 1

            else:
                res = page_string[i:]

                try:
                    maximum = res.index(memory_state[0])
                    try:
                        for k in range(0, len(memory_state)):
                            maximum = max(maximum, res.index(memory_state[k]))
                        delete = memory_state.index(res[maximum])
                    except:
                        delete = memory_state.index(memory_state[k])
                except ValueError:
                    delete = memory_state.index(memory_state[0])


                memory_state[delete] = page_string[i]
                page_fault = page_fault + 1

        print(memory_state)

    print("총 page fault 횟수: ",page_fault )
```

FIFO

- 새로 들어오는 페이지가 기존의 메모리를 할당받고 있으면 그대로 진행한다.
- memory에 자리가 있는 경우 memory를 할당하고 자리가 없는 경우는 남은 reference string을 확인하여 현재 memory에 존재하는 페이지 중 가장 먼저 메모리를 할당받은 페이지를 빼고, 새로운 페이지에 메모리를 할당한다.

In [50]:
```
def FIFO_replacement(N, M, W, K, page_string):
    print("FIFO")
    print("메모리 상태 변화 과정")

    memory_state = []
    queue = Queue()
    page_fault = 0

    for i in range(0, K):
        exist = False

        for j in range(0,len(memory_state)):
            if(page_string[i] == memory_state[j]):
                exist = True

        if(not exist):
            if(len(memory_state) < M):
                memory_state.append(page_string[i])
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1

            else:
                ## 가장 먼저 들어온 페이지를 제거
                delete = memory_state.index(queue.dequeue())

                memory_state[delete] = page_string[i]
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1

        print(memory_state)

    print("총 page fault 횟수: ",page_fault )
```

LRU

- 새로 들어오는 페이지가 기존의 메모리를 할당받고 있으면 그대로 진행한다.
- memory에 자리가 있는 경우 memory를 할당하고 자리가 없는 경우는 남은 reference string을 확인하여 현재 memory에 존재하는 페이지 중 최근에 참조된 시간이 가장 오래된 페이지의 메모리를 빼서 새로운 페이지에 할당한다.
- 이미 참조된 페이지가 다시 참조될 경우 queue에서 순서를 변경하여 원래의 위치에서 값을 삭제하고 맨 뒤로 이동

In [29]:
```
def LRU_replacement(N, M, W, K, page_string):
    print("LRU")
    print("메모리 상태 변화 과정")

    memory_state = []
    queue = Queue()
    page_fault = 0

    for i in range(0, K):
        exist = False

        for j in range(0,len(memory_state)):
            if(page_string[i] == memory_state[j]):
                exist = True

        if(not exist):
            if(len(memory_state) < M):
                memory_state.append(page_string[i])
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1

            else:
                delete = memory_state.index(queue.dequeue())

                memory_state[delete] = page_string[i]
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1
        else:
            queue.remove(page_string[i])
            queue.enqueue(page_string[i])

        print(memory_state)

    print("총 page fault 횟수: ",page_fault )
```

LFU

- 새로 들어오는 페이지가 기존의 메모리를 할당받고 있으면 그대로 진행한다.
- memory에 자리가 있는 경우 memory를 할당하고 자리가 없는 경우는 남은 reference string을 확인하여 현재 memory에 존재하는 페이지 중 참조된 횟수가 가장 적은 페이지를 빼서 새로운 페이지에 메모리를 할당한다.
- 페이지마다 reference_time을 설정하여 참조될 때마다 count를 1씩 증가

In [35]:
```
def LFU_replacement(N, M, W, K, page_string):
    print("LFU")
    print("메모리 상태 변화 과정")

    memory_state = []
    page_fault = 0
    queue = Queue()

    result = []

    for i in range(0, N):
        result.append(LFU(i))

    for i in range(0, K):
        result[page_string[i]-1].increase()
        exist = False


        for j in range(0,len(memory_state)):
            if(page_string[i] == memory_state[j]):
                exist = True

        if(not exist):
            if(len(memory_state) < M):
                memory_state.append(page_string[i])
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1

