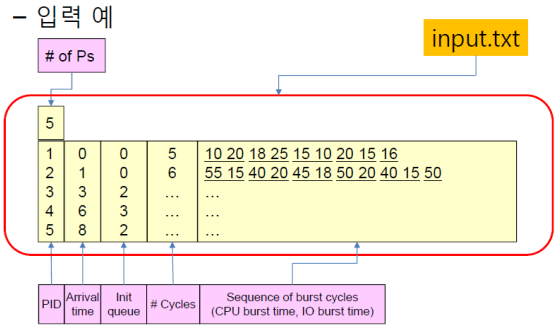
Project 1: MFQ Scheduling 구현

2017314626 이재훈

이 프로그램은, 다양한 scheduling 기법 중 MFQ 방식으로 input.txt에서 입력 받은 프로세스를 scheduling한다. 그 후 time에 따른 CPU-burst중인 process를 Gantt Chart의 형식으로 보여준다. 또한 Gantt Chart의 출력이 끝난 후에는, 각 process의 Turnaround Time과 Waiting Time이 제공되며, 모든 process들의 평균 Turnaround Time과 평균 Waiting Time이 보이며 프로그램이 종료된다.

Input.txt



input.txt의 형식은 오른쪽과 같다. 1행에는 process의 개수가 입력되고, 2행부터는 각 process 별 PID, Arrival time(도착 시간), Init queue(최초 진입 queue), number of cycles, sequence of burst cycles가 입력된다. 단, 각 process의 마지막 cycle은 항상 CPU-burst로 끝난다고 가정한다. 즉 sequence of burst cycles은 cycle*2 -1의 개수만큼 입력된다.

고려사항

Scheduling 과정에 있어서 몇 가지 선제조건들이 존재한다. 먼저, 모든 process의 IO burst는 병렬 진행이 가능하다. 즉, 다른 process가 IO를 하는 것과 상관없이 독립적으로 자신의 IO burst를 진행한다. 또한, 각 프로세스가 Ready Queue_i에서 time quantum을 모두 소모할 경우, Ready Queue_i+1에 진입한다. 만약 Ready Queue_i에서 스케쥴 받아 실행한 process가 IO burst를 마치고 wakeup 되는 경우, Ready Queue_i-1로 진입한다. (Ready Queue_0의 경우 Ready Queue_0으로 진입한다.) 또한, Ready Queue_3에 진입한 process는, IO burst를 마치더라도 무조건 ReadyQueue_3으로 진입한다. 만약 CPU-burst가 끝남과 동시에 time quantum을 다 썼다면, 그 process는 IO burst가 끝날 때, time quantum을 다 쓴 것과 IO burst가 끝난 것이 상충되어 기존의 Ready Queue로 진입하는 것으로 규칙을 정했다. 또한, 만약 a라는 프로세스가 time slice를 다 사용하여 Ready Queue를 옮기고 b라는 프로세스는 io burst를 마쳤거나 arrival time이 돼서 진입한다면, a 프로세스보다 b 프로세스를 우선적으로 Ready Queue에 진입시키는 규칙을 정했다. 그리고 만약 io burst를 마친 후 wake up되어 다시 Ready Queue에 들어가는 다수의 프로세스가 있다면, input.txt에서의 프로세스 순서대로 Ready Queue에 진입시키는 규칙을 정했다. 마지막으로, 각 Ready Queue의 우선순위는 Q0>Q1>Q2>Q3의 순이며, scheduling은 항상 높은 우선순위의 queue부터 이루어진다.

Ready Queue

각 Ready Queue는 독자적인 scheduling 방식을 가지며, 그것은 다음과 같다.

