# 프로그래밍 수행

# 결과 보고서

-Assignment 3-





**과목명** | 운영체제

**담당 교수** | 차호정 교수님

**학과** | 교육학과

**학번** | 2018182058

**이름** | 박주언


**제출일** | 2020 년 6 월 12 일

# 개발 환경

✔ 메모리 정보

```
mingbee@mingbee-VirtualBox:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.8G        910M        2.3G         28M        651M        2.7G
Swap:          1.4G          0B        1.4G
```

**$free -h** 명령어를 사용해 메모리 사용 현황을 확인했다. -h 옵션으로 가독성을 높였다.

```
mingbee@mingbee-VirtualBox:~$ cat /proc/meminfo
MemTotal:        4029820 kB
MemFree:         2339316 kB
MemAvailable:    2821128 kB
Buffers:           97380 kB
Cached:           573548 kB
SwapCached:            0 kB
Active:          1119648 kB
Inactive:         355268 kB
Active(anon):     804904 kB
Inactive(anon):    28384 kB
Active(file):     314744 kB
Inactive(file):   326884 kB
Unevictable:          32 kB
Mlocked:              32 kB
SwapTotal:       1459804 kB
SwapFree:        1459804 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        804112 kB
Mapped:           241640 kB
Shmem:             29300 kB
KReclaimable:      86488 kB
Slab:             121616 kB
SReclaimable:      86488 kB
SUnreclaim:        35128 kB
KernelStack:        7808 kB
```

```
PageTables:         30004 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:      3474712 kB
Committed_AS:     3792756 kB
VmallocTotal:   34359738367 kB
VmallocUsed:        32920 kB
VmallocChunk:           0 kB
Percpu:              2592 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
DirectMap4k:       137152 kB
DirectMap2M:      4057088 kB
```

**$cat /proc/meminfo** 명령어를 사용해 전체 메모리 정보 또한 확인할 수 있었다.

✔ CPU 정보

```
mingbee@mingbee-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
stepping        : 10
cpu MHz         : 1992.001
cache size      : 8192 KB
physical id     : 0
siblings        : 6
core id         : 0
cpu cores       : 6
apicid          : 0
```

**$cat /proc/cpuinfo** 명령어를 사용해 CPU 정보를 확인했다. CPU 코어는 6 개다.

✔ 컴파일러 버전

```
mingbee@mingbee-VirtualBox:~$ gcc --version
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
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.
```

**$ gcc --version** 명령어를 사용하여 gcc 컴파일러의 버전을 확인하였다.

```
mingbee@mingbee-VirtualBox:~$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
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.
```

**$ g++ --version** 명령어를 사용하여 g++ 컴파일러의 버전을 확인하였다.

채점환경과 동일한 컴파일 환경을 만들기 위해, gcc 5, g++ 5 패키지도 설치하여 테스트해보았다.

```
mingbee@mingbee-VirtualBox:~$ sudo update-alternatives --config gcc
There are 2 choices for the alternative gcc (providing /usr/bin/gcc).

  Selection    Path            Priority   Status
------------------------------------------------------------
  0            /usr/bin/gcc-7   80        auto mode
* 1            /usr/bin/gcc-5   60        manual mode
  2            /usr/bin/gcc-7   80        manual mode
```

```
mingbee@mingbee-VirtualBox:~$ sudo update-alternatives --config g++
There are 2 choices for the alternative g++ (providing /usr/bin/g++).

  Selection    Path            Priority   Status
------------------------------------------------------------
  0            /usr/bin/g++-7   80        auto mode
* 1            /usr/bin/g++-5   60        manual mode
  2            /usr/bin/g++-7   80        manual mode
```

각각 **$sudo update-alternatives --config gcc** 그리고 **$sudo update-alternatives --config g++** 커맨드로 우분투에 깔려 있는 여러가지 컴파일러 버전을 확인할 수 있었다. 버전을 확인하는 명령어를 입력하는 경우에는 priority 가 높은 7.5 버전이 출력되게 설정하였다.

✔ 시스템 정보

```
mingbee@mingbee-VirtualBox:~$ uname -a
Linux mingbee-VirtualBox 5.3.0-46-generic #38~18.04.1-Ubuntu SMP Tue Mar 31 04:1
7:56 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

**$uname -a** 명령어를 사용하여 과제 수행을 진행한 시스템의 정보를 확인하였다.

# Makefile

✔ Makefile 작성

