OS Assignment #3

2015147531 서기원

1. 개발 환경

```
giwon@giwon-VirtualBox:~$ uname -a
Linux giwon-VirtualBox 5.4.28-2015147531 #1 SMP Sun Mar 29 02:17:02 KST 2020 x86_64 x86_64 x86_64 GNU/Linux
giwon@giwon-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.

giwon@giwon-VirtualBox:~$ grep -c processor /proc/cpuinfo
2
```

```
giwon@giwon-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz
stepping        : 3
cpu MHz         : 2594.012
cache size      : 3072 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_know
_freq pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2 invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass litf mds swapgs itlb_multihit
bogomips        : 5188.02
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz
stepping        : 3
cpu MHz         : 2594.012
cache size      : 3072 KB
physical id     : 0
```

```
processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz
stepping        : 3
cpu MHz         : 2594.012
cache size      : 3072 KB
physical id     : 0
siblings        : 2
core id         : 1
cpu cores       : 2
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_know
_freq pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2 invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass litf mds swapgs itlb_multihit
bogomips        : 5188.02
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

giwon@giwon-VirtualBox:~$
```

g++ 컴파일러의 버전은 7.5.0 버전이었으며 cpu는 2개의 코어로 설정하여 프로그래밍을 진행하였다. cpu 정보는 2장에 걸쳐 그림으로 나타내었다.

```
giwon@giwon-VirtualBox:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           3.8G        879M        2.0G         38M        1.0G        2.7G
스왑:          2.0G          0B        2.0G
```

메모리 정보는 위의 그림과 같다.

2. 프로그램의 기본 구성

－구조체

[Instruction] : 'input' 파일을 읽으면서 정보를 저장하는 구조체

time : 시간 cycle

pid : "INPUT" 명령어를 실행 시, 프로세스의 id를 구별하기 위해 저장하는 변수

proc_name : 명령어의 문자열, "INPUT"이 아닐 시, 프로세스의 이름을 저장.

[Process] : 각 프로세스 별 내용을 저장하는 구조체

name : 프로세스의 이름

pid : 프로세스의 ID

set : 명령어 배열

inst_num : 프로세스의 명령어 총 개수

job_index : 프로세스 프로그램 카운터(PC)

is_finished : 프로세스의 종료 여부

sleep_cycle : 프로세스 sleep 동작을 위한 카운터

sleep : 프로세스의 sleep 여부

wait : 프로세스의 IO-wait 여부

elapsed_time : 프로세스가 "run"하고 나서부터의 시간

cpu_burst : 그 전 burst 시점부터 runtime동안 흐른 시간 (T[n])

burst_num : cpu_burst를 한 횟수

is_first : cpu_burst를 경험했는지 여부

busy_waiting : lock이 걸려 busy-waiting있는 경우

pg_aid : page table[allocation id]

pg_valid : page table[valid]

－클래스

[Scheduler] : 스케줄링과 Paging을 모두 담당하는 클래스

SR : Scheduled Process

RP : Running Process

rq : Running Queue

Sl : sleeping list

IO : IO-wait list

input : 메인 input 파일에 적혀있는 명령어들을 저장

INPUT의 정보를 instruction 구조체에 저장하여 필요한 프로세스의 수만큼 Process 구조체를 할당하였다. 프로세스들의 정보를 바탕으로 스케줄링 및 페이징을 하는 클래스를 정의하였다.

3. 프로그램의 동작 과정과 구현 방법

```
switch(this->soption) {
        case 1:
                FCFS();
                break;
        case 2:
                RR();
                break;
        case 3:
                SJF_simple();
                break;
        case 4:
                SJF_EXP();
                break;
}
```

```
// check if every Process is finished
void Scheduler::checkProcess() {
        for (int i=0; i<proc_num; i++) {
                if (proc[i]->is_finish) {
                        sch_finish = true;
                        continue;
                }
                else {
                        sch_finish = false;
                        break;
                }
        }
}
```

메인 함수에서 인자로 받은 스케줄링 옵션을 통해 스케줄링 방법을 정하고 해당 함수를 실행한다. 또한, 매 사이클마다 모든 프로세스가 종료되었는지 여부를 검사하는 함수를 설정하였다.

