REPORT

Process-scheduler simulator 제작

교 과 목	운영체제(SW)
분 반	3
담 당 교 수	최종무 교수님
학 과	소프트웨어학과
학 번	32141868
	32144697
이 름	박유현
	최광진
제 출 일	2019.04.15
	2019.04.18(수정)

목 차

1. 알고리즘에 필요한 Queue 구현

저희 팀은 먼저 알고리즘 구현에 필요한 Queue를 직접 구현하였습니다.

구현에는 구조체를 사용하고 상황에 맞는 Method를 구현해 두었습니다.

```
struct Queue {
   int max, front, rear;
   int *data;
};
// 구조체에는 어떤 프로세스를 실행하고 있는지에 대한 프로세스 이름만을 입력해주게 됩니다.
```

Method

```
void qInit(struct Queue *q, int max)   // Queue 초기화
void qDestroy(struct Queue *q)   // Queue 삭제
void qPush(struct Queue *q, int param)   // Queue data 삽입
int qPop(struct Queue *q)   // Queue 원소 삭제 return data
int qPeek(struct Queue *q)   // Queue 가장 앞에 있는 원소를 확인 return data
int qSize(struct Queue *q)   //Queue의 크기를 확인
void qPrint(struct Queue *q)   //개발용 Queue의 현재 상태 출력
```

Queue Push, Pop, Peek, Size를 중점적으로 사용하였으며 Debug를 용이하게 할 수 있도록 qPrint를 사용하여 계속해서 큐의 상태를 체크하여 사용했습니다.

해당 Queue의 구성을 어떻게 할지 계속 고민한 결과, 가장 간단한 형태로 프로세스의 이름을 부여하고 해당 프로세스의 이름을 Queue에 넣어주는 것으로 하였습니다. 프로세스의 이름은 기본적으로 도착한 순서로 부여되거나 Input Data의 순서대로 부여됩니다.

2. FCFS

CPU의 입장에서 프로세스는 차례대로 오는 것으로 보일 것이라 생각하여 Sort를 하여 진행했습니다.

2.1 핵심코드

```
while(1) {
    tempQSize = qSize(&q);
    for (int i = 0; i < col; i++)
        if (realTime >= tempData[i][0] && checkProcess[i] == 0) { // It queues when a process arrives.
            for (int j = 0; j < serviceData[i]; j++) {
                qPush(&q, i);
                realTime++;
                leftProcessTime--;
            }
            checkProcess[i] = 1;
            }
        if (qSize(&q) == tempQSize){ // If no jobs have arrived, wait.
            qPush(&q, -1);
            realTime++;
            totalProcessTime++;
        }
        if (leftProcessTime == 0)// Exit when all execution is finished.
            break;
}
```

FCFS는 특별히 어려웠던 점은 없었습니다. 실제 시간처럼 동작할 realTime 변수를 기준으로 프로그램을 작성하였습니다. 먼저, 첫 if문에서 처음 도착한 프로세스인지 확인하고 큐에 넣어줍니다. 큐에 넣어줄 때 시간이 흐릅니다. 이때, while에 시작부분에 qSize를 미리 저장해 두어, qSize가 변하는지 봅니다. 변한 다면 도착한 프로세스가 있었던 것이고, 만약 도착한 프로세스가 없다면 공백을 뜻하는 -1을 넣어주고 시간이 흐릅니다. 결과적으로 FCFS의 실행 순서와 같은 Queue가 생성되며 프로세스가 도착하면 Queue에 넣어주고 Pop하여 결과 값을 1차원 배열로 넘겨줍니다.