```
1   project3 : schedAl.o
2       g++ -o project3 project3.cpp schedAl.o
3
4   schedAl.o : schedAl.h schedAl.cpp
5       g++ -c -o schedAl.o schedAl.cpp
6
7   clean :
8       rm -f project3 schedAl.o
9
10
```

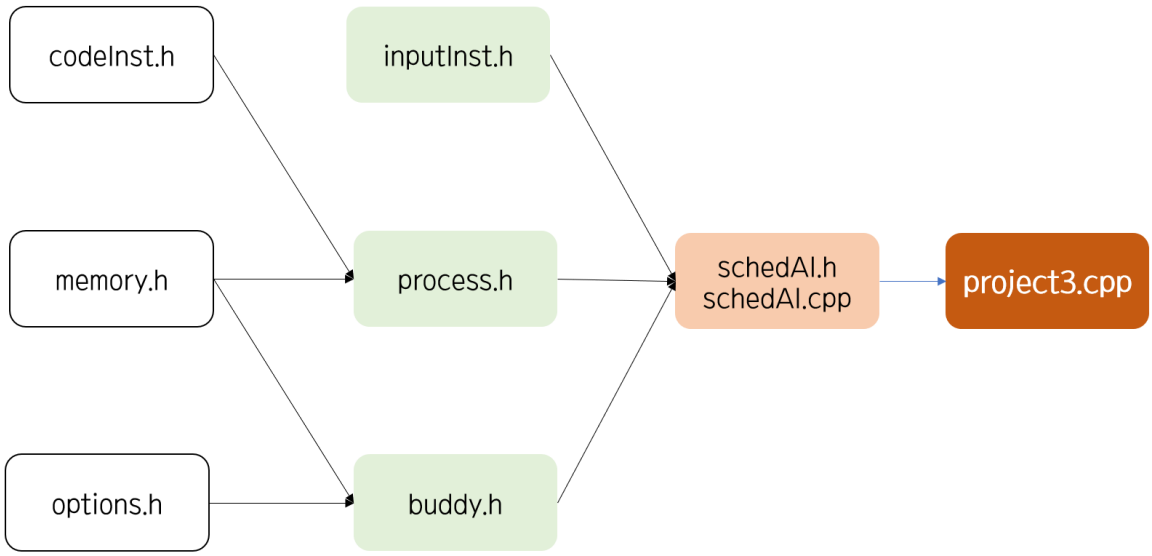
저번 과제에서 makefile 작성법을 조금 익혔기 때문에 조금 수월하게 작성할 수 있었다.

```
mingbee@mingbee-VirtualBox:~/Desktop/OS/kkkk$ make
g++ -c -o schedAl.o schedAl.cpp
g++ -o project3 project3.cpp schedAl.o
```

```
mingbee@mingbee-VirtualBox:~/Desktop/OS/kkkk$ make clean
rm -f project3 schedAl.o
```

# 프로그램 구조 설명

✔ 프로그램 전체 구조




✔ codeInst.h

인풋으로 들어오는 프로그램 파일의 코드를 스트럭트 codeInst 속 opcode 와 arg 에 나누어 저장한다.

✔ memory.h

가상메모리와 물리메모리의 정보를 담는 스트럭트를 담고 있다. 메모리의 사이즈와 메모리 테이블을 가지고 있다. 메모리 테이블은 이중벡터로 구성했다. 메모리 사이즈는 입력받는 수가 아닌 페이지(프레임)의 사이즈로 나누어 그 총 개수(페이지 혹은 프레임의 열의 개수)로 저장했다.

✔ options.h

Option 스트럭트에 프로그램을 실행시킬 때 명령어로 받는 옵션들을 저장한다. page, sched, dir 의 값을 저장한다.

✔ inputInst.h

인풋파일의 명령어를 저장하는 inputInst 스트럭트가 선언되어 있다. 들어오는 cycleTime 과 codename, IO 오퍼레이션인 경우 pid 까지 저장한다. 편의상 IO 명령일때의 코드네임은 'INPUT'으로 했다.

✔ process.h

프로세스 스트럭트가 선언되어있다. 프로세스를 실행할 때 필요한 정보를 모두 담고 있다. 각 프로세스 별 코드리스트를 만들어서 관리하고, 프로세스 별 가상메모리와 가상메모리 포인터도 설정해주었다.

✔ buddy.h

프로그램이 물리 메모리에 접근하려고 할 때 해당 파일 속 Buddy 클래스를 거친다.

Pages 라는 스트럭트에 물리 메모리에 적재된 페이지와 그 정보를 저장한다. 페이지에는 각 페이지 교체 알고리즘 별 필요한 정보들(시간, 카운트 등)을 저장한다.

- *allocatePM()*

  aid 와 필요한 프레임의 사이즈를 받아온 뒤, 물리 메모리의 정보를 담고 있는 노드를 통해 탐색한다. 만약 search 를 통해 비어있는 노드를 발견했을 경우 그 노드에 aid 값을 저장하고, 1 을 리턴한다(페이지 폴트 개수를 구하기 위해).

  만약 비어 있는 노드가 없다면 페이지 폴트가 발생하여, 알고리즘별로 함수를 호출한다.