과제에서 주어진 작업들의 우선 순위에 따랐다.

```
//check if there is a process to awake
for (int i=0; i<Sl.size(); i++) {
        Sl[i]->sleep_cycle--;
        if (Sl[i]->sleep_cycle == 0) {
                // awake
                Sl[i]->sleep = false;
```

우선, 매 사이클마다 sleep list에 존재하는 프로세스들의 sleep cycle을 1씩 줄이고 만약 0이 되었다면 list에서 빠져나와 runQueue로 등록이 되는 awake 과정을 구현하였다.

그다음 instruction set이 저장되어 있는 구조체 input에서 cycle과 time이 일치하는지 확인 후 io 작업이 있을 시 실행한다.

```
if (!strcmp(input[event_index-1].proc_name, "INPUT")) {
```

두 번째 인자의 이름이 INPUT인 것이 확인되면 io 작업을 진행한다.

그 외에 process 생성작업을 시행하여 runQueue의 뒤에 삽입한다. 생성 작업이 없을 시 스케줄링을 통해 수행할 process를 획득하는 과정을 거친다.

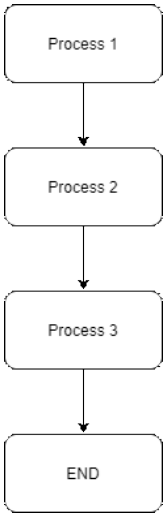
<스케줄링>

1_ FCFS

스케줄링 된 프로세스가 끝나거나 cpu burst가 일어날 시 프로세스 교체가 일어난다.

```
else if (RP!=NULL) {
        if(RP->is_finish || RP->sleep)  RP = NULL;
}
```

RP의 is_finish가 true가 되거나 io, sleep 명령을 실행할 시 프로세스 교체가 일어난다.



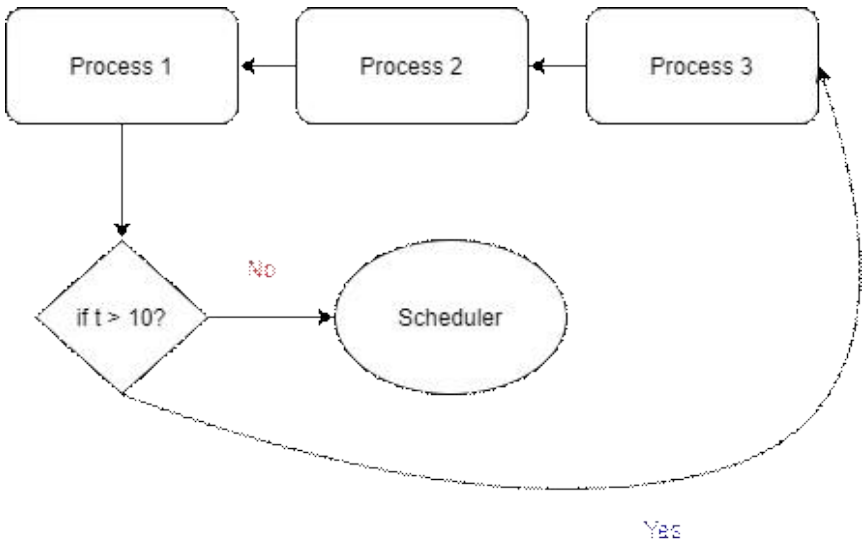
FCFS 순서도

2_ Round Robin

```
const unsigned int time_quantum = 10;
```

time quantum을 10으로 정의하여 해당 프로세스가 다른 io/sleep/finish 명령을 수행하지 않고 10 cycle동안 runtime을 갖게 된다면 다른 프로세스에게 running process의 권한을 주었다.

```
else if (RP->elapsed_time == time_quantum) {
        // initialize the elapsed time
        RP->elapsed_time = 0;
        // store in runQueue
        rq.push_back(RP);
        RP = NULL;
}
```



