KOREA

[COSE341-01] Operating Systems

Term Project

산업경영공학부 2020170856 이우진

고려대학교
KOREA UNIVERSITY

I. 서론

1.1. CPU 스케줄러 개요

CPU 스케줄러는 운영체제의 핵심 구성 요소 중 하나로, 시스템에서 실행 가능한 여러 프로세스 중 하나를 선택하여 CPU를 할당하는 역할을 한다. 이는 시스템의 효율성을 높이고, 응답 시간을 최소화하며, 공정한 자원 분배를 보장하기 위해 필수적입니다. CPU 스케줄러는 크게 비선점형(non-preemptive) 스케줄러와 선점형(preemptive) 스케줄러로 나눌 수 있다.

1.1.1. 비선점형 스케줄링 (Non-Preemptive Scheduling)

비선점형 스케줄링에서는 현재 실행 중인 프로세스가 스스로 종료하거나 I/O 작업을 위해 스스로 CPU를 양보하지 않는 한, 다른 프로세스가 CPU를 사용할 수 없다. 비선점형 스케줄링 알고리즘의 주요 유형은 다음과 같다.

1. FCFS (First-Come, First-Served)

    프로세스가 도착한 순서대로 CPU를 할당한다.

    구현이 간단하지만, 긴 프로세스가 먼저 도착하면 전체 시스템의 응답 시간이 길어질 수 있다. 이를 Convoy Effect라고 한다.

2. SJF (Shortest Job First)

    CPU 버스트 타임이 가장 짧은 프로세스를 먼저 실행한다.

    이론적으로 최적의 평균 대기 시간을 제공하지만, 각 프로세스의 CPU 버스트 타임을 미리 알아야 하는 단점이 있다.

3. Priority Scheduling

    각 프로세스에 우선순위를 부여하고, 우선순위가 높은 프로세스에 먼저 CPU를 할당한다.

    우선순위가 낮은 프로세스는 오랫동안 CPU를 할당 받지 못하는 Starvation 문제가 발생할 수 있다. 이를 해결하기 위해 Aging 기법이 사용되기도 한다.

1.1.2. 선점형 스케줄링 (Preemptive Scheduling)

선점형 스케줄링에서는 현재 실행 중인 프로세스가 CPU를 독점하지 않으며, 더 높은 우선순위를 가진 프로세스가 도착하면 현재 프로세스를 중단시키고 CPU를 할당할 수 있다. 선점형 스케줄링

알고리즘의 주요 유형은 다음과 같다.

1.Round Robin (RR)

각 프로세스는 동일한 크기의 time quantum을 할당 받아 순환한다.

공정성을 보장하지만, time quantum의 크기에 따라 context switch 오버헤드가 발생할 수 있다.

2. Preemptive SJF (Shortest Remaining Time First, SRTF)

가장 짧은 남은 실행 시간을 가진 프로세스에 CPU를 할당한다.

SJF와 유사하지만, 실행 중인 프로세스를 선점할 수 있어 더 효율적인 스케줄링이 가능하다.

3. Preemptive Priority Scheduling

우선순위가 높은 프로세스가 도착하면 현재 실행 중인 프로세스를 선점한다.

우선순위가 낮은 프로세스는 여전히 Starvation 문제를 겪을 수 있으며, Aging 기법을 사용하여 이를 완화할 수 있다.

1.1.3. 스케줄러의 성능 평가 기준

1. CPU 이용률 (CPU Utilization)

CPU가 작업을 처리하는 시간의 비율을 나타낸다. 높을수록 효율적이다.

2. 응답 시간 (Response Time)

프로세스가 처음으로 CPU를 할당 받은 시점까지의 시간을 의미한다. 짧을수록 좋다.

3. 대기 시간 (Waiting Time)

프로세스가 준비 큐에서 기다리는 총 시간을 의미한다. 짧을수록 효율적이다.

4. 반환 시간 (Turnaround Time)

프로세스가 제출된 시점부터 완료된 시점까지의 시간을 의미한다. 짧을수록 좋다.

1.2. 큐 (Queue)

CPU 스케줄러는 프로세스를 효과적으로 관리하기 위해 큐(Queue)를 사용한다. 큐는 FIFO(First In, First Out) 원칙에 따라 데이터가 삽입되고 제거되는 자료 구조이다. 다양한 스케줄링 알고리즘은 각기 다른 종류의 큐를 사용하여 프로세스를 정리하고 처리한다.

1.2.1. 준비 큐 (Ready Queue)

준비 큐는 CPU 할당을 기다리는 프로세스들을 저장하는 큐이다. 모든 프로세스는 생성된 후 준비 큐에 들어가며, 스케줄러는 준비 큐에서 프로세스를 선택하여 CPU를 할당한다.

준비 큐는 모든 스케줄링 알고리즘에서 사용되며, 프로세스가 도착하면 그 순서에 맞게 준비 큐에 들어가게 된다.

1.2.2. 대기 큐 (Waiting Queue)

대기 큐는 I/O 작업을 기다리는 프로세스를 저장한다. 프로세스가 I/O 작업을 요청하면 준비 큐에서 제거되어 대기 큐에 들어가고, I/O 작업이 완료되면 다시 준비 큐로 돌아간다.

대기 큐는 비선점형 및 선점형 스케줄링 알고리즘 모두에서 중요한 역할을 한다. I/O 바운드 프로세스는 주로 대기 큐에 머무르며, 이를 통해 CPU 바운드 프로세스와의 균형을 맞춘다.