2.2 실행 스크린샷

```
 # FCFS

   Option setup
-----------------
 2. Choose sample data.

 1.       A  B  C  D  E
arrival   0  2  4  6  8
duration  3  6  4  5  2

 2.       A  B  C
arrival   0  1  2
duration 15  1  1

Enter your command : 1

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E
        |     |     |     |     |
        V     V     V     V     V
-------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
-------------------------------------------------------------------
   A |  O  O  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  -  O  O  O  O  O  O  -  -  -  -  -  -  -  -  -  -  -
   C |  -  -  -  -  -  -  -  -  -  O  O  O  O  -  -  -  -  -  -  -
   D |  -  -  -  -  -  -  -  -  -  -  -  -  -  O  O  O  O  O  -  -
   E |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  O
(SUM)|  A  A  A  B  B  B  B  B  B  C  C  C  C  D  D  D  D  D  E  E
-------------------------------------------------------------------

-------------------------------------------------
 (Analyzation)  |   A    B    C    D    E     AVG
-------------------------------------------------
  Response time |   0    1    5    7   10    4.60
Turnaround time |   3    7    9   12   12    8.60
-------------------------------------------------
```

```
 2.       A  B  C
arrival   0  1  2
duration 15  1  1

Enter your command : 2

 # 'Process-scheduler simulator' RESULT REPORT

        A  B  C
        |  |  |
        V  V  V
-------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
-------------------------------------------------------------
   A |  O  O  O  O  O  O  O  O  O  O  O  O  O  O  O  -  -
   B |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -
   C |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O
(SUM)|  A  A  A  A  A  A  A  A  A  A  A  A  A  A  A  B  C
-------------------------------------------------------------


---------------------------------------
 (Analyzation)  |   A    B    C     AVG
---------------------------------------
  Response time |   0   14   14    9.33
Turnaround time |  15   15   15   15.00
---------------------------------------
```

3. Round Robin

CPU의 입장에서 프로세스는 차례대로 오는 것으로 보일 것이라 생각하여 Sort를 하여 진행했습니다.

3.1 핵심코드

```
while (1) { //Loop
    for (int i = 0; i < col; i++)
        if (realTime >= tempData[i][0] && checkProcess[i] == 0) {     // It queues when a process arrives.
            for (int j = 0; j < serviceData[i]; j++)
                qPush(&q, i);
            checkProcess[i] = 1;
        }
    if (temp == qPeek(&q)) // Processes that were performed without any new arrivals are also backward.
        for (int w = 0; w < serviceData[temp]; w++)
            qPush(&q, qPop(&q));
    for (int i = 0; i < timeQuantum; i++) { // Executes as much as the time quantum.
        if (qSize(&q) == 0) // Ends the loop if the queue is empty.
            break;
        leftServiceTime--;
        temp = qPop(&q);
        resultData[realTime] = temp;
        serviceData[temp]--;
        realTime++;       // If the process is successful, it will take time.
        if(temp != qPeek(&q)) // When the process has finished running, it exits the loop.
            break;
        }
    if (qSize(&q) == 0) { // The part that handles spaces between processes.
        emptyCounter++;
        if(emptyCounter == 3) {
                    resultData[realTime] =- 1;
                    realTime++;
                    totalProcessTime++;
                    emptyCounter = 0;
        }
    }
    else
        emptyCounter = 0;
    if(leftServiceTime == 0) // Exit when all execution is finished.
        break;
}
```

특별히 어려운 점이 있었다면 순서였습니다. 순서를 틀리면 원하는 결과를 얻을 수 없었습니다. Queue의 삽입, 삭제, 이동에 집중하여 코딩하였습니다. 정확한 포인트는 주석을 확인해 주시면 감사하겠습니다. 먼저, 처음 도착한 프로세스가 있는지 확인하고 있다면 Queue에 넣어줍니다. 그

뒤, 이전에 실행하던 프로세스 Time Quantum을 전부 사용한 프로세스면 Queued의 뒷부분으로 이동됩니다. 이동이 끝나면 Queue를 POP하며 실행합니다. 현재 실행한 프로세스를 Temp에 잡아주어 확인하고 해당 프로세스가 끝났는지, 또는 실행할 프로세스가 없는지 체크합니다. 정상적으로 실행된다면 realTime이 증가하며 시간이 흐릅니다. 프로세스가 아무도 동작하지 않고 아무도 도착하지 않았을 때의 코드가 마지막 부분입니다. Emptycounter를 만들어 해당 counter가 3이 되면 현재 시간에는 아무도 도착하지 않고 실행이 안되는 것으로 간주하여 시간을 흐르게 합니다. result에는 -1을 넣어줍니다.

