운영체제 기초

MFQ Scheduling 과제 보고서

2022312957 조준범

목차

1. 과제 분석 및 사용 도구
2. 해결 과정과 코드 설명
3. 3가지 예제에 대한 결과
4. 실행 결과 확인 방법
5. 결론

1. 과제 분석 및 사용 도구

본 과제에서 요구하는 바는 3개의 ReadyQueue를 가지는 MFQ 기반 스케쥴링을 구현하는 것이다. 각 큐를 Q0, Q1, Q2라 지칭하였을 때, 처음 두가지는 Round-Robin 기법을 따르되, Q0의 경우 time quantum이 2이고, Q1은 4이다.

Round-Robin은 FCFS(First-Come-First-Service)와 유사하게 먼저 큐에 진입한 순서대로 CPU를 할당하는 것인데, time quantum, 즉 CPU 사용 시간 제한을 두는 것이다. 이때 자신에게 할당된 모든 시간을 사용하였을 경우, 해당 프로세스는 running 상태에서 다시 ready 상태로 돌아오되 다음 큐(Q_{i+1})에 배치되게 된다.

마지막 큐인 Q2는 SPN 스케쥴링 기법을 적용하게 된다. SPN이란 Short-Process-Next의 줄임말로 큐에 진입한 순서가 아닌 큐에 존재하는 모든 프로세스들 중 잔여 시간이 가장 짧은 프로세스에게 가장 높은 우선순위를 부여하는 방식이다. Q2는 time quantum이 존재하지 않아 Q2를 벗어나 실행 상태에 도달한 프로세스는 종료될 때까지 CPU를 점유한다.

큐들 사이의 우선순위는 Q0 > Q1 > Q2로 지정되며, 하위 큐에 프로세스가 존재하여도 우선순위에 기반하여 상위 큐에서부터 스케쥴링이 진행된다.

프로세스 개수와 각 프로세스 별 도착시간 (arriveTime)과 수행시간 (burstTime)이 포함된 input.txt 파일을 입력으로 한다. 위의 조건대로 스케쥴링을 수행해 시간대별 CPU 점유 프로세스에 대한 정보, 종료 이후 각 프로세스 별 Turnaround Time, Waiting Time 및 전체적인 평균을 반환하며, Gantt Chart를 함께 출력하여 스케쥴링의 경과를 시각적으로 표현한다.

문제를 해결하기 위하여 실행 속도가 비교적 빠른 C언어를 사용해 구현하고자 하며, Visual Studio 2022 플랫폼에서 코드를 작성하였다. Intel 기반 CPU가 아닌 ARM 환경에서 개발하였음을 함께 고지한다.

2. 해결 과정과 코드 설명

입력을 받기 전, 프로그램은 두 가지 구조체를 정의하여 사용한다. 첫 번째 구조체는 개별 프로세스의 정보를 담기 위해 사용되며, 프로세스의 고유한 식별자인 id, 도착 시간인 arriveTime, 총 실행 시간을 의미하는 burstTime, 현재 남아 있는 실행 시간인 remainTime, 그리고 프로세스가 종료된 시점을 저장하는 finishTime과 같은 정수형 변수를 포함한다. 입력 파일을 통해 읽어들인 각 프로세스는 이 구조체를 기반으로 메모리에 선언되고, 이후 스케줄링 과정에서 큐를 통해 관리된다.

두 번째 구조체는 레디 큐의 정보를 저장하는 데 사용된다. 레디 큐는 실행 대기 중인 프로세스들을 저장하는 일종의 선입선출(FIFO) 구조로, 앞서 정의한 프로세스 구조체를 가리키는 포인터들을 배열 형태로 담고 있다. 또한 큐의 범위를 나타내기 위해 시작 위치와 종료 위치를 의미하는 start와 fin 변수를 함께 정의함으로써, 현재 큐에 존재하는 유효한 프로세스의 개수를 계산하거나, 새로운 프로세스를 추가하거나 제거하는 작업을 보다 쉽게 수행할 수 있도록 설계되었다.