1.3. 구현한 스케줄러에 대한 요약 설명

먼저, 사용자로부터 process의 개수, 각 process의 PID, CPU Burst Time, Arrival Time, Priority를 입력 받고, 마지막으로 Round Robin scheduling을 위한 time quantum을 입력 받는다. I/O 발생은 랜덤으로 1회 발생하게 하였다. 이때, I/O 발생 타이밍은 CPU burst 중간에 발생하게 하였으며, I/O Burst Time은 1~3 범위의 랜덤한 수로 설정하였다.

입력 받은 정보를 바탕으로 다음과 같은 스케줄링 알고리즘들을 실행하고, Gantt 차트를 비롯한 스케줄러의 성능 결과를 출력한다. 아래에 CPU scheduling simulator에 구현한 알고리즘을 간략하게 소개하고자 한다.

1. FCFS (First-Come, First-Served): 도착 순서대로 프로세스를 실행하는 가장 간단한 비선점형 스케줄링 알고리즘

2. Non-Preemptive SJF (Shortest Job First): 가장 짧은 CPU Burst Time을 가진 프로세스를 먼저 실행하는 비선점형 스케줄링 알고리즘

3. Non-Preemptive Priority: 우선순위가 높은 프로세스를 먼저 실행하는 비선점형 스케줄링 알고리즘

4. Round Robin: 각 프로세스가 일정 시간(quantum) 동안 순서대로 CPU를 할당 받는 선점형 스케줄링 알고리즘

5. Preemptive SJF: 실행 중인 프로세스보다 더 짧은 CPU Burst Time을 가진 프로세스가 도착하면 CPU를 선점하는 스케줄링 알고리즘

6. Preemptive Priority: 실행 중인 프로세스보다 높은 우선순위를 가진 프로세스가 도착하면 CPU를 선점하는 스케줄링 알고리즘

7. Priority-based Round Robin: 우선순위 큐에 따라 프로세스를 분류하고, 각 우선순위 큐 내에서 Round Robin 방식을 적용하는 스케줄링 알고리즘

이때, 구현한 스케줄러에는 Waiting Queue와 Ready Queue가 존재한다. 프로세스가 생성될 때 각각 인덱스가 부여되며, 해당 인덱스를 사용하여 프로세스를 큐에 삽입하고 제거한다. 각 알고리즘은 이 두 개의 큐를 사용하여 프로세스를 관리하게 된다.

이와 같이 다양한 스케줄링 알고리즘을 통해 각 프로세스의 Waiting Time, Turnaround Time 등의 성능 지표를 평가하고, 이를 Gantt 차트로 시각화 하여 출력한다. 이를 통해 각 알고리즘의 특성과 장단점을 분석하고, 다양한 상황에서의 성능을 비교할 수 있다.

II. 본론

2.1. 다른 CPU 스케줄링 시뮬레이터에 대한 소개 (CPUSim)[1]

CPU 스케줄링 시뮬레이터는 운영체제에서 프로세스들을 효율적으로 관리하기 위해 사용되는 알고리즘들을 시뮬레이션 하는 프로그램이다. 이러한 시뮬레이터는 다양한 스케줄링 알고리즘을 구현하고, 프로세스의 동작과 CPU 할당을 실시간으로 시각화 하여 사용자에게 보여준다.

대표적인 CPU 스케줄링 시뮬레이터로 CPUSim이 존재한다. CPUSim은 CPU 스케줄링 알고리즘을 이해하고 학습하는 데 도움을 주기 위해 설계된 시뮬레이션 소프트웨어이다. 이 도구는 교육 목적으로 사용되며, 다양한 스케줄링 알고리즘을 시각적으로 이해하고 비교하는 데 유용하다. 다음은 CPUSim의 주요 기능과 특징에 대한 설명이다.

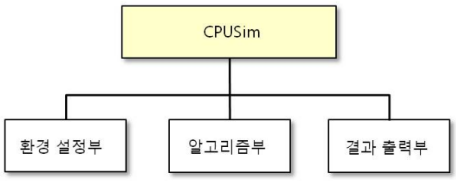

그림 1. CPUSim의 구성
Fig. 1 The construction of CPUSim

CPUSim은 위의 그림과 같이 환경 설정부, 알고리즘부, 결과 출력부 세 부분으로 구성된다.

2.1.1. 환경 설정부

먼저, 환경 설정부는 CPUSim의 구동에 필요한 시뮬레이션 매개변수 설정 기능을 제공한다. 설정 가능한 매개변수들은 프로세스 생성 방법과 생성할 프로세스 수이다. '자동'을 선택하고 프로세스 수를 입력하면 설정된 수만큼 프로세스 관련 정보(프로세스명, 도착시간, 서비스 시간, 우선순위)들이 생성된다. '수동'을 선택하면 사용자가 해당 정보를 직접 입력한다. RR 알고리즘을 선택하면, Quantum time을 추가로 입력한다. 또한, 매개변수의 값을 재설정할 수 있는 기능도 제공한다.

2.1.2. 알고리즘부

CPUSim은 대표적인 6가지 알고리즘(FCFS, HRN, SJF, SRT, 우선순위, 라운드 로빈)을 지원한다. 사용자가 매개변수의 설정을 마친 후 알고리즘을 선택하고 '실행' 버튼을 누르면 시뮬레이터가 구동된다.

---

[1] 고정국. (2012). CPUSim: CPU 스케줄링 알고리즘 교육을 지원하는 시뮬레이터. 한국정보통신학회 논문지, 16(4), 835-842. DBpia.

2.1.3. 결과 출력부

결과 출력부는 설정된 프로세스 관련 정보에 근거하여 알고리즘의 처리 과정을 단계별로 보여준다. 즉, 프로세스들의 CPU 사용시간을 Gantt 차트로 출력한 후 해당 알고리즘의 성능 수치(평균 대기시간과 평균 반환시간)를 표시한다.

