OS 과제 3

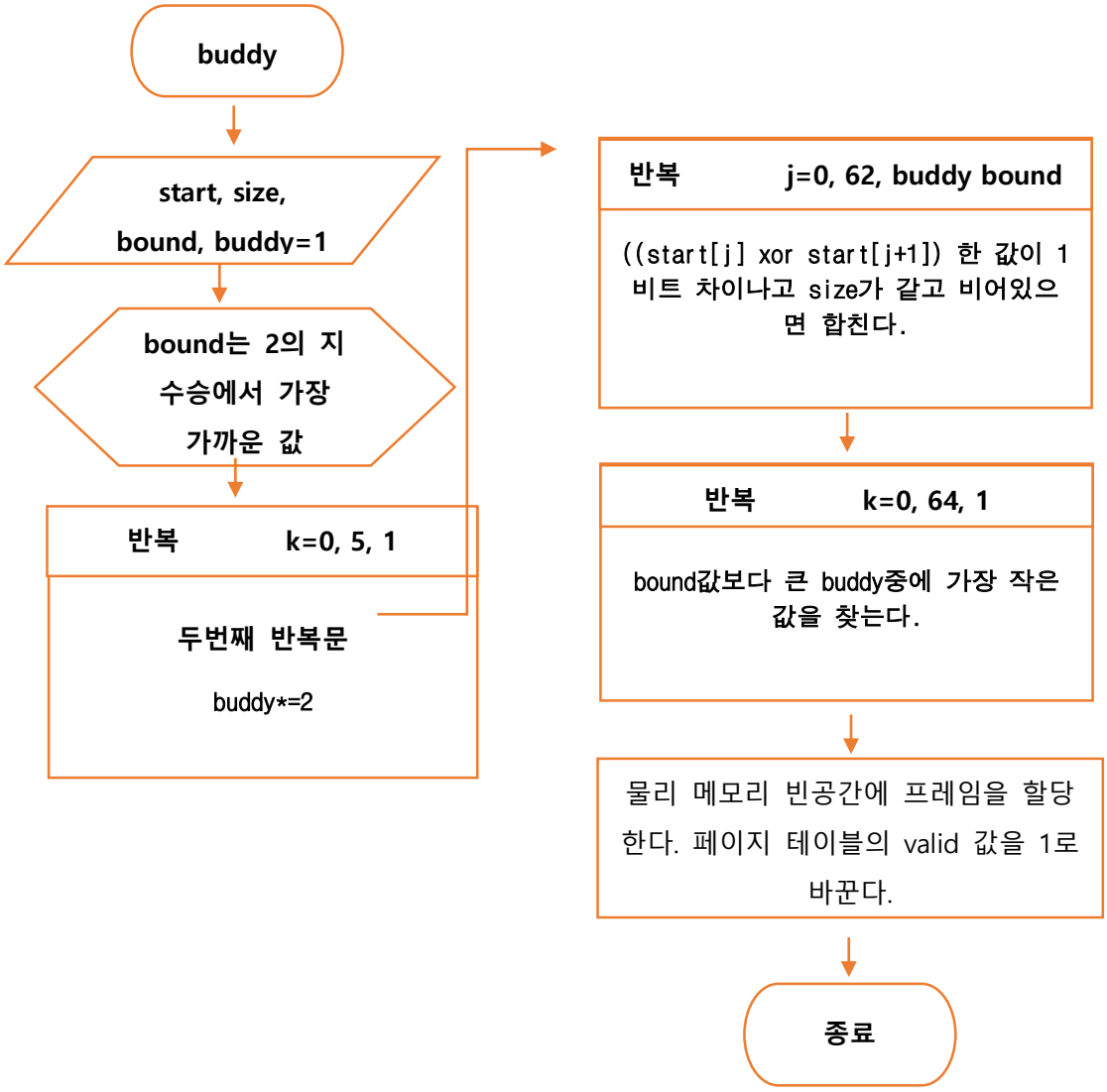
2015147574 백진우

1. 프로그램밍 수행 결과 보고서

1) 작성한 프로그램의 동작 과정과 구현 방법

Allocation 일때는 page_table에 페이지를 할당한후 valid bit을 0으로 초기화 한다. Access 일때는 버디시스템으로 물리메모리에 페이지를 할당하고 page table의 valid값을 1로 바꾼다. 만약 물리 메모리에 공간이 부족하면 5가지의 Page replacement 정책에 따라 페이지 교체가 일어난다.

