REPORT

(프로세스 스케줄링 심층 연구)

과 목 명	운영체제
담 당 교 수	조영석
학 과	컴퓨터정보공학과
학 번	202244012
이 름	노성민
제 출 일	2025.12.11.

차례

1. 서론 ---- 3
2. 프로세스 스케줄링 개요 ---- 3
3. 스케줄링 알고리즘의 목표와 기준 ---- 3
4. 스케줄링 알고리즘 유형 ---- 4
5. 주요 스케줄링 알고리즘 ---- 4
6. 스케줄링 알고리즘의 성능 평가 ---- 7
7. 실시간 스케줄링 ---- 7
8. 결론 ---- 8
9. 참고문헌 ---- 9
10. 레포트 작성에서 느낀점 ---- 9

1. 서론

현대의 컴퓨터 시스템은 다중 프로그래밍 환경에서 여러 프로세스를동시에 실행한다. 제한된 CPU 자원을 여러 프로세스가 효율적으로 공유하기 위해서는 적절한 프로세스 스케줄링 기법이 필수적이다. 프로세스 스케줄링은 운영체제의 핵심 기능 중 하나로, CPU를 어떤 프로세스에게 언제 할당할 것인지를 결정하는 메커니즘이다.

효과적인 스케줄링 알고리즘은 시스템의 전반적인 성능을 향상시키고, 사용자 경험을 개선하며, 시스템 자원을 최적으로 활용할 수 있게 한다. 본 레포트에서는 프로세스 스케줄링의 기본 개념부터 다양한 스케줄링 알고리즘, 성능 평가 기준, 그리고 실시간 시스템에서의 스케줄링까지 체계적으로 살펴보고자 한다.

특히, 각 스케줄링 알고리즘의 동작 원리와 장단점을 이론적으로 분석하고, 실제 운영체제에서 어떻게 적용되는지에 대해 깊이 있게 다룰 것이다. 이를 통해 프로세스 스케줄링이 운영체제의 성능과 효율성에 미치는 영향을 종합적으로 이해할 수 있을 것이다.

2. 프로세스 스케줄링의 개요

프로세스 스케줄링은 실행 가능한 프로세스 중에서 CPU를 할당받을 프로세스를 선택하는 과정이다. 운영체제는 스케줄러(Scheduler)라는 모듈을 통해 이러한 결정을 내리며, 스케줄링의 목적은 CPU 이용률을 최대화하고 응답 시간을 최소화하는 등 시스템의 성능 목표를 달성하는 것이다.

프로세스는 생명 주기 동안 여러 상태를 거치게 되는데, 크게 생성(New), 준비(Ready),실행(Running), 대기(Waiting), 종료(Terminated) 상태로 구분된다. 이 중 스케줄링과 직접적으로 관련된 상태는 준비 상태와 실행 상태이다. 준비 큐(ReadyQueue)에 있는 프로세스들은 CPU 할당을 기다리고 있으며, 스케줄러는 이 큐에서 다음에 실행할 프로세스를 선택한다.

스케줄링이 발생하는 시점은 크게 네 가지로 분류할 수 있다. 첫째, 실행 중인 프로세스가 실행 상태에서 대기 상태로 전환될 때이다. 이는 입출력 작업을 요청하거나 자식 프로세스의 종료를 기다리는 경우에 발생한다. 둘째, 프로세스가 실행 상태에서 준비 상태로 전환될 때인데, 이는 인터럽트가 발생했을 때 나타난다. 셋째, 프로세스가 대기 상태에서 준비 상태로 전환될 때로, 입출력 작업이 완료되었을 때 발생한다. 마지막으로, 프로세스가 종료될 때 스케줄링이 필요하다.

스케줄러는 크게 장기 스케줄러(Long-term Scheduler), 중기 스케줄러(Medium-termScheduler), 단기 스케줄러(Short-term Scheduler)로 구분된다. 장기 스케줄러는 작업 스케줄러라고도 불리며, 디스크에서 메모리로 어떤 프로세스를 가져올지 결정한다. 중기 스케줄러는 메모리 관리와 관련되어 있으며, 스와핑(Swapping)을 통해 메모리 공간을 확보한다. 단기 스케줄러는 CPU 스케줄러로,준비 큐에 있는 프로세스 중 어떤 것을 다음에 실행할지 결정하는 가장 빈번하게 실행되는 스케줄러이다.