이렇듯, CPUSim은 직관적인 인터페이스와 다양한 기능을 통해 CPU 스케줄링 알고리즘을 효과적으로 학습하고 이해하는 데 큰 도움을 주며, 대학교 등에서 운영체제 수업의 일환으로 사용된다. 따라서 교육자와 학생 모두에게 유용한 도구라고 할 수 있다.

2.2. 구현한 시뮬레이터의 시스템 구성도

구현한 시뮬레이터에는 여러 스케줄링 알고리즘을 구현하기 위해 필요한 다양한 함수와 자료 구조들이 정의되어 있다. 또한, 각 알고리즘을 구현하고 출력하기 위한 기능들도 존재한다. 각 주요 구성 요소는 다음과 같다.

2.2.1. 프로세스 구조체

프로세스 구조체는 각 프로세스의 기본 정보를 저장하기 위한 자료구조이다. PID, CPU Burst Time, Arrival Time, Priority 등의 기본 속성을 가지고 있고, 스케줄링 알고리즘 및 성능 평가를 위해 waiting time, turnaround time, 남은 CPU burst time 정보도 가지고 있다. 또한, arrival time이 되었을 때, Ready Queue로 들어갔는지 확인하기 위한 ready completed라는 Bool 타입 변수도 추가하였고, I/O operation을 위한 I/O Burst Time 및 Start Time과 같은 정보도 추가하였다.

2.2.2. 큐 구조체

큐 구조체는 프로세스를 관리하기 위한 자료 구조로, Ready Queue와 Waiting Queue, Gantt Queue가 포함된다. Ready Queue는 CPU 할당을 기다리는 프로세스들을 관리하며, Waiting Queue는 I/O 작업을 대기하는 프로세스들을 관리한다. 추가로 Gantt차트 출력을 위한 Gantt Queue도 만들었는데, 매 time마다 실행한 process의 인덱스를 큐에 넣고, 알고리즘이 끝나면, 해당 큐를 확인하여 Gantt차트 출력을 하게 하였다.

큐에 front, rear 속성이 존재하는데, 이때, front는 큐의 맨 앞에 위치한 노드를 가리키는 포인터이고, rear는 큐의 맨 뒤에 위치한 노드를 가리키는 포인터이다. 큐는 노드로 구성되어 있는데, 각

노드는 데이터와 다음 노드를 가리키는 포인트로 구성되며, 데이터는 프로세스의 인덱스가 된다.

마지막으로 큐 동작을 위한 getQueueSize, initQueue, isEmpty, enqueue, dequeue 함수들이 존재한다.

2.2.3. 프로세스 생성 및 초기화 함수

Create_Process() 함수: 사용자로부터 PID, Burst Time, Arrival Time, Priority를 입력 받아 프로세스의 초기 상태를 설정한다. 또한, CPU Burst Time에 따라 랜덤으로 I/O Burst time과 I/O Start time을 생성한다.

Process_init() 함수: 새로운 알고리즘이 시작될 때마다, 사용자가 입력한 정보를 다시 불러오기 위한 함수이다. 프로세스 속성들인 waiting time, around time, remaining time, I/O remaining time을 초기화하고, Ready Queue에 들어갔는지 확인하기 위한 변수는 다시 False로 바꿔준다.

2.2.4. 스케줄링 알고리즘 함수

다양한 스케줄링 알고리즘을 구현한 함수들이 포함되어 있다. 각 함수는 특정한 스케줄링 정책을 따라 프로세스를 처리하고 결과를 출력한다. 구현된 함수들은 다음과 같다.

1. FCFS(): 선입선출(FCFS) 스케줄링을 구현하였다. 프로세스가 도착한 순서대로 CPU를 할당하여 처리한다.

2. SJF(): 최단 작업 우선(SJF) 스케줄링을 구현하였다. CPU Burst Time이 가장 짧은 프로세스를 우선적으로 실행한다.

3. Priority(): 우선순위 스케줄링을 구현하였다. 우선순위가 높은 프로세스를 먼저 실행한다.

4. Preem_SJF(): 선점형 SJF 스케줄링을 구현하였다. 더 짧은 CPU Burst Time을 가진 프로세스가 도착하면 실행 중인 프로세스를 선점한다.

5. Preem_Priority(): 선점형 우선순위 스케줄링을 구현하였다. 더 높은 우선순위를 가진 프로세스가 도착하면 실행 중인 프로세스를 선점한다.

6. RoundRobin(): Round Robin 스케줄링을 구현하였다. 고정된 시간 양자(Quantum)만큼 각 프로세스를 순환하면서 실행한다.

7. RoundRobin_priority(): 우선순위 기반 Round Robin 스케줄링을 구현하였다. 우선순위가 높은 프로세스에게 우선적으로 CPU를 할당하며, 같은 우선순위를 가진 프로세스들에게는 time quantum을 적용한다.

2.2.5. 출력 함수

결과를 출력하기 위한 3가지 함수를 정의하였다.

print_table() 함수: 각 프로세스의 정보(PID, Burst Time, Arrival Time, Priority, I/O Burst Time, I/O start Time, Waiting time, Turnaround time)을 테이블 형태로 출력하게 하였다.

printGanttChart() 함수: Gantt 차트를 출력하여 시간 결과에 따른 프로세스 실행 순서를 시각적으로 보여준다. 이때, GanttQueue를 확인하여, 프로세스의 실행 순서를 확인한다.

evaluate() 함수: 프로세스들의 평균 Waiting time과 평균 Turnaround time을 계산하고 출력한다.