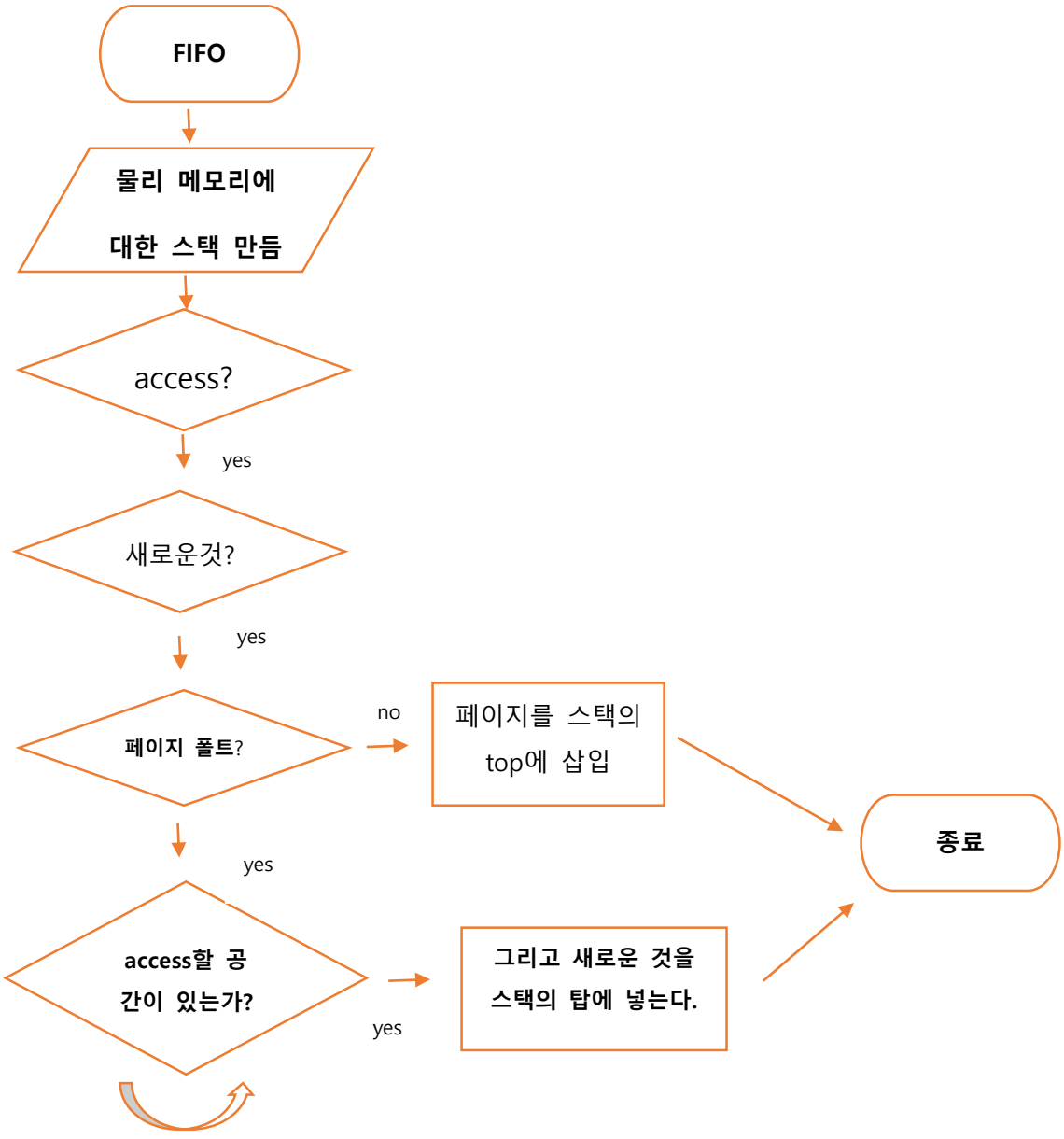
a) buddy



버디가 서로 비어져있으면 합치는 과정을 반복하여 넣을 수 있는 공간중에 가장 작은 것에다가 frame을 할당한다. 할당하면 그에 해당하는 페이지 테이블의 valid 값을 1로 바꾼다.

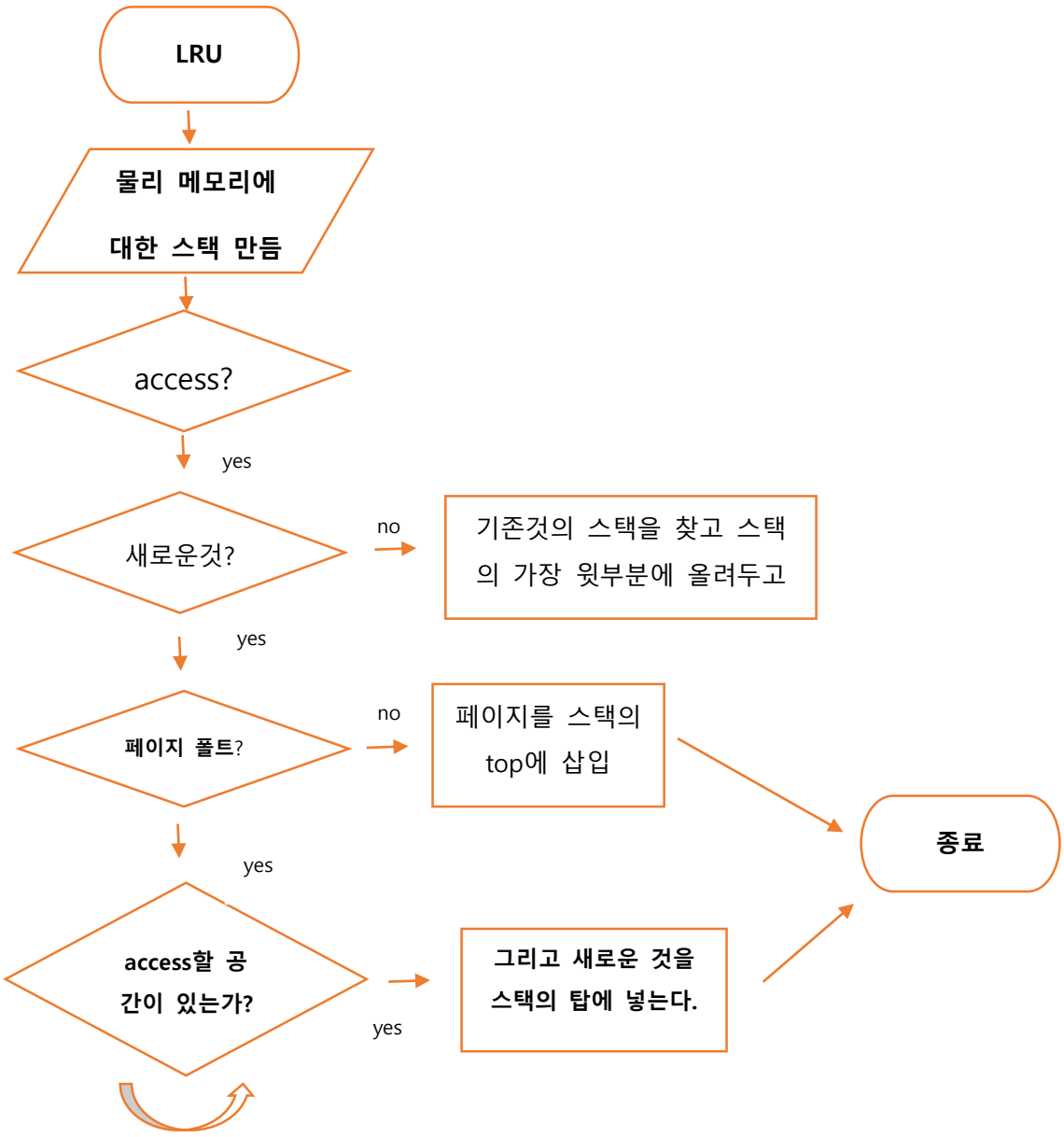
앞으로 설명할 5개의 알고리즘 전부 다 page replacement할것이 여러개 있으면 alloc id가 가장 작은것을 뺀다.