```
if (options.page=="fifo") {
    pageF=fifo(PM, VM , demandSize, aid, idx);
} else if (options.page=="lru") {
    pageF=lru(PM, VM , demandSize, aid, idx);
} else if (options.page=="lru-sample") {
    pageF=lruS(PM, VM, demandSize, aid, idx);
} else if (options.page=="lfu") {
    pageF=lfu(PM, VM, demandSize, aid, idx);
} else if (options.page=="mfu") {
    pageF=mfu(PM, VM, demandSize, aid, idx);
} else if (options.page=="optimal") {
    pageF=optimal(PM, VM, demandSize, aid, idx, accessCodeList, cycle);
}
```

- *split()*

search()를 진행하다가 노드가 사이즈가 큰 경우(필요한 사이즈보다) 노드를 나누어 준다. 본래 노드의 left 와 right 로 split 하고 각각 노드들의 파렌트 노드로 원래 노드를 연결해준다.

- *merge()*

나누어 졌던 노드를 합한다. right 노드와 계속해서 합하여서 더 이상 합할 수 없을 때까지 합한다.

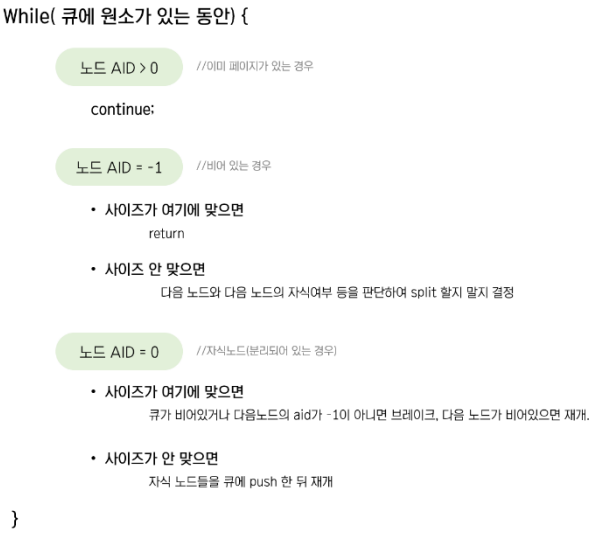
- *mergeCheck()*

merge 를 하기에 앞서 합할 수 있는지 여부를 검사한다. 전달되는 노드가 왼쪽인지 오른쪽 노드인지에 따라 검사방식을 다르게 하고, 각 오른쪽/왼쪽의 노드가 비어있는지 여부를 검사하여 합할 수 있는 지 없는지를 반환한다.