3.2 실행 스크린샷

```
 # RR

   Option setup
-----------------
  1. time quantum
->2. input process data
-----------------
 2.2 Choose sample data.

 1.       A  B  C  D  E
arrival   0  2  4  6  8
duration  3  6  4  5  2

 2.       A  B  C  D  E  F  G  H
arrival   0  2  4  6  8 10 12 14
duration  4  4  4  4  4  4  4  4

Enter your command : 1

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E
        |     |     |     |     |
        V     V     V     V     V
-----------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
-----------------------------------------------------------------
   A |  O  O  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  O  -  O  -  O  -  -  O  -  -  -  O  -  -  -  O  -  -
   C |  -  -  -  -  -  O  -  -  O  -  -  -  O  -  -  -  O  -  -  -
   D |  -  -  -  -  -  -  -  O  -  -  -  O  -  -  -  O  -  -  O  O
   E |  -  -  -  -  -  -  -  -  -  -  O  -  -  -  O  -  -  -  -  -
(SUM)|  A  A  B  A  B  C  B  D  C  B  E  D  C  B  E  D  C  B  D  D
-----------------------------------------------------------------


-------------------------------------------------
 (Analyzation)  |   A    B    C    D    E     AVG
-------------------------------------------------
  Response time |   0    0    1    1    2    0.80
Turnaround time |   4   16   13   14    7   10.80
-------------------------------------------------
```

```
 2.       A  B  C  D  E  F  G  H
arrival   0  2  4  6  8 10 12 14
duration  4  4  4  4  4  4  4  4

Enter your command : 2

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E     F     G     H
        |     |     |     |     |     |     |     |
        V     V     V     V     V     V     V     V
-------------------------------------------------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
-------------------------------------------------------------------------------------------------------------
   A |  O  O  -  O  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  O  -  O  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   C |  -  -  -  -  -  O  -  -  -  O  -  -  -  -  O  -  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -
   D |  -  -  -  -  -  -  -  -  O  -  -  -  O  -  -  -  -  O  -  -  -  -  -  O  -  -  -  -  -  -  -  -
   E |  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  O  -  -  -  -  -  O  -  -  -  -  O  -  -  -  -  -
   F |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  -  -
   G |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  -  O  -  -
   H |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  -  O  -  -  -  O  -  -  O
(SUM)|  A  A  B  A  B  C  A  B  D  C  E  B  D  F  C  E  G  D  H  F  C  E  G  D  H  F  E  G  H  F  G  H
-------------------------------------------------------------------------------------------------------------


-------------------------------------------------------------
 (Analyzation)  |   A    B    C    D    E    F    G    H     AVG
-------------------------------------------------------------
  Response time |   0    0    1    2    2    3    4    4    2.00
Turnaround time |   7   10   17   18   19   20   19   18   16.00
-------------------------------------------------------------
```

FCFS보다 Response time이 눈에 띄게 빨라진 것이 보입니다.

