운영체제 (SWE3004-43)

Project 1

[MFQ Scheduling Simulation]

2016313923 한상욱

1. 설계/구현에 사용한 도구 및 입, 출력 형식

A. 사용한 언어 : C++

i. 오직 신입생 때만 C언어를 사용해 보았으며, 그 이후로는 C++을 사용하였습니다. 군대에서도 C++을 계속 사용하였고, 군복학 후 처음 듣는 수업인지라 C++을 사용하였습니다.

B. 새로 정의한 규칙

i. 어떤 프로세스가 I/O Burst를 마치고 ReadyQueue로 진입하는 일과, ReadyQueue에서 다음 실행할 프로세스를 고르는 일이 동일한 시간에 벌어진다면, 프로세스가 I/O Burst를 마치고 ReadyQueue로 진입하는 일을 먼저 처리해 주었습니다.

ii. 새롭게 ReadyQueue로 들어오는 프로세스와 CPU Burst를 마치고 다른 큐로 들어가야되는 프로세스가 시간이 같을 경우, 새롭게 ReadyQueue로 들어오는 프로세스를 먼저 처리하도록 작성하였습니다.

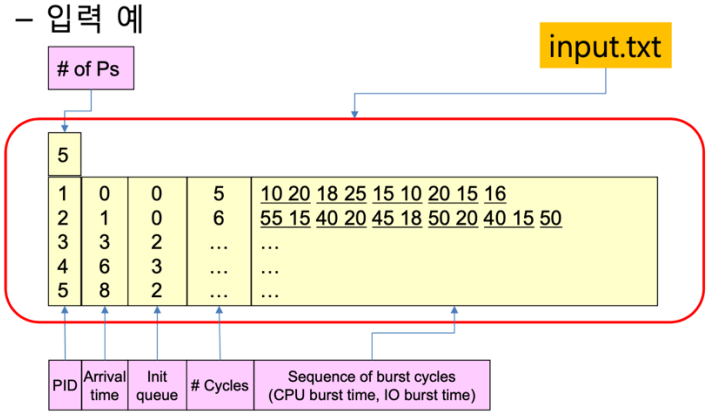
iii. 새롭게 ReadyQueue로 들어오는 프로세스와 IO Burst를 마치고 ReadyQueue로 들어오는 프로세스가 시간이 같을 경우, 새롭게 ReadyQueue로 들어오는 프로세스를 먼저 처리하도록 작성하였습니다.

iv. 입력되는 프로세스의 PID는 1부터 시작한다고 가정하였습니다.

C. 입력 값 형식

입력 값 형식은 과제 내 입력 형식과 같습니다.
Pid, Arrival time, Init queue, Number of Cycles, Sequence of burst cycles
의 형식대로 들어옵니다.

– 입력 예



Ex)
5
1 0 0 5 10 20 18 25 15 10 20 15 16
2 1 0 6 55 15 40 20 45 18 50 20 40 15 50
3 3 2 4 32 20 41 12 49 33 19
4 6 3 8 44 32 12 31 28 17 13 32 41 75 33 56 62 12 51

5 8 2 2 82 11 31

D. 출력 값 형식

```
|---P1(7)---||--P2(5)--||-P1(3)-||-P2(2)-||---------0(18)---------||---------P1(18)---------|
P1's TT : 53s WT : 5s
P2's TT : 13s WT : 5s

Average Turnaround Time : 33s
Average Waiting Time : 5s
```

출력은 Gantt 차트, 각 프로세스 별 Turnaround Time(TT) 및 Waiting Time (WT), 그리고 전체 프로세스의 평균 Turnaround Time(TT) 및 Waiting Time (WT)으로 구성되어 있습니다.

Gantt 차트는 수업 PDF 자료에 있는 그림과 같은 형태입니다. 실행되는 프로세스의 PID를 막대 가운데에 적어 두었고, CPU를 점유한 시간을 숫자로 나타내었습니다. 0은 해당시간동안 IO작업만 있었을 뿐 CPU를 점유한 프로세스가 없다는 의미입니다.

2. 설계/구현 내용 설명

A. 전역 변수 및 자료구조

```
priority_queue<process, vector<process>, compare> rq;
priority_queue<process, vector<process>, compare> MFQ[4];
priority_queue<process, vector<process>, compare> IO, IOTemp, IOReadyQ;
vector<process> exitV;
vector<processTimeStruct> processTimeVec;
vector<int> CPU;
int timeQuantum[4] = {1, 2, 4, INF};
int curTime = 1, totalProcNum = 0;
```

RQ : 모든 process들이 들어있는 우선순위 큐입니다.
Arrival Time을 기준으로 정렬되어 있습니다.