- *binaryCheck()*

버디시스템의 원리에 따라 필요한 크기와 2^U 의 관계를 검사한다. 해당 노드에 들어갈 수 있는지를 검사한다.

- *search()*




버디 시스템 클래스를 생성할 때 루트 노드도 하나 함께 생성하여, 그 루트 노드를 기준으로 모든 search 를진행한다. Node 에는 노드의 시작 위치와 끝위치, 사이즈, 부모노드가 저장되어 있다.

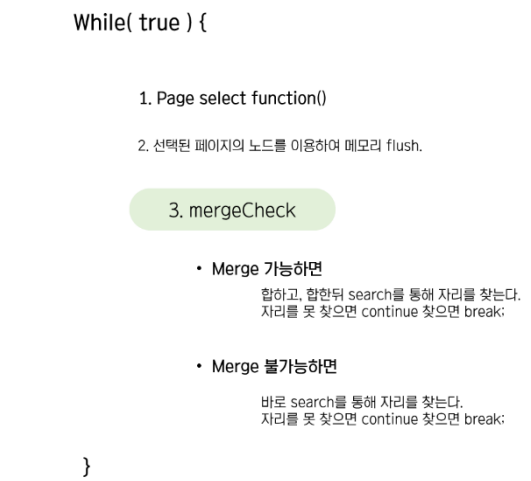
search 는 bfs 의 방식을 채택하여 부모노드부터 자식노드별로 순서대로 빈 위치를 찾아나간다.

노드의 aid 는 0, -1, 1 보다 큰 정수로 이루어져 있으며, 0은 자식이 있다는 것을 의미하고, -1은 비어 있다는 것을 의미한다. 1 보다 큰 정수는 그 안에 저장된 페이지의 aid 이다.

- *fillMem() & flushMem() & releaseMem() & eraseMem()*

노드와 페이지 테이블 혹은 물리메모리를 저장하고 있는 memory 의 메모리테이블의 값을 연동시켜준다. 각각 메모리 테이블을 채우기, 비우기이며 상황에따라 전달되는 인자나 바꾸는 값 등이 달라서 비슷한 기능이지만 이름을 다르게 만들었다.

- *페이지 교체 함수 & 페이지 선택 함수*

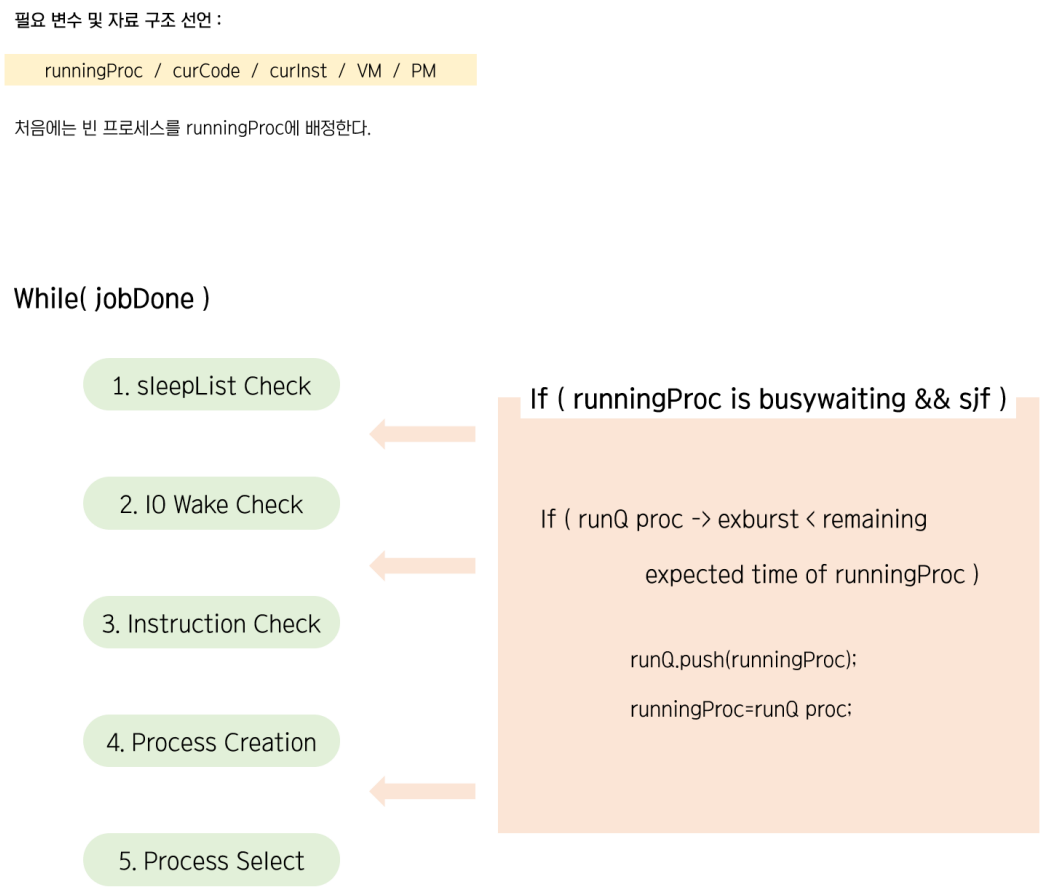



페이지 교체 알고리즘 별로 페이지 선택을 무엇으로 할 것인지 달라진다. 따라서 페이지 별로 페이지를 선택한 다음, 해당 페이지의 노드를 기준으로 페이지 교체를 진행한다. 먼저 반환된 페이지의 노드가 merge 될 수 있는지 체크하고, 될 수 있다면 합하고 난 뒤 다시 search 하고, merge 될 수 없다면 바로 search 를 진행한다.