4. Multi-Level Feedback Queue

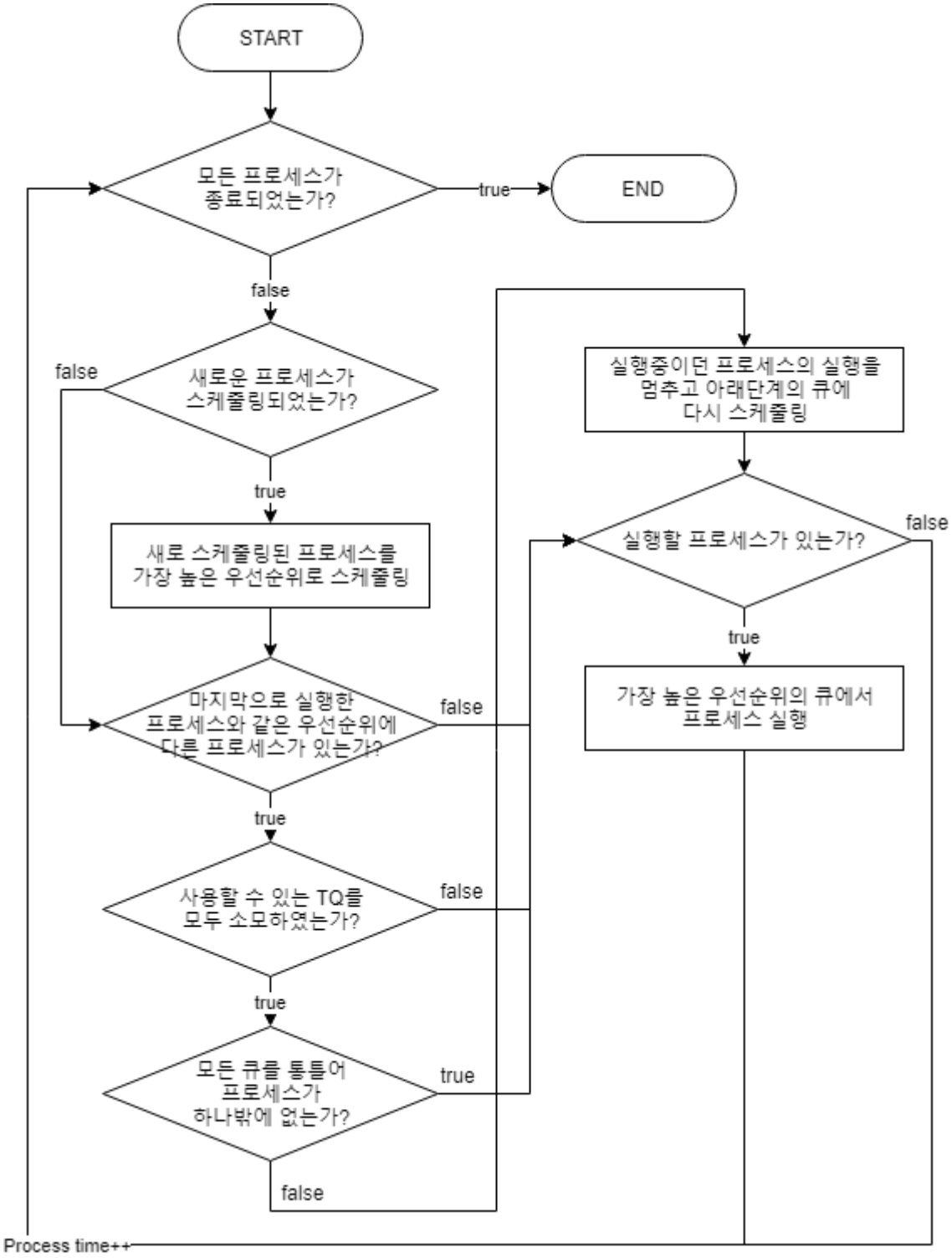
4.1 핵심코드

```
    while (getLeftTime(leftServiceTimeArr, col) > 0) {
        //Add new process to highest priority queue when there is a process currently in procTime.
        //If there are multiple processes arriving at the same time, a process with a smaller array index is added to the queue first.
        // Because the process is not named in the current simulator.
        int pushCnt = -1;
        for (int i = 0; i < col; i++)
            if (data[i][0] == procTime) {
                for (int j = 0; j < data[i][1]; j++)
                    qPush(&queueList[0], i);
                if (pushCnt == -1)
                    pushCnt = i;
            }

        //When there are remaining jobs and there are another processes scheduled in scheduler, drop it to the queue below.
        int totalQueueSize = 0;
        for (int i = 0; i < queueSize; i++)
            totalQueueSize += qSize(queueList+i);
        if (leftServiceTimeArr[curProc] == 0)
```

```
            quantumTimer = timeQuantum;
        for (int i = 0; i < queueSize; i++)
            if (qSize(&queueList[i]) > 0) {
                if (curProc == qPeek(&queueList[i]))
                    quantumTimer--;
                if (quantumTimer <= 0 && totalQueueSize != leftServiceTimeArr[curProc]) {
                    for (int j = 0; j < leftServiceTimeArr[curProc]; j++)
                        if (i != queueSize - 1)
                            qPush(&queueList[i+1], qPop(&queueList[i]));
                        else //Push to itself because there is no queue below.
                            qPush(&queueList[i], qPop(&queueList[i]));
                    quantumTimer = timeQuantum;
                    break;
                }
            }

        //Execute process in the highest priority queue.
        for (int i = 0; i < queueSize; i++)
            if (qSize(&queueList[i]) > 0) {
                curProc = qPop(&queueList[i]);
                break;
            }
            else
                curProc = -1;
        if (curProc > -1) { //Continue roof when there are no processes to execute.
            result[procTime] = curProc; //Push to result.
            leftServiceTimeArr[curProc]--;
        }
        procTime++;
    }
    *resSize = procTime;
    return result;
}
```

