Report 01
CPU Scheduler Analysis

Spring, 2025
Department of Computer Engineering

2025.04.17 컴퓨터공학과 22312072 차민경

First-Come, First-Served (FCFS) Scheduling
스케줄링 알고리즘

```
BEGIN
    SORT processes by arrival time
    FOR each process in ready queue
        EXECUTE the process until completion
    END FOR
END
```

First-Come, First-Served (FCFS) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time
P1	0ms	6ms
P2	2ms	4ms
P3	4ms	5ms

- 대기 시간

Process	Waiting time
P1	0ms
P2	6ms
P3	10ms

- 작동 순서
  1. P1이 먼저 도착하여 6ms 동안 실행한다
  2. 그 후 P2가 실행되며 4ms 동안 실행한다
  3. 마지막으로 P3이 실행되며 5ms 동안 실행한다

First-Come, First-Served (FCFS) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  각 프로세스는 도착 순서대로 실행되었고, 대기 시간이 CPU Burst Time의 길이에 비례해서 늘어났다. 프로세스 A가 먼저 실행되었기 때문에 B와 C는 각각 A와 B가 실행되는 동안 대기해야 했다.

First-Come, First-Served (FCFS) Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  프로세스 도착 시간, CPU Burst Time, 대기 시간

- 독특한 동작 혹은 현상

  Convey Effect

  비선점형으로 우선순위와 관계없이 도착 순서대로 실행한다

- 구현 시, 특징적 고려사항

  프로세스 도착 순서와 CPU Burst Time을 정확히 알고 있어야 효율적인 FCFS 스케줄링을 구현할 수 있다.

  Convoy Effect를 피하려면, 다른 스케줄링 알고리즘을 사용하는 것이 좋다

First-Come, First-Served (FCFS) Scheduling
장점 및 단점

- 장점

  구현이 간단하고 직관적이다

  공정성을 보장한다 (첫 번째로 도착한 프로세스가 먼저 실행되므로 선택 기준이 명확하다)

- 단점

  Convoy Effect

  비효율적 : 긴 대기 시간을 초래할 수 있어 응답 시간이 길어질 수 있다

  우선순위 고려 부족 : FCFS는 우선순위나 중요도를 고려하지 않고 도착 순서대로 처리되므로, 중요한 작업이 나중에 실행될 수 있다

Shortest Job First (SJF) Scheduling
스케줄링 알고리즘

- 비선점형 SJF

```
BEGIN
    SORT processes by burst time (shortest burst time first)
    FOR each process in the ready queue
        EXECUTE the process with the shortest burst time
    END FOR
END
```

Shortest Job First (SJF) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time
P1	0ms	7ms
P2	2ms	5ms
P3	4ms	2ms

- 대기 시간

Process	Waiting time
P1	0ms
P2	9ms
P3	7ms

- 작동 순서 (비선점형 SJF)

1. P1이 먼저 도착하여 7ms 동안 실행한다
2. 그 후 Burst Time이 짧은 P3 실행한다
3. 마지막으로 P2 실행한다

Shortest Job First (SJF) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  처음에는 먼저 도착한 프로세스 실행되었고, 그 후 CPU Burst Time의 길이가 짧은 프로세스가 실행 되었다. 프로세스 P1이 먼저 실행되었기 때문에 P2와 P3는 각각 P3과 P1가 실행되는 동안 대기해야 한다.