MFQ[4] : 0~2번째 까지는 Round Robin t=1, 2, 4 큐이고, 3번째는 FCFS 큐입니다.
이 또한 Arrival Time을 기준으로 정렬되어 있습니다.

IO, IOTemp, IOReadyQ : IO Burst를 위해 사용한 큐입니다.
(IO Burst는 병렬적으로 처리가 가능하기 때문에, 시간이 1초 지날 때마다 IO Burst 과정을 처리해주기 위해 만들었습니다)

exitV : terminate 된 프로세스들을 저장해두는 벡터입니다.

processTimeVec : 각 프로세스별 Turnaround Time, Waiting Time을 처리해주기 위해 만든 벡터입니다.

CPU : Gantt Chart 생성을 위해, 현재 어떤 프로세스가 CPU를 점유하고 있는지에

대해 저장해 둔 벡터입니다.

timeQuantum: 각 Ready큐 별 timeQuantum을 배열에 저장해 두었습니다. FCFS는 TimeQuantum을 무제한으로 설정해 두었습니다.

curTIme : 현재 시간입니다.

totalProcNum : 전체 프로세스의 갯수입니다.

B. 구조체

```
struct process {
    int pid, at, Q, progress, step, occup;
    vector<int> burst;
};
struct processTimeStruct{
    int pid, AT, BT, TT, WT;
};
struct compare {
    bool operator()(process A, process B) {
        if(A.at == B.at) return A.pid < B.pid;
        else return A.at > B.at; //arriving time 작은 순서
    }
};
```

Process : 프로세스의 정보에 대한 구조체입니다. Pid, Arrival Time, 현재 있어야 될 Q, 현재 Burst에 대한 진행 정도, 현재 어떤(몇 번째) Burst 진행중인지 여부, Burst 목록을 담고있습니다.

processTimeStruct : 프로세스의 pid, arrival time, Burst time, Turnaround Time, Waiting Time에 대한 정보를 계산하기 위해 만들었습니다.

Compare : arrival time을 기준으로 큐를 정렬하기 위해 만들었습니다.

C. 기본 함수

```
23   void input();
24   void printCPU();
25   void IOToReady();
26   void createToReady();
27   vector<int> parseStrToVec(string str, char ch);
```

Input함수는 파일을 읽은 후 파싱하는 과정을 거쳐 프로세스 구조체를 만들고, rq에 넣어주는 역할을 하는 함수입니다.

printCPU 함수는 간트차트를 출력해주는 함수입니다.

IOToReady 함수는 IO Burst를 마친 프로세스들을 큐로 다시 넣어주는 함수입니다.

createToReady 함수는 초기에 입력받은 프로세스들을 탐색하며, AT이 되었을 때 적적한 큐로 넣어주는 함수입니다.

parseStrToVec 함수는 파일을 읽은 후 string을 파싱해주는 함수입니다.

D. 메인 함수

```
36  int main(){
37      input();
38      totalProcNum = rq.size();
39      createToReady();
40      while(1) {
41          if(exitV.size() == totalProcNum) break;
42          while(!IO.empty()) {
43              process cur = IO.top(); IO.pop();
44              cur.progress++;
45              if(cur.progress == cur.burst[cur.step]){
46                  cur.step++;
47                  cur.progress = 0;
48                  cur.at = curTime;
49                  IOReadyQ.push(cur);
50                  continue;
51              }
52              IOTemp.push(cur);
53          }
54          IO = IOTemp;
55          while(!IOTemp.empty()) IOTemp.pop();
```

메인함수의 시작은 input 함수를 실행시켜 파일 내용들을 구조체에 맞게 저장합니다.