3. 스케줄링 알고리즘의 목표와 기준

스케줄링 알고리즘을 평가하고 비교하기 위해서는 명확한 성능기준이 필요하다. 시스템의 유형과 목적에 따라 중요시되는 기준이 다를 수 있지만, 일반적으로 다음과 같은 기준들이 사용된다.

CPU 이용률 (CPU Utilization)

CPU 이용률은 전체 시스템 시간 중에서 CPU가 실제로 작업을 수행한 시간의 비율을 의미한다. 이상적으로는 CPU가 항상 바쁘게 동작하는 것이 좋으며, 일반적으로 40%에서 90% 사이의 이용률을 목표로 한다. CPU 이용률이 높다는 것은 시스템 자원을 효율적으로 활용하고 있다는 의미이다.

처리량(Throughput)

처리량은 단위 시간당 완료된 프로세스의 개수를 나타낸다. 시간당 몇 개의 작업을 처리할 수 있는지를 측정하는 지표로, 처리량이 높을수록 시스템의 성능이 우수하다고 평가할 수 있다. 특히 배치 처리 시스템에서 중요한 성능 지표이다.

반환 시간(Turnaround Time)

반환 시간은 프로세스가 시스템에 제출된 시점부터 완료될 때까지의총 시간을 의미한다. 여기에는 준비 큐에서 대기한 시간, CPU에서 실행된 시간, 입출력 작업 시간 등이 모두 포함된다. 사용자 관점에서 중요한 지표로, 반환 시간이 짧을수록 사용자 만족도가 높아진다.

대기 시간(Waiting Time)

대기 시간은 프로세스가 준비 큐에서 CPU 할당을 기다린 시간의 총합이다. 스케줄링 알고리즘의 성능을 직접적으로 반영하는 지표로, 대기 시간을 최소화하는 것이 스케줄링 알고리즘의 주요 목표 중 하나이다. 공평성과도 관련이 있어, 특정 프로세스가 과도하게 긴 대기 시간을 겪지 않도록 하는 것이 중요하다.

응답 시간(Response Time)

응답 시간은 프로세스가 제출된 후 첫 번째 응답이 나올 때까지의시간을 의미한다. 대화형 시스템에서 특히 중요한 지표로, 사용자가 명령을 입력한 후 시스템이 반응하기 시작하는 시간이다. 응답 시간이 짧을수록 시스템이 더 반응적으로 느껴진다.

이러한 성능 기준들은 서로 상충되는 경우가 많다. 예를 들어, CPU 이용률을 최대화하려다 보면 일부 프로세스의 대기 시간이 길어질 수 있다. 따라서 스케줄링 알고리즘을 설계할 때는 시스템의 목적과 특성에 맞는 적절한 균형점을 찾는 것이 중요하다.

4. 스케줄링 알고리즘 유형

스케줄링 알고리즘은 크게 선점형(Preemptive)과 비선점형(Non-preemptive)으로 구분된다. 이 분류는 실행 중인 프로세스로부터 CPU를 강제로 빼앗을 수 있는지 여부에 따라 결정된다.

4.1 선점형 스케줄링

선점형 스케줄링은 현재 실행 중인 프로세스로부터 CPU를 강제로 빼앗아 다른 프로세스에게 할당할 수 있는 방식이다. 우선순위가 높은 프로세스가 도착하거나 타임 슬라이스가 만료되면 현재 프로세스의 실행을 중단하고 다른 프로세스에게 CPU를 할당한다.

선점형 스케줄링의 주요 장점은 응답 시간이 빠르고 대화형시스템에 적합하다는 것이다. 또한 우선순위가 높은 작업을 신속하게 처리할 수 있어 시스템의 반응성이 향상된다. 그러나 문맥 교환(Context Switching)이 빈번하게 발생하여 오버헤드가 증가할 수 있으며, 공유 데이터에 대한 동기화 문제가 발생할 수 있다는 단점이 있다.