RR 순서도

3_ SJF - simple

```
double* sT = new double[proc_num];
for (int i=0; i<proc_num; i++) sT[i] = 0;
```

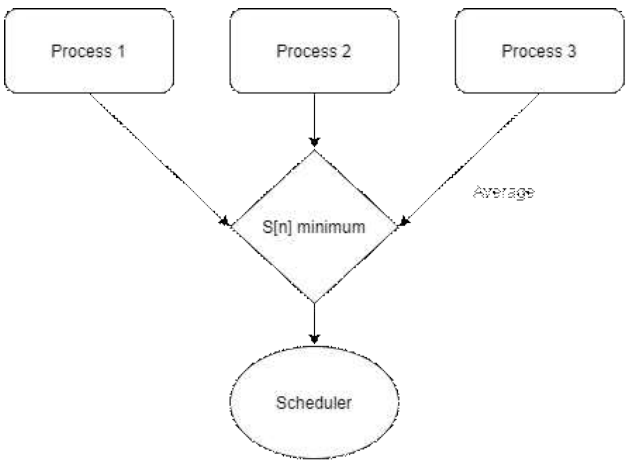
프로세스 개수만큼의 크기를 가진 double형 배열을 선언하여 각 프로세스별 예상 burst time을 매 사이클마다 갱신하여 비교하였다.

```
sT[p->pid] = 5;
```

프로세스를 생성할 당시 초기 burst 예상 값을 5로 초기화하였다.

```
if(RP->is_first) {
        sT[RP->pid] = RP->cpu_burst;
        RP->cpu_burst = 0;
        RP->is_first = false;
        Sl.push_back(RP);
}
else {
        sT[RP->pid] *= ((RP->burst_num - 1) / RP->burst_num);
        sT[RP->pid] += (RP->cpu_burst / RP->burst_num);
        RP->cpu_burst = 0;
        Sl.push_back(RP);
}
```

burst가 일어났을 때, S[0]를 고려하지 않기 위해 이것이 첫 번재 busrt인지를 검사하고 프로세스 구조체 안의 변수 burst_num과 함께 S[n+1]값을 계산하였다. 이후 사이클이 바뀌었을 때, running process를 정하는 과정에서 갱신된 S값을 비교하여 running process를 결정하였다.



Text

SJF -simple 순서도

4_ SJF－exponential

```
double* sT = new double[proc_num];
for (int i=0; i<proc_num; i++) sT[i] = 0;
```

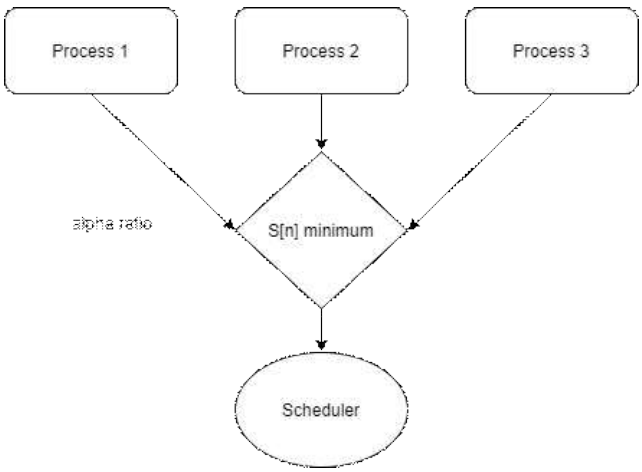
```
sT[p->pid] = 5;
```

여기까지의 방법은 SJF－simple과의 방법이 같다.

```
const double alpha = 0.6;
```

```
if(RP->is_first) {
        sT[RP->pid] = RP->cpu_burst;
        RP->cpu_burst = 0;
        RP->is_first = false;
        Sl.push_back(RP);
}
else {
        sT[RP->pid] *= (1-alpha);
        sT[RP->pid] += RP->cpu_burst * alpha;
        RP->cpu_burst = 0;
        Sl.push_back(RP);
}
```

하지만, S[n+1]을 계산하는 과정에서 burst_num이 아닌 일정한 상수 알파(=0.6)을 사용하여 갱신되는 S[n+1]을 구했다.



