Homework3 결과 보고서

2018147558. 김정주

1. 프로그램 동작 과정 및 구현 방법

   A. 과정에 필요한 구조체

      i. pageTable{bool Vbit, int frameIdx, int pageIdx, int size, int AID} : 각 프로세스가 가지고 있는 페이지로서, 프레임에 맵핑 여부를 알려주는 Vbit, 프레임에서의 index를 가지는 frameIdx, 가상 메모리의 index를 가지는 pageIdx, 페이지의 크기인 size, 할방 번호인 AID를 가진다. Process의 ptable로 사용된다.

      ii. Process{bool* isempty, vector<pageTable> ptable, int ntable} : 프로세스의 개수마다 존재하며, 가상 메모리의 공간이 사용되는지의 여부를 나타내는 isempty, page를 유지하는 ptable, page의 할당 번호를 정해주는 ntable로 구성된다. Process 벡터의 원소로서 사용된다.

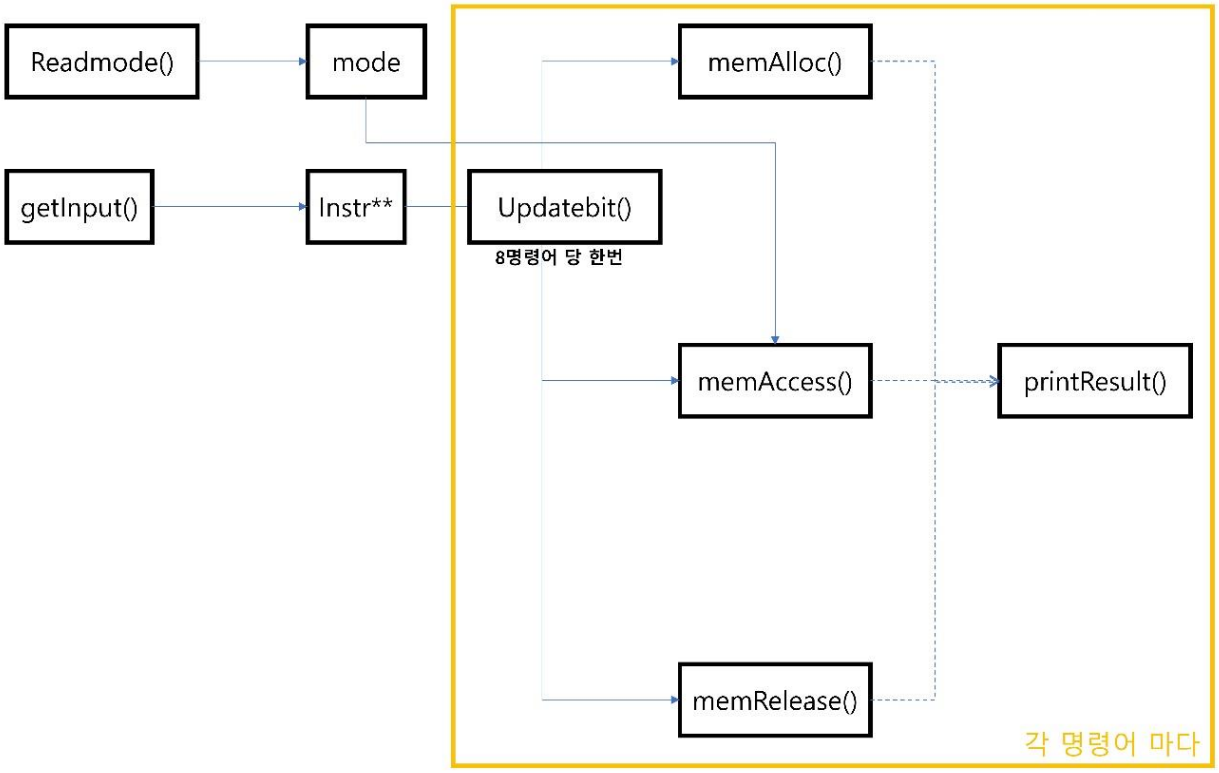
      iii. Frame{int AID, int PID, bool refer, bool use, int count, int rbyte} : 물리 메모리에 할당된 프레임을 나타내며, 프로세스 ID인 PID, sampled LRU에서 사용되는 refer, second-chance에서 사용되는 use, LFU/MFU에서 사용되는 count, sampled LRU에서 사용되는 rbyte가 있다. Physframe 벡터의 원소로서 사용된다.

      iv. Buddy{int AID, int PID, bool use, int size, int addr} : 버디 시스템으로 나누어진 buddy가 사용되는지 여부를 알려주는 use, 버디의 사이즈인 size, 버디의 시작 주소를 나타내는 addr이 있다. Buddysys 벡터의 원소로서 사용된다.

