목차

그림 목차

1. INTRODUCTION

A. Assignment Purpose

MFQ 기법을 리눅스 환경에서 C언어로 구현할 수 있다.

B. Problem Statement

1) Describe what is the problem[i]

a) 4개의 ready Queue를 갖는 MFQ 스케쥴링 기법을 구현한다.

b) 우선순위는 Q0 > Q1 > Q2 > Q3이다.

c) Q0, Q1, Q2는 RR 스케쥴링이 적용되는 queue로 time slice는 각각 2, 4, 8 이다.

d) Q3은 FCFS 스케쥴링이 적용된다.

e) 만약 Q_{i+1}의 time slice 내에 Q_i에 새로운 process가 등장하여도 Q_{i+1}의 time slice는 존중되어야 한다. 예로 Q1의 time slice 내에 Q0에 새로운 process가 등장하여도 Q1의 time slice가 끝날 때까지 Q0에 있는 process 가 실행되지 않는다.

f) 해당 cycle의 cpu burst를 모두 사용하면 i/o burst를 채우기 위해 sleep 상태가 된다.

g) 해당 cycle의 cpu burst를 모두 채우지 못한 채 time slice가 다 차면 preempt된다.

h) preempt된 process는 Q_{i+1}로 옮겨진다. 만약 현재 Q가 Q3이었다면 Q3으로 옮겨진다.

i) 해당 cycle의 cpu burst가 time slice 내에 모두 채워졌을 때 이 cycle이 마지막 cycle이었으면 finish, 아니라면 sleep(마지막 cycle이 아니라면 후에 i/o burst가 있으므로)상태가 된다.

j) i/o burst를 모두 채운 process는 Q_{i-1}로 wake up 된다. 만약 현재 Q가 Q0 이었다면 Q0으로 옮겨진다.

k) process의 pid, arrival time, number of cycle, cpu – i/o burst, initial Q와 총 process 개수는 input 파일에서 주어진다.

l) Gantt chart, process당 waiting time, turnaround time, 전체 process의 평균 waiting time, turnaround time를 실행결과로 보인다.

2) Describe how you solve the problem

a) 4개의 ready Q를 자료구조 QUEUE의 형태로 구현한다.
b) context switching 시간은 0으로 간주한다.
c) time이라는 변수로 simulation내 시간을 나타낸다.
d) input 파일의 정보를 PROCESS 라는 struct에 저장한다. 저장된 PROCESS type의 struct는 plist라는 pointer array에 저장한다.
e) plist의 process들을 arrival time을 기준으로 오름차순 정렬한다.
f) 만약 arrival time이 되면 해당 process를 알맞은 ready Q에 삽입한다.
g) 만약 wake up time과 arrival time이 같은 process가 있다면 arrive한 process가 먼저 queue에 들어간다.
h) 만약 wake up time이 되면(i/o burst가 다 차면) 해당 process를 알맞은 다음 Q로 삽입한다.
i) 해당 시간에 실행할 process가 있는 ready Q를 찾는다.
j) 찾은 Q에 있는 process의 cpu burst time을 1씩 줄이고 runtime은 1씩 증가시킨다.
k) Q0, Q1, Q2의 경우 만약 runtime이 time slice와 같아지면 남아있는 cpu burst time에 따라 preempt(burst time이 남아있음), finish(burst time이 남아있지 않고 마지막 cpu burst였음), sleep(burst time이 남아있지 않고 마지막 cycle이 아님)중 상태를 결정한다.
l) Q0, Q1, Q2의 경우 만약 runtime이 time slice보다 작은데 cpu burst time이 남아있지 않다면 finish(burst time이 남아있지 않고 마지막 cpu burst였음), sleep(burst time이 남아있지 않고 마지막 cycle이 아님)중 상태를 결정한다.
m) Q3의 경우 cpu burst time이 남아있지 않을 때까지 cpu burst time을 줄인다. finish(burst time이 남아있지 않고 마지막 cpu burst였음), sleep(burst time이 남아있지 않고 마지막 cycle이 아님)중 상태를 결정한다.
n) finish일 때는 해당 시간을 process struct에 저장한다.
o) 결정된 상태에 따라 다음 진입할 Q(Q0, Q1, Q2, Q3, sleep Q 중 1.B.1에서 명시된 규칙에 따라)를 선택한다.
p) 상태가 finish인 process의 개수를 매시간 확인한다.
q) 만약 finish 상태 process의 개수가 전체 process의 개수보다 작고 모든 Q가 비어 있을 때는 아직 arrival time이 되지 않은 Q를 기다리거나 wakeup

을 기다리고 있는 것이기 때문에 cpu run을 하지 않고 time을 증가시킨다.

r) 만약 finish 상태 process의 개수가 전체 process의 개수와 같고 모든 Q가 비어 있으면 모든 process에 대한 스케쥴링이 끝난 것으로 간주한다.

s) 스케쥴링이 끝나면 process의 waiting time과 turnaround time을 계산한다. turnaround time은 finish time – arrival time으로 waiting time은 turnaround time – (cpu burst time + i/o burst time)으로 계산한다.

t) gantt chart는 txt 파일의 형식으로 출력한다.

u) process 별 정보(pid, arrival time, total cycle number, finish time, waiting time, rest burst time – 이상적으로 0이어야 함, turnaround time, 마지막 Q)를 출력한다.

v) 평균 turnaround time, waiting time을 계산해 출력한다.