대표적인 선점형 스케줄링 알고리즘으로는 라운드 로빈(Round Robin), 선점형 우선순위 스케줄링(PreemptivePriority Scheduling), 선점형 SJF(Shortest Remaining TimeFirst) 등이 있다. 현대의 대부분의 운영체제는 선점형 스케줄링을 사용하고 있으며, 이는 멀티태스킹 환경에서 공정성과 반응성을 보장하기 위함이다.

4.2 비선점형 스케줄링

비선점형 스케줄링은 한 프로세스가 CPU를 할당받으면 그 프로세스가 자발적으로 CPU를 반납할 때까지 다른 프로세스가 CPU를 사용할 수 없는 방식이다. 프로세스는 작업을 완료하거나 입출력 작업을 위해 대기 상태로 전환될 때만 CPU를 반납한다.

비선점형 스케줄링의 장점은 구현이 간단하고 문맥 교환의 오버헤드가적다는 것이다. 또한 프로세스가 중단 없이 실행을 완료할 수 있어 특정 작업의 처리 시간을 예측하기 쉽다. 그러나 응답 시간이 길어질 수 있고, 긴 프로세스가 CPU를 독점하면 다른 프로세스들이 무한정 기다릴 수 있다는 단점이 있다.

대표적인 비선점형 스케줄링 알고리즘으로는 FCFS(First-Come, First-Served), 비선점형SJF(Shortest Job First), 비선점형 우선순위 스케줄링 등이 있다. 비선점형 스케줄링은 배치 처리 시스템이나 특정 임베디드 시스템에서 주로 사용된다.

5. 주요 스케줄링 알고리즘

다양한 스케줄링 알고리즘들이 개발되어 왔으며, 각각은 특정 상황과 목적에 최적화되어 있다. 여기서는 가장 널리 사용되고 있는 주요 스케줄링 알고리즘들의 동작 원리와 특징을 살펴본다.

5.1 FCFS (First-Come, First-Served)

FCFS는 가장 간단한 스케줄링 알고리즘으로, 프로세스가 준비 큐에 도착한 순서대로CPU를 할당하는 방식이다. 선입선출(FIFO) 큐를 사용하여 구현되며, 비선점형 알고리즘이다.

FCFS의 가장 큰 장점은 구현이 매우 간단하고 이해하기 쉽다는 것이다. 또한 공평성 측면에서 모든 프로세스가 도착한 순서대로 처리되므로 기아 상태(Starvation)가 발생하지 않는다.그러나 호위 효과(Convoy Effect)라는 심각한 문제점이 있다. 호위 효과란 실행 시간이 긴 프로세스가 먼저 도착하면 뒤에 있는 짧은 프로세스들이 모두 기다려야 하는 현상을 말한다.

예를 들어, CPU 버스트 시간이 24초인 프로세스 P1이 먼저 도착하고, 그 다음 3초와 3초의 버스트 시간을 가진 P2와 P3가 도착했다고 가정하자. FCFS 방식에서는 P1이 완료될 때까지 P2와 P3가 기다려야 하므로 평균 대기 시간이 (0 + 24 + 27) / 3 = 17초가 된다. 만약 P2, P3, P1 순서로 처리했다면 평균 대기 시간은 (0 + 3 + 6)/ 3 = 3초로 크게 감소한다.

따라서 FCFS는 평균 대기 시간이 길고, 특히 CPU 집약적인 프로세스와 입출력 집약적인 프로세스가 혼재된 환경에서는 비효율적이다. 현대의 운영체제에서는 단독으로 사용되지 않으며, 다른 스케줄링 기법과 결합하여 사용되는 경우가 많다.

5.2 SJF (Shortest Job First)

SJF 스케줄링은 준비 큐에 있는 프로세스 중 CPU 버스트 시간이 가장 짧은 프로세스에게 먼저 CPU를 할당하는 방식이다. 이 알고리즘은 이론적으로 평균 대기 시간을 최소화하는 최적의 알고리즘임이 증명되어 있다.