프로그램이 시작되면 전역변수로 세 개의 레디 큐(Q0, Q1, Q2), 최대 100개의 프로세스를 담을 수 있는 구조체 배열, 현재 시간을 나타내는 변수 time, 그리고 입력된 전체 프로세스의 수를 저장할 processCount가 선언된다. 이 외에도 스케줄링이 진행되는 동안 각 시간 단위마다 CPU를 점유한 프로세스의 ID를 저장하기 위한 배열 gantt[]와, 전체 실행 시간(즉, Gantt 차트의 길이)을 의미하는 ganttCount 변수가 함께 정의된다. 이들 변수는 시뮬레이션 전반에 걸쳐 공유되며, 최종적으로 Gantt 차트를 통해 시각적인 결과를 출력하는 데 사용된다.

프로세스의 큐 출입을 반복적으로 수행해야 하므로, 프로세스를 큐에 추가하거나 제거하는 기능은 별도의 함수로 분리하여 구현되었다. 먼저 enqueue() 함수는 특정 레디 큐의 주소와 추가될 프로세스의 주소를 인자로 받아 큐의 끝에 해당 프로세스를 삽입하며, 큐의 종료 지점을 나타내는 fin 값을 1 증가시킨다. 단, Q2 큐의 경우에는 우선순위가 남은 실행 시간에 따라 달라지므로, 삽입 시 잔여 실행 시간을 기준으로 오름차순 정렬을 수행한다. 이를 위해 반복문과 조건문을 활용하여 큐 내부에서 프로세스 간의 위치를

적절히 조정하며, Q2는 결과적으로 SPN 스케줄링처럼 동작하게 된다.

반대로 dequeue() 함수는 지정된 큐의 가장 앞에 위치한 프로세스를 반환하고, 해당 큐의 시작 지점(start) 값을 1 증가시킴으로써 해당 프로세스를 더 이상 사용할 수 없도록 처리한다. 이와 함께 getQueueSize() 함수는 특정 큐 내에 남아 있는 유효한 프로세스의 개수를 반환하며, 이는 스케줄링 도중 큐가 비었는지 여부를 확인하거나 프로그램 종료 조건을 판단하는 데 사용된다. 또한 showQueue() 함수는 디버깅 및 상태 확인을 위해 큐 내에 존재하는 각 프로세스의 정보를 출력하는 기능을 제공하며, 스케줄링이 시작되기 전 Q0의 초기 상태를 확인하는 데 주로 사용된다.

프로그램 실행 시, loadFile() 함수는 외부 텍스트 파일로부터 프로세스의 수와 각 프로세스의 도착 시간 및 실행 시간을 입력받고, 이를 기반으로 각 프로세스를 초기화한 뒤 도착 시간 기준으로 정렬하여 Q0 큐에 삽입한다. 이는 시뮬레이션 시작 시점에 도착 시간 순으로 프로세스를 처리하기 위함이며, 앞서 설명한 것과 같이 각 레디 큐는 FIFO 구조로 설계되었기 때문이다.

이후 schedule() 함수에서는 전체 스케줄링이 수행된다. 스케줄링은 Q0, Q1, Q2로 구성된 멀티 레벨 피드백 큐 정책에 따라 실행되며, 각 큐는 서로 다른 타임퀀텀을 사용한다. 우선 Q0에서 도착 시간이 현재 시간 이하인 프로세스를 선택하여 실행하며, 이때 time quantum은 2이다. 프로세스가 주어진 시간 내에 완료되지 못할 경우 Q1으로 이동하게 되고, Q1에서는 time quantum 4를 사용한다. 마찬가지로 주어진 시간 내에 완료되지 않으면 Q2로 이동한다. Q2는 잔여 시간을 모두 소진할 때까지 실행하는 방식으로, SPN 기반의 정책을 따른다. 만약 Q0에 도착하지 않은 프로세스만 존재하고 나머지 큐가 모두 비어 있는 경우에는 CPU는 대기 상태로 진입하며 시간은 1씩 증가한다. 이 모든 실행 과정에서 각 시간 단위별로 실행 중인 프로세스의 ID는 gantt[] 배열에 기록된다. (CPU가 idle 상태인 경우 -1이 기록되어 구분이 가능토록 한다.)

Q0과 Q1의 라운드 로빈(Round Robin) 방식의 스케줄링은 schedule_RR() 함수에서 구현되며, Q0 또는 Q1에서 현재 실행 중인 프로세스를 quantum 만큼 실행한 뒤, 남은 시간이 있을 경우 다음 큐로 이관하고, 그렇지 않으면 프로세스를 종료 처리한다. 종료 시에는 해당 프로세스의 finishTime을 기록하며, Gantt 차트 역시 이에 따라 갱신된다.

