실행

Windows) cmd 또는 PowerShell 실행 후 해당 폴더의 경로로 이동 -> make -> ./mfq

MacOS, Linux) Terminal 실행 후 해당 폴더의 경로로 이동 -> make -> ./mfq

개요

텍스트 파일 “*input.txt”* 에서 입력을 받아, 정해진 스케줄링 기법으로 시뮬레이션. 이후 Gantt Chart ,프로세스별 Turnaround Time, Wating Time, 전체 프로세스의 평균 Turnaround Time, 평균 Wating Time을 출력.

입력

1행) 프로세스의갯수

2행 이후) 각 프로세스 정보

1열) 해당 프로세스 **ID**

2열) 해당 프로세스 최초진입큐

3열) 해당 프로세스 A**rrival Time**

4열) 해당 프로세스 **Cycle** 수

5열 이후) **CPU burst time**, **I/O burst time**

(주의) 항상 CPU burst 종료되어야 함.

출력

Gantt Chart

프로세스별 Turnaround Time, Wating Time

전체 프로세스의 평균 Turnaround Time, 평균 Wating Time

설계및구현

Q0) Round-Robin, Time Quantum = 2

Q1) Round-Robin, Time Quantum = 4

Q2) Shortest Process Next

우선 순위는 Q0 -> Q1 -> Q2

**< main.h >**

시뮬레이션에 필요한 정적 라이브러리와 사용자 정의 구조체가 명시.

**< main.c >**

main 함수에서 다음과 같이 실행.

1. 파일 입력을 통한 프로세스 정보를 얻음.
2. 시뮬레이션을 실시.
3. 결과 출력.
4. 메모리 해제 및 종료.

**< setter.c >**

set\_sim() 함수에서 파일을 읽고, 프로세스 구조체에 메모리를 할당하여 정보 입력.

**< simulator.c >**

start\_sim() 함수에서 시뮬레이션을 다음과 같이 실행.

1. job\_queue에 있는 프로세스의 도착 시간을 확인하고 ready queue로 push.
2. I/O를 요청한 프로세스가 있다면, I/O burst time을 확인하고 ready queue로 push.
3. 실행중인 프로세스가 있는지 확인.

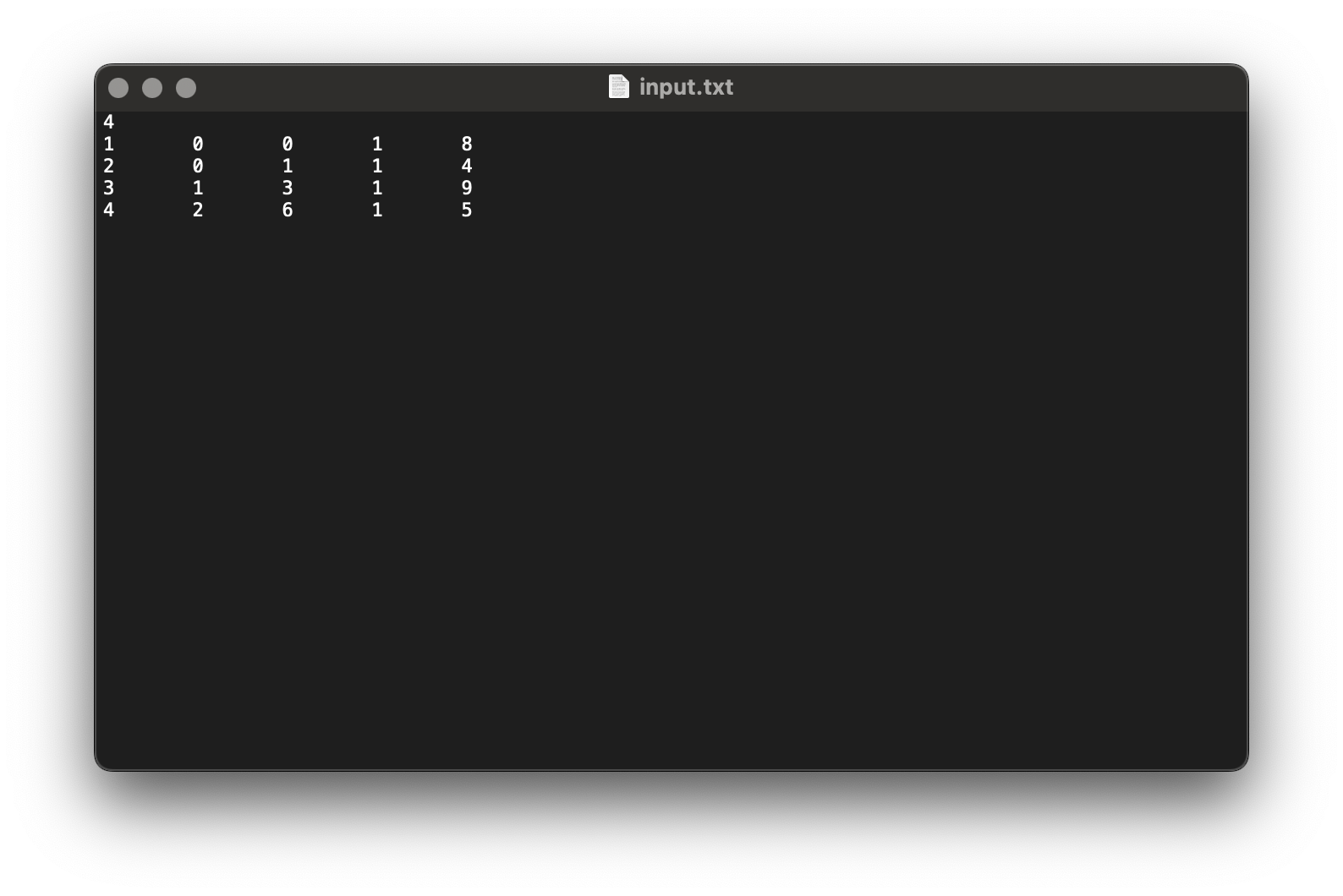
* 없다면, 새로운 프로세스를 스케줄링.
  + 새로운 프로세스를 스케줄링하는데 실패했고, 남아있는 프로세스가 없다면, 시뮬레이션 종료.
  + 새로운 프로세스를 스케줄링하는데 실패했고, I/O 요청을 대기중인 프로세스가 있다면, 대기.
* 실행중인 프로세스가 있다면, Q2에서 스케줄링받아 실행중인 프로세스인지 확인합니다.