Shortest Job First (SJF) Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  도착 시간, CPU Burst Time, 대기 시간, 완료 시간

- 독특한 동작 혹은 현상

  Convoy Effect, Starvation, 최적화된 평균 대기 시간

- 구현 시, 특징적 고려사항

  CPU Burst Time 예측의 어렵다 -> 가중 평균을 통해 예측값으로 해결한다

  Starvation 문제 -> Aging 기법 사용하여 해결한다

Shortest Job First (SJF) Scheduling

장점 및 단점

- 장점

  최소화된 대기 시간

  짧은 프로세스들 우선 실행으로 인한 효율적인 자원 관리

- 단점

  CPU Burst Time 예측하기 어렵다 T_{n+1} = alpha * T_{n} + (1 - alpha) * T_n

  Starvation 문제

Shortest Remaining Time First (SRTF) Scheduling

스케줄링 알고리즘

- 선점형 SJF - SRTF (Shortest-Remaining-Time-First)

```
BEGIN
    WHILE there are processes in the ready queue
        SELECT the process with the shortest remaining time
        IF a new process arrives with a shorter burst time
            PREEMPT the current process and run the new process
    END WHILE
END
```

Shortest Remaining Time First (SRTF) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time
P1	0ms	7ms
P2	2ms	5ms
P3	4ms	2ms

- 대기 시간

Process	Waiting time
P1	9ms
P2	2ms
P3	0ms

- 작동 순서 (선점형 SJF)

1. P1이 0ms에 도착하여 실행한다

2. P2이 2ms에 도착하고, P2의 CPU Burst Time이 5ms이다. P2는 P1의 남은 시간(5ms)보다 짧기 때문에 P1을 선점하고 P2이 실행된다. P2는 실행된다.

3. P3이 4ms에 도착하고, P3의 CPU Burst Time은 2ms이다. P2가 실행 중인데, P3은 P2보다 짧기 때문에 P3이 먼저 실행되고, 이후 P3과 P2 중 남은 Burst Time이 짧은 P이 실행된다.

4. P3 실행 후 P2의 남은 시간 (3ms)이 실행된다

5. 마지막으로 P1의 남은 시간 (5ms)이 실행된다

Shortest Remaining Time First (SRTF) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  남은 CPU 시간이 짧은 프로세스를 우선 실행하여 시스템 전체의 대기 시간을 최소화할 수 있지만, 선점을 통해 실행 순서가 변경되므로 대기 시간이 길어질 수 있는 프로세스가 발생할 수 있다.

Shortest Remaining Time First (SRTF) Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  도착 시간, CPU Burst Time, 대기 시간, 완료 시간

- 독특한 동작 혹은 현상

  선점, Starvation, 문맥 교환

- 구현 시, 특징적 고려사항

  남은 시간 추적, 프로세스 도착 시 스케줄링 재계산, 비효율적인 컨텍스트 스위칭, Starvation 문제 해결

Shortest Remaining Time First (SRTF) Scheduling

장점 및 단점

- 장점

  평균 대기 시간 최소화

  효율적인 리소스 관리

- 단점

  Context Switing 오버헤드

  프로세스 관리 복잡성

  Starvation 문제

Priority Scheduling
스케줄링 알고리즘

- 선점 Priority Scheduling

```
BEGIN
    WHILE there are processes in the ready queue
        SELECT the process with the highest priority
        IF a higher priority process arrives
            PREEMPT the current process and allocate CPU to the higher priority process
    END WHILE
END
```

Priority Scheduling
스케줄링 알고리즘

- 비선점 Priority Scheduling

```
BEGIN
    SORT processes by priority (highest priority first)
    FOR each process in ready queue
        EXECUTE the process with highest priority
    END FOR
END
```

Priority Scheduling (선점형)

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time	Priority
P1	0ms	5ms	2
P2	2ms	4ms	1
P3	4ms	3ms	3

- 대기 시간

Process	Waiting time
P1	4ms
P2	0ms
P3	5ms

- 작동 순서
  1. P1이 0ms에 도착하고, 우선순위 2로 실행된다
  2. P2가 2ms에 도착하고, 우선순위 1로 P1이 실행 중일 때 선점하여 실행된다
  3. P3가 4ms에 도착하고, 우선순위 3으로 P2 실행 후 실행된다

Priority Scheduling (비선점형)

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time	Priority
P1	0ms	5ms	2
P2	2ms	4ms	1
P3	4ms	3ms	3