2.2.6. 메인 함수

프로그램의 시작점으로, 위에서 정의된 함수를 활용하여 사용자로부터 프로세스 수와 각각의 프로세스의 정보들을 입력 받고, 프로세스 배열을 할당한다. time quantum까지 입력 받은 후, 각 스케줄링 알고리즘 함수를 호출하고, 마지막으로 할당된 메모리를 해제한다.

2.3. 각 모듈에 대한 설명

다음은 각 스케줄링 알고리즘에 대한 구체적인 설명으로, Pseudo Code로 작성하였다.

2.3.1. FCFS 스케줄링 알고리즘

1. 변수 및 큐 초기화

   - `current_time`, `completed_processes`를 0 으로 초기화.

   - `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)

   - 사용자가 입력한 정보들로 process 속성 초기화.

   - readyQueue, waitingQueue, ganttQueue 초기화.

2. 프로세스 도착 시간 기준으로 정렬

   - 프로세스 리스트를 `arrive_time` 기준으로 정렬.

3. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

  각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

    - 프로세스의 `remaining_time`을 1 감소.

    - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

    - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

  if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

    - ready 큐에 가장 먼저 들어온 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

- 간트 Queue 업데이트

  현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

  `current_time`을 1 증가.

4. 결과 출력

- 간트 차트 출력

- 프로세스 테이블 출력

- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.2. SJF 스케줄링 알고리즘

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.

- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)

- 사용자가 입력한 정보들로 process 속성 초기화.

- readyQueue, waitingQueue, ganttQueue 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

  각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

  - 프로세스의 `remaining_time`을 1 감소.

  - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

  - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

  if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

  - ready 큐에서 '남은 cpu burst time'이 가장 짧은 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

- 간트 Queue 업데이트

  현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

  `current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력

- 프로세스 테이블 출력

- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.3. Priority 스케줄링 알고리즘 (priority 값이 작을수록 우선순위가 높다)

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.

- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)

- 사용자가 입력한 정보들로 process 속성 초기화.

- readyQueue, waitingQueue, ganttQueue 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

  각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

  - 프로세스의 `remaining_time`을 1 감소.

- 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

- 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

- ready 큐에서 우선순위가 가장 높은 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

- 간트 Queue 업데이트

현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

`current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력
- 프로세스 테이블 출력
- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.4. Preem_SJF 스케줄링 알고리즘

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.
- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)
- 사용자가 입력한 정보들로 process 속성 초기화.
- readyQueue, waitingQueue, ganttQueue 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

    - 프로세스의 `remaining_time`을 1 감소.

    - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

    - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

  선점 조건 확인 (ready 큐에서 '남은 cpu burst time'이 가장 짧은 프로세스를 찾기)

  if 위의 프로세스가 현재 실행 중인 process 보다 '남은 cpu burst time'가 짧은 경우

    - 현재 실행 중인 프로세스를 ready 큐에 추가하고, 해당 프로세스를 꺼내 실행.

    - 해당 프로세스의 waiting time 을 업데이트.

  if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

    - ready 큐에서 '남은 cpu burst time'이 가장 짧은 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

- 간트 Queue 업데이트

  현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

  `current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력

- 프로세스 테이블 출력

- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.5. Preem_Priority 스케줄링 알고리즘 (priority 값이 작을수록 우선순위가 높다)

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.
- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)
- 사용자가 입력한 정보들로 process 속성 초기화.
- readyQueue, waitingQueue, ganttQueue 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

  각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

  - 프로세스의 `remaining_time`을 1 감소.
  - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

  - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

  선점 조건 확인 (ready 큐에서 우선순위가 가장 높은 프로세스를 찾기)

  if 위의 프로세스가 현재 실행 중인 process 보다 우선순위가 높은 경우

- 현재 실행 중인 프로세스를 ready 큐에 추가하고, 해당 프로세스를 꺼내 실행.

- 해당 프로세스의 waiting time 을 업데이트.

if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

- ready 큐에서 우선순위가 가장 높은 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

- 간트 Queue 업데이트

현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

`current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력

- 프로세스 테이블 출력

- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.6. RoundRobin 스케줄링 알고리즘

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.

- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)

- 사용자가 입력한 정보들로 process 속성 초기화.

- readyQueue, waitingQueue, ganttQueue 초기화.

- remaining_quantum 을 사용자가 입력한 quantum 값으로 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

    - 프로세스의 `remaining_time`을 1 감소.

    - 남은 quantum 시간을 1 감소.

    - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

    - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

  if 프로세스가 끝나지 않았지만, 남은 quantum time 이 0 이 된 경우

    - 현재 실행 중인 프로세스를 ready 큐에 추가, `running_index`를 -1 로 설정.

  if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

    - ready 큐에 가장 먼저 들어온 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.

    - remaining_quantum 을 quantum 값으로 초기화.

- 간트 Queue 업데이트

  현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

  `current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력

- 프로세스 테이블 출력

- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.3.7. RoundRobin_priority 스케줄링 알고리즘 (priority 값이 작을수록 우선순위 높다)

1. 변수 및 큐 초기화

- `current_time`, `completed_processes`를 0 으로 초기화.
- `running_index`를 -1 로 초기화 (현재 실행 중인 프로세스 인덱스를 나타내는 변수로, 실행 중인 프로세스가 없다면, -1)
- 사용자가 입력한 정보들로 process 속성 초기화.
- readyQueue, waitingQueue, ganttQueue 초기화.
- remaining_quantum 을 사용자가 입력한 quantum 값으로 초기화.