allocatePM()에서 search 를 하고 난뒤, 자리가 없는 경우에 이 함수가 불리게 된다.

## ✔ schedAl.h & schedAl.cpp

buddy에서 버디시스템을 구현하였다면, schedAl 에서는 스케쥴러가 구현되어 있다. scheduler() 라는 함수와 sjf 인 경우에 예상 burst를 계산할 수 있는 두가지 함수로 구성되어 있다.




sjf 인 경우 러닝 프로세스가 비지 웨이팅하고 있을 경우에 무한루프를 돌 수 있기 때문에, 런큐에 새로운 프로세스가 푸쉬 될 수 있는 부분의 뒤에선 런큐에 있는 프로세스들이 비지웨이팅 러닝 함수를 대체하여 선택될 수 있는지 체크하였다.

그 후에는 프로세스 별로 opcode를 받아와 코드를 실행하고 결과를 출력했다.

## ✔ project3.cpp

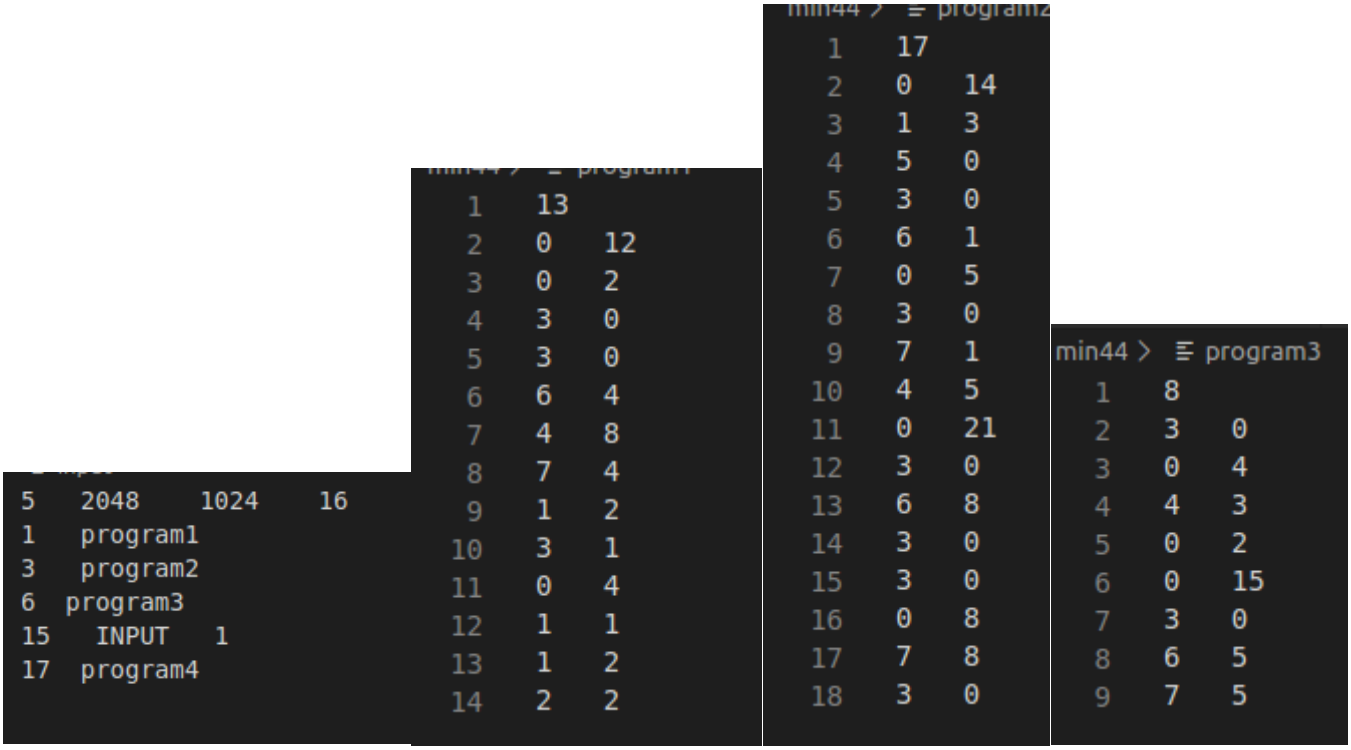
메인 함수가 있는 곳으로, 나머지 모든 헤더파일을 참조하여 실제 인풋도 처리하고, 시뮬레이터를 수행한다.

- *optimalSimulator()*

옵티멀 페이지 교체 알고리즘을 처리하기 위한 함수이다. vector<pair<int, int>>에 액세스를 시도하는 경우의 aid 를 싸이클 수와 함께 저장하여, schedAl 속 simulator 에 전달한다. 이 벡터는 buddy.h 에서 옵티멀 방식으로 페이지 교체를 진행할 때도 사용된다.

## 프로그래밍 결과

✔ input1

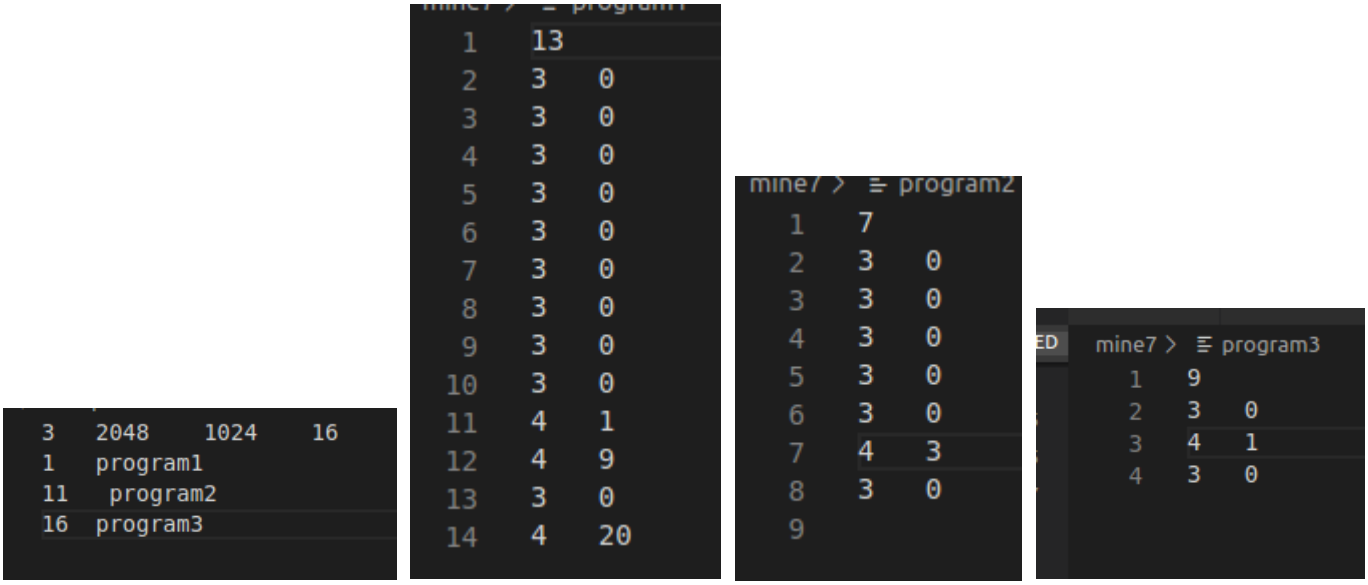



| input1 | |
|---|---|
| | test |