MLFQ는 크게 세가지로 나눌 수 있는데 먼저 하나의 큐는 RR과 똑같이 동작한다는 것. 둘째는 여러 단계의 큐가 서로 다른 우선순위를 갖고 프로세스들을 스케줄링 한다는 것이고, 세번째는 하나의 프로세스가 실행되면 여러 조건에 따라 우선순위가 낮은 단계로 다시 스케줄링 된다는 것이었습니다.

따라서 다음 순서도와 같은 논리를 따라 코딩을 하였습니다.



<MLFQ 알고리즘 구현 순서도>

4.2 실행 스크린샷

```
 # MLFQ

   Option setup
-----------------
  1. time quantum
  2. queue size
->3. input process data
-----------------
 3.2 Choose sample data.

 1.        A  B  C  D  E
arrival    0  2  4  6  8
duration   3  6  4  5  2

 2.        A  B  C  D  E  F  G  H
arrival    4  2  4  6  8 29 32 33
duration   8  6  4  5  2  8  1  6

Enter your command : 1

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E
        |     |     |     |     |
        V     V     V     V     V

-------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
-------------------------------------------------------------------
   A |  O  O  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  O  -  -  O  -  -  -  -  -  O  -  -  O  -  -  O  -  O
   C |  -  -  -  -  O  -  -  O  -  -  -  -  O  -  -  O  -  -  -  -
   D |  -  -  -  -  -  -  O  -  -  O  -  -  -  O  -  -  O  -  O  -
   E |  -  -  -  -  -  -  -  -  O  -  O  -  -  -  -  -  -  -  -  -
(SUM)|  A  A  B  A  C  B  D  C  E  D  E  B  C  D  B  C  D  B  D  B
-------------------------------------------------------------------

-------------------------------------------------
 (Analyzation)  |   A    B    C    D    E     AVG
-------------------------------------------------
  Response time |   0    0    0    0    0    0.00
Turnaround time |   4   18   12   13    3   10.00
-------------------------------------------------
```

```
 2.        A  B  C  D  E  F  G  H
arrival    4  2  4  6  8 29 32 33
duration   8  6  4  5  2  8  1  6

Enter your command : 2

 # 'Process-scheduler simulator' RESULT REPORT

              B    A..    D     E                                                          F        G  H
              |     |     |     |                                                          |        |  |
              V     V     V     V                                                          V        V  V

----------------------------------------------------------------------------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
----------------------------------------------------------------------------------------------------------------------------------------
   A |  -  -  -  -  O  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  -  -  O  -  O  O  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  O  O  -  -  -  O  -  -  -  -  -  O  -  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   C |  -  -  -  -  -  O  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   D |  -  -  -  -  -  -  O  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   E |  -  -  -  -  -  -  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   F |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  O  O  -  -  O  -  O  -  O  -  O  -  O  -
   G |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -
   H |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  O  -  O  -  O  -  O  -  O
(SUM)|  -  -  B  B  A  C  D  B  E  A  C  D  E  B  A  C  D  B  A  C  D  B  A  D  A  A  A  -  -  F  F  F  G  H  F  H  F  H  F  H  F  H  F  H
----------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------
 (Analyzation)  |   A    B    C    D    E    F    G    H     AVG
-------------------------------------------------------------------
  Response time |   0    0    1    0    0    0    0    0    0.12
Turnaround time |  23   20   16   18    5   14    1   11   13.50
-------------------------------------------------------------------
```