2. while 시뮬레이션 루프 (모든 프로세스가 완료될 때까지 반복)

- 도착한 프로세스를 ready 큐에 추가

  각 프로세스에 대해, `current_time`이 `arrive_time`보다 크거나 같고, 해당 프로세스가 아직 ready 상태가 아니라면, 프로세스 인덱스를 ready 큐에 추가하고 ready 상태 표시.

- waiting 큐에 있는 프로세스의 남은 I/O 시간 업데이트

  waiting 큐에 있는 각 프로세스의 `io_remaining_time`을 1 감소.

  프로세스의 `io_remaining_time`이 0 이 되면, 해당 프로세스를 waiting 큐에서 제거하고 ready 큐에 추가.

- 프로세스 실행

  if 현재 실행 중인 프로세스가 있을 경우

  - 프로세스의 `remaining_time`을 1 감소.
  - 남은 quantum 시간을 1 감소.
  - 프로세스가 I/O 시작 시간에 도달하고 `io_remaining_time`이 0 보다 크다면, 해당 프로세스를 waiting 큐에 추가, `running_index`를 -1 로 설정.

  if 프로세스의 `remaining_time`이 0 이 되어 끝난 경우

  - 완료로 표시하고 turnaround time 업데이트, `running_index`를 -1 로 설정.

if 프로세스가 끝나지 않았지만, 남은 quantum time 이 0 이 된 경우

- 현재 실행 중인 프로세스를 ready 큐에 추가, `running_index`를 -1 로 설정.

선점 조건 확인 (ready 큐에서 우선순위가 가장 높은 프로세스를 찾기)

if 위의 프로세스가 현재 실행 중인 process 보다 우선순위가 높은 경우

- 현재 실행 중인 프로세스를 ready 큐에 추가하고, 해당 프로세스를 꺼내 실행.
- 해당 프로세스의 waiting time 을 업데이트.
- remaining_quantum 을 quantum 값으로 초기화.

if 현재 실행 중인 프로세스가 없고, ready 큐가 비어 있지 않는 경우

- ready 큐에 가장 먼저 들어온 프로세스를 꺼내 실행. 해당 프로세스의 waiting time 을 업데이트.
- remaining_quantum 을 quantum 값으로 초기화.

- 간트 Queue 업데이트

현재 `running_index` (또는 실행 중인 프로세스가 없을 경우 -1)를 간트 Queue 에 추가.

- 현재 시간 증가

`current_time`을 1 증가.

3. 결과 출력

- 간트 차트 출력
- 프로세스 테이블 출력
- 성능 지표 출력 (평균 waiting time, 평균 turnaround time)

2.4. 시뮬레이터 실행 결과 화면

먼저, 시뮬레이터 실행을 위해 입력한 값들은 다음과 같다.

```
Enter the number of processes: 5
Enter PID, Burst Time, Arrival Time, Priority (separated by spaces): 1 5 2 4
Enter PID, Burst Time, Arrival Time, Priority (separated by spaces): 2 6 2 2
Enter PID, Burst Time, Arrival Time, Priority (separated by spaces): 3 4 4 4
Enter PID, Burst Time, Arrival Time, Priority (separated by spaces): 4 4 4 1
Enter PID, Burst Time, Arrival Time, Priority (separated by spaces): 5 5 5 2
Enter the time quantum for Round Robin scheduling: 2
```

다음으로, 구현한 시뮬레이터의 실행 결과 화면은 아래와 같다.

2.4.1. FCFS Scheduling 실행 결과

```
FCFS Scheduling
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P1  |  P2  |  P3  |  P4  |  P5  |  P1  |  P2  |  P3  |  P4  |  P5  |
+------+------+------+------+------+------+------+------+------+------+------+
0      2      5      9      10     13     14     16     18     21     22     26
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   1 |     5    |      2     |    4   |      3    |      3    |       6    |       14      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   2 |     6    |      2     |    2   |      1    |      4    |       9    |       16      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   3 |     4    |      4     |    4   |      1    |      1    |      12    |       17      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   4 |     4    |      4     |    1   |      2    |      3    |      12    |       18      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   5 |     5    |      5     |    2   |      1    |      1    |      15    |       21      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
Average Waiting Time: 10.800
Average Turnaround Time: 17.200
```

2.4.2. Non-preemptive SJF Scheduling 실행 결과

```
Non-preemptive SJF Scheduling
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P1  |  P3  |  P4  |  P1  |  P4  |  P3  |  P5  |  P2  |  P5  |  P2  |
+------+------+------+------+------+------+------+------+------+------+------+
0      2      5      6      9      11     12     15     16     20     24     26
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   1 |     5    |      2     |    4   |      3    |      3    |       1    |        9      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   2 |     6    |      2     |    2   |      1    |      4    |      17    |       24      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   3 |     4    |      4     |    4   |      1    |      1    |       6    |       11      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   4 |     4    |      4     |    1   |      2    |      3    |       2    |        8      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   5 |     5    |      5     |    2   |      1    |      1    |      13    |       19      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
Average Waiting Time: 7.800
Average Turnaround Time: 14.200
```

2.4.3. Non-preemptive Priority Scheduling 실행 결과

```
Non-preemptive Priority Scheduling
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P2  |  P4  |  P5  |  P2  |  P4  |  P5  |  P1  |  P3  | IDLE |  P3  |  P1  |
+------+------+------+------+------+------+------+------+------+------+------+------+
0      2      6      9      10     12     13     17     20     21     22     25     27
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   1 |     5    |      2     |    4   |      3    |      3    |      17    |       25      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   2 |     6    |      2     |    2   |      1    |      4    |       3    |       10      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   3 |     4    |      4     |    4   |      1    |      1    |      16    |       21      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   4 |     4    |      4     |    1   |      2    |      3    |       3    |        9      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   5 |     5    |      5     |    2   |      1    |      1    |       6    |       12      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
Average Waiting Time: 9.000
Average Turnaround Time: 15.400
```