SJF는 비선점형과 선점형 두 가지 버전이 있다. 비선점형 SJF는 일단 CPU가 할당되면 해당 프로세스가 완료될 때까지 실행된다. 반면 선점형 SJF는 SRTF(Shortest Remaining Time First)라고도 불리며, 새로운 프로세스가 도착했을 때 그 프로세스의 버스트 시간이 현재 실행 중인 프로세스의 남은 시간보다 짧으면 선점이 발생한다.

SJF의 주요 장점은 평균 대기 시간과 평균 반환 시간을 최소화할 수 있다는 것이다. 짧은 작업들이 먼저 처리되므로 전체적인 시스템 효율성이 향상된다. 그러나 치명적인 단점이 있는데, 바로 프로세스의 실제 CPU 버스트 시간을 사전에 정확히 알 수 없다는 것이다.

실제 시스템에서는 과거의CPU 버스트 길이를 이용하여 다음 CPU 버스트를 예측하는 방법을 사용한다. 지수 평균(Exponential Averaging) 기법이 대표적인데, 최근의 버스트에 더 큰 가중치를 두어 다음 버스트 시간을 추정한다. 또 다른 문제점은 긴 프로세스가 무한정 기다릴 수 있다는 기아 현상이다. 짧은 프로세스가 계속 도착하면 긴 프로세스는 영원히 CPU를 할당받지 못할 수 있다.

5.3 우선순위 스케줄링 (PriorityScheduling)

우선순위 스케줄링은 각 프로세스에 우선순위를 부여하고, 가장 높은 우선순위를 가진 프로세스에게 CPU를 할당하는 방식이다. 우선순위는 정수로 표현되며, 일반적으로 작은 숫자가 높은 우선순위를 의미한다.

우선순위는 내부적으로 또는 외부적으로 정의될 수 있다. 내부적 우선순위는 시간 제한, 메모리 요구량, 열린 파일 수, 평균 CPU 버스트 대 평균 입출력 버스트의 비율 등 측정 가능한 양을 사용하여 계산된다. 외부적 우선순위는 프로세스의 중요도, 사용료를 지불한 컴퓨터 사용 시간의 유형과 양, 작업을 후원하는 부서 등 운영체제 외부의 기준에 의해 설정된다.

우선순위 스케줄링도 선점형과 비선점형으로 구현될 수 있다. 선점형 우선순위 스케줄링에서는 새로 도착한 프로세스의 우선순위가 현재 실행 중인 프로세스보다 높으면 선점이 발생한다. 비선점형에서는 새로운 프로세스가 준비 큐의 맨 앞에 배치되지만 현재 실행 중인 프로세스는 계속 실행된다.

우선순위 스케줄링의 주요 문제점은 무한 봉쇄(Indefinite Blocking) 또는 기아(Starvation)이다. 낮은 우선순위 프로세스가 높은 우선순위 프로세스들에 의해 무한정 대기할 수 있다. 이 문제를 해결하기 위한 방법이 에이징(Aging)이다. 에이징은 시간이 지남에 따라 프로세스의 우선순위를 점진적으로 증가시키는 기법으로, 오래 기다린 프로세스는 결국 높은 우선순위를 얻게 되어 실행될 수 있다.

SJF는 사실 우선순위 스케줄링의 특수한 경우로 볼 수 있다. 여기서 우선순위는 다음 CPU 버스트 시

간의 역수이며, CPU 버스트가 클수록 우선순위가 낮아진다.

5.4 라운드 로빈 (Round Robin)

라운드 로빈 스케줄링은 시분할 시스템을 위해 특별히 설계된알고리즘으로, 선점형 FCFS라고 볼 수 있다. 각 프로세스는 타임 퀀텀(Time Quantum) 또는 타임 슬라이스(Time Slice)라고 불리는 작은 CPU 시간 단위를 할당받는다. 타임 퀀텀이 만료되면 프로세스는 선점되어 준비 큐의 끝에 배치되고, 다음 프로세스가 CPU를 할당받는다.