모든 스케줄링이 완료되면 getResult() 함수가 호출되어 각 프로세스의 Turnaround Time(종료 시간 - 도착 시간)과 Waiting Time(대기 시간 = Turnaround Time - Burst Time)을 계산하여 출력하며, 전체 및 평균 값도 함께 계산된다. 마지막으로 ganttChart() 함수는 시간과 함께 Gantt 차트를 시각적으로 출력함으로써 각 프로세스가 어느 시점에 CPU를 점유했는지를 직관적으로 확인할 수 있도록 한다. 또한, 파일 입력부터 스케줄링까지의 모든 과정을 로그로 출력하여 프로그램의 흐름을 명확하게 확인할 수 있도록 한다.

3. 3가지 예제에 대한 결과

예제 1: 기본적인 예제 (프로세스 수 적고 도착 시간 다름)

[input.txt]
5
0 8
1 4
2 9
3 5
4 2

출력결과

-----input.txt-----
0번 프로세스 -> 도착시간 : 0 / 버스트시간 : 8
1번 프로세스 -> 도착시간 : 1 / 버스트시간 : 4
2번 프로세스 -> 도착시간 : 2 / 버스트시간 : 9
3번 프로세스 -> 도착시간 : 3 / 버스트시간 : 5
4번 프로세스 -> 도착시간 : 4 / 버스트시간 : 2

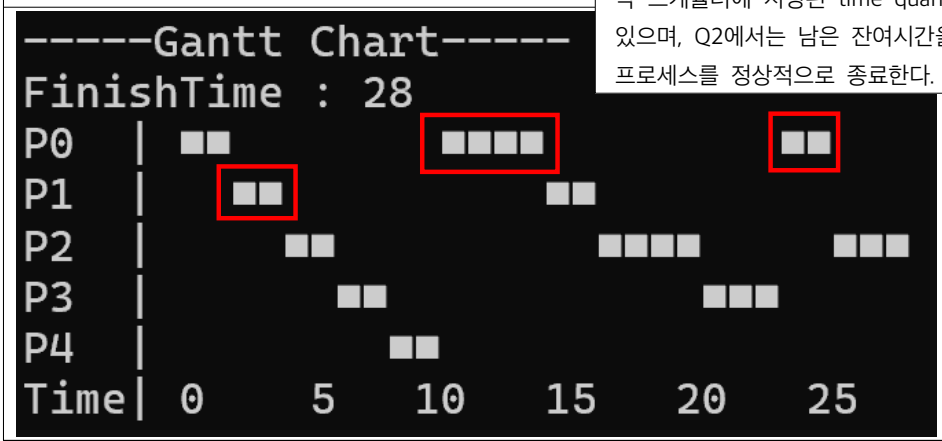
-----도착시간을 기준으로 재설정한 Q0 큐-----
할당된 프로세스 개수 : 5
0번 프로세스 -> 도착시간 : 0 / 버스트시간 : 8 / 남은시간 : 8
1번 프로세스 -> 도착시간 : 1 / 버스트시간 : 4 / 남은시간 : 4
2번 프로세스 -> 도착시간 : 2 / 버스트시간 : 9 / 남은시간 : 9
3번 프로세스 -> 도착시간 : 3 / 버스트시간 : 5 / 남은시간 : 5
4번 프로세스 -> 도착시간 : 4 / 버스트시간 : 2 / 남은시간 : 2

-----스케쥴링 시작-----
0 | 프로세스 0 Q0에서 스케쥴되어 실행 중..
2 | 프로세스 1 Q0에서 스케쥴되어 실행 중..
4 | 프로세스 2 Q0에서 스케쥴되어 실행 중..
6 | 프로세스 3 Q0에서 스케쥴되어 실행 중..
8 | 프로세스 4 Q0에서 스케쥴되어 실행 중..
10 | 프로세스 4 완료!
10 | 프로세스 0 Q1에서 스케쥴되어 실행 중..
14 | 프로세스 1 Q1에서 스케쥴되어 실행 중..
16 | 프로세스 1 완료!
16 | 프로세스 2 Q1에서 스케쥴되어 실행 중..
20 | 프로세스 3 Q1에서 스케쥴되어 실행 중..
23 | 프로세스 3 완료!
23 | 프로세스 0 Q2에서 스케쥴되어 실행 중..
25 | 프로세스 0 완료!
25 | 프로세스 2 Q2에서 스케쥴되어 실행 중..
28 | 프로세스 2 완료!
프로세스가 존재하지 않습니다.