b) FIFO



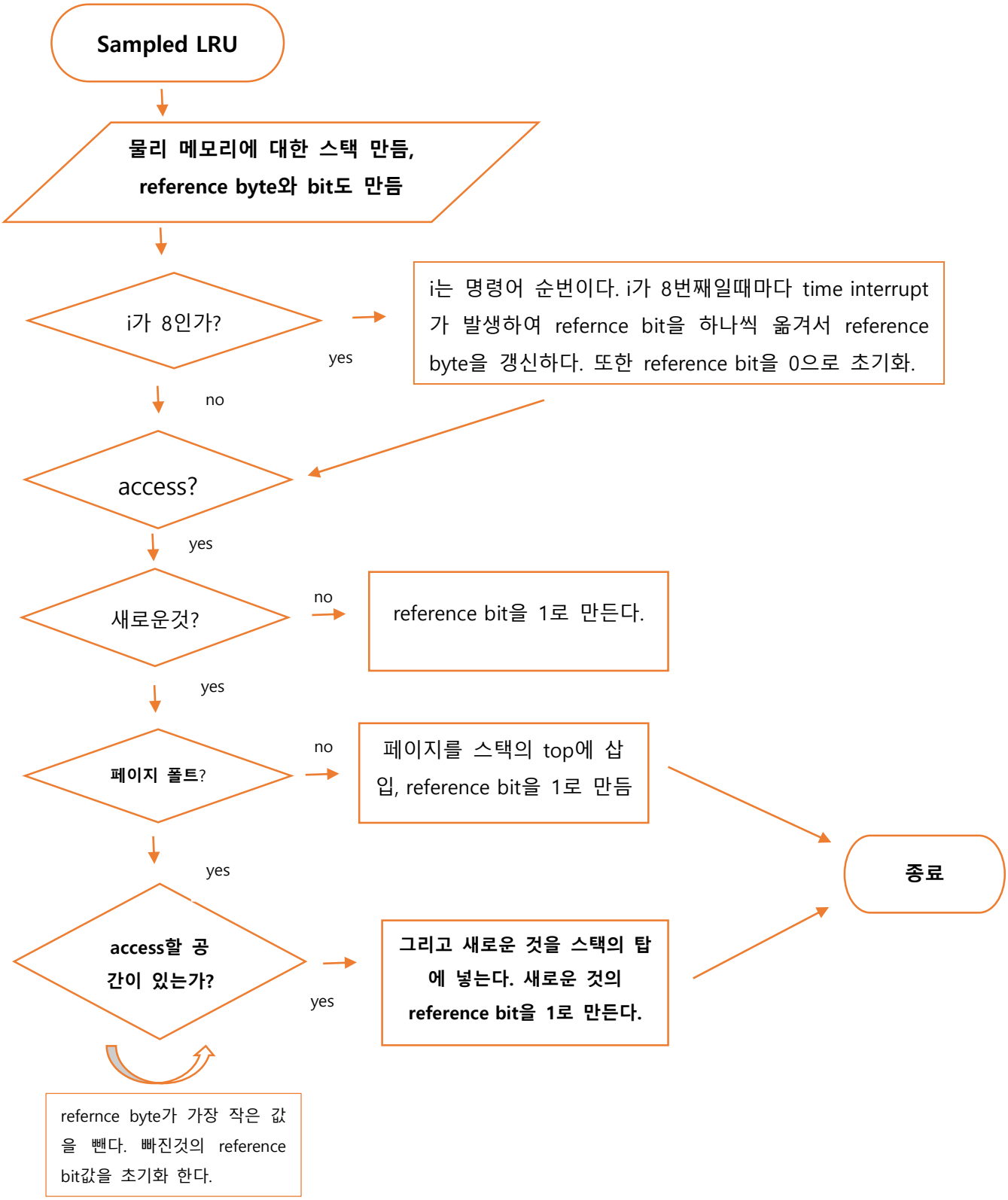
Fifo는 기본적으로 가장 먼저 들어온 것이 가장 먼저 나가게 짜는 것이다. 여기서는 스택으로 구현하여 스택의 가장 밑부분을 빼고 가장 윗부분에 새로운 것을 추가하였다.