라운드 로빈의 성능은 타임 퀀텀의 크기에 크게 의존한다. 타임 퀀텀이 매우 크면 FCFS와 동일하게 동작하고, 매우 작으면 문맥 교환 오버헤드가 과도하게 증가한다. 일반적으로 타임 퀀텀은 10~100밀리초 정도로 설정되며, 문맥 교환 시간은 이의 10% 미만이어야 한다.

라운드 로빈의 주요 장점은 모든 프로세스가 공정하게 CPU 시간을 할당받으며, 응답 시간이 예측 가능하다는 것이다. n개의 프로세스가 준비 큐에 있고 타임 퀀텀이 q일 때, 각 프로세스는 최대 (n-1) × q 시간 단위 이상 기다리지 않는다. 또한 기아 현상이 발생하지 않는다.

그러나 평균 반환 시간은SJF보다 길 수 있으며, 문맥 교환 오버헤드가 있다는 단점이 있다. 특히 모든 프로세스의 CPU 버스트가 타임 퀀텀보다 작으면 불필요한 문맥 교환이 발생한다. 또한 프로세스의 CPU 버스트가 타임 퀀텀의 배수가 아니면 마지막 타임 슬라이스에서 CPU가 낭비될 수 있다.

라운드 로빈은 현대의 시분할 시스템에서 가장 널리 사용되는스케줄링 알고리즘 중 하나이며, 특히 대화형 시스템에서 우수한 성능을 보인다. 많은 운영체제가 라운드 로빈을 기반으로 하여 다양한 변형과 개선된 버전을 구현하고 있다.

5.5 다단계 큐 스케줄링 (Multilevel QueueScheduling)

다단계 큐 스케줄링은 준비 큐를 여러 개의 별도 큐로 분할하는방식이다. 프로세스들은 메모리 크기, 프로세스 우선순위, 프로세스 유형 등의 속성에 따라 하나의 큐에 영구적으로 할당된다. 각 큐는 자신만의 스케줄링 알고리즘을 가질 수 있다.

일반적인 분류는 포그라운드(대화형) 프로세스와 백그라운드(배치) 프로세스로 나누는 것이다. 포그라운드 큐는 라운드 로빈 스케줄링을 사용하여 빠른 응답 시간을 보장하고, 백그라운드 큐는 FCFS를 사용하여 문맥 교환 오버헤드를 줄인다. 더 세분화된 시스템에서는 시스템 프로세스, 대화형 프로세스, 대화형 편집 프로세스, 배치 프로세스, 학생 프로세스 등으로 구분하기도 한다.

큐들 간의 스케줄링도 필요한데, 이는 고정 우선순위 선점형 스케줄링으로 구현되는 것이 일반적이다. 예를 들어, 포그라운드 큐가 백그라운드 큐보다 절대적인 우선순위를 가질 수 있다. 또 다른 방법은 타임 슬라이스 방식으로, 각 큐에 CPU 시간의 일정 비율을 할당하는 것이다. 예를 들어 포그라운드 큐에 80%, 백그라운드 큐에 20%의 CPU 시간을 할당할 수 있다.

다단계 큐 스케줄링의 장점은 서로 다른 유형의 프로세스를효율적으로 처리할 수 있다는 것이다. 각 큐에 최적화된 스케줄링 알고리즘을 적용할 수 있어 전체 시스템 성능이 향상된다. 그러나 프로세스가 한 번 큐에 할당되면 다른 큐로 이동할 수 없어 유연성이 떨어진다는 단점이 있다.

5.6 다단계 피드백 큐 스케줄링 (MultilevelFeedback Queue Scheduling)

다단계 피드백 큐 스케줄링은 다단계 큐 스케줄링의 확장으로, 프로세스가 큐들 사이를 이동할 수 있게 한다. 이 알고리즘은 CPU 버스트의 특성에 따라 프로세스를 분리하는데, 프로세스가 CPU 시간을 너무 많이 사용하면 낮은 우선순위 큐로 이동시킨다. 반대로 낮은 우선순위 큐에서 너무 오래 대기한 프로세스는 높은 우선순위 큐로 이동시켜 기아 현상을 방지한다.