2.4.4. Preemptive SJF Scheduling 실행 결과

```
Preemptive SJF Scheduling
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P1  |  P3  |  P4  |  P1  |  P4  |  P3  |  P5  |  P2  |  P5  |  P2  | IDLE |  P2  |
+------+------+------+------+------+------+------+------+------+------+------+------+------+
0      2      5      6      9      11     12     15     16     17     21     24     25     27
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   1 |     5    |      2     |    4   |      3    |      3    |       1    |        9      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   2 |     6    |      2     |    2   |      1    |      4    |      18    |       25      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   3 |     4    |      4     |    4   |      1    |      1    |       6    |       11      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   4 |     4    |      4     |    1   |      2    |      3    |       2    |        8      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   5 |     5    |      5     |    2   |      1    |      1    |      10    |       16      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
Average Waiting Time: 7.400
Average Turnaround Time: 13.800
```

2.4.5. Preemptive Priority Scheduling 실행 결과

```
Preemptive Priority Scheduling
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P2  |  P4  |  P2  |  P4  |  P5  |  P2  |  P5  |  P1  |  P3  | IDLE |  P3  |  P1  |
+------+------+------+------+------+------+------+------+------+------+------+------+------+
0      2      4      7      9      10     11     13     17     20     21     22     25     27
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   1 |     5    |      2     |    4   |      3    |      3    |      17    |       25      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   2 |     6    |      2     |    2   |      1    |      4    |       4    |       11      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   3 |     4    |      4     |    4   |      1    |      1    |      16    |       21      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   4 |     4    |      4     |    1   |      2    |      3    |       0    |        6      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
       |   5 |     5    |      5     |    2   |      1    |      1    |       6    |       12      |
       +-----+----------+------------+--------+-----------+-----------+------------+---------------+
Average Waiting Time: 8.600
Average Turnaround Time: 15.000
```

2.4.6. Round Robin Scheduling with Quantum 2 실행 결과

```
Round Robin Scheduling with Quantum 2
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P1  |  P2  |  P3  |  P4  |  P1  |  P5  |  P2  |  P3  |  P4  |  P5  |  P1  |  P2  |  P3  |  P4  |  P5  |
+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+
0      2      4      6      7      9      10     11     13     15     16     18     20     22     23     24     26
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   1 |     5    |      2     |    4   |      3   |      3   |      10    |       18      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   2 |     6    |      2     |    2   |      1   |      4   |      13    |       20      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   3 |     4    |      4     |    4   |      1   |      1   |      14    |       19      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   4 |     4    |      4     |    1   |      2   |      3   |      14    |       20      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   5 |     5    |      5     |    2   |      1   |      1   |      15    |       21      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
Average Waiting Time: 13.200
Average Turnaround Time: 19.600
```

2.4.7. Round Robin + Priority Scheduling with Quantum 2 실행 결과

```
Round Robin + Priority Scheduling with Quantum 2
Gantt Chart:
+------+------+------+------+------+------+------+------+------+------+------+------+------+
| IDLE |  P2  |  P4  |  P2  |  P4  |  P5  |  P2  |  P5  |  P1  |  P3  |  P1  |  P3  |  P1  |
+------+------+------+------+------+------+------+------+------+------+------+------+------+
0      2      4      7      9      10     11     13     17     19     20     21     24     26
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        | PID |Burst Time|Arrival Time|Priority| I/O Burst | I/O start |Waiting Time|Turnaround Time|
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   1 |     5    |      2     |    4   |      3   |      3   |      16    |       24      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   2 |     6    |      2     |    2   |      1   |      4   |       4    |       11      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   3 |     4    |      4     |    4   |      1   |      1   |      15    |       20      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   4 |     4    |      4     |    1   |      2   |      3   |       0    |        6      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
        |   5 |     5    |      5     |    2   |      1   |      1   |       6    |       12      |
        +-----+----------+------------+--------+----------+----------+------------+---------------+
Average Waiting Time: 8.200
Average Turnaround Time: 14.600
```

위의 시뮬레이션 결과들을 하나씩 확인해 본 결과, Gantt Chart 및 Waiting time, Turnaround Time 이 모두 잘 나온 것을 알 수 있다. 결론적으로, 작성한 CPU Scheduling Simulator 의 모든 스케줄링 알고리즘이 잘 구현되었다고 할 수 있다.

2.5. 알고리즘들 간의 성능 비교

다음으로, 위의 시뮬레이션 결과를 보고 알고리즘들 간의 성능 비교를 위해 아래와 같은 표를 만들어 보았다.

	FCFS	SJF	Priority	Preem SJF	Preem Priority	RR	RR + Priority
Average Waiting TIme	10.800	7.800	9.000	7.400	8.600	13.200	8.200
Average Turnaround Time	17.200	14.200	15.400	13.800	15.000	19.600	14.600

2.5.1. Average Waiting Time 기준

가장 작은 값을 보이는 알고리즘부터 나타낸 순서는 아래와 같다.

Preem-SJF > SJF > RR+Priority > Preem-Priority > Priority > FCFS > RR

2.5.2. Average Turnaround Time 기준

가장 작은 값을 보이는 알고리즘부터 나타낸 순서는 아래와 같다.

Preem-SJF > SJF > RR+Priority > Preem-Priority > Priority > FCFS > RR

2.5.3. CPU Utilization 기준

가장 높은 값을 보이는 알고리즘부터 나타낸 순서는 아래와 같다.