| rr | 31 |
| sjf-s | 20 |
| sjf-e | 20 |

위 표에서 볼 수 있듯이, busywaiting 을 하는 경우가 포함된 경우에는 rr 과 sjf 계열 스케쥴러간의 차이가 꽤 났다. 이는 sjf 계열에서 위의 프로그램 동작 방법에서 설명한 것처럼, 런큐를 계속해서 체크할 수 있어서 그런 것이라고 추측할 수 있었다. 비지웨이팅을 하는 경우이기 때문에 fcfs 는 제외했다.

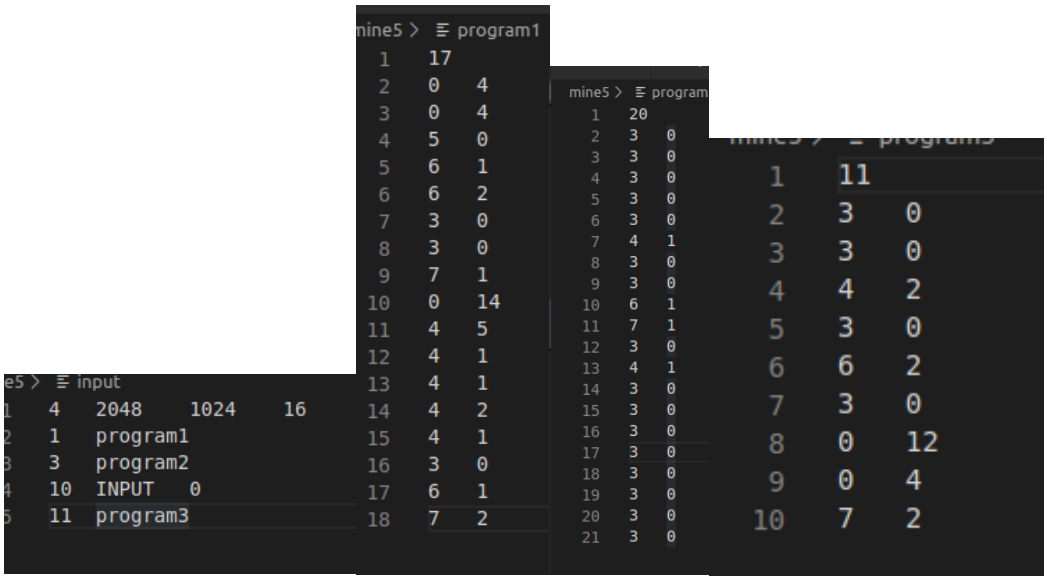
✔ input2

| Input2 | |
|---|---|
| fcfs | 37 |
| rr | 35 |
| sjfs | 41 |
| sjfe | 41 |




위의 예시와는 다르게, 이번에는 비슷비슷한 결과가 나오거나 오히려 fcfs 와 rr 이 성능이 더 좋았다.

✔ input3




| input3 | |
|---|---|
| fcfs | |
| rr | 97 |
| sjfs | 42 |
| sjfe | 59 |

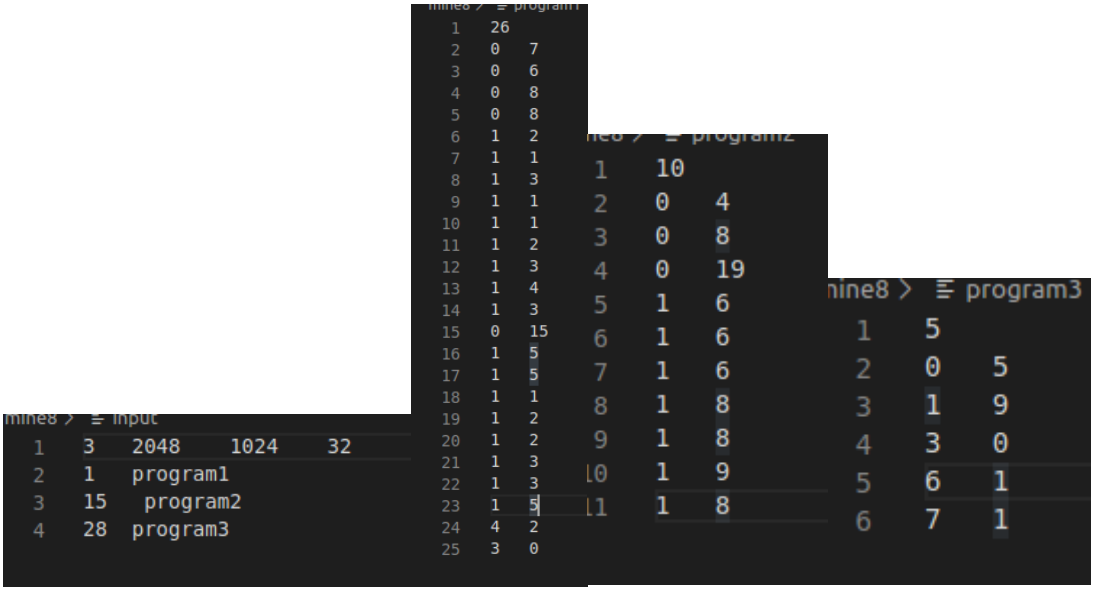
여기서도 마찬가지로 비지웨이팅을 하는 상황을 설정해보았다. 인풋1에서는 한나의 비지웨이팅 프로세스가 발생하였지만, 이번에는 두번을 발생하게 했더니 성능차이가 훨씬 눈에 띄게 보였다. 이렇게 비지웨이팅이 생길 수 있는 경우는 sjf가 성능면에서 더 우수하다고 볼 수 있을 것 같다.

✔ input4

| Input4 | |
|---|---|
| | |
| fcfs | 29 |
| rr | 28 |
| sjfs | 37 |
| sjfe | 37 |