SJF - exponential 순서도

<페이징>

```
void Scheduler::paging() {
        switch(this->poption) {
                case 1:
                        FIFO();
                        break;
                case 2:
                        LRU();
                        break;
                case 3:
                        LRU_SAMPLED();
                        break;
                case 4:
                        LFU();
                        break;
                case 5:
                        MFU();
                        break;
                case 6:
                        OPTIMAL();
                        break;
        }
}
```

스케줄링과 비슷한 방법으로 paging option 변수를 통해 페이지 교체 알고리즘을 적용했다.

```
page_number.push_back(RP->set[RP->job_index][1]);
```

빠른 접근을 위해 memory allocate을 하는 과정에서 page number을 vector에 저장하였다.

```
// if buddy is empty
if (isbuddyempty()){
```

```
else if (!checkExist()) {
```

memory access 과정에서 buddy가 아예 비었는지, 비어 있지 않다면 불필요한 access를 막기 위해 buddy 안에 페이지가 잘 매칭되어 있는지를 검사한다.

```
(!checkExist()) {
index = findlocation(RP->set[RP->job_index][1]);
if (index != -1) {
```

findlocation 함수를 통해 해당 aid가 들어갈 만한 충분한 공간을 buddy가 가지고 있는지를 확인하고 index가 -1일 경우 페이지 교체를 진행한다.

1_ FIFO

```
// first in
sw_out = frame_load[0];
```

frame에 lodaing 된 순서대로 frame_load 벡터에 저장하였고, 그 첫 번째 aid를 sw_out으로 복사하여 swapping을 구현하였다.

```
for (int i=0; i<buddy_size; i++) {
        if (buddy[i] == sw_out) {
                buddy[i] = -1;
        }
}
```

```
page_fault++;
start = findlocation(RP->set[RP->job_index][1]);
for (int i=start; i<start + U + 1; i++) {
        buddy[i] = RP->set[RP->job_index][1];                                   // swap in
}
frame_load.push_back(RP->set[RP->job_index][1]);
for (int i=0; i<proc_num; i++) {
        for (int j=0; j<pg_size; j++) {
                if (proc[i]->pg_aid[j] == sw_out) proc[i]->pg_valid[j] = 0;     // valid bit control
        }
}
```

페이지를 교체하는 과정이므로 page_fault값을 증가시키고, buddy에서 해당 aid를 지우고 valid bit를 0으로 초기화하면서 access하고자 하는 값을 buddy에 옮긴다.

-1은 비어있다는 뜻이다.

2_ LRU

3_ LRU_SAMPLED

4_ LFU

5_ MFU

6_ OPTIMAL

구현하지 못했다.

4. 결과

※ 결과는 마지막 사이클의 결과가 보이는 장면을 첨부하였다.

[input 1]

```
3       2048    1024    32
1       sjfProc1
8       sjfProc2
11      sjfProc3
```

```
13
3       0
3       0
3       0
3       0
3       0
3       0
3       0
6       2
6       4
3       0
3       0
7       2
7       4
```

```
15
3       0
3       0
3       0
3       0
3       0
3       0
3       0
3       0
3       0
3       0
3       0
4       1
3       0
3       0
```

```
16
3       0
3       0
3       0
3       0
6       2
3       0
3       0
3       0
6       4
3       0
3       0
3       0
3       0
7       4
7       2
3       0
```

```
[42 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 13(op 4, arg 1)
RunQueue: Empty
SleepList: 2(sjfProc3)
IOWait List: Empty

[43 Cycle] Scheduled Process: 2 sjfProc3
Running Process: Process#2 running code sjfProc3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<FCFS 결과>