2. BODY

A. Implementation

1) typedef

a) enum STATE

define enum of the process state

PREEMPT = 0

WAKEUP = 1

SLEEP = 2

FINISH = 3

RUN = 4

b) struct PROCESS

int pid // pid of a process

int at // arrival time

int ft // finish time (initialized as -1)

int Q // current Q (initialized as initQ)

int numCycle // total no. of the burst cycle

int *bc // pointer array to store burst time

int cpu_burst // total cpu_burst time

int io_burst // total i/o burst time

int wt // waiting time (initialized as 0)

int tt // turnaround time (initialized as)

int curCycle // current burst cycle (initialized as 1)

c) struct NODE

PROCESS *data // data of the linked list

struct __node *next // next node's address (last node points NULL)

d) struct QUEUE

NODE *front // front of the queue

NODE *rear // back of the queue

2) main

a) in this function

make Q0, Q1, Q2, Q3, readyQ(where process that sleeps wait) and initialize count time

count no. of finished process

get information from input file

check arrival and wake up

find next Q to run

print the element of the Qs

if all process is scheduled, calculate waiting time, turnaround time

b) output

output file "gantt_chart.txt"

print the element of the queue for each time

print the information of processes

if the function end properly, return int 0

c) precondition

none

3) findstate

a) in this function

find the state of the process in cpu (preempt, sleep, finish)

push to the next queue

b) output

return STATE state

if state is FINISH, store finish time to the PROCESS runP (since the whole time in the program increase after this function, store 'time + 1')

c) precondition

runtime should be returned from runCPU

number of the queue should be given

4) findnxtQ

a) in this function

find next Q to run by using empty information and the value of busy

busy 0: process in Q0 is running

1: process in Q1 is running

2: process in Q2 is running

3: process in Q3 is running

-1: time slice is over or previously running process sleeps or finishes

when busy is i, only process in Qi can use cpu

-1, all process can use cpu

therefore, return the number of queue whose previous queue is empty and itself is not empty & busy is -1 or the number of queue.

b) output

return Q number

if no Q to be run, return -1 (when all queue is empty or cpu cannot run other process)

c) precondition

busy should be set before this function (in main(), nxtQ(), wakeup(), runCPU())

5) runCPU

a) in this function

find process to be run

set busy

increase runtime, decrease cpu burst time

if running queue is changed or running process is changed, runtime becomes 0

print gantt chart to outfile

b) output

value of busy

print gantt chart to output file

return runtime

c) precondition

the number of the next queue to run should be given

6) calculatewaiting

a) in this function

calculate waiting time, turnaround time of the process

calculate average waiting time, turnaround time of whole process

turnaround time = finish time – arrival time

waiting time = turnaround time – (cpu burst time + i/o burst time)

b) output

store the result in each PROCESS

store the average time in variable avg_wt, avg_tt

c) precondition

none

7) wakeup

a) in this function

if there is process that wakes up, return int 1

if readyQ is not empty, decrease I/O burst rest time 1

if rest time == 0, put the process to proper next Q and increase current cycle no.

b) output

if there is process that wakes up, return 1 (else 0), delete the process from readyQ, push the process to proper next Q

c) precondition

none

8) nxtQ

a) in this function

find next Q

delete the process from current Q

push to the next Q

set busy

if state is PREEMPT, SLEEP, set busy -1 (let all Q can use cpu) else, hold

b) output

change busy

delete Q from previous Q and push them to proper next Q

c) precondition

none

9) delCurNode

a) in this function

delete the process from readyQ whose i/0 burst = 0

b) output

return deleted process address

delete Q from readyQ

c) precondition

pid of process to be deleted should be given

10) enterQ

a) in this function

when arrival time, push the process to initial Q

b) output

push the process in proper Q

change current Q number in PROCESS

c) precondition

plist should be sorted in ascending order with arrival time

11) sortQ

a) in this function

sort process in the plist with arrival time (ascending order)

using bubble sort

b) output

sorted plist

c) precondition

total number of process should be given

12) printQ

a) in this function

print information of all element int the plist

b) output
print the information

c) precondition
none

13) getP

a) in this function
get information from the input file
calculate total cpu burst time, i/o burst time of each process
initialize value of process

b) output
return pointer array of process address PROCESS **plist

c) precondition
input file form
first line: total number of process
from second line,
pid, arrival time, initial Q to enter, number of cycle, cpu-i/o burst time