- 대기 시간

Process	Waiting time
P1	0ms
P2	2ms
P3	4ms

- 작동 순서
  1. P1이 0ms에 도착하고, 우선순위 2로 실행된다
  2. P2가 2ms에 도착하고, 우선순위 1로 P1 실행 완료 후 실행된다
  3. P3가 4ms에 도착하고, 우선순위 3으로 P2 실행 완료 후 실행된다

Priority Scheduling (선점형)

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  프로세스중에 우선순위가 높은 프로세스가 실행된다

Priority Scheduling (비선점형)

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  실행 중인 프로세스가 끝날 때까지 대기한 후 우선순위가 높은 프로세스가 실행된다

Priority Scheduling
각 알고리즘 별 특징

- 개별 입력 요소

  우선순위, 도착 시간, CPU Burst Time, 대기 시간, 완료 시간

- 독특한 동작 혹은 현상

  Starvation 문제

  우선순위에 따른 실행 순서

- 구현 시, 특징적 고려사항

  우선순위의 동적 변화

Highest Response Ratio Next (HRRN) Scheduling

스케줄링 알고리즘

```
BEGIN
  WHILE there are processes in the ready queue
    Calculate the Response Ratio for each process:
      Response Ratio = (Waiting Time + CPU Burst Time) / CPU Burst Time
    SELECT the process with the highest Response Ratio
    EXECUTE the selected process
  END WHILE
END
```

Highest Response Ratio Next (HRRN) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Waiting Time	Burst Time
P1	16ms	4ms
P2	10ms	5ms
P3	7ms	7ms

- Response Ratio

Process	Response Ratio
P1	5ms
P2	3ms
P3	2ms

- 작동 순서
  1. P1이 대기 시간 5ms로 가장 높은 응답 비율을 가지고 있어 먼저 실행된다
  2. P2가 대기 시간 3ms로 두 번째로 높은 응답 비율을 가지고 있어 실행된다
  3. 마지막으로 P3이 실행된다 P3이 응답 비율은 2ms로 가장 낮기 때문에 마지막으로 실행된다

Highest Response Ratio Next (HRRN) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  각 프로세스의 대기 시간과 CPU Burst Time을 고려하여 응답 비율(Response Ratio)을 계산하고, 응답 비율이 가장 큰 프로세스를 우선적으로 실행한다.

Highest Response Ratio Next (HRRN) Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  도착 시간, CPU Burst Tiime, 대기 시간

- 독특한 동작 혹은 현상

  응답 비율을 계산하여, 대기 시간이 긴 프로세스를 우선 실행하여 기아 문제를 해결한다

- 구현 시, 특징적 고려사항

  대기 시간 추적, 동적 우선순위

Highest Response Ratio Next (HRRN) Scheduling

장점 및 단점

- 장점

  기아 해결

  효율적인 대기 시간 관리

- 단점

  CPU Burst Time 예측의 어려움

  시스템 자원 관리의 복잡성

Round Robin (RR) Scheduling
스케줄링 알고리즘

```
BEGIN
  Initialize ready_queue
  WHILE there are processes in the ready queue
    SELECT the first process from ready_queue
    ASSIGN the process to CPU for time quantum (time slice)

    IF the process completes within the time quantum
      REMOVE the process from the ready_queue
    ELSE
      REINSERT the process at the end of ready_queue with updated remaining time

  END WHILE
END
```

Round Robin (RR) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	Arrival Time	Burst Time	Time Quantum
P1	0ms	8ms	4ms
P2	2ms	6ms	4ms
P3	4ms	5ms	4ms

- 작동 순서

1. P1이 먼저 도착하여 4ms 실행된다 실행 후 P1은 남은 시간 4ms와 함께 ready queue의 끝으로 이동한다

2. P2가 4ms 실행된다 실행 후 P2는 남은 시간 2ms와 함께 ready queue의 끝으로 이동한다

3. P3이 4ms 실행된다 실행 후 P3은 남은 시간 1ms와 함께 ready queue의 끝으로 이동한다

4. P1은 남은 4ms를 실행하여 완료된다

5. P2는 남은 2ms를 실행하여 완료된다

6. P3은 1ms를 실행하여 완료된다

Round Robin (RR) Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  Round Robin 방식은 각 프로세스에 공평하게 CPU 시간을 나누어 주며, time quantum이 다되면 프로세스는 큐 뒤로 이동하여 다른 프로세스가 CPU 자원을 할당받을 수 있도록 한다.