일반적인 다단계 피드백 큐의 구조는 다음과 같다. 새로운 프로세스는 가장 높은 우선순위 큐에 들어간다. 이 큐에서 타임 퀀텀을 모두 사용하면 한 단계 낮은 큐로 이동한다. 이 과정은 가장 낮은 큐에 도달할 때까지 반복된다. 가장 낮은 큐에서는 FCFS 방식으로 처리되며, 프로세스가 완료될 때까지 실행된다.

예를 들어, 세 개의 큐 Q0, Q1, Q2가 있다고 가정하자. Q0는 타임 퀀텀이 8밀리초인 라운드 로빈, Q1은 16밀리초인 라운드 로빈, Q2는FCFS를 사용한다. 새 프로세스는 Q0에 들어가고, 8밀리초 내에 완료되지 않으면 Q1로 이동한다. Q1에서 16밀리초를 모두 사용하면 Q2로 이동하여 FCFS 방식으로 처리

된다. 이러한 방식은 짧은 프로세스를 우대하고 입출력 집약적인 프로세스에게 높은 우선순위를 부여한다.

다단계 피드백 큐 스케줄러는 다음과 같은 매개변수로 정의된다. 큐의 수, 각 큐의 스케줄링 알고리즘, 프로세스를 상위 큐로 승격시키는 시기를 결정하는 방법, 프로세스를 하위 큐로 강등시키는 시기를 결정하는 방법, 그리고 프로세스가 서비스를 필요로 할 때 들어갈 큐를 결정하는 방법이 그것이다.

이 알고리즘은 가장 일반적인 CPU 스케줄링 알고리즘이며, 가장 복잡하기도 하다. 시스템의 특성에 맞게 매개변수를 조정하여 최적화할 수 있다는 장점이 있다. 짧은 프로세스와 입출력 집약적인 프로세스를 우대하면서도 긴 프로세스의 기아를 방지할 수 있다. 많은 현대 운영체제가 이 방식을 기반으로 한 스케줄링 정책을 사용하고 있다.

6. 스케줄링 알고리즘의 성능 평가

스케줄링 알고리즘을 선택하거나 개선하기 위해서는 다양한 방법으로성능을 평가해야 한다. 주요 평가 방법으로는 결정론적 모델링, 큐잉 모델, 시뮬레이션, 그리고 실제 구현이 있다.

결정론적 모델링(Deterministic Modeling)

결정론적 모델링은 특정 워크로드에 대해 각 알고리즘의 성능을분석하는 방법이다. 사전에 정의된 프로세스 집합과 그들의 도착 시간,CPU 버스트 시간을 사용하여 각 알고리즘을 적용했을 때의 평균 대기 시간, 평균 반환 시간 등을 계산한다. 이 방법은 간단하고 빠르며, 알고리즘의 상대적 성능을 명확히 비교할 수 있다는 장점이 있다. 그러나 특정 워크로드에만 적용되므로 일반화하기 어렵고, 실제 시스템의 동적인 특성을 반영하지 못한다는 한계가 있다.

큐잉 모델(Queueing Models)

큐잉 이론은 수학적 분석 방법으로, 프로세스 도착률과 서비스율의 확률 분포를 이용하여 평균 대기 시간, 평균 큐 길이 등을 계산한다. 리틀의 공식(Little's Formula)이 대표적인 예로, 'L = λW' (L은 시스템 내 평균 프로세스 수,λ는 평균 도착률, W는 평균 대기 시간)의 관계를 나타낸다. 큐잉 모델은 이론적으로 우아하고 일반적인 경향을 파악하는 데 유용하지만, 실제 시스템의 복잡성을 모두 반영하기 어렵고 정확한 확률 분포를 알아야 한다는 제약이 있다.

시뮬레이션(Simulation)

시뮬레이션은 컴퓨터 시스템의 모델을 프로그램으로 구현하고, 다양한 워크로드에 대해 스케줄링 알고리즘의 동작을 모의 실험하는 방법이다. 난수 생성기를 사용하여 프로세스 도착 시간, CPU 버스트 시간 등을 생성하거나, 실제 시스템에서 수집한 추적 데이터(trace)를 사용할 수 있다. 시뮬레이션은 복잡한 시스템을 모델링할 수 있고, 다양한 시나리오를 테스트할 수 있다는 장점이 있다. 그러나 시뮬레이션 프로그램 개발에 많은 시간이 소요되고, 결과가 입력 데이터의 정확성에 크게 의존한다는 단점이 있다.