14) closeP

a) in this function
free dynamic memory used in plist

b) output
none

c) precondition
none

15) Qinit

a) in this function
initialize QUEUE queue by making front and rear point NULL

b) output
front and rear point NULL

c) precondition
none

16) isQempty

a) in this function
if front of the queue is NULL (means empty), return TRUE, else FALSE

b) output
if Q is empty, return TRUE, else FALSE

c) precondition
none

17) EnQ

a) in this function
perform push operation to Q

b) output
add PROCESS x to rear of the Q

c) precondition
none

18) DelQ

a) in this function
perform delete operation to Q
free dynamic memory of the NODE

b) output
return deleted PROCESS

c) precondition
Q to be deleted is not empty

19) Qprint

a) in this function
print the pid of processes in the queue

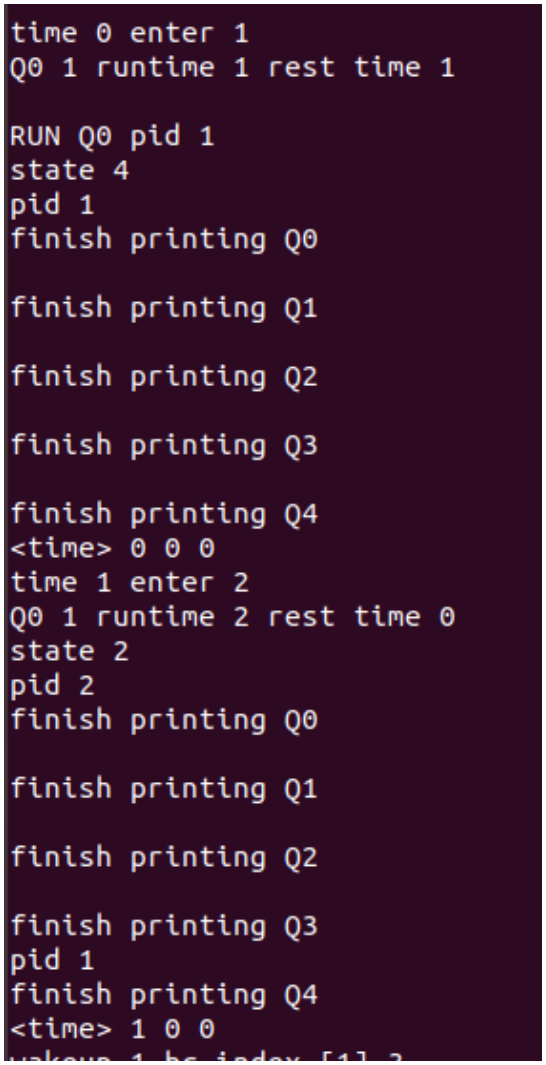
b) output
print the pid

c) precondition

none

B. Result

1) Console output (print)

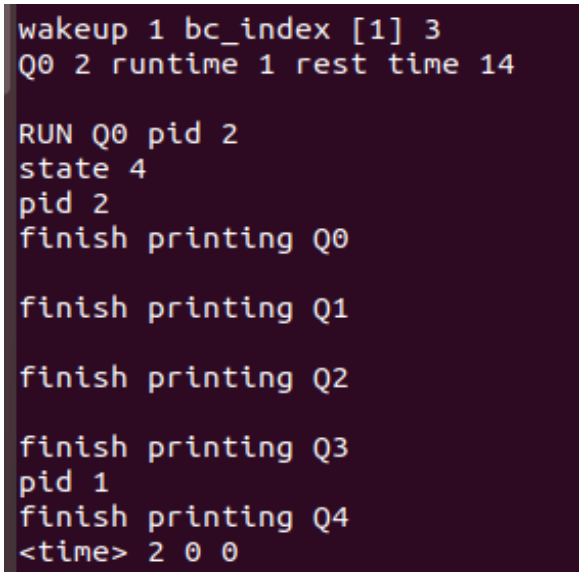


← arrival time이 되어 enter한 process
(현재 시간, enter pid)

← 이 time에 run한 process의 상태
| ← 현재 Queue 내 process의 pid
| readyQ(i/o burst를 기다리는 Q)는 Q4로
| 표현
|
| ← Q가 비어 있다면 pid가 출력되지 않음
|
|
|
← 현재 시간

← 현재 run 중인 process가 가장 최근에 있었던 Queue, run 중인 process의 pid, 이번 cycle의 runtime, 이번 cpu cycle의 남은 burst time