하지만 비지웨이팅이 없이, 한 번에 세 프로세스가 모두 슬립에 들어서게끔 했을 때, 오히려 fcfs와 rr이 더 좋은 성능을 보이기도 했다. 이는 바로 프로그램1은 실행시간을 길게, 2와 3은 짧게 하고서 끝부분에 sleep 코드를 집어넣었기 때문이다. 이렇게 상황에 따라 스케줄러의 상황은 천차만별로 달라질 수 있다.

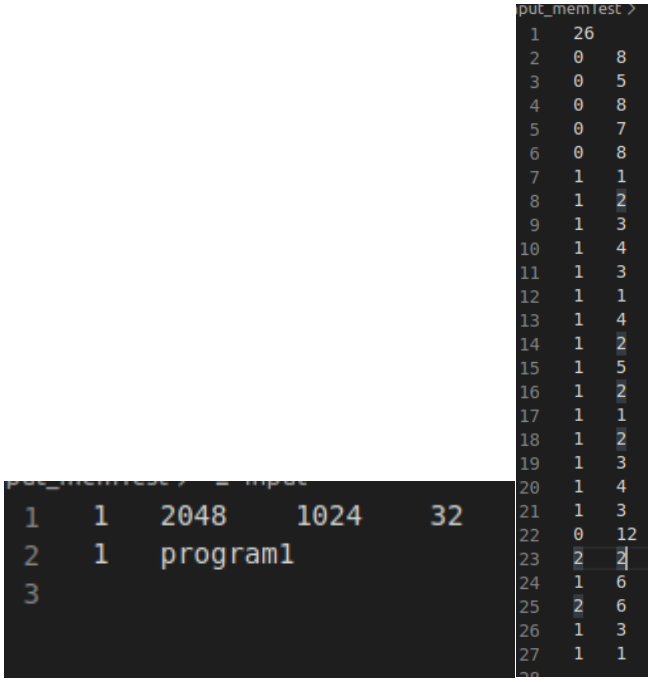
✔ input5




| Input5 | |
|---|---|
| lru-s | 13 |
| lru | 11 |
| mfu | 12 |
| lfu | 12 |
| fifo | 11 |
| optimal | 11 |

스케줄러와 달리 메모리 부분에서는 인풋을 짜는 것이 쉽지 않았다. 그만큼 의미있는 성능을 내기도 힘들었다. 예상과는 다르게 fifo의 성능이 준수하게 나왔다. 예시 인풋에서도 그렇고, 프로그램의 크기가 더 커지지 않는 이상 더 의미있는 차이를 만들기엔 어려워 보였다. 더욱이 여러가지 프로세스와 메모리를 한번에 고려해야하는 점에서 인풋을 짜기가 더 어려웠다.

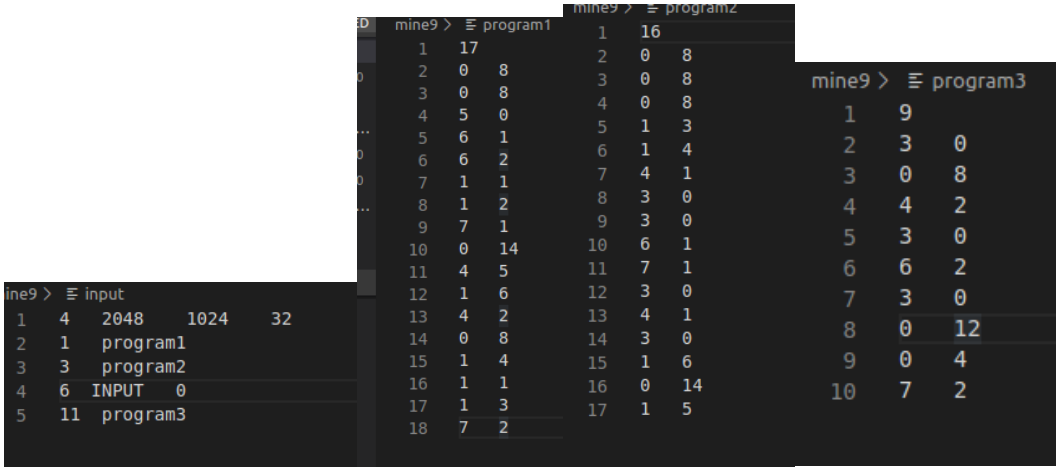
✔ input6




| Input6 | |
|---|---|
| lru-s | 8 |
| lru | 9 |
| mfu | 12 |
| lfu | 8 |
| fifo | 13 |
| optimal | 10 |

따라서 하나의 프로그램을 가지고 알고리즘별 성능을 조금 더 분석해보았다. 그랬더니 아까보다는 확연하게 알고리즘별 성능이 눈에 띄게 차이났다. 이번에는 파이포가 성능이 가장 낮게 나왔고, lru와 lfu가 성능이 가장 좋게 보였다. 옵티멀 또한 미래를 합당한 근거로 예측했다는 점에서 괜찮은 성적이 나왔다.

✔ input7




| Input7 | |
|---|---|
| lru-s | 4 |