Round Robin (RR) Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  도착 시간, CPU Burst Time, 시간 할당량

- 독특한 동작 혹은 현상

  시간 할당량만큼 실행 후 프로세스가 ready queue의 끝으로 이동한다

  전체 프로세스가 공평하게 CPU 시간을 사용하므로 응답 시간이 짧고, 대기 시간이 공정하게 분배된다

- 구현 시, 특징적 고려사항

  시간 할당량 설정

Round Robin (RR) Scheduling

장점 및 단점

- 장점

  공평한 CPU 할당

  응답 시간 단축

- 단점

  Context switiching 오버헤드

  CPU 시간의 비효율적인 사용 (시간 할당량이 너무 크면 CPU 자원 낭비)

Multilevel Queue Scheduling

스케줄링 알고리즘

```
BEGIN
  Initialize ready_queue
  WHILE there are processes in the ready_queue
    SELECT the first process from ready_queue
    ASSIGN the process to CPU for time quantum (time slice)

    IF the process completes within the time quantum
      REMOVE the process from the ready_queue
    ELSE
      UPDATE remaining time for the process
      REINSERT the process at the end of ready_queue

  END WHILE
END
```

Multilevel Queue Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	알고리즘	
Q1 : Foreground	RR	Time Slice : 6ms
Q2 : Background	FCFS	

- 작동 순서
  1. Foreground Queue에서 첫 번째 프로세스가 6ms 동안 실행된다
  2. Background Queue로 넘어가서 FCFS로 처리된 프로세스들이 순차적으로 실행된다

Multilevel Queue Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  각 큐의 우선순위는 Foreground Queue가 Background Queue보다 우선순위가 높다. 즉, 사용자 인터랙티브 프로세스는 우선적으로 처리된다. Time Slice의 비율은 Foreground Queue가 75%, Background Queue가 25%의 CPU 시간을 할당받는다

Multilevel Queue Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  프로세스 분류, 큐 우선순위, 큐 선택 기준

- 독특한 동작 혹은 현상

  프로세스의 큐 이동, 반응 시간

- 구현 시, 특징적 고려사항

  큐 우선순위 관리, 프로세스 간 이동, 각 큐의 스케줄링 알고리즘 설정

Multilevel Queue Scheduling

장점 및 단점

- 장점

  효율적인 리소스 관리

  우선순위 기반 스케줄링

- 단점

  복잡성

  자원 낭비

Multilevel Feedback Queue Scheduling

스케줄링 알고리즘

```
BEGIN
  Initialize multiple queues (Q1, Q2, Q3, ...)
  WHILE there are ready processes in any queue
    IF there is a process in the highest priority queue (Q1)
      SELECT the first process from Q1
      ASSIGN the process to CPU for time quantum (time slice)

      IF the process completes within the time quantum
        REMOVE the process from Q1
      ELSE
        MOVE the process to the next lower priority queue (Q2)

    ELSE IF there is a process in the next queue (Q2)
      SELECT the first process from Q2
      ASSIGN the process to CPU for a larger time quantum

      IF the process completes within the time quantum
        REMOVE the process from Q2
      ELSE
        MOVE the process to an even lower priority queue (Q3)

    ELSE IF there is a process in the lowest priority queue (Qn)
      SELECT the first process from Qn
      ASSIGN the process to CPU for the largest time quantum

      IF the process completes within the time quantum
        REMOVE the process from Qn
      ELSE
        WAIT for the next round (process does not move to another queue)

  END WHILE
END
```