While 반복문을 돌면서, 먼저는 IO burst 중인 프로세스들이 있는지 점검하고, 있다면 IO Burst를 진행시킵니다.

```
for(int i=0; i<4; i++){
    if(!MFQ[i].empty()){
        process cur = MFQ[i].top(); MFQ[i].pop();
        CPU.push_back(cur.pid); flag = true;
        cur.progress++;
        cur.occup++;
        if(cur.progress == cur.burst[cur.step]){
            createToReady();
            IOToReady();
            cur.at = curTime;
            cur.step++;
            cur.occup = 0;
            cur.progress = 0;
            if(cur.Q == 3) cur.Q = 3;
            else if (cur.Q == 0) cur.Q = 0;
            else cur.Q = cur.Q - 1;
            if(cur.step == cur.burst.size()) {
                for(int j=0; j<processTimeVec.size(); j++){
                    if(processTimeVec[j].pid == cur.pid) processTimeVec[j].TT = curTime - processTimeVec[j].AT;
                }
                exitV.push_back(cur);
            } else {
                IO.push(cur);
            }
            break;
```

그 후, MFQ를 Q0부터 돌면서 가장 우선순위가 높고 Arrival Time이 빠른 프로세스에 대해 CPU Burst를 진행시킵니다. 만약 TimeQuantum 내에 진행이 다 되었다면 후에 IO Burst 후에 Wakeup 되었을 때 한 단계 우선순위가 높은 큐에 배치시키기 위한 처리를 합니다.

```
} else if(cur.occup == timeQuantum[i]){
    createToReady();
    IOToReady();
    cur.at = curTime;
    cur.occup = 0;
    if(cur.Q == 3) cur.Q = 3;
    else cur.Q = cur.Q + 1;
    MFQ[cur.Q].push(cur);
    break;
}
MFQ[i].push(cur);
break;
```

만약 프로세스가 해당 queue의 TimeQuantum을 소모할 경우, 현재 큐보다 한단계 낮은 우선순위의 큐로 배치시킵니다.

```
if(!flag) {
    CPU.push_back(0);
    IOToReady();
}
curTime++;
```

ReadyQueue에 프로세스가 없을 경우 점유하지 않고 있음을 표시해주고, 시간을 증가시킵니다.

3. 다양한 입력에 대한 실행 결과
   A. 입력되는 프로세스가 1개일 경우

입력

```
1
1 0 0 5 10 20 18 25 15 10 20 15 16
```

결과

```
|-----P1(10)-----||----------0(20)----------||---------P1(18)---------||------------0(25)------------||-------P1
(15)-------||-----0(10)-----||----------P1(20)----------||-------0(15)-------||--------P1(16)--------|
P1's TT : 149s WT : 0s

Average Turnaround Time : 149s
Average Waiting Time : 0s
```

첫 프로세스가 0초에 도착하여 프로세스를 점유하기 시작하고, 첫 10초간 CPU를 사용한 후 20초 동안 IO Burst 진행하므로 CPU는 사용되지 않는다. 그 이후 18초간 CPU 사용, 25초간 IO Burst, 15초간 다시 CPU사용, 10초간 IO Burst 식으로 반복된다.

프로세스의 TT는 149초, WT는 0초.
프로세스가 하나뿐이므로 Average Turnaround time은 149초,
Average Waiting Time은 0초이다.

B. 입력되는 프로세스의 수가 2개일 경우
입력

```
2
1 0 0 2 10 20 18
2 4 0 2 5 1 2%
```

결과

```
|---P1(7)---||--P2(5)--||-P1(3)-||-P2(2)-||---------0(18)---------||---------P1(18)---------|
P1's TT : 53s WT : 5s
P2's TT : 13s WT : 5s

Average Turnaround Time : 33s
Average Waiting Time : 5s
```

먼저 AT = 0일 때 P1이 레디큐에 들어오고, CPU를 점유한다. Q0 -> Q1 -> Q2 총 7초를 점유하게 된다.
4초 때 P2가 레디큐에 들어오고, Q2를 마친 P1보다 우선순위가 높으므로 P2가 CPU를 점유하게 된다.
Q0 -> Q1 -> Q2 사용 중 timeQuanTum이 지나기 전에 마치고 sleep 상태로 넘어간다.
P1이 다시 점유하고, 3초 뒤 P1도 sleep 상태로 넘어간다.
이 때, P2가 wake up되고, Q1으로 진입한다.
P2가 CPU를 2초간 사용하게 된 후, 모두 끝났으므로 exit 한다.
P1은 현재 IO를 2만큼 수행하였으며, 18초간 IO를 더 수행하는 동안 CPU는 점유되지 않은 상태로 유지된다.
후에 P1이 wake up되고, 모든 burst time을 수행 후 exit 한다.