실제 구현(Implementation)

가장 확실한 평가 방법은 실제 운영체제에 스케줄링 알고리즘을구현하여 실제 워크로드에서 테스트하는 것이다. 이 방법은 가장 정확한 결과를 제공하지만, 구현에 많은 비용과 시간이 소요되고, 시스템을 변경하는 것이 어려울 수 있다. 또한 사용자들이 새로운 시스템에 적응하는 시간이 필요하며,환경 변화에 따라 성능이 달라질 수 있다.

실제로는 이러한 방법들을 조합하여 사용한다. 초기 설계 단계에서는 결정론적 모델링과 큐잉 모델을 사용하여 알고리즘의 이론적 특성을 파악하고, 시뮬레이션을 통해 다양한 시나리오에서의 성능을 검증한 후, 최종적으로 실제 시스템에 구현하여 성능을 측정한다.

7. 실시간 스케줄링

실시간 시스템(Real-TimeSystem)은 시간 제약(time constraint) 내에 작업을 완료해야 하는 시스템이다. 일반 시스템과 달리 실시간 시스템에서는 단순히 빠른 응답이 아니라 정해진 시간 내에 작업을 완료하는 것이 중요하다. 실시간 시스템은 경성 실시간 시스템(Hard Real-Time System)과 연성 실시간 시스템(Soft Real-Time System)으로 구분된다.

경성 실시간 시스템은 데드라인을 절대적으로 지켜야 하는 시스템이다. 데드라인을 넘기면 시스템 오류나 치명적인 결과를 초래할 수 있다. 의료 기기, 항공기 제어 시스템, 원자로 제어 시스템 등이 대표적인 예이다. 연성 실시간 시스템은 데드라인을 지키는 것이 중요하지만 가끔 놓쳐도 시스템이 계속 동작할 수 있는 시스템이다. 멀티미디어 스트리밍, 온라인 거래 시스템 등이 이에 해당한다.

실시간 스케줄링에서는 각 태스크의 특성을 명확히 정의해야한다. 주기적 태스크는 일정한 간격으로 반복 실행되며, 각 태스크는 도착 시간, 실행 시간, 데드라인, 주기 등의 속성을 가진다. 스케줄링 알고리즘은 이러한 제약 조건을 만족하면서 모든 태스크를 스케줄할 수 있어야 한다.

RateMonotonic Scheduling (RMS)

RMS는 정적 우선순위 선점형 알고리즘으로, 주기가 짧은 태스크에게 높은 우선순위를 부여한다. 이 알고리즘은 최적의 정적 우선순위 알고리즘임이 증명되어 있다. 즉, RMS로 스케줄할 수 없는 태스크 집합은 다른 어떤 정적 우선순위 알고리즘으로도 스케줄할 수 없다. 그러나 CPU 이용률이 약 69%이상이면 스케줄 가능성을 보장할 수 없다는 한계가 있다.

EarliestDeadline First (EDF)

EDF는 동적 우선순위 선점형 알고리즘으로, 데드라인이 가장 가까운 태스크에게 가장 높은 우선순위를 부여한다. EDF는 이론적으로 CPU 이용률100%까지 스케줄 가능하며, 최적의 동적 우선순위 알고리즘이다. 즉, EDF로 스케줄할 수 없는 태스크 집합은 다른 어떤 알고리즘으로도 스케줄할 수 없다. 그러나 구현이 복잡하고, 오버로드 상황에서의 동작이 예측하기 어렵다는 단점이 있다.

실시간 스케줄링에서는 우선순위 역전(Priority Inversion) 문제도 중요하다. 이는 낮은 우선순위 태스크가 높은 우선순위 태스크가 필요로 하는 자원을 점유하고 있을 때, 중간 우선순위 태스크가 실행되어 높은 우선순위 태스크가 오래 기다리게 되는 현상이다. 이를 해결하기 위해 우선순위 상속(Priority Inheritance) 프로토콜이나 우선순위 천장(PriorityCeiling) 프로토콜 등이 사용된다.