FCFS = SJF = RR = RR+Priority > Priority = Preem-SJF = Preem-Priority

이때, 앞의 4개의 알고리즘은 시간이 26일 때 모든 프로세스가 종료되고, 따라서 CPU Utilization은 약 92.31% 라고 할 수 있다. 뒤의 3개의 알고리즘은 시간이 27일 때 모든 프로세스가 종료되고, 따라서 CPU Utilization은 약 88.89% 라고 할 수 있다.

2.5.4. 종합 평가

위의 결과를 확인해보면, Average Waiting Time이 짧은 알고리즘일수록 Average Turnaround Time

도 짧을 것을 알 수 있다. 재미있는 사실은 Average Waiting Time과 Average Turnaround Time에서 낮은 성능을 보였던 FCFS, RR이 CPU Utilization에서는 높은 수치를 보인다는 점이다. 그렇지만, 각 알고리즘에서 모든 프로세스가 끝나는 시간 차이가 1 이하이기 때문에, CPU Utilization에서 각 알고리즘마다 유의미한 차이가 있다고 보기엔 다소 어려워 보인다.

결론적으로, 위의 7 개의 알고리즘이 모두 비슷한 CPU Utilization 을 가지기 때문에, Average Waiting Time 과 Average Turnaround Time 을 기준으로 종합적인 평가를 하면, 성능 순위는 다음과 같다.

1. Preemptive SJF
2. Non-preemptive SJF
3. Round Robin + Priority
4. Preemptive Priority
5. Non-preemptive Priority
6. FCFS
7. Round Robin

Preemptive SJF (Shortest Job First) 알고리즘과 Non-preemptive SJF (Shortest Job First) 알고리즘의 성능이 높은 이유는 다음과 같이 설명할 수 있다.

1. Preemptive SJF는 남아있는 실행 시간이 가장 짧은 프로세스를 먼저 실행함으로써, 대기 시간을 최소화한다. 특히, 짧은 작업들이 긴 작업에 의해 지연되지 않도록 하기 때문에, 일반적으로 전체적인 Average Waiting Time과 Average Turnaround Time이 가장 낮게 나타난다.

2. Non-preemptive SJF는 프로세스가 시작되면 끝날 때까지 계속 실행되지만, 위의 알고리즘과 비슷하게 CPU burst time이 짧은 작업을 우선 실행하므로, 대기 시간을 효과적으로 줄인다. 그러나 Preemptive SJF보다 유연성이 떨어지기 때문에, 약간 더 높은 대기 시간과 반환 시간이 나타날 수 있다.

반면, FCFS (First-Come, First-Served) 알고리즘과 Round Robin 알고리즘의 성능이 낮은 이유는 다음과 같이 설명할 수 있다.

1. FCFS 알고리즘은 도착 순서대로 프로세스를 실행하기 때문에, 도착 시간이 빠른 프로세스

가 긴 작업일 경우, 뒤에 도착한 짧은 작업들이 불필요하게 대기하게 된다. 이는 waiting time 과 turnaround time을 증가시키는 주요 요인이다.

2. Round Robin 알고리즘은 각 프로세스에 동일한 CPU 시간을 할당하므로, 긴 작업이 자주 분할되어 실행된다. 이는 response time 을 줄이는 데 효과적이지만, 전체적인 waiting time 과 turnaround time 을 증가시키는 결과를 초래한다.

주의해야 할 점은, 위와 같은 결과는 현재 입력 받은 값을 토대로 나온 결과라는 것이다. 즉, 입력 받은 프로세스의 개수, 각 프로세스의 속성값, time quantum, I/O 발생 타이밍 및 시간 등이 달라지면 결과가 달라질 수 있음에 유의해야 한다. 또한, waiting time, turnaround time, CPU utilization 외에도 다른 성능 평가 지표를 같이 활용한다면, 위의 결과가 달라질 수 있다.

III. 결론

3.1. 구현한 시뮬레이터에 대한 정리

이번 프로젝트에서는 다양한 CPU 스케줄링 알고리즘을 구현하고 비교 분석하는 시뮬레이터를 개발하였다. 구현된 알고리즘은 다음과 같다:

- FCFS (First-Come, First-Served)
- Non-preemptive SJF (Shortest Job First)
- Preemptive SJF
- Non-preemptive Priority
- Preemptive Priority
- Round Robin
- Round Robin with Priority

각 알고리즘의 성능을 평가하기 위해 Average Waiting Time, Average Turnaround Time, 그리고 CPU Utilization 을 측정하였다. 이 과정에서 각 스케줄링 알고리즘이 어떻게 작동하는지, 그리고 다양한 조건에서의 성능 차이 및 각 알고리즘의 장단점을 확인할 수 있었다.

주요 결과를 요약하면 다음과 같다.

Average Waiting Time 과 Average Turnaround Time 은 Preemptive SJF, SJF 가 가장 우수한 성능을 보였고, FCFS 와 RR 이 가장 낮은 성능을 보였다.

CPU Utilization 은 대부분의 알고리즘에서 유사한 수치를 보였으나, FCFS 와 SJF, RR, RR+Priority 가 상대적으로 높은 수치를 기록했다. 하지만, 모든 프로세스가 끝나는 시간 차이가 1 이하로 매우 작기 때문에 유의미한 차이라고 보기 어렵다.

이때, 본 프로젝트에서 도출한 결과는 특정 케이스에 기반한 것이며, 이는 주어진 프로세스의 특성에 따라 다르게 나타날 수 있다.

- 프로세스의 개수: 프로세스의 수가 많아지면 스케줄링 알고리즘의 효율성에 대한 평가가 달라질 수 있다.