1. 글로벌 시간을 1 증가.
2. I/O 요청을 대기중인 프로세스만 남았다면, 대기.
3. 실행중인 프로세스가 할당받은 시간을 1 소모.

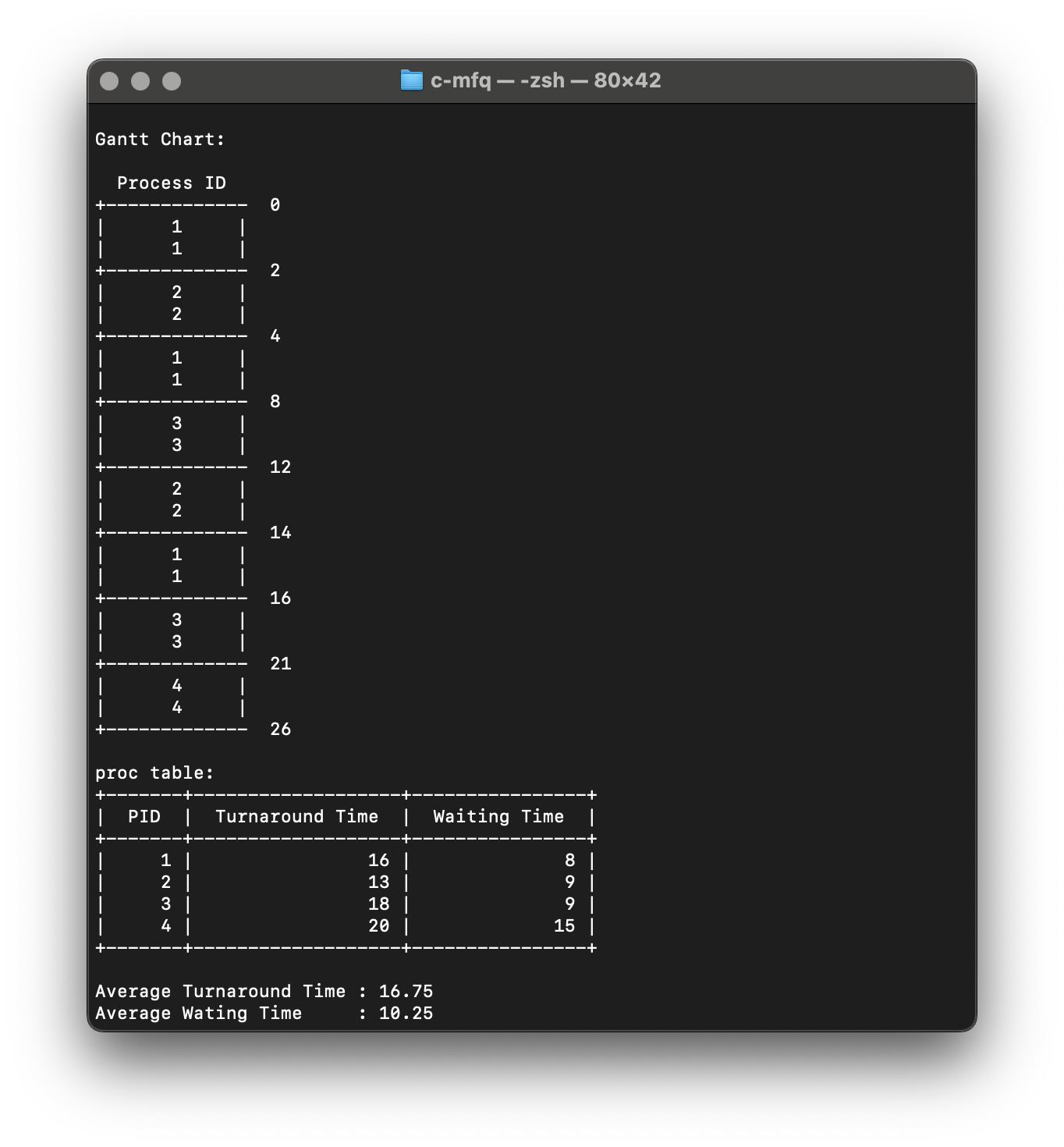
* 만약 할당받은 시간을 모두 소모했음에도 burst time이 남았다면, 적절한 ready queue로 push.
* I/O를 요청했다면, sleep queue로 push.
* 프로세스가 할당받은 시간을 모두 소모했고, burst time이 남아있지 않다면, 해당 프로세스를 종료.
* 이외의 경우, 반복.