| lru | 7 |
| mfu | 7 |
| lfu | 7 |
| fifo | 7 |
| optimal | 8 |

마지막으로 비지웨이팅 하는 경우를 체크해보았다. 그랬더니 확연하게 lru-sample 의 성능이 우수하게 나왔다. 앞전에서 계속해서 좋은 성능을 보이던 옵티멀은 안 좋은 모습을 보였다.

이를 통해 알 수 있는 것은, 절대적으로 어떠한 스케쥴러 혹은 페이지 교체 알고리즘이 낫다는 것을 미리 파악하기는 힘들다는 것이다. 그래서 더더욱 소프트웨어의 발전을 위해 이러한 알고리즘의 연구가 필요하다는 것을 느낄 수 있었다. 이렇게 작은 규모의 인풋말고, 조금 더 큰 인풋을 만들 수 있다면 조금 더 경향성이 보였을 것이라고 생각한다.

## 💻 과제 수행 중 애로 사항

가장 어려웠던 것은 아무래도 개념 자체의 이해였다. 그 구조가 복잡해서 구조를 파악하는 것 부터가 쉽지 않았고, 구조를 파악하고 난 뒤에도 구현에서 많은 어려움이 있었다.

먼저 버디시스템을 구현할 때 이진트리 자료구조를 활용해야 했기 때문에 조금 어려웠다. 어떠한 순회 방식을 사용해야 하는지 시뮬레이션을 많이 해보았는데 그 과정에서 많은 어려움을 겪었다.

두번째로, c++ 언어 사용의 미숙이 큰 걸림돌이 되었다. 포인터의 사용법도 잘 모르기 때문에, 이번 과제에서 특히나 애를 먹었다. 포인터로 자료를 전달하거나, 배열의 동적할당의 위험성 등을 잘 몰랐기 때문에 많은 시행착오가 있었다. 특히나 벡터의 값이 전달이 안되거나 하는 등의 문제가 많아서 많이 힘들었지만, 이번 과제를 통해 포인터의 사용법 등을 확실하게 익힐 수 있었다.

## 참고 사이트

강의 슬라이드

강의 교재

https://adnoctum.tistory.com/216