c) LRU



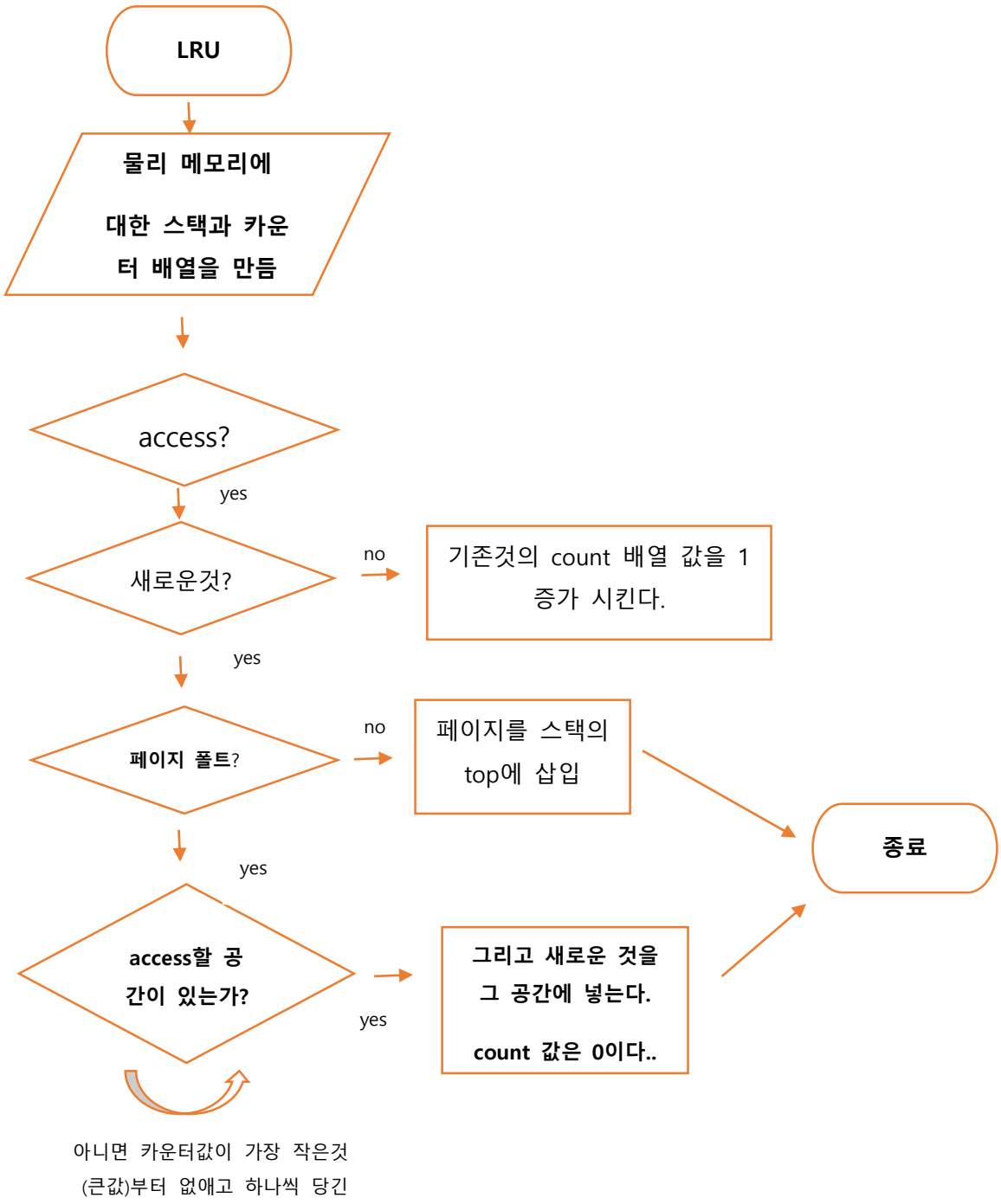
LRU는 least recently used로서 가장 오랫동안 참조되지 않은 페이지를 교체하는 것이다. 만약 기존의 있는 것을 access하면 그것을 스택의 탑에 올리고 스택을 하나씩 당긴다.