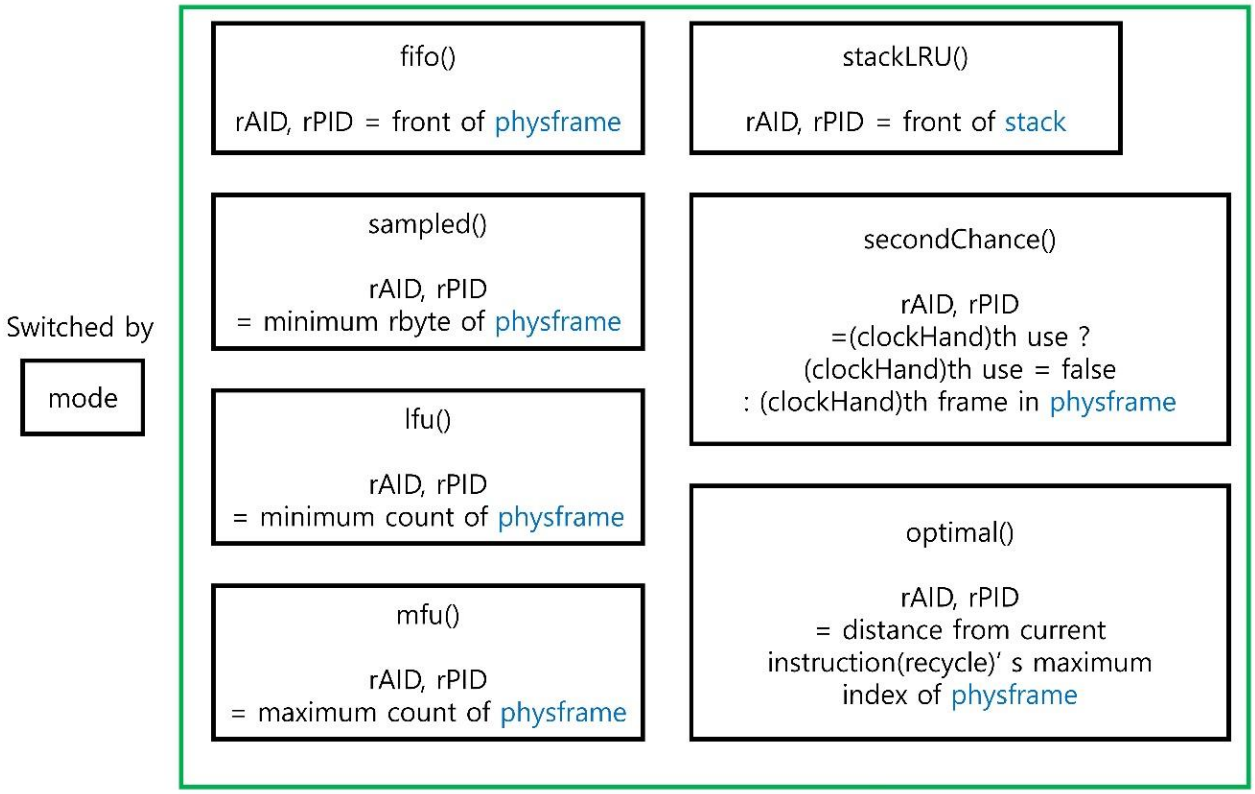
      v. miniF{int AID, int PID} : 오로지 stack LRU만을 위한 구조체이다. Stack 벡터의 원소로서 사용된다.

B. 프로그램 구조 도식도

i. 전체적인 flow



ii. 각 교체 알고리즘별 방식(파란색 글자는 구조체의 vector 형식)



2. 개발환경

```
y2018147558@y2018147558-VirtualBox:~$ uname -a
Linux y2018147558-VirtualBox 5.15.25-2018147558 #1 SMP Sat Mar 5 11:28:44 KST 2022 x8
6_64 x86_64 x86_64 GNU/Linux
y2018147558@y2018147558-VirtualBox:~$ lshw -short
WARNING: you should run this program as super-user.
H/W path    Device      Class       Description
============================================
                        system      Computer
/0                      bus         Motherboard
/0/0                    memory      7960MiB System memory
/0/1                    processor   11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40G
/0/100                  bridge      440FX - 82441FX PMC [Natoma]
/0/100/1                bridge      82371SB PIIX3 ISA [Natoma/Triton II]
/0/100/1.1              storage     82371AB/EB/MB PIIX4 IDE
/0/100/2                display     SVGA II Adapter
/0/100/3    enp0s3      network     82540EM Gigabit Ethernet Controller
/0/100/4                generic     VirtualBox Guest Service
/0/100/5                multimedia  82801AA AC'97 Audio Controller
/0/100/6                bus         KeyLargo/Intrepid USB
/0/100/7                bridge      82371AB/EB/MB PIIX4 ACPI
/0/100/d                storage     82801HM/HEM (ICH8M/ICH8M-E) SATA Controller
/0/2        scsi1       storage
/0/2/0.0.0  /dev/cdrom  disk        CD-ROM
WARNING: output may be incomplete or inaccurate, you should run this program as super
-user.
```