결과

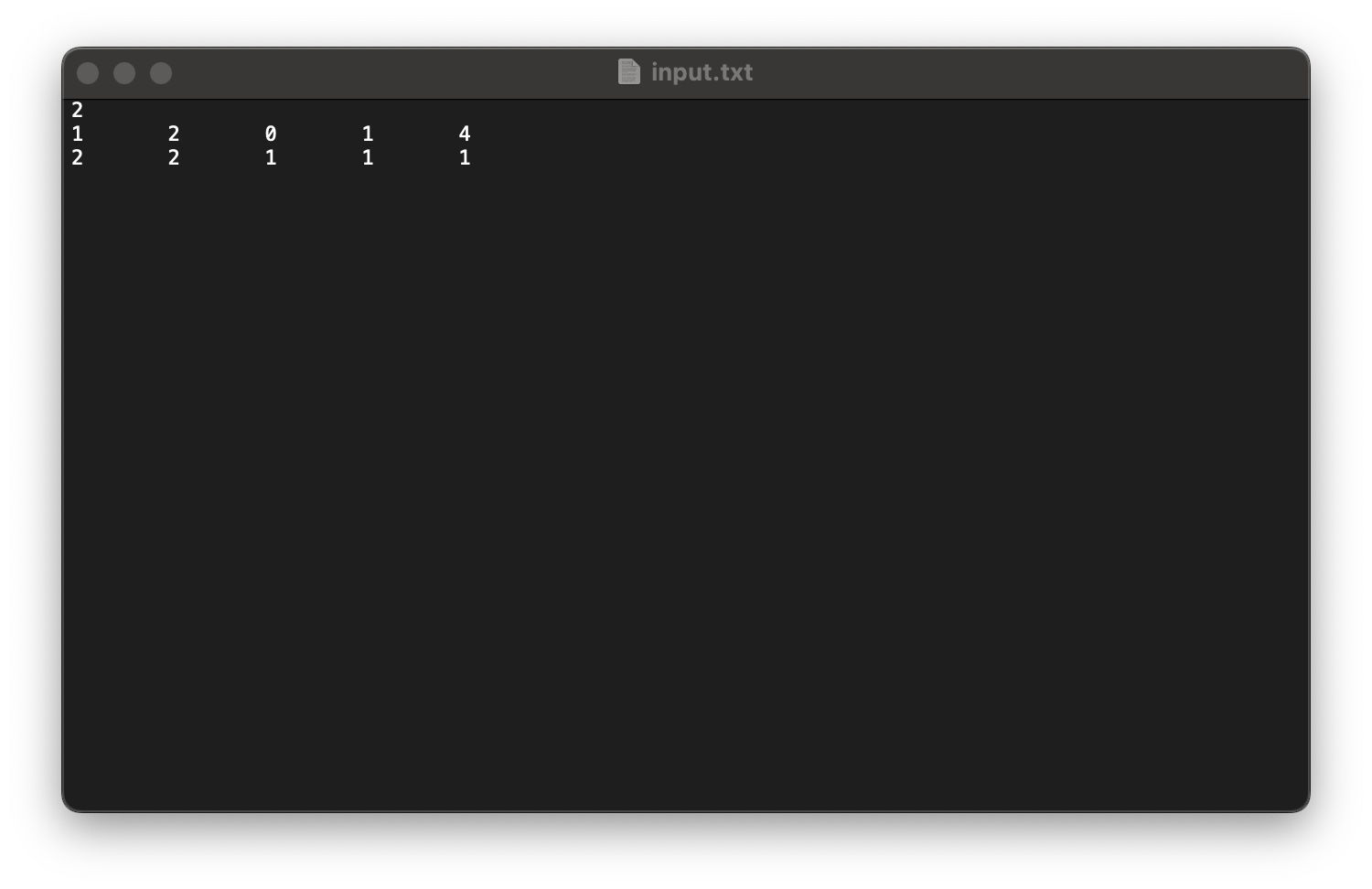
입력 1)