- Q0: Time quantum 1의 RR scheduling
- Q1: Time quantum 2의 RR scheduling
- Q2: Time quantum 1의 RR scheduling
- Q3: FCFS scheduling

프로그램 설계

이 프로그램은 ps와 Node라는 구조체를 사용하며, ps_num, global, slice라는 전역변수를 사용한다. 또한 RQ_0, RQ_1, RQ_2, RQ_3 총 4개의 Ready Queue와, 프로세스들의 초기 저장소인 befQ와 io 작업중인 프로세스를 넣을 ioQ를 구현하였다. 또한, schedule, setPs, move_queue, io_check 총 4개의 함수를 사용하였으며 함수에서 구현되지 않은 기능들은 main함수 안에서 구현하였다.

- ps

  Pid, at(arrival time), initq(진입할 큐), cycle(cycle의 수), burst_time(각 burst time을 저장), c_index(몇 번째 cycle인가에 대한 정보), num(결과값 출력순서를 위한 값)의 정보를 가지고 있는 구조체다.

- Node

  Ready Queue의 구현을 위해 생성된 연결 리스트 구조체다. 데이터로는 ps형 포인터를 가진다.

- ps_num, global, slice

  ps_num은 파일의 첫번째 행에서 얻은 process의 개수를 저장한다. Global은 현재 시간을 저장한다. 즉, 첫 시작할 때는 0이며, 1회의 scheduling마다 1씩 증가시킨다. (CPU 또는 io burst를 하지 않더라도 이것은 증가한다.) slice는 현재 CPU-burst를 진행중인 ready queue의 time quantum을 저장한다. 만약 ready queue가 time quantum을 가지지 않는다면 (Q3의 경우) -1을 저장한다.

main함수는 다음의 과정을 가진다.

1. 파일을 입력 받아 프로세스의 개수를 저장하고, 그 프로세스의 개수만큼 프로세스를 읽는다. 그 후, 프로세스를 befQ에 넣는다.

2. Scheduling을 위한 무한루프 while(1)을 실행한다. While 안에서의 과정은 다음과 같다.

    1) befQ안의 프로세스의 arrival time이 되었으면 그 프로세스를 적합한 ready queue에 할당한다.

    2) io_check를 통해 io burst time이 0이 된 프로세스가 있다면 그 프로세스를 적합한 ready queue에 할당한다. 이 때, 기존의 ready queue가 Q3이 아니라면 한 단계 낮은 ready queue로 할당한다. 즉, RQ_i이었다면 RQ_i-1로 할당한다.

    3) 만약 현재 CPU-burst 진행중인 프로세스가 없다면, schedule 함수를 통해 CPU-burst 할 프로세스를 할당 받는다. 만약 CPU-burst할 프로세스가 없고 io burst중인 프로세스도 없으며 아직 arrival time이 되지 않아 befQ에서 대기중인 프로세스도 없다면 scheduling이 끝난 것이므로 while문을 탈출한다.

    4) 현재 진행중인 프로세스를 출력한다. 만약 현재 CPU-burst가 진행중이지 않고 io burst가 끝나기를 기다리는 중이라면, "cpu idle"을 출력한다.

    5) Global 시간을 1 증가시킨다. 또한, time quantum(slice 변수)을 1 감소시킨다.

    6) io burst와 CPU-burst를 진행한다. ioQ에 있는 모든 프로세스의 남은 io burst time을 1 감소시킨다. (io는 병렬적으로 수행되기 때문이다.) 또한 현재 CPU-burst중인 프로세스의 남은 CPU-burst time도 1 감소시킨다. 만약 프로세스가 time quantum을 다 소모하였으며 현재 ready queue가 Q3이 아니라면, ready queue를 한 단계 높인다. 즉, RQ-i에서 RQ_i+1로 옮긴다. 만약 time quantum을 소모하지 않고 CPU burst가 끝났다면 ready queue를 변경하지 않는다. 만약 프로세스의 남은 CPU-burst가 0이 된다면, 그 프로세스를 io 작업을 위한 ioQ에 할당한다.

    이러한 반복적인 과정을 거쳐 Gantt chart의 형식으로 scheduling이 출력된다.