            else:
                min_value = result[memory_state[0]-1].reference_times
                pos = 0
                when = queue.items.index(memory_state[0])
                for re in range(0, len(memory_state)):
                    if(result[memory_state[re]-1].reference_times < min_value):
                        min_value = result[memory_state[re]-1].reference_times
                        pos = re
                        when = queue.items.index(memory_state[re])
                    elif(result[memory_state[re]-1].reference_times == min_value):
                        if(queue.items.index(memory_state[re]) > when):
                            min_value = result[memory_state[re]-1].reference_times
                            pos = re
                            when = queue.items.index(memory_state[re])


                memory_state[pos] = page_string[i]
                queue.enqueue(page_string[i])
                page_fault = page_fault + 1

        else:
            queue.remove(page_string[i])
            queue.enqueue(page_string[i])

        print(memory_state)

    print("총 page fault 횟수: ",page_fault )
```

WS

- 해당 page가 memory에 들어오면 그 페이지의 memory state를 on 시킴
- 해당 각 time마다 window size를 바탕으로 그 이전에 memory에 존재했던 페이지의 state를 "-"으로 변경
- 초기 page frame은 비어 있는 것으로 가정하였으므로 time 3까지 page가 들어오는 것 역시 page_fault로 설정

In [67]:
```
def WS_replacement(N, M, W, K, page_string):
    print("WS")
    print("메모리 상태 변화 과정")
    result = []
    page_fault = 0
    num_of_allocated_frame = []

    for i in range(1, N+1):
        result.append(WS(i))

    for i in range(0, K):
        if(i < W):
            result[page_string[i]-1].memory_on()
            page_fault = page_fault + 1

        else:
            if(result[page_string[i]-1].state == "-"):
                result[page_string[i]-1].memory_on()
                page_fault = page_fault + 1

            page = []
            for k in range(i-W, i+1):
                page.append(page_string[k])

            if (not page_string[i-W-1] in page):
                result[page_string[i-W-1]-1].memory_off()


        memory_state = []
        for y in range(0, N):
            memory_state.append(result[y].state)

        num_of_allocated_frame.append(sum(np.array(memory_state) != "-"))
        print(memory_state)


    print("총 page fault 횟수: ",page_fault)
    print("average allocated page frame: ", sum(num_of_allocated_frame)/len(num_of_a
llocated_frame))
```

Simulate1 - 교안에 있는 예시

In [60]:
```
N, M, W, K, page_string = read_input("input1.txt")
print(N, M, W, K, page_string)
```

```
6 4 3 14 [1, 2, 6, 1, 4, 5, 1, 2, 1, 4, 5, 6, 4, 5]
```

In [30]:
```
N, M, W, K, page_string = read_input("input1.txt")
MIN_replacement(N, M, W, K, page_string)
```

```
MIN
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 6]
[1, 2, 6]
[1, 2, 6, 4]
[1, 2, 5, 4]
[1, 2, 5, 4]
[1, 2, 5, 4]
[1, 2, 5, 4]
[1, 2, 5, 4]
[1, 2, 5, 4]
[6, 2, 5, 4]
[6, 2, 5, 4]
[6, 2, 5, 4]
총 page fault 횟수:  6
```

In [31]:
```
N, M, W, K, page_string = read_input("input1.txt")
FIFO_replacement(N, M, W, K, page_string)
```

```
FIFO
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 6]
[1, 2, 6]
[1, 2, 6, 4]
[5, 2, 6, 4]
[5, 1, 6, 4]
[5, 1, 2, 4]
[5, 1, 2, 4]
[5, 1, 2, 4]
[5, 1, 2, 4]
[5, 1, 2, 6]
[4, 1, 2, 6]
[4, 5, 2, 6]
총 page fault 횟수:  10
```

In [32]:
```
N, M, W, K, page_string = read_input("input1.txt")
LRU_replacement(N, M, W, K, page_string)
```