- 각 프로세스의 속성값: 프로세스의 도착 시간, 실행 시간, 우선순위 등이 달라지면 특정 알고리즘의 성능이 변화할 수 있다.
- Time Quantum: Round Robin 알고리즘의 경우, time quantum 의 크기에 따라 waiting time 과 turnaround time 이 크게 달라질 수 있다.
- I/O 발생 타이밍 및 시간: 프로세스가 I/O 작업을 언제, 얼마나 자주, 그리고 얼마나 오래 수행하는지에 따라 스케줄링 알고리즘의 성능이 영향을 받을 수 있다.

따라서, 실제 시스템에서는 주어진 조건에 맞는 최적의 스케줄링 알고리즘을 선택하는 것이 중요하다. 본 시뮬레이터는 다양한 시나리오에서 각 알고리즘을 테스트하고, 이를 통해 얻은 결과를 바탕으로 시스템의 요구사항에 가장 적합한 스케줄링 방법을 찾는 데 유용한 도구가 될 것이다.

3.2. 프로젝트 수행 소감

이번 프로젝트는 CPU 스케줄링 알고리즘의 이론적 이해를 바탕으로 실습을 통해 깊이 있는 학습을 할 수 있는 기회를 제공해 주었다. 여러 스케줄링 알고리즘을 동일한 환경에서 비교함으로써, 각 알고리즘의 특성과 성능 차이를 명확하게 이해할 수 있었다. 특히, 알고리즘 간의 성능 차이가 특정 조건에서 어떻게 변화하는지를 확인하며, 각 알고리즘의 장단점을 실감할 수 있었다.

또한, 파이썬 언어는 많이 다뤄봐서 익숙한 반면, C 언어는 상대적으로 익숙하지 않았기 때문에, C 언어에 대해 다소 두려움이 있었다. 그러나 이번 과제를 통해 이러한 두려움을 극복할 수 있었다. 알고리즘을 구현하고 디버깅하는 과정에서 많은 여러 과제를 마주했지만, 이를 해결해 나가면서 프로그래밍 능력과 문제 해결 능력이 크게 향상되었다. 특히, 큐 자료구조를 활용하여 프로세스를 관리하는 부분에서 많은 성장을 이루었다.

3.3. 향후 발전 방향

프로젝트를 통해 많은 것을 배웠지만, 앞으로 개선하거나 추가할 수 있는 몇 가지 발전 방향을 제안하고자 한다.

1. 다양한 시나리오 테스트

더 다양한 프로세스 조합에서 스케줄링 알고리즘을 테스트하여, 알고리즘의 성능을 보다 포괄적으로 평가할 필요가 있다. 예를 들어, 프로세스의 도착 시간이 밀집된 경우와 분산된 경우, I/O 작업이 빈번한 경우 등 다양한 시나리오를 고려한다면, 더 보편적이고 일반화된 결과를 얻을 수 있을 것으로 기대된다.

2. 실제 시스템에서의 테스트

시뮬레이터 외에도 실제 운영 체제 환경에서 스케줄링 알고리즘을 적용하고 테스트하여, 실제 시스템에서의 성능을 평가할 필요가 있다. 이를 통해 시뮬레이터의 결과와 실제 시스템 간의 차이를 분석할 수 있다. 예를 들어, SJF(Shortest Job First) 알고리즘은 프로세스의 burst time을 미리 알고 있어야 최적의 성능을 발휘할 수 있지만, 실제 시스템에서는 프로세스의 burst time을 사전에 정확히 알 수 없기 때문에, 예측된 값이나 과거 데이터를 바탕으로 동작하게 되어 시뮬레이터에서 얻은 결과와 다를 수 있다. 이러한 차이를 분석함으로써 알고리즘의 현실적인 성능과 한계를 이해할 수 있을 것으로 기대된다.

3. 알고리즘 최적화

현재 작성한 시뮬레이션 코드는 중복된 코드가 많고, 이해하기 쉽게 작성하려다 보니 다소 복잡해진 부분이 존재한다. 특히, Queue를 구현할 때, 양방향 Queue를 사용한다면 preemptive 알고리즘에서 더 손쉽게 선점(preemptive)할 수 있을 것이다. 또한, 중복된 코드를 제거하고 함수로 분리하여, 코드의 재사용성을 높이고, 가독성을 높일 수 있다. 이러한 점을 고려하여, 각 알고리즘의 구현을 최적화함으로써 성능을 더욱 향상시킬 수 있는 방법을 연구할 필요가 있다.

4. 알고리즘 추가

현재 구현된 7개의 알고리즘 외에도 다양한 스케줄링 알고리즘을 추가하여 비교해볼 수 있다. 예를 들어, 다단계 큐(Multilevel Queue), 다단계 피드백 큐(Multilevel Feedback Queue), 공정 공유 스케줄링(Fair-Share Scheduling) 등의 알고리즘들을 추가로 구현하여, 더욱 다양한 상황에서의 알고리즘 성능 비교가 가능해지고, 보다 더 효율적인 최적의 스케줄링 알고리즘을 찾는 것이 가능해질 것이다.

이번 프로젝트는 저에게 많은 것을 가르쳐 준 소중한 경험이었으며, 향후 기회가 된다면 위에서 작성한 것과 같이 더욱 발전된 연구와 프로젝트로 나아가도록 하겠습니다!

IV. 부록

소스코드는 아래 github 링크에 들어가면 확인할 수 있다.

이때, 2020170856_woojin_final_extra.c 코드가 최종적으로 모든 기능이 구현되어 있는 코드이다.

https://github.com/Woojin0118/20241R0136COSE34101/tree/main/term1