3. 각 프로세스 별 Turnaround Time과 Waiting Time을 출력한다. Burst Time은 input.txt에서 입력 받은 sequences of burst cycles를 통해 계산하며 Turnaround Time은 각 프로세스의 arrival time과 프로세스가 끝나는 시간의 차이를 통해 계산한다. 마지막으로 Waiting Time은 Turnaround Time에서 Burst Time을 빼서 계산한다.

4. 모든 프로세스의 Turnaround Time과 Waiting Time의 평균을 출력한다.

main함수를 도와주는 함수들의 기능은 다음과 같다.

- ps* setPs(int pid, int at, int initq, int cycle)

    함수 인자로 받은 정보를 토대로 프로세스를 생성한 후 return 한다.

- int move_queue(ps* ps)

  함수 인자로 받은 프로세스의 initq에 맞는 ready queue로 프로세스를 할당한다.

- int io_check()

  ioQ에 있는 프로세스들이 io burst를 마쳤다면 그 프로세스의 initq에 맞는 ready queue로 프로세스를 할당한다. 이 때, 기존의 ready queue가 Q3이 아니라면 initq가 1 감소하고 그에 맞는 프로세스를 할당한다.

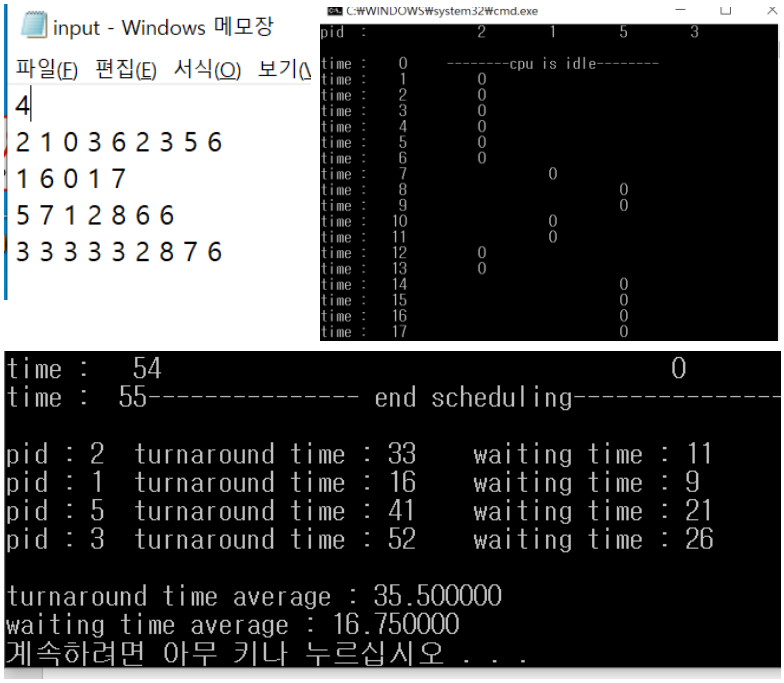
- ps* schedule()

  Ready Queue를 RQ_0부터 RQ_3까지 탐색한다. 만약 CPU-burst를 기다리는 프로세스가 Ready Queue 안에 존재한다면, 그에 맞는 time quantum 설정 후 프로세스를 return한다. Ready Queue의 우선순위는 고려사항에 서술되어 있다.

출력 예시 및 출력 보는 법

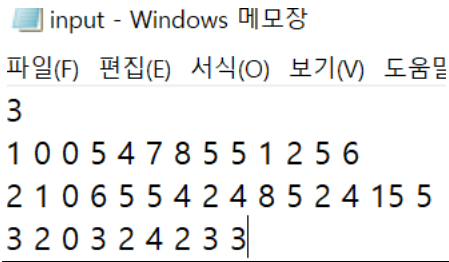
1. pid의 순서가 정렬되어 있지 않고, io burst를 진행하지 않는 프로세스가 있는 경우.

   input.txt와 이에 대한 결과는 다음과 같다.