Multilevel Feedback Queue Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 동작 사례

Process	알고리즘	
Q1 : Foreground	RR	Time Slice : 6ms
Q2 : Background	FCFS	

- 작동 순서
  1. Foreground Queue에서 첫 번째 프로세스가 6ms 동안 실행된다
  2. Background Queue로 넘어가서 FCFS로 처리된 프로세스들이 순차적으로 실행된다

Multilevel Feedback Queue Scheduling

알고리즘의 특징을 효과적으로 보여주는 동작 사례

- 제시된 사례에 대한 설명 분석

  각 큐의 우선순위는 Foreground Queue가 Background Queue보다 우선순위가 높다. 즉, 사용자 인터랙티브 프로세스는 우선적으로 처리된다. Time Slice의 비율은 Foreground Queue가 75%, Background Queue가 25%의 CPU 시간을 할당받는다

Multilevel Feedback Queue Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  프로세스 우선순위, 시간 슬라이스, 큐의 이동 규칙

- 독특한 동작 혹은 현상

  Dynamic Priority Adjustment, Starvation 방지

- 구현 시, 특징적 고려사항

  시간 슬라이스의 크기, 큐 이동 규칙, Starvation 해결

Multilevel Feedback Queue Scheduling

장점 및 단점

- 장점

  다양한 프로세스 처리

  동적 우선순위 조정

  Starvation 방지

- 단점

  복잡성

  응답 시간 지연

  구현 오버헤드

Fair Share Scheduling
스케줄링 알고리즘

```
BEGIN
  Initialize resource allocation for each user (user_share)
  Initialize ready_queue for all processes

  WHILE there are processes in ready_queue
    SELECT the first process from the ready_queue

    IF the process belongs to User X
      ASSIGN the process to CPU for the share allocated to User X

      IF the process completes within its time slice
        REMOVE the process from ready_queue
      ELSE
        DECREASE the remaining share for User X
        REINSERT the process in ready_queue

    ELSE
      SELECT next available process from ready_queue for other users

  END WHILE
END
```

Fair Share Scheduling

각 알고리즘 별 특징

- 개별 입력 요소

  사용자별 자원 할당량, 프로세스 우선순위, 프로세스 상태, 프로세스 실행 시간

- 독특한 동작 혹은 현상

  자원 공평 분배, 사용자 간 자원 경쟁 해결, 자원 할당 비율 유지

- 구현 시, 특징적 고려사항

  자원 할당 비율 설정, 실시간 자원 모니터링, 다중 사용자 관리, 우선순위 및 시간 관리

Fair Share Scheduling

장점 및 단점

- 장점

  공정성

  다중 사용자 환경에서 효율적

- 단점

  복잡성

  성능 저하

  자원 낭비

Summary

First-Come, First-Served (FCFS) Scheduling

- **특징**: 가장 간단한 알고리즘으로, **도착 순서**대로 프로세스를 실행합니다.
- **장점**: 구현이 매우 간단합니다.
- **단점**: **Convoy Effect** 발생 가능. 즉, 긴 CPU Burst Time을 가진 프로세스가 먼저 실행되면 그 뒤의 짧은 프로세스들이 불필요하게 대기하게 됩니다.

Summary

Shortest Job First (SJF) Scheduling

- **특징**: **CPU Burst Time**이 가장 짧은 프로세스를 우선 실행합니다. 비선점형으로 실행됩니다.
- **장점**: 평균 대기 시간이 가장 적은 알고리즘입니다.
- **단점**: **Burst Time**을 예측하기 어려워 **기아(Starvation)**가 발생할 수 있습니다. 긴 작업이 계속해서 뒤로 밀리는 문제.