```
LRU
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 6]
[1, 2, 6]
[1, 2, 6, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
총 page fault 횟수:  7
```

In [36]:
```
N, M, W, K, page_string = read_input("input1.txt")
LFU_replacement(N, M, W, K, page_string)
```

```
LFU
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 6]
[1, 2, 6]
[1, 2, 6, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 2, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
[1, 5, 6, 4]
총 page fault 횟수:  7
```

In [62]:
```
N, M, W, K, page_string = read_input("input2.txt")
print(N, M, W, K, page_string)
```

```
5 3 3 13 [5, 4, 1, 3, 3, 4, 2, 3, 5, 3, 5, 1, 4]
```

In [68]:
```
N, M, W, K, page_string = read_input("input2.txt")
WS_replacement(N, M, W, K, page_string)
```

```
WS
메모리 상태 변화 과정
['-', '-', '-', '-', '5']
['-', '-', '-', '4', '5']
['1', '-', '-', '4', '5']
['1', '-', '3', '4', '5']
['1', '-', '3', '4', '-']
['1', '-', '3', '4', '-']
['-', '2', '3', '4', '-']
['-', '2', '3', '4', '-']
['-', '2', '3', '4', '5']
['-', '2', '3', '-', '5']
['-', '-', '3', '-', '5']
['1', '-', '3', '-', '5']
['1', '-', '3', '4', '5']
총 page fault 횟수:  8
average allocated page frame:  2.923076923076923
```

Simulate2 - HW pdf에 존재하는 예시

In [69]:
```
N, M, W, K, page_string = read_input("input3.txt")
print(N, M, W, K, page_string)
```

```
5 3 3 15 [1, 2, 3, 2, 3, 4, 5, 4, 1, 3, 4, 2, 3, 4, 5]
```

In [70]:
```
N, M, W, K, page_string = read_input("input3.txt")
MIN_replacement(N, M, W, K, page_string)
FIFO_replacement(N, M, W, K, page_string)
LRU_replacement(N, M, W, K, page_string)
LFU_replacement(N, M, W, K, page_string)
WS_replacement(N, M, W, K, page_string)
```

```
MIN
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
[1, 4, 3]
[1, 4, 5]
[1, 4, 5]
[1, 4, 5]
[3, 4, 5]
[3, 4, 5]
[3, 4, 2]
[3, 4, 2]
[3, 4, 2]
[5, 4, 2]
총 page fault 횟수:  8
FIFO
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
[4, 2, 3]
[4, 5, 3]
[4, 5, 3]
[4, 5, 1]
[3, 5, 1]
[3, 4, 1]
[3, 4, 2]
[3, 4, 2]
[3, 4, 2]
[5, 4, 2]
총 page fault 횟수:  10
LRU
메모리 상태 변화 과정
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
[4, 2, 3]
[4, 5, 3]
[4, 5, 3]
[4, 5, 1]
[4, 3, 1]
[4, 3, 1]
[4, 3, 2]
[4, 3, 2]
[4, 3, 2]
[4, 3, 5]
총 page fault 횟수:  9
LFU
메모리 상태 변화 과정
[1]
```

```
[1, 2]
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
[4, 2, 3]
[5, 2, 3]
[4, 2, 3]
[4, 1, 3]
[4, 1, 3]
[4, 1, 3]
[4, 2, 3]
[4, 2, 3]
[4, 2, 3]
[4, 5, 3]
총 page fault 횟수:  9
WS
메모리 상태 변화 과정
['1', '-', '-', '-', '-']
['1', '2', '-', '-', '-']
['1', '2', '3', '-', '-']
['1', '2', '3', '-', '-']
['-', '2', '3', '-', '-']
['-', '2', '3', '4', '-']
['-', '2', '3', '4', '5']
['-', '-', '3', '4', '5']
['1', '-', '-', '4', '5']
['1', '-', '3', '4', '5']
['1', '-', '3', '4', '-']
['1', '2', '3', '4', '-']
['-', '2', '3', '4', '-']
['-', '2', '3', '4', '-']
['-', '2', '3', '4', '5']
총 page fault 횟수:  9
average allocated page frame:  3.0
```