C. 입력되는 프로세스의 수가 2개이며 두 프로세스가 다른 초기 Queue를 가질 때.
입력

```
2
1 0 2 2 17 19 15
2 3 0 2 20 21 20%
```

결과

```
|--P1(4)--||---P2(7)---||------P1(13)------||------P2(13)------||---0(6)---||-------P1(15)-------||---------
-P2(20)----------|
P1's TT : 58s WT : 7s
P2's TT : 75s WT : 14s

Average Turnaround Time : 66.5s
Average Waiting Time : 10.5s
```

P1이 0초에 도착하여, 4초간 초기 배정된 Q2에서 CPU를 사용한다.

3초가 되었을 때, P2가 도착하였고 우선순위가 더 높으므로, 4초 때 P1은 preemption되어 FCFS로 넘어가고 P2가 CPU를 사용하기 시작한다.
P2가 7초간 Q0->Q1->Q2로 이동하며 CPU를 사용하고, FCFS로 넘어가게 된다.
FCFS에는 P1이 먼저 도착 하였으므로 P1이 CPU를 13초간 사용하고 sleep 상태로 넘어간다.
P2가 남은 13초간 CPU를 사용하고 sleep 상태로 넘어간다.
6초간 CPU는 사용되지 않다가, 19초간의 IO Burst를 마친 P1이 wakeup 되고 15초간 다시 CPU를 점유한 뒤 exit된다.
Wake up되어 기다리고 있던 P2가 뒤이어 CPU를 20초간 사용한 뒤 exit 된다.

D. 입력되는 프로세스의 수가 4개일 경우

입력

```
4
2 1 0 4 4 5 9 11 3 12 10
1 0 0 1 8
3 3 2 3 9 4 15 12 10
4 6 3 2 5 10 6%
```

출력

```
|P1(1)||P2(1)||-P1(2)-||-P2(2)-||--P3(4)--||--P1(4)--||P2(1)||--P4(5)--||---P2(6)---||--P3(5)--||P1(1)||-P2(
3)-||---P4(6)---||-------P3(15)-------||-P2(3)-||----0(9)----||-----P3(10)-----||-----P2(10)-----|
P1's TT : 32s WT : 24s
P2's TT : 87s WT : 33s
P3's TT : 75s WT : 25s
P4's TT : 35s WT : 14s

Average Turnaround Time : 57.25s
Average Waiting Time : 24s
```

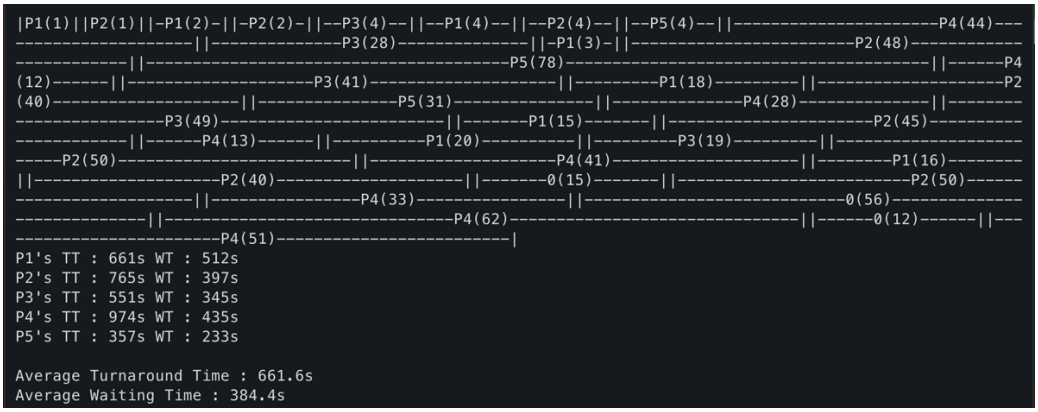
4개의 복잡한 process도 잘 동작함을 확인할 수 있다.
(수기로 작성한 Gantt Chart와 결과가 같았습니다)

E. 입력되는 프로세스의 수가 5개일 경우

입력

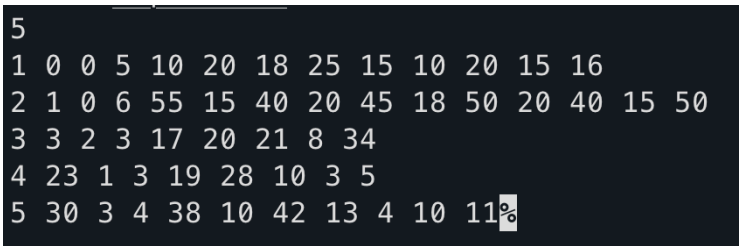
```
5
1 0 0 5 10 20 18 25 15 10 20 15 16
2 1 0 6 55 15 40 20 45 18 50 20 40 15 50
3 3 2 4 32 20 41 12 49 33 19
4 6 3 8 44 32 12 31 28 17 13 32 41 75 33 56 62 12 51
5 8 2 2 82 11 31%
```