그림 1 매시간 enter process, running process, Q의 element 표시



← 현재 sleep 중인 process의 pid, burst cycle[index]의 남은 i/o burst time

그림 2 sleep 중인 process의 남은 i/o burst time 표시

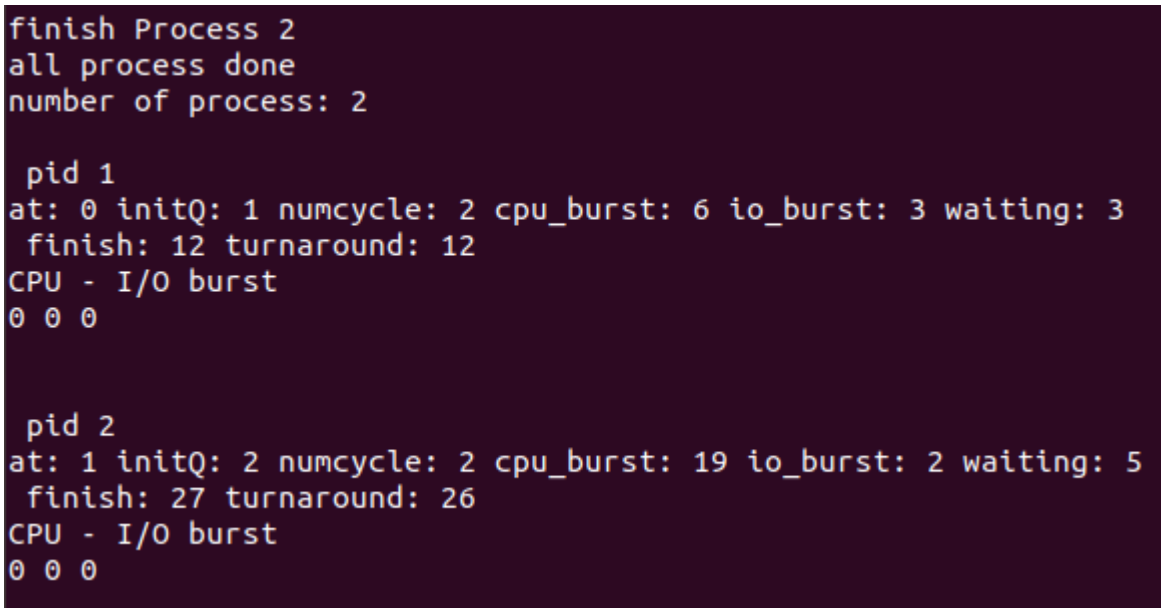


그림 3 모든 process의 스케쥴링을 마친 후

모든 process의 스케쥴링을 마친 후 최종 process의 개수와 process의 information을 출력한다.

CPU – I/O burst 는 스케쥴링 중 감소시켰으므로 스케쥴링이 적절히 끝났다면 0인 것이 이상적이다.

2) file output

```
2
1, 0, 0, 2,  2 3 4
2, 1, 0, 2, 15 2 4
```

그림 4 input file

```
----------
|   1   |0
|   1   |1
----------
|   2   |2
|   2   |3
----------
|   2   |4
|   2   |5
|   2   |6
|   2   |7
----------
|   1   |8
|   1   |9
----------
|   1   |10
|   1   |11
----------
|   2   |12
|   2   |13
|   2   |14
|   2   |15
|   2   |16
|   2   |17
|   2   |18
|   2   |19
----------
|   2   |20
|  wait  |21
|  wait  |22
----------
|   2   |23
|   2   |24
|   2   |25
|   2   |26
average waiting time: 4.00
average turnaround time: 19.00
```

그림 5 gantt chart in output file

Gantt chart 에는 현재 실행중인 process의 pid와 실행을 시작한 시간을 옆에 표시한다. (그림 4의 첫 줄에서 | 1 |0은 0과 1시간 단위 사이에 pid가 1인 process를 run했다는 것을 의미한다) 만약 실행 중인 process가 바뀌거나 queue가 바뀌면 -------로 표현한다. wait은 sleep 중인 process가 wake up하길 대기 중이거나 arrival time이 아직 되지 않은 process를 기다리는 중이라는 뜻이다. 끝에 평균 wait time, turnaround time을 표시한다.

3) 다양한 input file 처리

다양한 상황의 input을 테스트할 수 있도록 input파일을 경우별로 나눠보았다. 처음 프로그램을 실행하면 input.txt가 실행되고 다른 input파일을 실행하고 싶다면 프로그램 코드 내 main()에서 input파일의 이름을 변경하면 된다.

input rule

i. 첫 줄에는 process의 개수를 기재한다
ii. 두 번째 줄부터 pid, arrival time, initial Q, total burst cycle, cpu-i/o burst 순으로 작성한다.
iii. burst time은 , 없이 띄어쓰기로만 구별한다.
iv. 마지막 cycle에는 cpu burst 만 존재한다.
v. arrival time순으로 작성할 필요는 없다.
vi. initial Q가 0일 필요는 없다. 이론적으로 첫 process는 Q0으로 진입하는 것이 맞으나 구현 상 그렇지 않아도 작동되도록 설계하였다.
vii. arrival time은 겹칠 수 없다.
viii. input file의 형식에 대한 error 처리를 하지 않았기 때문에 적절히 작성하는 것이 중요하다.

i. 모두 Q0으로 진입, arrival time이 오름차순으로 정리되지 않음(input.txt)

```
3
1, 0, 0, 5,  2 3 4 3 8 3 2 3 2
3, 2, 0, 2, 2 2 2
2, 1, 0, 4, 15 2 4 3 7 3 5
```