-----결과 출력-----
프로세스 0 | TurnAroundTime : 25, WaitTime : 17
프로세스 1 | TurnAroundTime : 15, WaitTime : 11
프로세스 2 | TurnAroundTime : 26, WaitTime : 17
프로세스 3 | TurnAroundTime : 20, WaitTime : 15
프로세스 4 | TurnAroundTime : 6, WaitTime : 4

전체 TurnAroundTime : 92, 전체 WaitTime : 64
평균 TurnAroundTime : 18.40, 평균 WaitTime : 12.80

각 스케쥴러에 지정된 time quantum을 지키고 있으며, Q2에서는 남은 잔여시간을 모두 소비해 프로세스를 정상적으로 종료한다.



예제 2: 도착 시간 격차 큼 + burst time 다양

[input.txt]
7
0 6
3 2
10 4
12 8
15 3
20 7
25 5

출력결과

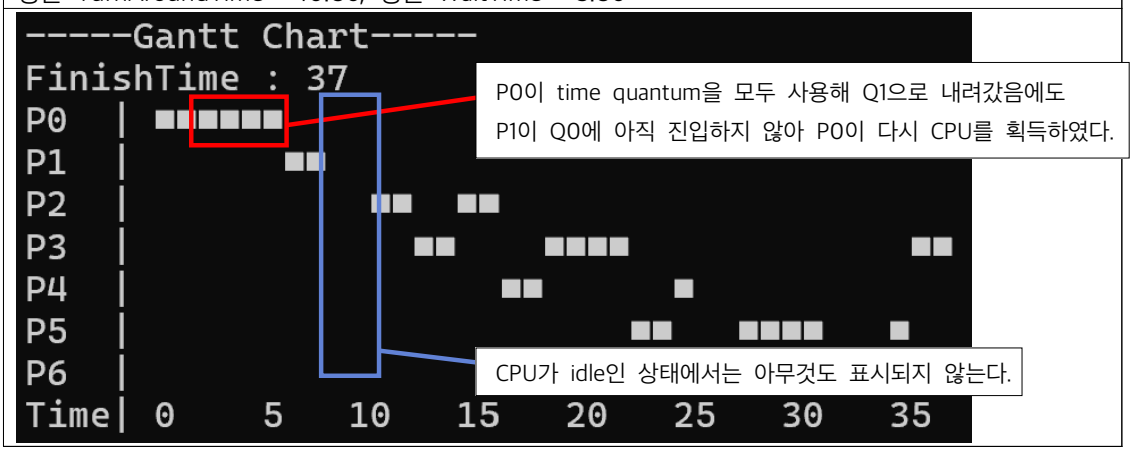
-----input.txt-----
0번 프로세스 -> 도착시간 : 0 / 버스트시간 : 6
1번 프로세스 -> 도착시간 : 3 / 버스트시간 : 2
2번 프로세스 -> 도착시간 : 10 / 버스트시간 : 4
3번 프로세스 -> 도착시간 : 12 / 버스트시간 : 8
4번 프로세스 -> 도착시간 : 15 / 버스트시간 : 3
5번 프로세스 -> 도착시간 : 20 / 버스트시간 : 7
6번 프로세스 -> 도착시간 : 25 / 버스트시간 : 5

-----도착시간을 기준으로 재설정한 Q0 큐-----
할당된 프로세스 개수 : 7
0번 프로세스 -> 도착시간 : 0 / 버스트시간 : 6 / 남은시간 : 6
1번 프로세스 -> 도착시간 : 3 / 버스트시간 : 2 / 남은시간 : 2
2번 프로세스 -> 도착시간 : 10 / 버스트시간 : 4 / 남은시간 : 4
3번 프로세스 -> 도착시간 : 12 / 버스트시간 : 8 / 남은시간 : 8
4번 프로세스 -> 도착시간 : 15 / 버스트시간 : 3 / 남은시간 : 3
5번 프로세스 -> 도착시간 : 20 / 버스트시간 : 7 / 남은시간 : 7
6번 프로세스 -> 도착시간 : 25 / 버스트시간 : 5 / 남은시간 : 5