출력

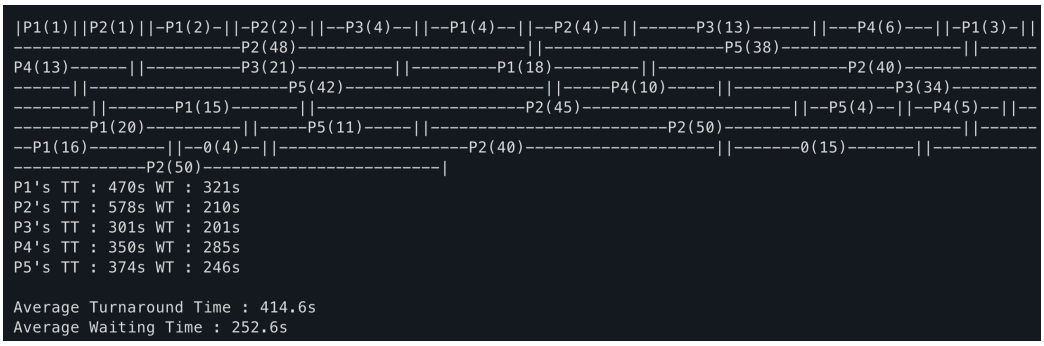


5개의 복잡한 process도 잘 동작함을 확인할 수 있다.

입력



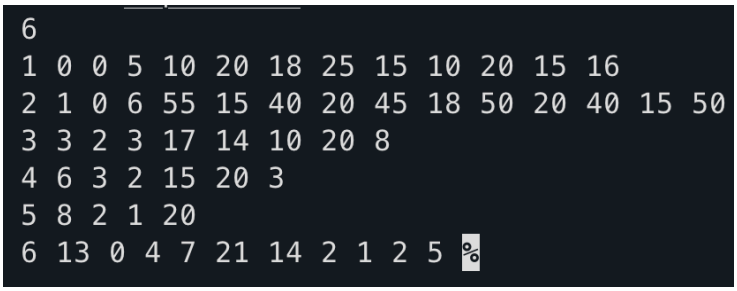
출력



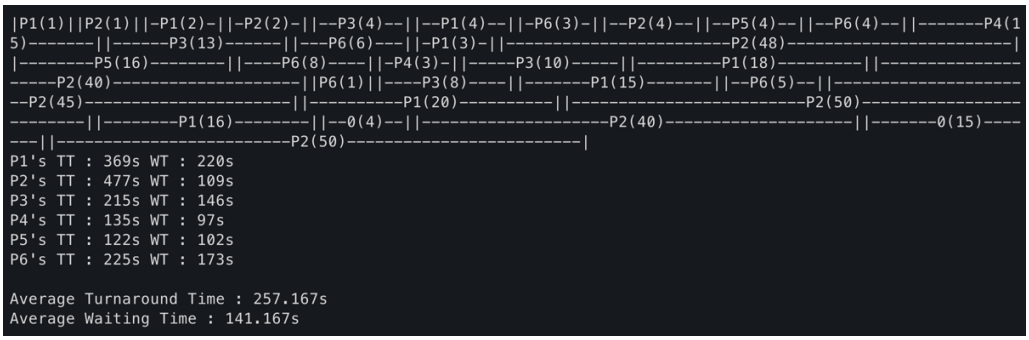
또 다른 5개의 복잡한 process도 잘 동작함을 확인할 수 있다.

F. 입력되는 프로세스의 수가 6개일 경우

입력



결과



6개의 복잡한 process도 잘 동작함을 확인할 수 있다.

4. 실행 결과를 보는 방법에 대한 설명
   실행 결과는 Gantt 차트, 각 프로세스 별 Turnaround Time(TT) 및 Waiting Time (WT), 그리고 전체 프로세스의 평균 Turnaround Time(TT) 및 Waiting Time (WT)으로 구성되어 있습니다.

   Gantt 차트는 수업 PDF 자료에 있는 그림과 같은 형태입니다. 실행되는 프로세스의 PID를 막대 가운데에 적어 두었고, CPU를 점유한 시간을 숫자로 나타내었습니다. 0은 해당시간동안 IO작업만 있었을 뿐 CPU를 점유한 프로세스가 없다는 의미입니다.