Summary

Shortest Remaining Time First (SRTF) Scheduling

- **특징**: **SJF**의 선점형 버전으로, 프로세스가 실행 중일 때 더 짧은 남은 실행 시간이 있는 프로세스가 들어오면, 현재 실행 중인 프로세스를 중단하고 새로운 프로세스를 실행합니다.
- **장점**: SJF에 비해 **기아 문제**가 다소 완화됩니다.
- **단점**: 남은 시간이 짧은 프로세스를 계속해서 우선 실행하기 때문에 **빈번한 컨텍스트 스위칭**이 발생할 수 있습니다.

Summary

Priority Scheduling (선점형, 비선점형)

- **특징**: 각 프로세스에 우선순위를 부여하고, 우선순위가 높은 프로세스를 먼저 실행합니다.
- **선점형**: 높은 우선순위를 가진 프로세스가 실행 중인 프로세스를 선점할 수 있습니다.
- **비선점형**: 실행 중인 프로세스가 종료되거나 대기 상태로 전환될 때까지 우선순위가 높은 프로세스는 대기합니다.
- **장점**: 중요한 프로세스를 우선적으로 처리할 수 있습니다.
- **단점**: **기아(Starvation)** 문제 발생 가능, 우선순위가 낮은 프로세스가 계속해서 실행되지 않을 수 있습니다. **Aging** 기법을 사용해 해결할 수 있습니다.

Summary

Highest Response Ratio Next (HRRN) Scheduling

- **특징**: **대기 시간**과 **CPU Burst Time**을 모두 고려하여 **응답 비율**이 가장 큰 프로세스를 우선 실행합니다.
- **응답 비율**: {Response Ratio} = {(Waiting Time + CPU Burst Time)} / {CPU Burst Time}
- **장점**: **기아 문제**를 해결하며, **대기 시간이 길어질수록 우선순위**가 올라갑니다.
- **단점**: 대기 시간이 너무 길어지면 **응답 비율**이 급격히 증가하여 너무 자주 실행될 수 있습니다.

Summary

Round Robin (RR) Scheduling

- **특징**: 각 프로세스에 동일한 시간 할당량(Time Quantum)을 부여하고, 그 시간 동안 프로세스를 실행한 후 **큐의 끝으로 이동**시킵니다.
- **장점: 응답 시간**이 짧고, **공정한 CPU 자원 분배**를 제공합니다.
- **단점**: **Time Quantum**이 너무 크면 **FCFS**와 비슷하게 동작하고, 너무 작으면 **컨텍스트 스위칭**이 자주 발생하여 효율성이 떨어질 수 있습니다.

Summary

Multilevel Queue Scheduling

- **특징**: 프로세스를 여러 큐로 나누어 각 큐에 다른 스케줄링 알고리즘을 적용하고 우선순위에 따라 실행합니다.
- **장점:** 우선순위가 높은 프로세스를 빠르게 처리할 수 있어 리소스를 효율적으로 관리합니다.
- **단점**: 낮은 우선순위 큐의 프로세스가 실행되지 않거나 지연될 수 있어 기아 현상이 발생할 수 있습니다.

Summary

Multilevel Feedback Queue Scheduling

- **특징**: 프로세스가 큐를 이동하며 동적으로 우선순위가 조정되어 실행됩니다.
- **장점:** 대화형 프로세스는 빠르게 처리되고, CPU 집약적 프로세스는 효율적으로 관리됩니다.
- **단점**: 큐 관리와 우선순위 조정이 복잡하고, 낮은 우선순위 프로세스는 응답 시간이 지연될 수 있습니다.

Summary

Fair Share Scheduling

- **특징**: 사용자에게 공평하게 자원을 할당하여 자원 독점 문제를 방지하고, 자원 할당 비율을 유지합니다.
- **장점:** 다중 사용자 환경에서 효율적이며, 각 사용자에게 공정한 자원 분배를 보장합니다.
- **단점**: 구현이 복잡하고, 성능 저하와 자원 낭비가 발생할 수 있습니다.