- 프로세스는 4코어, 메모리용량은 7960MiB이다.
- 컴파일러는 g++ 9.5.0을 사용하였다.

input1

```
56 1 2048 1024
0 0 3
0 0 7
0 0 8
0 0 10
0 0 6
0 0 3
0 0 7
0 0 5
0 0 4
1 0 0
1 0 1
1 0 2
1 0 3
1 0 1
1 0 3
1 0 1
1 0 2
1 0 0
1 0 1
1 0 2
1 0 0
1 0 1
1 0 0
1 0 2
1 0 3
1 0 4
1 0 5
1 0 6
1 0 8
1 0 8
1 0 7
1 0 3
1 0 7
1 0 6
1 0 5
1 0 6
1 0 7
1 0 0
1 0 7
1 0 1
1 0 7
1 0 8
1 0 2
1 0 6
1 0 6
1 0 5
1 0 7
1 0 0
1 0 8
1 0 7
1 0 6
1 0 5
1 0 4
1 0 7
1 0 0
1 0 5
```

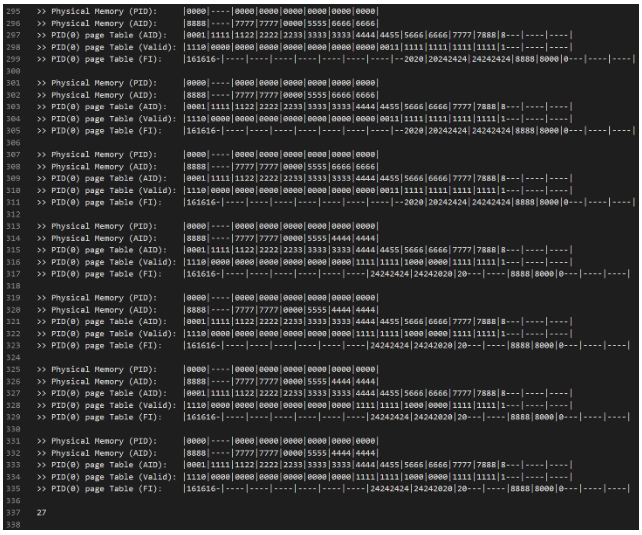
input2

```
42 1 2048 1024
0 0 2
0 0 9
0 0 6
0 0 8
0 0 3
0 0 8
0 0 5
0 0 1
0 0 9
1 0 0
1 0 1
1 0 2
1 0 3
1 0 4
1 0 5
1 0 5
1 0 7
1 0 6
1 0 8
1 0 3
1 0 4
1 0 1
1 0 7
1 0 8
1 0 3
1 0 6
1 0 5
1 0 4
1 0 6
1 0 7
1 0 6
1 0 8
1 0 8
1 0 7
1 0 4
1 0 6
1 0 7
1 0 6
1 0 8
1 0 8
1 0 6
1 0 7
```

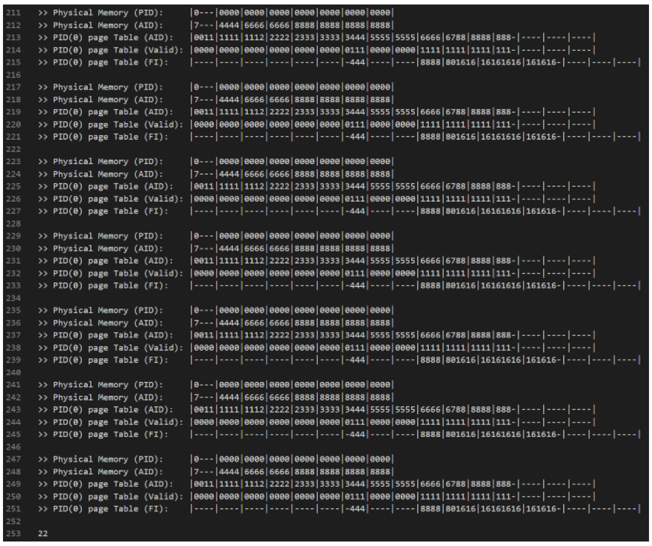
input3

```
48 1 2048 512
0 0 2
0 0 4
0 0 6
0 0 8
0 0 3
0 0 8
0 0 5
0 0 1
0 0 9
1 0 4
1 0 7
1 0 1
1 0 3
1 0 6
1 0 5
1 0 8
1 0 7
1 0 2
1 0 3
1 0 3
1 0 6
1 0 7
1 0 4
1 0 5
1 0 3
1 0 6
1 0 6
1 0 4
1 0 1
1 0 0
1 0 0
1 0 8
1 0 5
1 0 7
1 0 3
1 0 5
1 0 2
1 0 7
1 0 6
1 0 4
1 0 5
1 0 5
1 0 7
1 0 3
1 0 2
1 0 4
1 0 6
1 0 7
```