d) Sampled LRU



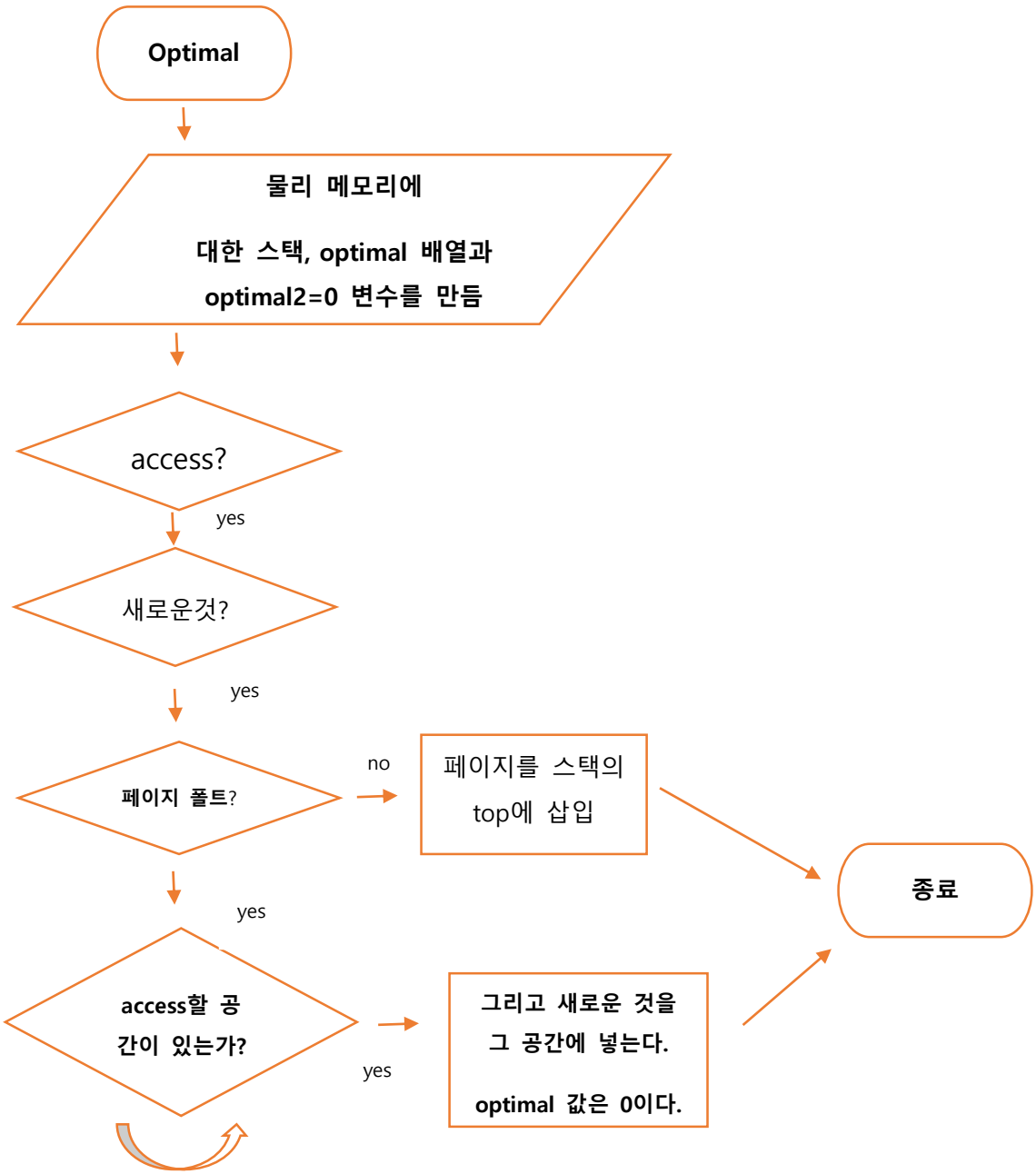
Sample LRU는 access하면 reference bit을 1로 release하면 0으로 초기화 시켜서 한다. 그리고 8번째 명령어마다 타임인터럽트가 발생하는데 그럴때마다 reference byte을 bit을 shift right해서 갱신하고 refernce bit을 0으로 초기화한다. page fault시 refernce byte가 작은 것을 내보낸다.

c) LFU(MFU)



LFU(MFU)에서는 기존의 것을 access 하면 count 값을 1 증가시켜서 count 값이 가장 작은 것(LFU), 가장 큰것(MFU)을 빼는 것이다. count 값은 물리메모리에 연동되는 배열을 만들어서 구현하였다.

c) Optimal



Optimal은 가장 이상적인 알고리즘이며 원래는 구현이 불가능하나 우리는 명령어를 받을 것을 미리 다 알고 있어서 가능하다. 페이지 폴트시 미래를 보아서 가장 나중에 사용할 것을 뺀다.

2) 개발 환경 명시

a) uname –a

```
baek@ubuntu:~$ uname -a
Linux ubuntu 4.19.27-2015147574 #1 SMP Thu Mar 14 11:01:30 PDT 2019 x86_64 x86_6
4 x86_64 GNU/Linux
```

b) CPU, 메모리 정보, 사용한 컴파일러 버전

```
baek@ubuntu:~$ grep -c processor /proc/cpuinfo
4
```

cpu 코어 전체 개수: 4

```
baek@ubuntu:~$ cat /etc/issue
Ubuntu 18.04.2 LTS \n \l
```

OS : Ubuntu 18.04.2 LTS

```
baek@ubuntu:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 78
model name      : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
stepping        : 3
microcode       : 0xba
cpu MHz         : 2592.000
cache size      : 4096 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pg
e mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonst
op_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic m
ovbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor l
ahf_lm abm 3dnowprefetch cpuid_fault invpcid_single pti fsgsbase tsc_
adjust bmi1 avx2 smep bmi2 invpcid mpx rdseed adx smap clflushopt xsa
veopt xsavec xsaves arat
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypas
s l1tf
bogomips        : 5184.00
clflush size    : 64
cache_alignment : 64
address sizes   : 43 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 78
model name      : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
stepping        : 3
microcode       : 0xba
cpu MHz         : 2592.000
cache size      : 4096 KB
physical id     : 2
siblings        : 1
core id         : 0
```

```
baek@ubuntu:~$ cat /proc/meminfo
MemTotal:        4012248 kB
MemFree:         1615964 kB
MemAvailable:    2336496 kB
Buffers:          164640 kB
Cached:           715896 kB
SwapCached:            0 kB
Active:          1331724 kB
Inactive:         405452 kB
Active(anon):     858016 kB
Inactive(anon):    22716 kB
Active(file):     473708 kB
Inactive(file):   382736 kB
Unevictable:          16 kB
Mlocked:              16 kB
SwapTotal:       2097148 kB
SwapFree:        2097148 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        856680 kB
Mapped:           238780 kB
Shmem:             24096 kB
Slab:             207272 kB
SReclaimable:     137524 kB
SUnreclaim:        69748 kB
KernelStack:       12688 kB
PageTables:        42656 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     4103272 kB
Committed_AS:    4643548 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
Percpu:            34816 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
```