그림 6 arrival time이 순서대로가 아닌 input file

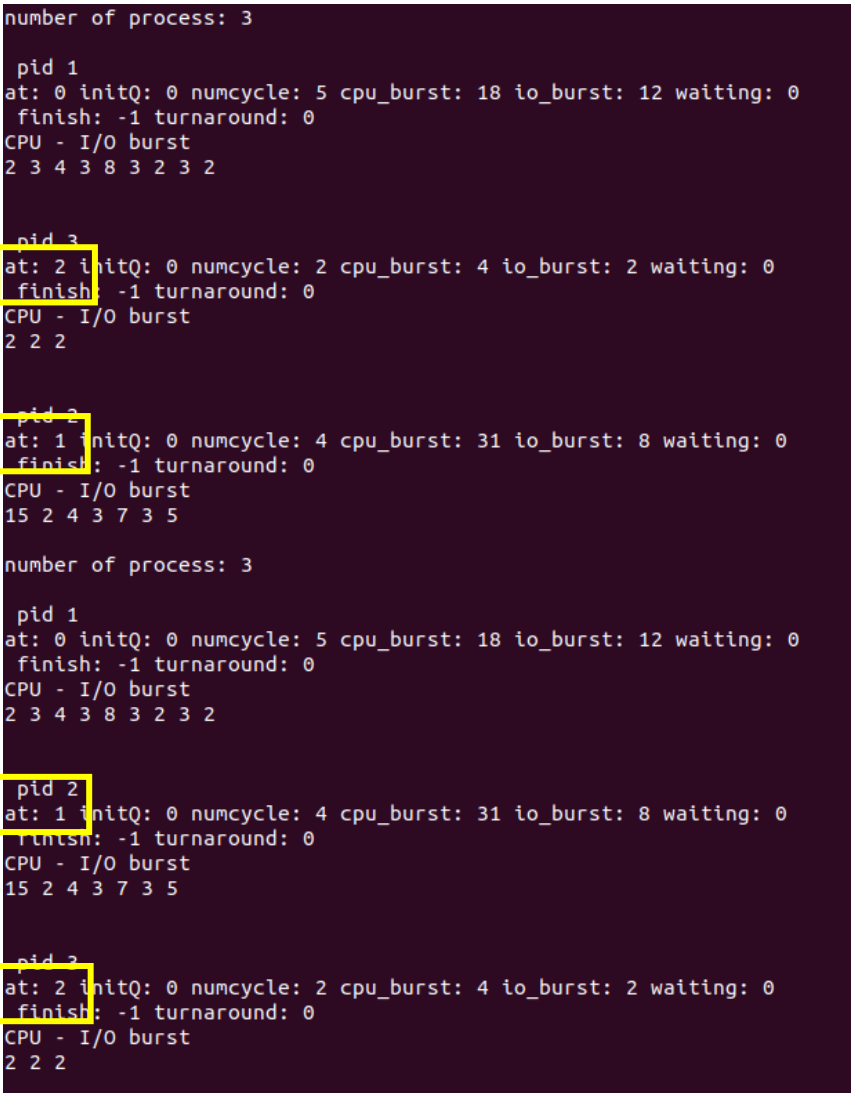


그림 7 run을 하기 전 process를 arrival time기준으로 오름차순 정렬

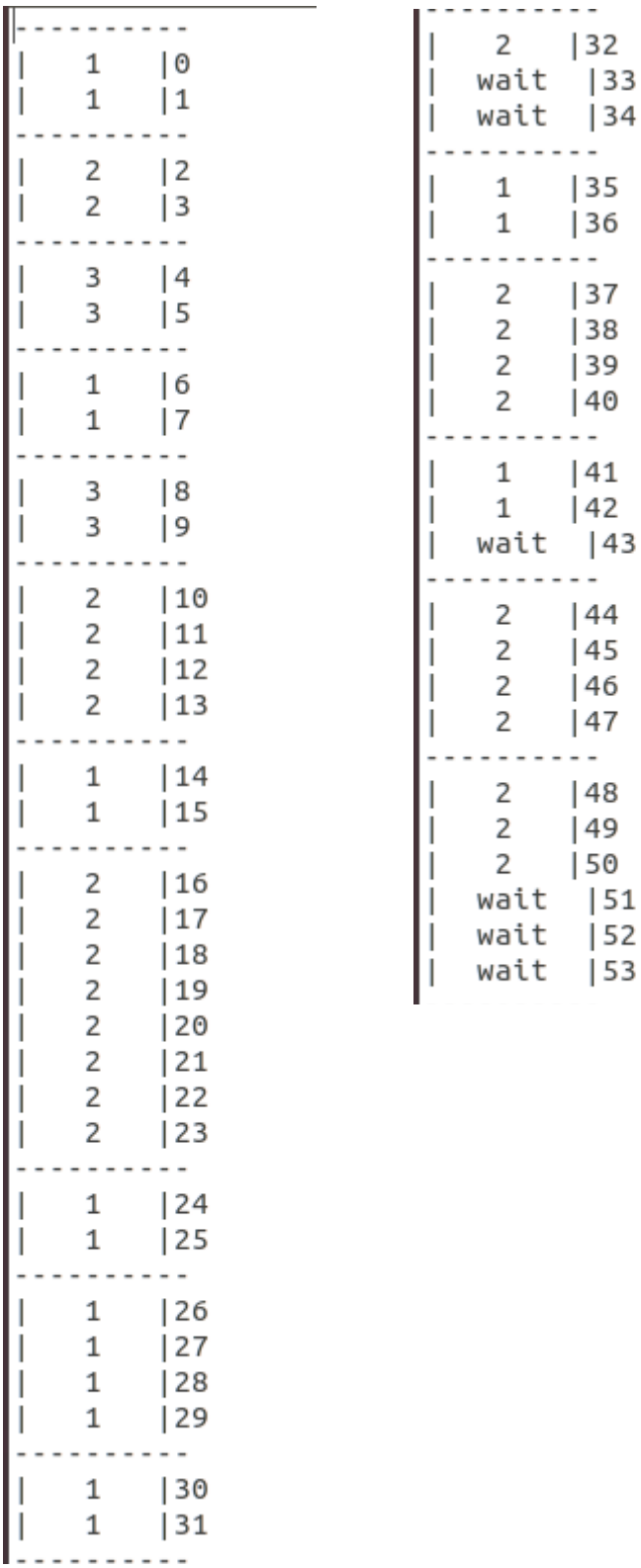


그림 8 그림 6 output

ii. 모두 Q0으로 진입, I/O를 하지 않음 (input2.txt)

```
5
1, 0, 0, 1, 3
2, 1, 0, 1, 7
3, 3, 0, 1, 2
4, 5, 0, 1, 5
5, 6, 0, 1, 3
```

그림 9 모두 Q0으로 진입, i/o를 하지 않음

```
finish Process 5
all process done
number of process: 5

 pid 1
at: 0 initQ: 1 numcycle: 1 cpu_burst: 3 io_burst: 0 waiting: 8
 finish: 11 turnaround: 11
CPU - I/O burst
0


 pid 2
at: 1 initQ: 2 numcycle: 1 cpu_burst: 7 io_burst: 0 waiting: 12
 finish: 20 turnaround: 19
CPU - I/O burst
0


 pid 3
at: 3 initQ: 0 numcycle: 1 cpu_burst: 2 io_burst: 0 waiting: 1
 finish: 6 turnaround: 3
CPU - I/O burst
0


 pid 4
at: 5 initQ: 1 numcycle: 1 cpu_burst: 5 io_burst: 0 waiting: 8
 finish: 18 turnaround: 13
CPU - I/O burst
0


 pid 5
at: 6 initQ: 1 numcycle: 1 cpu_burst: 3 io_burst: 0 waiting: 10
 finish: 19 turnaround: 13
CPU - I/O burst
0
```

그림 10 그림 9의 결과

```
----------
|   1   |0
|   1   |1
----------
|   2   |2
|   2   |3
----------
|   3   |4
|   3   |5
----------
|   4   |6
|   4   |7
----------
|   5   |8
|   5   |9
----------
|   1   |10
----------
|   2   |11
|   2   |12
|   2   |13
|   2   |14
----------
|   4   |15
|   4   |16
|   4   |17
----------
|   5   |18
----------
|   2   |19
average waiting time: 7.80
average turnaround time: 11.80
```

그림 11 그림 9의 Gantt chart

iii. Q0, Q1로 진입, FCFS의 작동 확인 (input3.txt)

```
2