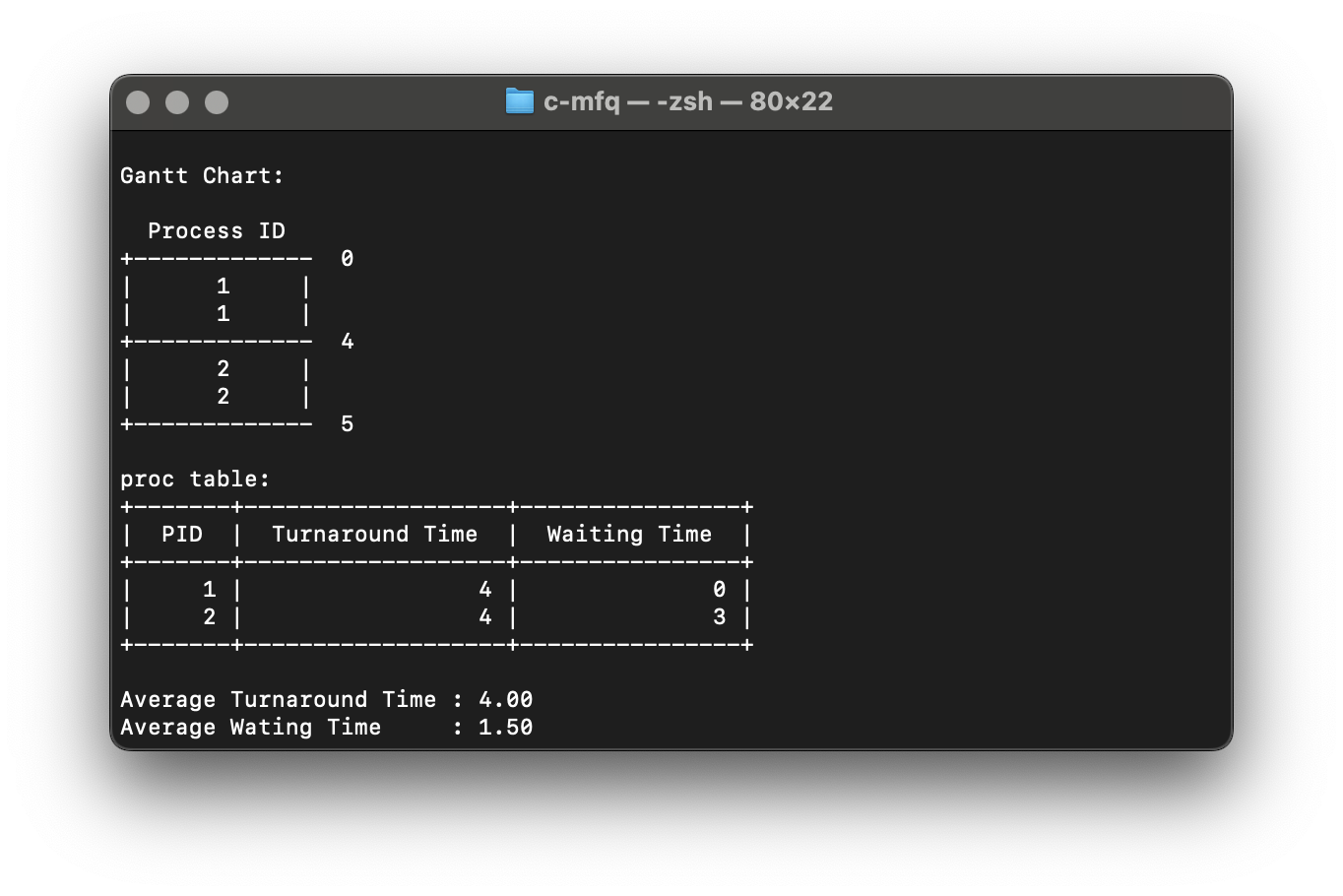
출력 1)

****

입력 2)

****

출력 2)

****