Response time이 0으로 굉장히 빨랐던 것이 눈에 띄었습니다.

5. Lottery

5.1 핵심코드

```
while (getLeftTime(leftServiceTimeArr, col) > 0) {
    //Add new process when there is a process currently in procTime.
    int pushCnt = -1;
    for (int i = 0; i < col; i++)
        if (data[i][0] == procTime) {
            for (int j = 0; j < data[i][1]; j++)
                ticketArr[i] = data[i][1];
            if (pushCnt == -1)
                pushCnt = i;
        }

    //Executes a process that randomly picks one of the tickets.
    int ticketAmount = 0;
    for (int i = 0; i < col; i++)
        ticketAmount += ticketArr[i];
    if (ticketAmount > 0) { //Continue roof when there are no tickets to pick.
        int r = rand() % ticketAmount;
        for (int i = 0; i < col; i++)
            if (r >= ticketArr[i])
                r -= ticketArr[i];
            else {
                curProc = i;
                break;
            }
        result[procTime] = curProc; //Push to result.
        leftServiceTimeArr[curProc]--;

        for (int i = 0; i < col; i++)
            if (leftServiceTimeArr[i] == 0)
                ticketArr[i] = 0;
    }
    procTime++;
}
```

Lottery는 MLFQ만큼 복잡한 논리를 포함하고 있지 않았습니다. Lottery의 구현은 난수를 생성하여 ticket을 뽑아 다음 실행될 프로세스를 정하고 하나의 프로세스가 종료되면 ticket을 제거하여 다음 프로세스들이 실행될 수 있도록 구현하였습니다.