1, 0, 0, 1,  40
2, 1, 1, 1, 30
```

그림 12 FCFS 작동 확인을 위한 긴 CPU burst

```
finish Process 2
all process done
number of process: 2

 pid 1
at: 0 initQ: 3 numcycle: 1 cpu_burst: 40 io_burst: 0 waiting: 29
 finish: 69 turnaround: 69
CPU - I/O burst
0


 pid 2
at: 1 initQ: 3 numcycle: 1 cpu_burst: 30 io_burst: 0 waiting: 12
 finish: 43 turnaround: 42
CPU - I/O burst
0
```

그림 13 그림 12의 결과

```
----------
|    1    |0
|    1    |1
----------
|    2    |2
|    2    |3
|    2    |4
|    2    |5
----------
|    1    |6
|    1    |7
|    1    |8
|    1    |9
----------
|    2    |10
|    2    |11
|    2    |12
|    2    |13
|    2    |14
|    2    |15
|    2    |16
|    2    |17
----------
|    1    |18
|    1    |19
|    1    |20
|    1    |21
|    1    |22
|    1    |23
|    1    |24
|    1    |25
----------
|    2    |26
|    2    |27
|    2    |28
|    2    |29
|    2    |30
|    2    |31
|    2    |32
|    2    |33
|    2    |34
|    2    |35
|    2    |36
|    2    |37
|    2    |38
|    2    |39
|    2    |40
|    2    |41
|    2    |42
|    2    |43
----------
|    1    |44
|    1    |45
|    1    |46
|    1    |47
|    1    |48
|    1    |49
|    1    |50
|    1    |51
|    1    |52
|    1    |53
|    1    |54
|    1    |55
|    1    |56
|    1    |57
|    1    |58
|    1    |59
|    1    |60
|    1    |61
|    1    |62
|    1    |63
|    1    |64
|    1    |65
|    1    |66
|    1    |67
|    1    |68
|    1    |69
average waiting time: 20.50
average turnaround time: 55.50
```