<왼쪽 그림이 cpu정보, 오른쪽 그림이 메모리정보>

사용한 컴파일러 버전

```
baek@ubuntu:~/Desktop/hw3$ g++ --version
g++ (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3) 결과 화면과 결과에 대한 토의 내용

1. input1

```
5
2
41
0	1	1	13
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
1	1	6	6
1	0	6
1	0	6
1	0	6
1	0	6
0	1	8	5
0	0	8
1	1	3	6
0	1	9	2
1	1	2	7
0	1	4	1
1	0	2
1	0	2
1	1	5	8
1	1	7	5
0	0	1
0	0	9
1	0	6
0	0	1
1	0	2
0	0	4
0	0	1
1	0	5
1	0	7
1	0	6
0	0	1
0	0	8
1	0	3
1	0	6
0	0	9
1	0	2
0	0	4
1	0	6
1	0	5
```

a) FIFO 알고리즘 결과

```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |6666|6666|994-|----|5555|5555|2222|2222|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0011|1---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1100|0000|1111|1111|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 21
```

b) LRU 알고리즘 결과

```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |6666|6666|994-|----|5555|5555|2222|2222|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0011|1---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1100|0000|1111|1111|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 20
```

c) Sampled LRU 알고리즘 결과

```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |1111|1111|1111|1111|6666|6666|5555|5555|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1000|0000|0---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1100|0000|0000|0001|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 19
```

d) LFU 알고리즘 결과

```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |1111|1111|1111|1111|6666|6666|5555|5555|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1000|0000|0---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1100|0000|0000|0001|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 15
```

e) MFU 알고리즘 결과

```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |5555|5555|6666|6666|--4-|----|3333|3333|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|1---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1111|1111|0000|0001|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 21
```

f) OPTIMAL 알고리즘 결과

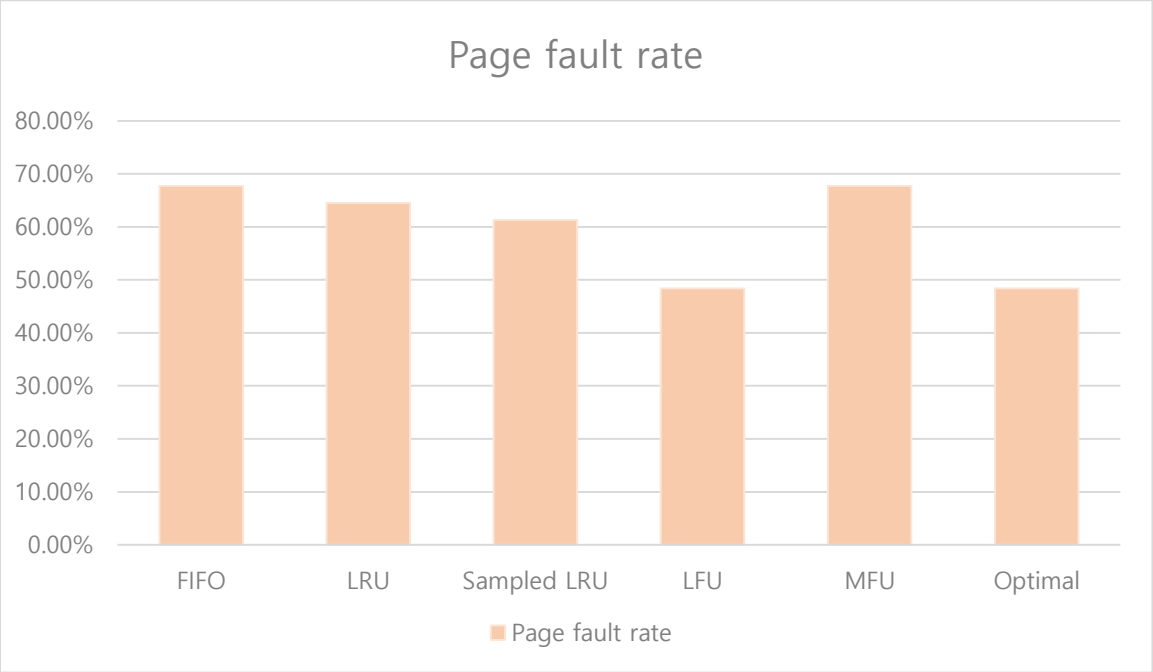
```
* Input : Pid [1] Function [ACCESS] Alloc ID [6] Page Num[0]
>> Physical Memory :          |8888|8888|994-|----|6666|6666|5555|5555|
>> pid(0) Page Table(AID) :   |1111|1111|1111|1888|8899|4---|----|----|----|----|----|----|----|----|----|----|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0111|1111|1---|----|----|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(AID) :   |6666|6633|3333|2222|2225|5555|5557|7777|----|----|----|----|----|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1100|0000|0000|0001|1111|1110|0000|----|----|----|----|----|----|----|----|

page fault = 15
```

g) 5가지 알고리즘의 Page fualt rate 비교

Page fault rate = (페이지폴트수 / 전체 access한 수)*100

알고리즘	Page fault rate
FIFO	67.74%
LRU	64.52%
Sampled LRU	61.29%
LFU	48.38%
MFU	67.74%
Optimal	48.38%



Optimal 과 LFU 가 가장 좋은 성능을 보이고 MFU 와 FIFO 가 가장 안 좋은 성능이 보인다. 여기서는 가장 많이 참조되는 것이 나중에 사용될 가능성이 커서 성능이 안좋다. 또한 FIFO 는 단순하여서 먼저 들어간 것이 나중에 참조를 많이 해서 성능이 안좋다. 만약 이 인풋의 형태로 프로그램을 짠다면 LFU 로 짜는 것이 이상적일 것이다.(Optimal 은 미래를 예측하지 못하면 구현하기 힘들기 때문이다). 또한 여기서 LRU 가 Sampled LRU 보다 성능이 안 좋은데 그 이유는 physical memory 가 작아서 이다.

2. input2