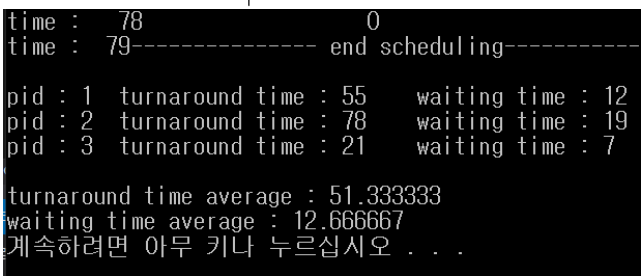


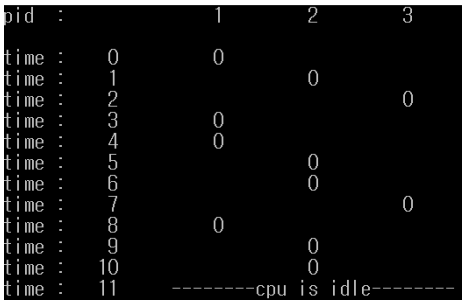
결과창의 경우 스크롤이 길어서 첫 부분과 끝부분만 편집하였다. 먼저, 위에 프로세스 별 pid들이 출력된다. 순서는 input.txt에 넣은 순서와 동일하다. 그 후 각 time 별로 cpu burst 중인 프로세스에 O 표시가 출력된다. 예를 들어 time: 1에 pid 2 열에 O가 나타났으므로, pid2 프로세스가 time 1~2 동안 CPU burst를 진행한 것이다. 또한, cpu burst 중인 프로세스가 없다면 "cpu is idle" 문구가 나타난다. scheduling에 대한 표가 출력이 된 후 각 프로세스 별 turnaround time과 waiting time이 출력된다. 마지막으로, 평균 turnaround time과 평균 waiting time이 출력되며 프로그램이 종료된다. 위의 경우 평균 turnaround time이 35.5, 평균 waiting time이 16.75이다.

2. Pid의 순서가 정렬되어 있고, io burst를 모든 프로세스가 진행할 경우.

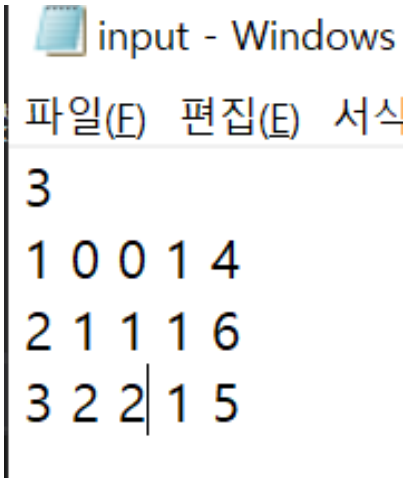


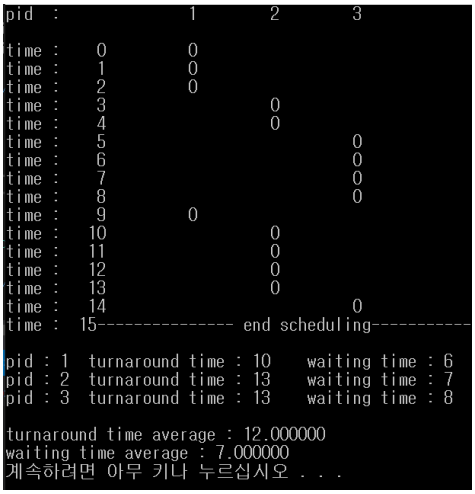
이 경우 평균 turnaround time이 51.3, 평균 waiting time이 12.6이 나왔다.





3. Pid의 순서가 정렬되어 있고, 모든 프로세스가 io burst를 진행하지 않을 경우





이 경우 io burst가 없기 때문에 "cpu is idle"문구가 출력되지 않았다. 평균 turnaround time이 12, 평균 waiting time이 7이었다.