그림 14 그림 12의 Gantt chart

iv. 첫 진입 Q가 Q0이 아님, 이전 process의 스케쥴링이 모두 끝난 후 등장하는 process 처리 (input4.txt)

```
2
1, 0, 1, 2,  2 2 2
2, 7, 0, 2, 2 2 2
```

그림 15 이전 process의 스케쥴링이 끝난 후 arrival time이 있는 process 처리

```
finish Process 2
all process done
number of process: 2

 pid 1
at: 0 initQ: 0 numcycle: 2 cpu_burst: 4 io_burst: 2 waiting: 0
 finish: 6 turnaround: 6
CPU - I/O burst
0 0 0


 pid 2
at: 7 initQ: 0 numcycle: 2 cpu_burst: 4 io_burst: 2 waiting: 0
 finish: 13 turnaround: 6
CPU - I/O burst
0 0 0
```

그림 16 그림 15 결과

```
----------
|   1   |0
|   1   |1
|  wait  |2
|  wait  |3
----------
|   1   |4
|   1   |5
|  wait  |6
----------
|   2   |7
|   2   |8
|  wait  |9
|  wait  |10
----------
|   2   |11
|   2   |12
average waiting time: 0.00
average turnaround time: 6.00
```

그림 17 그림 15 Gantt chart

v. Q3까지 사용하여 스케쥴링(input5.txt)

```
3
1, 0, 0, 5,  2 3 4 3 8 3 2 3 2
3, 2, 0, 2, 2 2 2
2, 1, 0, 4, 30 2 4 3 7 3 5
```

그림 18 Q3 FCFS까지 사용하는 스케쥴링

```
finish Process 3
all process done
number of process: 3

 pid 1
at: 0 initQ: 0 numcycle: 5 cpu_burst: 18 io_burst: 12 waiting: 26
 finish: 56 turnaround: 56
CPU - I/O burst
0 0 0 0 0 0 0 0 0


 pid 2
at: 1 initQ: 3 numcycle: 4 cpu_burst: 46 io_burst: 8 waiting: 17
 finish: 72 turnaround: 71
CPU - I/O burst
0 0 0 0 0 0 0


 pid 3
at: 2 initQ: 0 numcycle: 2 cpu_burst: 4 io_burst: 2 waiting: 2
 finish: 10 turnaround: 8
CPU - I/O burst
0 0 0
```

그림 19 그림 18의 결과

```
wakeup 2 bc_index [1] 1
Q1 1 runtime 2 rest time 0
state 2

finish printing Q0

finish printing Q1

finish printing Q2

finish printing Q3
pid 2 pid 1
finish printing Q4
<time> 49 0 1
wakeup 2 bc_index [1] 0
Q3 2 runtime 1 rest time 3

RUN Q3 pid 2
state 4

finish printing Q0

finish printing Q1

finish printing Q2
pid 2
finish printing Q3
pid 1
finish printing Q4
<time> 50 1 1
```