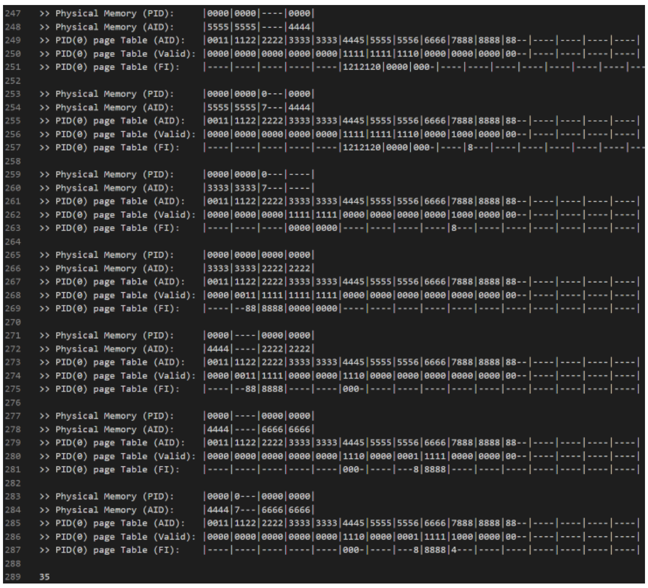
fifo input1 output(마지막 일부)



sampled input2 output(마지막 일부)



second-chance input3 output(마지막 일부)



3. 결과 분석

	input1	input2	input3
fifo	27	21	35
stack	25	22	35
sampled	29	17	34
second-chance	27	21	35
lfu	29	21	34
Mfu	30	24	35
optimal	19	16	29

A. Input1에서는 stack LRU 알고리즘이 좋은 결과를 냈다. 이는 LRU 알고리즘이 최근의 메모리 사용에 대한 패턴에 민감하게 반응하기 때문이여 수집된 데이터에 의한 결과라고 생각한다.

B. Input2에서는 sampled LRU 알고리즘 좋은 결과를 냈다. Input1과는 다르게 stack이 아닌 sampled LRU 알고리즘이 좋은 결과를 낸 이유는 input의 데이터를 살펴보면 알 수 있다. Stack의 경우는 frame을 사용한 순서만을 기억하지만 sampled LRU의 경우 위에서는 8개의 명령어의 locality에 영향을 크게 받는다. 그렇기에 input2에서는 locality를 잘 활용하는 명령어라는 것을 알 수 있고, input1의 경우는 8개 명령어 이상에서의 다른 page의 이용이 많다는 것을 추측할 수 있었다.

C. Input3는 물리 메모리의 용량을 512로 설정하였다. 예상대로 아무리 교체 알고리즘이 좋아도 절대적인 물리 메모리 량의 감소는 모든 알고리즘의 성능의 저하로 이어진다는 것을 알 수 있었다.

D. 이 데이터를 토대로 보면 대체로 LRU 알고리즘의 성능이 좋으나 명령어의 유형이나 사용 패턴에 따라서 각 알고리즘의 성능이 차이를 보인다는 것을 알 수 있었다. 그렇기에 일부 데이터만으로 알고리즘의 성능을 가늠하는 것은 무리이며, 각 시스템의 사용 패턴과 명령어를 분석하여 해당 시스템에 최적한 알고리즘을 찾는 것이 가장 효율적인 알고리즘 선택이라고 생각했다.

E. 모든 데이터에서 예상한대로 optimal이 가장 성능이 좋다는 것도 알 수 있었다.

4. 과제 도중 발생한 에로사항

A. Vector의 iterator 기능 중 erase를 사용 중의 segment fault

i. 원인 : iterator가 가리키는 vector 원소를 지우기 위해서 erase를 사용하는 도중 segment fault 오류가 자주 발생하였다. 이는 for문이나 while 문 내부에서 iterator가 가리키는 원소를 지우면, 이후 iterator는 자동으로 다음 원소를 가리키는 특징 때문에 발생했다.

ii. 해결 : erase 사용 후 iterator가 다음 원소를 가리키는 것을 상정하여 if문을 배치함으로써 해결했다.

B. Iterator 사용 중 syntax 오류

i. 원인 : iterator가 가리키는 원소에 접근하기 위해 .(dot)을 썻을 때, 발생하는 에러였다.

ii. 해결 : iterator는 vector의 원소를 가리키는 pointer이기에 원래 원소를 접근했던 방식과 다르게 ->문자로 접근해야 한다.