5.2 실행 스크린샷

```
 # Lottery

   Option setup
-----------------
 2. Choose sample data.

 1.       A  B  C  D  E
arrival   0  2  4  6  8
duration  3  6  4  5  2

 2.       A  B  C  D  E  F
arrival   0  0  0  0  0  0
duration  5  5  5  5  5  5

Enter your command : 1

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E
        |     |     |     |     |
        V     V     V     V     V
-------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
-------------------------------------------------------------------
   A |  O  O  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  -  O  O  O  O  -  -  O  O  -  -  -  -  -  -  -  -  -
   C |  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  O  O  -  O
   D |  -  -  -  -  -  -  -  O  O  -  -  O  O  -  O  -  -  -  -  -
   E |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  O  -
(SUM)|  A  A  A  B  B  B  B  D  D  B  B  D  D  C  D  E  C  C  E  C
-------------------------------------------------------------------

-------------------------------------------------
 (Analyzation)  |   A    B    C    D    E     AVG
-------------------------------------------------
  Response time |   0    1    9    1    7    3.60
Turnaround time |   3    9   16    9   11    9.60
-------------------------------------------------
```

```
 1.       A  B  C  D  E
arrival   0  2  4  6  8
duration  3  6  4  5  2

 2.       A  B  C  D  E  F
arrival   0  0  0  0  0  0
duration  5  5  5  5  5  5

Enter your command : 1

 # 'Process-scheduler simulator' RESULT REPORT

        A     B     C     D     E
        |     |     |     |     |
        V     V     V     V     V
-------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
-------------------------------------------------------------------
   A |  O  O  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  -  O  -  -  -  O  O  -  O  -  O  -  O  -  -  -  -  -
   C |  -  -  -  -  O  O  -  -  -  -  -  O  -  O  -  -  -  -  -  -
   D |  -  -  -  -  -  -  O  -  -  O  -  -  -  -  -  O  -  O  O  -
   E |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  O
(SUM)|  A  A  A  B  C  C  D  B  B  D  B  C  B  C  B  D  E  D  D  E
-------------------------------------------------------------------

-------------------------------------------------
 (Analyzation)  |   A    B    C    D    E     AVG
-------------------------------------------------
  Response time |   0    1    0    0    8    1.80
Turnaround time |   3   13   10   13   12   10.20
-------------------------------------------------
```

```
 2.       A  B  C  D  E  F
arrival   0  0  0  0  0  0
duration  5  5  5  5  5  5

Enter your command : 2

 # 'Process-scheduler simulator' RESULT REPORT

       A..
        |
        V
-------------------------------------------------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
-------------------------------------------------------------------------------------------------------------
   A |  -  -  -  -  -  -  -  -  -  O  O  -  -  -  -  -  O  -  -  -  -  -  -  -  O  -  O  -  -  -
   B |  -  -  -  -  -  -  -  -  O  -  -  O  -  -  -  -  -  O  -  -  -  -  O  O  -  -  -  -  -  -
   C |  O  -  -  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  O  -  O  -  -
   D |  -  O  -  -  -  -  O  -  -  -  -  -  -  O  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  -
   E |  -  -  -  -  O  -  -  -  -  -  -  -  O  -  O  -  -  -  O  -  -  O  -  -  -  -  -  -  -  -
   F |  -  -  O  O  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  O
(SUM)|  C  D  F  F  E  F  D  C  B  A  A  B  E  D  E  D  A  B  E  D  C  E  B  B  A  C  A  C  F  F
-------------------------------------------------------------------------------------------------------------

-------------------------------------------------------
 (Analyzation)  |   A    B    C    D    E    F     AVG
-------------------------------------------------------
  Response time |   9    8    0    1    4    2    4.00
Turnaround time |  27   24   28   20   22   30   25.17
-------------------------------------------------------
```

```
 1.       A  B  C  D  E
arrival   0  2  4  6  8
duration  3  6  4  5  2

 2.       A  B  C  D  E  F
arrival   0  0  0  0  0  0
duration  5  5  5  5  5  5

Enter your command : 2

 # 'Process-scheduler simulator' RESULT REPORT

       A..
        |
        V
-------------------------------------------------------------------------------------------------------------
Time |  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
-------------------------------------------------------------------------------------------------------------
   A |  -  -  O  -  -  O  O  -  -  O  -  -  -  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -
   B |  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  O  -  -  -  O  -  O  O  -  -
   C |  -  -  -  -  -  -  -  -  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  O  -  O  -  -  O  O
   D |  -  O  -  -  -  -  -  -  O  -  O  O  -  -  -  -  O  -  -  -  -  -  -  -  -  -  -  -  -  -
   E |  O  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -  -  O  -  -  -  O  -  -  -  -  -  -  -  -
   F |  -  -  -  -  O  -  -  -  -  -  -  -  -  O  O  -  -  -  -  O  -  O  -  -  -  -  -  -  -  -
(SUM)|  E  D  A  E  F  A  A  E  D  A  D  D  C  F  F  B  D  A  E  F  B  F  E  C  B  C  B  B  C  C
-------------------------------------------------------------------------------------------------------------

-------------------------------------------------------
 (Analyzation)  |   A    B    C    D    E    F     AVG
-------------------------------------------------------
  Response time |   2   15   12    1    0    4    5.67
Turnaround time |  18   28   30   17   23   22   23.00
-------------------------------------------------------
```

Sample 1, Sample 2를 각각 두 번씩 실행한 스크린샷입니다.