```
[47 Cycle] Scheduled Process: 1 sjfProc2
Running Process: Process#1 running code sjfProc2 line 15(op 7, arg 2)
RunQueue: 2(sjfProc3)
SleepList: Empty
IOWait List: Empty

[48 Cycle] Scheduled Process: None
Running Process: Process#1 running code sjfProc2 line 16(op 3, arg 0)
RunQueue: 2(sjfProc3)
SleepList: Empty
IOWait List: Empty

[49 Cycle] Scheduled Process: 2 sjfProc3
Running Process: Process#2 running code sjfProc3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[50 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<RR 결과>

```
[42 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 13(op 4, arg 1)
RunQueue: Empty
SleepList: 2(sjfProc3)
IOWait List: Empty

[43 Cycle] Scheduled Process: 2 sjfProc3
Running Process: Process#2 running code sjfProc3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<SJF－simple 결과>

```
[41 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 12(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[42 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 13(op 4, arg 1)
RunQueue: Empty
SleepList: 2(sjfProc3)
IOWait List: Empty

[43 Cycle] Scheduled Process: 2 sjfProc3
Running Process: Process#2 running code sjfProc3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#2 running code sjfProc3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<SJF－exponential 결과>

[input 2]

```
3       2048    1024    32
1       program1
6       program2
15      program3
```

```
13
3       0
3       0
3       0
3       0
3       0
3       0
4       3
3       0
3       0
3       0
4       2
3       0
3       0
```

```
15
3       0
3       0
3       0
3       0
3       0
3       0
4       7
3       0
3       0
3       0
3       0
6       1
7       1
3       0
3       0
```

```
16
3       0
3       0
3       0
3       0
4       2
3       0
3       0
3       0
4       2
3       0
3       0
4       1
3       0
3       0
3       0
3       0
```

```
[43 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[45 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 16(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<FCFS 결과>

```
[43 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 14(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 15(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[45 Cycle] Scheduled Process: None
Running Process: Process#2 running code program3 line 16(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<RR 결과>

```
[42 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 15(op 3, arg 0)
RunQueue: 0(program1)
SleepList: Empty
IOWait List: Empty

[43 Cycle] Scheduled Process: 0 program1
Running Process: Process#0 running code program1 line 12(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#0 running code program1 line 13(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<SJF-simple 결과>

```
[42 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 15(op 3, arg 0)
RunQueue: 0(program1)
SleepList: Empty
IOWait List: Empty

[43 Cycle] Scheduled Process: 0 program1
Running Process: Process#0 running code program1 line 12(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty

[44 Cycle] Scheduled Process: None
Running Process: Process#0 running code program1 line 13(op 3, arg 0)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<SJF-exponential 결과>

[input 3]

```
5 2048 1024 32
1 program1
6 program2
8 INPUT 0
10 program3
14 INPUT 1
```

```
11
0 16
0 12
0 22
0 14
5 0
1 2
1 2
1 3
1 4
1 4
1 2
```

```
20
3 0
3 0
5 0
0 11
0 9
0 20
0 10
0 14
1 5
1 6
1 5
3 0
6 15
3 0
3 0
7 15
1 6
1 8
4 2
2 8
```

```
1
3 0
```

```
[31 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 19(op 4, arg 2)
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[32 Cycle] Scheduled Process: None
Running Process: None
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[33 Cycle] Scheduled Process: 1 program2
Running Process: Process#1 running code program2 line 20(op 2, arg 8)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<FCFS 결과>