그림 20 Q3에서 sleep으로 간 pid2, wake up 시 Q3에 배정

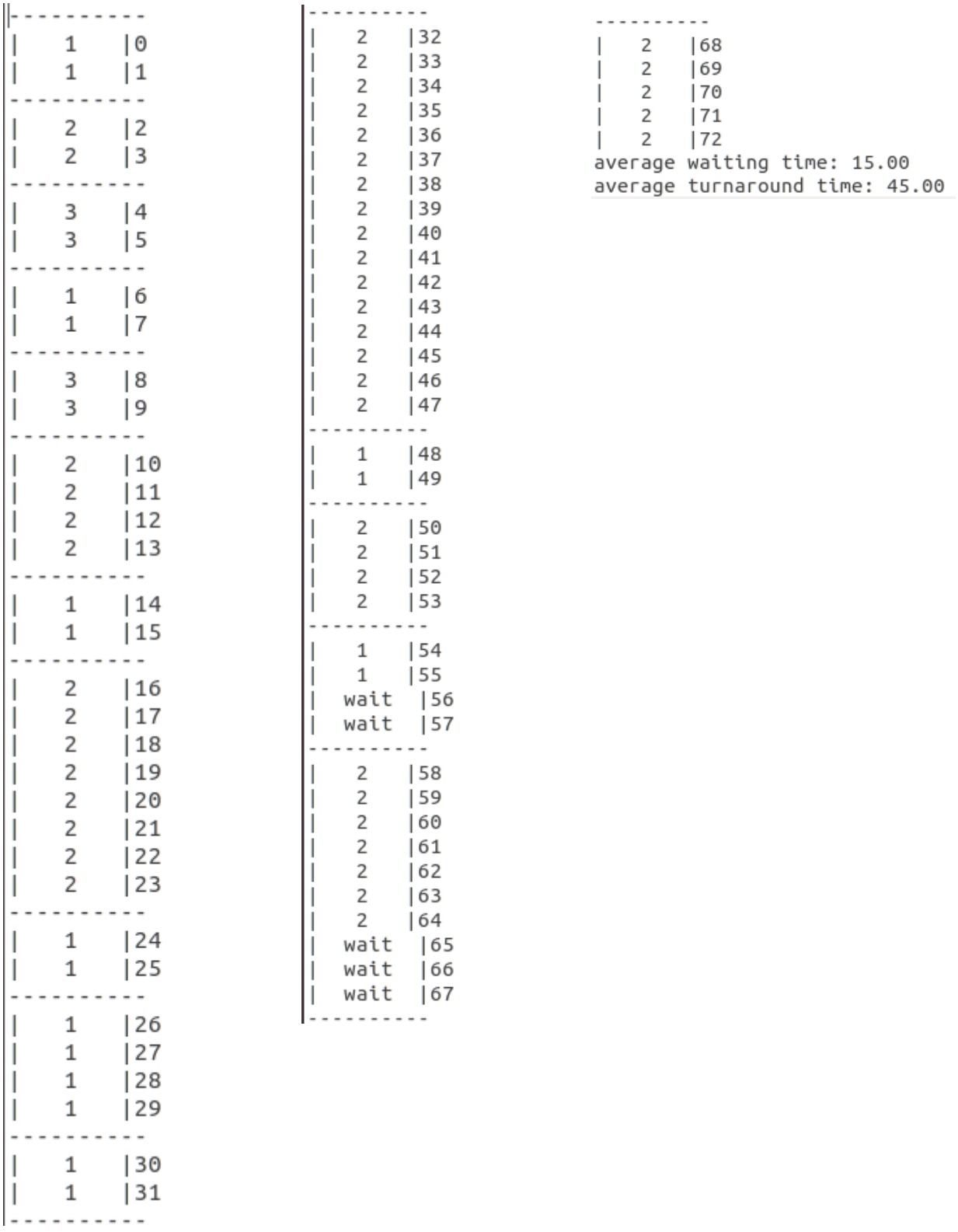


그림 21 그림 18의 Gantt Chart

그림 20에서 Q3에서 cpu burst time을 모두 채운 후 sleep 상태가 되었던 pid 2 process가 다시 wake up 할 때에도 Q3로 배정받는 것을 확인할 수 있다.

3. CONCLUSION

MFQ 기법의 rule에 따라 process가 진입할 다음 queue를 선택하고 time slice가 다 되면 preempt, burst time이 다 되면 sleep, finish 혹은 wake up하도록 case를 설계하였다.

runtime을 증가시키고 나서 프로그램 전체의 time을 증가시키기 때문에 이 관계를 정리하는 것에서 약간의 어려움이 있었다. 초반에는 RR과 FCFS를 실행하는 함수를 따로 작성하여 그 내에서 runtime과 cpu burst time을 조정했는데 이렇게 하니 프로그램 전체의 time과 연동시키기가 어려웠다. (특별히 preempt, sleep, finish 상태일 때는 이 일만 진행하고 다음 process를 run하지 않아 context switching 시간이 발생한 것처럼 구현되었다. 하지만 이는 의도한 것이 아니다.) 후에 문제를 해결하기 위해 다음에 실행할 Queue를 찾는 것, runtime과 cpu burst time을 조정하는 것, run 후 process의 상태를 결정하는 것, 다음 queue에 push하는 것을 각각 다른 함수로 나눠 구현하니 이러한 일이 발생하지 않았다. (이전 방법의 경우 RR의 output이 state였는데 이 때 PREEMPT, SLEEP, FINISH, RUN 중 하나만 선택할 수 있었고 이에 따라 runtime과 cpu burst를 조정하였다. 따라서 같은 시간 내에 PREEMPT와 다음 process RUN을 동시에 할 수 없었다.)

cpu가 낮은 우선순위의 queue에 있었던 process를 실행하는 동안 높은 우선순위의 queue에 process가 진입하면 preempt를 하지 않고 정해진 time slice 동안은 run하도록 설계하는 것에서 어려움을 겪었다. busy라는 개념을 통해 만약 busy가 -1이면 다른 queue의 process의 time slice내에 있지 않다는 뜻이므로 우선순위에 따라 process를 run하고 1, 2, 3 이라면 해당 queue에서 온 process가 time slice내에 실행 중이므로 다른 queue에서 온 process가 run되지 못하도록 설계하는 것으로 문제를 해결할 수 있었다.

input case로 다룬 것은

i. 모두 Q0으로 진입, arrival time이 오름차순으로 정리되지 않음
ii. 모두 Q0으로 진입, I/O를 하지 않음
iii. Q0, Q1로 진입, FCFS의 작동 확인
iv. 첫 진입 Q가 Q0이 아님, 이전 process의 스케쥴링이 모두 끝난 후 등장하는 process 처리
v. Q3까지 사용하여 스케쥴링

이다. 작은 input으로 구현했으나 static size array가 존재하지 않으므로 규모가 크거나 burst cycle이 많은 case에 대해서도 무리 없이 동작할 것이라

고 예상된다. 위의 5가지 case를 통해 이번 과제에서 고려하여 설계하였던 요소를 모두 포함하였기 때문에 다양한 input에 대한 test가 적절히 이뤄졌다 할 수 있다.

4. REFEFENCE

---

[i] 엄영익, "운영체제 OS-2020-2-Project-1(MFQ Scheduling Simulation)-안내서-2", 성균관대학교, 2020.09.27