-----스케줄링 시작-----
0 | 프로세스 0 Q0에서 스케줄되어 실행 중..
2 | 프로세스 0 Q1에서 스케줄되어 실행 중..
6 | 프로세스 0 완료!
6 | 프로세스 1 Q0에서 스케줄되어 실행 중..
8 | 프로세스 1 완료!
8 | CPU 대기 중..
9 | CPU 대기 중..
10 | 프로세스 2 Q0에서 스케줄되어 실행 중..
12 | 프로세스 3 Q0에서 스케줄되어 실행 중..
14 | 프로세스 2 Q1에서 스케줄되어 실행 중..
■ 중략
27 | 프로세스 5 Q1에서 스케줄되어 실행 중..
31 | 프로세스 6 Q1에서 스케줄되어 실행 중..
34 | 프로세스 6 완료!
34 | 프로세스 5 Q2에서 스케줄되어 실행 중..
35 | 프로세스 5 완료!
35 | 프로세스 3 Q2에서 스케줄되어 실행 중..
37 | 프로세스 3 완료!
프로세스가 존재하지 않습니다.

-----결과 출력-----
프로세스 0 | TurnAroundTime : 6, WaitTime : 0
프로세스 1 | TurnAroundTime : 5, WaitTime : 3
프로세스 2 | TurnAroundTime : 6, WaitTime : 2
프로세스 3 | TurnAroundTime : 25, WaitTime : 17
프로세스 4 | TurnAroundTime : 10, WaitTime : 7
프로세스 5 | TurnAroundTime : 15, WaitTime : 8
프로세스 6 | TurnAroundTime : 9, WaitTime : 4

전체 TurnAroundTime : 76, 전체 WaitTime : 41
평균 TurnAroundTime : 10.86, 평균 WaitTime : 5.86



예제 3: 도착 시간 분산 + burst time 중간 규모 혼재

[input.txt]
7
3 6
0 5
4 3
1 9
2 4
5 7
6 2

출력결과

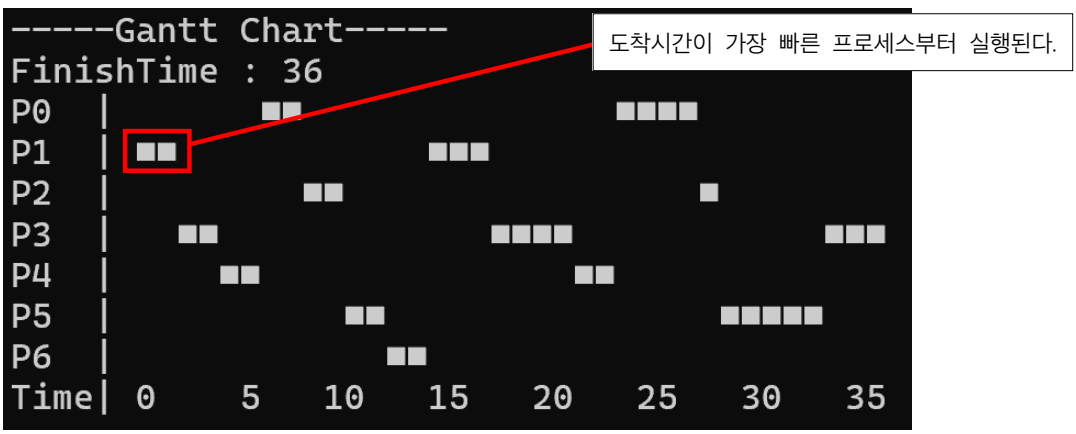
-----input.txt-----
0번 프로세스 -> 도착시간 : 3 / 버스트시간 : 6
1번 프로세스 -> 도착시간 : 0 / 버스트시간 : 5
2번 프로세스 -> 도착시간 : 4 / 버스트시간 : 3
3번 프로세스 -> 도착시간 : 1 / 버스트시간 : 9
4번 프로세스 -> 도착시간 : 2 / 버스트시간 : 4
5번 프로세스 -> 도착시간 : 5 / 버스트시간 : 7
6번 프로세스 -> 도착시간 : 6 / 버스트시간 : 2