실행할 때마다 다른 방식으로 스케줄링이 되는 것을 확인할 수 있습니다.

6. Main 함수

Main 함수에 대한 설명은 코드 없이 간단한 사용법만 소개하겠습니다.

```
# Process-scheduler simulator

1. FCFS
2. RR
3. MLFQ
4. Lottery
5. Exit
Enter your command :
```

먼저, 실행하면 어떤 Scheduler를 사용할지 결정하게 됩니다. 그 다음 각각 알고리즘에서 필요한 변수를 입력 받는 Option Setup으로 넘어갑니다. 알고리즘에 따라 입력 받는 변수의 개수가 다릅니다.

```
# FCFS

  Option setup
-----------------
1. Choose which input data to use.
  1. Type directly.
  2. Use sample process data.
Enter your command : 2
```

변수들을 입력해 준 다음 알고리즘 실행에 대입할 프로세스를 입력합니다. 이 때 사용자가 원하는 값을 직접 입력시켜줄지, 아니면 Sample Data를 사용할지 결정할 수 있습니다. Sample Data를 사용하여 결과를 빠르게 확인할 수 있습니다. 각 Sample Data는 PPT에 있는 데이터를 1번, 그 Algorithm에 특징이 두드러지게 나타날 만한 Data를 2번에 두었습니다.

이후 Option들을 모두 설정하면 자동으로 실행된 결과를 출력하게 됩니다.

7. 느낀 점

32144697 최 광진: FCFS, RR 스케줄러 구현.

간단하게 보면 정말 간단한 알고리즘이지만 그 안에서 고려해야할 문제가 꽤 많다는 걸 알게 된 재밌는 과제였습니다. 평소에 사용하던 객체를 사용하지 못하는 것에 대한 어려움도 꽤 있었습니다. Round Robin을 구현할 때에는 순서에 가장 많은 신경을 쓰게 되었습니다. 진행중인 프로세스가 Time Quantum을 전부 사용했는가를 체크하는 부분과 새로운 프로세스가 들어올 때 새 프로세스에게 비켜주어야 하는지 그대로 실행해야 하는지에 대한 부분에 가장 큰 초점을 맞추었습니다. 친구와 프로젝트를 진행함에 있어 Github을 사용하여 피드백 하였고 생각하는 바가 다를 수 있기 때문에 서로 맡은 부분만이 아니라 서로의 알고리즘에 계속해서 피드백을 하고 알고리즘을 수정하였습니다.

32141868 박 유현: 이번 프로젝트에서 MLFQ와 Lottery 스케줄러의 구현을 진행하며...

처음에 강의를 듣고 이해할 때는 구현하기에 별것 아닌 것 같았던 스케줄러의 개념들이 실제로 구현해볼 때는 생각보다 실제 스케줄러는 작업이 좀 더 세세한 단계로 나뉘고 그 작업들 간의 순서와 관계가 밀접하게 엮여 있다는 것을 알게 됐다. 구현에 대한 부분은 최대한 개념적으로 접근하려 노력했고 그 덕분에 구현을 함은 물론 코드의 길이도 많이 단축되고 가독성도 좋아진 것 같다. 그리고 평소 사용하던 객체지향언어를 사용하지 않고 C언어로 구현을 하기로 했는데 이부분에서 많은 애로사항이 있었다. 하지만 이번 기회에 C언어를 다시 한 번 상기할 수 있었고 어려운 부분은 팀원과 함께 풀어나갈 수 있어 다행이었다고 생각한다. 추가적으로 작업을 진행할 때 Git을 이용하여 팀원과 협업을 진행하였고 작업에서 협동의 요소가 중요함을 다시 한 번 느낄수 있었다. 참고로 해당 Github 페이지에 작업과정이 모두 남아있고 페이지 주소는 다음과 같다. "https://github.com/kevin0309/Process-scheduler"