```
5
2
30
0	1	0	8
0	1	1	8
0	1	2	8
0	1	3	8
0	1	4	8
0	1	5	8
0	1	6	8
0	1	7	8
0	1	8	8
1	1	9	8
1	1	10	8
1	1	11	8
1	1	12	8
1	1	13	8
1	1	14	8
0	0	0
0	0	1
0	0	2
0	0	3
0	0	4
0	0	5
0	0	6
0	0	7
0	0	8
1	1	9
1	1	10
1	1	11
1	1	12
1	1	13
1	1	14
```

a) FIFO 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

b) LRU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

c) Sampled LRU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

d) LFU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

e) MFU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

f) OPTIMAL 알고리즘 결과

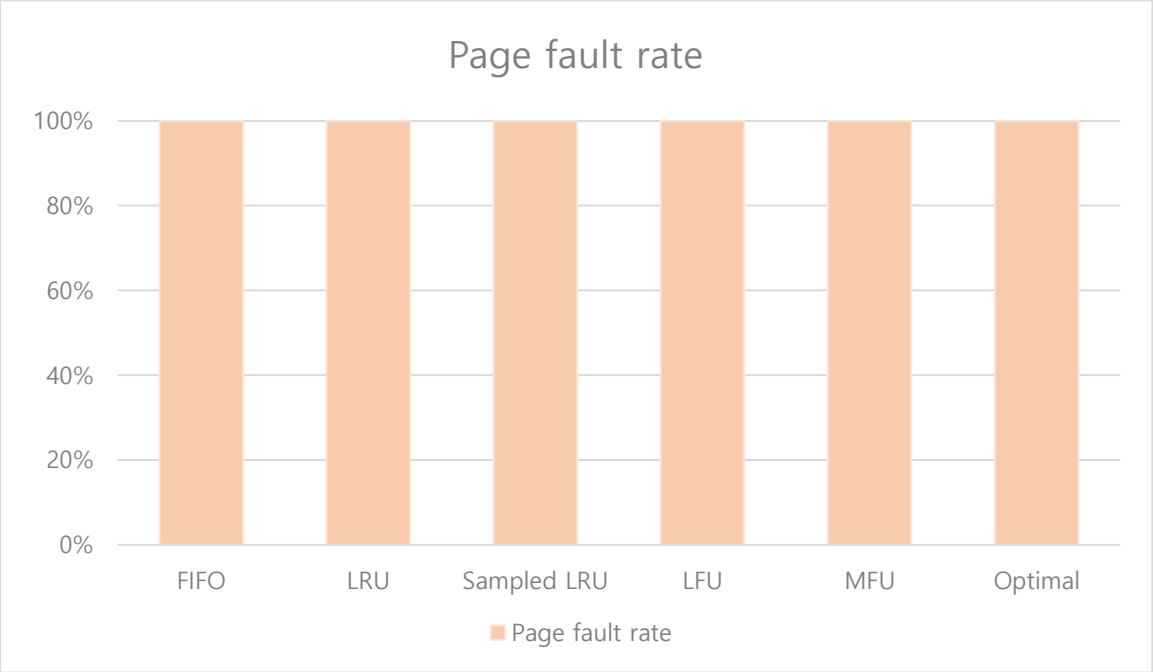
```
* Input : Pid [0] Function [ACCESS] Alloc ID [7] Page Num[0]
>> Physical Memory :          |4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|1111|1111|2222|2222|3333|3333|4444|4444|5555|5555|6666|6666|7777|7777|
>> pid(0) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|1111|1111|1111|1111|1111|1111|1111|1111|
>> pid(1) Page Table(AID) :   |9999|9999|10101010|10101010|11111111|11111111|12121212|12121212|13131313|13131313|14141414|14141414|----|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|----|----|----|----|

page fault = 8
```

g) 5가지 알고리즘의 Page fualt rate 비교

Page fault rate = (페이지폴트수 / 전체 access한 수)*100

알고리즘	Page fault rate
FIFO	100%
LRU	100%
Sampled LRU	100%
LFU	100%
MFU	100%
Optimal	100%



각각의 페이지가 한번씩만 access 되어서 모든 알고리즘의 성능이 같음을 볼 수 있다. 이럴경우에는 어떠한 알고리즘을 사용해도 상관없다.

3. input3

```
0
2
36
0	1	0	14
0	1	1	8
0	1	2	6
0	1	3	8
0	1	4	8
0	1	5	7
0	1	6	8
0	1	7	5
0	1	8	8
1	1	9	14
1	1	10	8
1	1	11	6
1	1	12	7
1	1	13	8
1	1	14	8
0	0	0
0	0	1
0	0	2
0	0	3
0	0	0
1	0	11
1	0	9
0	0	4
1	0	9
0	0	5
0	0	6
1	0	9
0	0	7
0	0	0
0	0	8
1	0	9
1	0	10
1	0	11
1	0	12
1	0	13
1	0	14
```

a) FIFO 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |0000|0000|0000|0000|----|----|7777|7777|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0001|1111|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 13
```

b) LRU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |----|----|7777|7777|0000|0000|0000|0000|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0001|1111|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 12
```

c) Sampled LRU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |0000|0000|0000|0000|7777|7777|----|----|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0001|1111|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 12
```

d) LFU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |0000|0000|0000|0000|9999|9999|9999|9999|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 12
```

e) MFU 알고리즘 결과