-----도착시간을 기준으로 재설정한 Q0 큐-----
할당된 프로세스 개수 : 7
1번 프로세스 -> 도착시간 : 0 / 버스트시간 : 5 / 남은시간 : 5
3번 프로세스 -> 도착시간 : 1 / 버스트시간 : 9 / 남은시간 : 9
4번 프로세스 -> 도착시간 : 2 / 버스트시간 : 4 / 남은시간 : 4
0번 프로세스 -> 도착시간 : 3 / 버스트시간 : 6 / 남은시간 : 6
2번 프로세스 -> 도착시간 : 4 / 버스트시간 : 3 / 남은시간 : 3
5번 프로세스 -> 도착시간 : 5 / 버스트시간 : 7 / 남은시간 : 7
6번 프로세스 -> 도착시간 : 6 / 버스트시간 : 2 / 남은시간 : 2

-----스케쥴링 시작-----
0 | 프로세스 1 Q0에서 스케쥴되어 실행 중..
2 | 프로세스 3 Q0에서 스케쥴되어 실행 중..
4 | 프로세스 4 Q0에서 스케쥴되어 실행 중..
6 | 프로세스 0 Q0에서 스케쥴되어 실행 중..
8 | 프로세스 2 Q0에서 스케쥴되어 실행 중..
10 | 프로세스 5 Q0에서 스케쥴되어 실행 중..
12 | 프로세스 6 Q0에서 스케쥴되어 실행 중..
14 | 프로세스 6 완료!
14 | 프로세스 1 Q1에서 스케쥴되어 실행 중..
■ 중략
27 | 프로세스 2 Q1에서 스케쥴되어 실행 중..
28 | 프로세스 2 완료!
28 | 프로세스 5 Q1에서 스케쥴되어 실행 중..
32 | 프로세스 5 Q2에서 스케쥴되어 실행 중..
33 | 프로세스 5 완료!
33 | 프로세스 3 Q2에서 스케쥴되어 실행 중..
36 | 프로세스 3 완료!
프로세스가 존재하지 않습니다.

-----결과 출력-----
프로세스 0 | TurnAroundTime : 24, WaitTime : 18
프로세스 1 | TurnAroundTime : 17, WaitTime : 12
프로세스 2 | TurnAroundTime : 24, WaitTime : 21
프로세스 3 | TurnAroundTime : 35, WaitTime : 26
프로세스 4 | TurnAroundTime : 21, WaitTime : 17
프로세스 5 | TurnAroundTime : 28, WaitTime : 21
프로세스 6 | TurnAroundTime : 8, WaitTime : 6

전체 TurnAroundTime : 157, 전체 WaitTime : 121
평균 TurnAroundTime : 22.43, 평균 WaitTime : 17.29



4. 실행 결과 확인 방법

본 과제를 C언어로 수행했기에 프로젝트 파일 전체를 함께 첨부한다. Visual Studio 2022와 같은 컴파일러를 이용해 이를 불러오고, [2022312957_조준범_MFQScheduling.sln] 솔루션 파일을 실행함으로써 (main.c를 실행하면 아무 출력도 발생하지 않는다.) 전체 프로그램의 흐름과 출력 결과를 확인할 수 있다. 프로그램은 input.txt 파일을 기반으로 동작하므로, 실행 전 동일 디렉터리에 해당 입력 파일이 존재해야 한다. 실행 결과는 콘솔 창에 순차적으로 출력되며, 스케쥴링 과정과 최종 통계 결과를 포함한다.

5. 결론

과제 수행을 위해 MFQ 스케줄링 알고리즘을 C 언어로 구현하고, 다양한 테스트 케이스를 통해 그 동작을 확인하였다. 구현 과정에서의 설계 의도와 알고리즘 흐름을 명확히 설명하고자 하였으며, 출력 결과를 통해 스케줄링 결과가 정확히 반영되었음을 확인할 수 있었다.

MFQ 스케줄러를 직접 구현하며, 프로세스가 큐 간을 어떻게 이동하고, 우선순위와 시간 분할이 실제로 어떤 영향을 주는지를 체감할 수 있었다. 또한 운영체제의 핵심 개념 중 하나인 CPU 스케줄링에 대한 실질적인 이해를 높이는 계기가 되었다.