```
[31 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 19(op 4, arg 2)
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[32 Cycle] Scheduled Process: None
Running Process: None
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[33 Cycle] Scheduled Process: 1 program2
Running Process: Process#1 running code program2 line 20(op 2, arg 8)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<RR 결과>

```
[30 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 19(op 4, arg 2)
RunQueue: 2(program3)
SleepList: 1(program2)
IOWait List: Empty

[31 Cycle] Scheduled Process: 2 program3
Running Process: Process#2 running code program3 line 1(op 3, arg 0)
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[32 Cycle] Scheduled Process: 1 program2
Running Process: Process#1 running code program2 line 20(op 2, arg 8)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

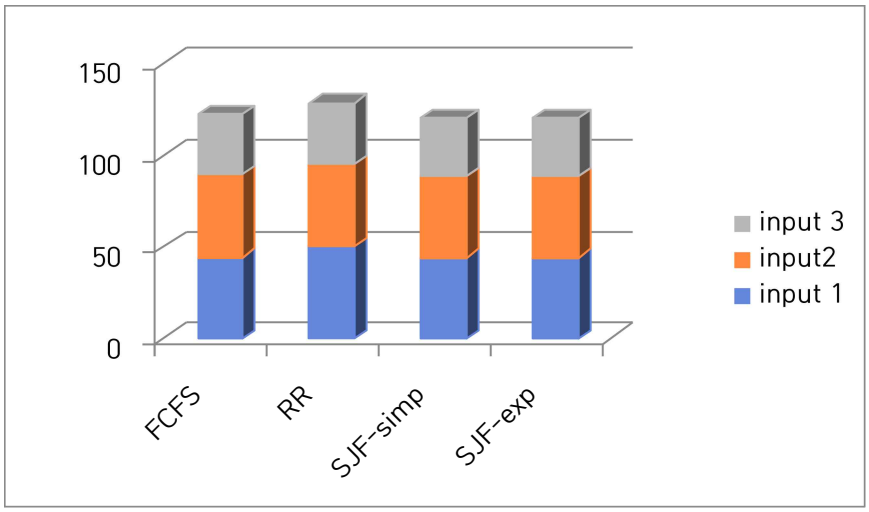
<SJF-simple 결과>

```
[30 Cycle] Scheduled Process: None
Running Process: Process#1 running code program2 line 19(op 4, arg 2)
RunQueue: 2(program3)
SleepList: 1(program2)
IOWait List: Empty

[31 Cycle] Scheduled Process: 2 program3
Running Process: Process#2 running code program3 line 1(op 3, arg 0)
RunQueue: Empty
SleepList: 1(program2)
IOWait List: Empty

[32 Cycle] Scheduled Process: 1 program2
Running Process: Process#1 running code program2 line 20(op 2, arg 8)
RunQueue: Empty
SleepList: Empty
IOWait List: Empty
```

<SJF-exponential 결과>

5. 토의

Scheduling 결과값 비교



RR은 time quantum 이 정해져있는 FCFS랑 같으므로 cycle의 수가 FCFS와 비슷하거나 더 많게 나왔다. 하지만 짧게 끝나는 프로세스를 예측하여 스케줄링하는 SJF 알고리즘은 비교적 빠른 시간 내에 프로세스가 종료됨을 알 수 있었다.

6. 과제 수행 시 겪었던 어려움과 해결 방법

✓ 초기 SJF 알고리즘을 구현 시, cpu_burst time 변수와 burst_num 변수가 모두 정수 이기 때문에 int로 정의하여 S의 값이 원활하게 갱신되지 않았다. 나눗셈 연산이 들어가는 것을 확인 후 double형 변수로 바꾸었다.

<해결>

✓ Page Fault가 발생한 후 물리 메모리에 access시, 해당 page number만큼만 크기를 할당하여 결과값이 다르게 나왔다. 버디 시스템을 이해하여 해당 페이지가 차지하는 버디의 크기를 findN함수를 활용하여 구한 후 적용하였다.

<해결>

✓ 버디 시스템에 의한 메모리 할당이 정확하게 이루어지지 않아 다른 페이지 교체 알고리즘을 구현하지 못했다.