```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |----|----|11111111|11111111|0000|0000|0000|0000|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |0000|0000|0000|0000|0000|0011|1111|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 13
```

f) OPTIMAL 알고리즘 결과

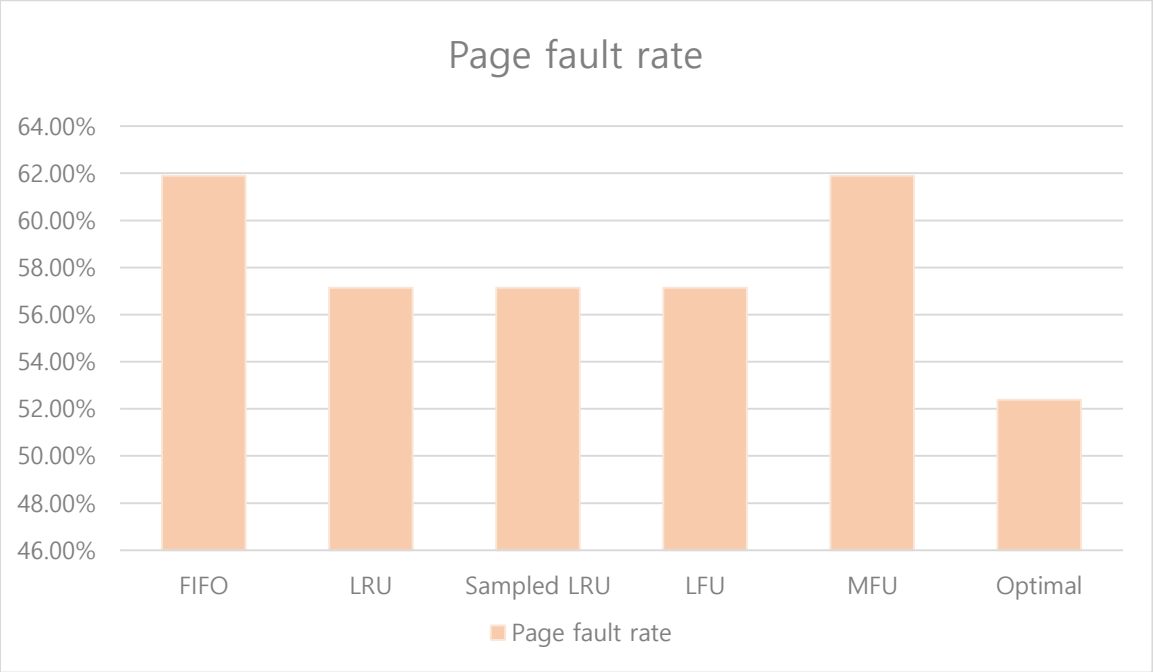
```
* Input : Pid [0] Function [ACCESS] Alloc ID [0] Page Num[0]
>> Physical Memory :          |0000|0000|0000|0000|9999|9999|9999|9999|
>> pid(0) Page Table(AID) :   |0000|0000|0000|0011|1111|1122|2222|3333|3333|4444|4444|5555|5556|6666|6667|7777|
>> pid(0) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(1) Page Table(AID) :   |9999|9999|9999|991010|10101010|10101111|11111111|12121212|12121213|13131313|13131314|14141414|141414-|----|----|----|
>> pid(1) Page Table(Valid) : |1111|1111|1111|1100|0000|0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|

page fault = 11
```

g) 5가지 알고리즘의 Page fualt rate 비교

Page fault rate = (페이지폴트수 / 전체 access한 수)*100

알고리즘	Page fault rate
FIFO	61.90%
LRU	57.14%
Sampled LRU	57.14%
LFU	57.14%
MFU	61.90%
Optimal	52.38%



Optimal 이 가장 성능이 좋고 그다음에 LRU, Sampled LRU, LFU 가 성능이 좋다. MFU 와 FIFO 는 둘다 성능이 좋지 않다. 이거 역시 가장 많이 참조되는 것이 나중에 사용될 가능성이 높은 input 이어서 이런 결과가 보여진 것 같다. FIFO 는 꾸준히 성능이 제일 안 좋다. 상황을 고려하지 않고 단순하게 고려해서 그런 것 같다. 우리는 알고리즘으로 짜서 LRU 에 대한 하드웨어적인 비용을 고려하지 않아도 되지만 실제로는 하드웨어의 도움이 필요하다. 그래서 이러한 하드웨어적인 cost 을 줄이기 위해서 나온 것이 Sampled LRU 이다. 위에서 LRU 와 성능 차이가 없으니 Sampled LRU 을 쓰는 것이 가장 효율적이다. LFU 같은 경우에는 직감적이기 때문에 LFU 와 MFU 중에 고르기가 힘들다. 그런데 input3 은 LFU 가 더 효율적이다.

4) 과제 수행 중 발생한 애로사항 및 해결방안

a) 버디 시스템 구현

버디시스템에서 buddy끼리 합쳐서 가장 작은 공간을 찾는 알고리즘을 구현하는 데 어려움을 겪었다. 잘 못 짜면 while문이 너무 많아져서 비효율적이기 때문이다. 하지만 OS 강의노트 8장 26쪽에서 The address of a block and its buddy differ in exactly one bit position. 라는 말에서 힌트를 얻어서 쉽게 구현이 가능하였다.

5) 참조 문헌

오에스 강의노트