8. 결론

프로세스 스케줄링은 운영체제의 핵심 기능으로, 시스템의 성능과 효율성에 직접적인 영향을 미친다. 본 레포트에서는 프로세스 스케줄링의 기본 개념부터 다양한 스케줄링 알고리즘, 성능 평가 방법, 그리고 실시간 스케줄링까지 폭넓게 살펴보았다.

각 스케줄링 알고리즘은 고유한 특성과 장단점을 가지고 있다. FCFS는 구현이 간단하지만 호위 효과로 인해 평균 대기 시간이 길어질 수 있다. SJF는 이론적으로 최적이지만 CPU 버스트 시간을 미리 알 수 없고 기아 현상이 발생할 수 있다. 우선순위 스케줄링은 중요한 작업을 먼저 처리할 수 있지만 낮은 우선순위 프로세스의 기아 문제가 있다. 라운드 로빈은 공정하고 응답 시간이 예측 가능하지만 문맥 교환 오버헤드가 있다.

현대의 운영체제는 단일 알고리즘을 사용하기보다는 다단계 피드백큐와 같이 여러 알고리즘을 결합한 복잡한 스케줄링 정책을 사용한다. 이를 통해 다양한 유형의 프로세스를 효과적으로 처리하고, CPU 이용률, 처리량, 응답 시간 등 여러 성능 지표 사이의 균형을 맞출 수 있다.

실시간 시스템에서는 일반 시스템과는 다른 스케줄링 접근법이필요하다. RMS와 EDF 같은 실시간 스케줄링 알고리즘은 시간 제약을 만족하면서 태스크를 스케줄하는 데 초점을 맞춘다. 특히 경성 실시간 시스템에서는 데드라인을 보장하는 것이 절대적으로 중요하다.

프로세스 스케줄링은 계속 진화하고 있다. 멀티코어 프로세서의 등장으로 다중 처리기 스케줄링이 중요해졌고, 에너지 효율을 고려한 스케줄링, 가상화 환경에서의 스케줄링 등 새로운 도전과제들이 제기되고 있다. 또한 인공지능과 머신러닝 기법을 활용하여 워크로드 패턴을 학습하고 최적의 스케줄링 결정을 내리는 연구도 진행되고 있다.

결론적으로, 효과적인 프로세스 스케줄링은 시스템의 목적과 특성을 명확히 이해하고, 다양한 알고리즘의 특성을 고려하여 적절한 스케줄링 정책을 선택하고 구현하는 것이다. 이를 통해 시스템 자원을 효율적으로 활용하고, 사용자에게 우수한 경험을 제공할 수 있다.

9. 참고문헌

1.Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating SystemConcepts (10th ed.). Wiley.

2. Tanenbaum,A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.

3. Stallings,W. (2018). Operating Systems: Internals and Design Principles (9th ed.).Pearson.

4. Liu, C. L.,& Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in aHard-Real-Time Environment. Journal of the ACM, 20(1), 46-61.

5. Buttazzo, G.C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithmsand Applications (3rd ed.). Springer.

6. Kleinrock,L. (1976). Queueing Systems, Volume 2: Computer Applications. Wiley.

7. Love, R.(2010). Linux Kernel Development (3rd ed.). Addison-Wesley Professional.

10. 느낀점

프로세스 스케줄링 레포트를 작성하면서 가장 크게 느낀 점은완벽한 알고리즘은 없다는 것이다. CPU 이용률을 높이면 응답 시간이 길어지고, 공정성을 추구하면 효율성이 떨어지는 등 항상 트레이드오프가 존재했다. 평소 스마트폰이나 컴퓨터로 여러 작업을 동시에 하는 것을 당연하게 여겼는데, 그 배경에 이렇게 복잡하고 정교한 스케줄링 알고리즘이 있다는 것을 알게 되었다.

특히 이론적으로 최적인SJF조차 실제 구현에는 어려움이 있다는 점에서 이론과 실제의 간극을 실감했다. 또한 기아 현상을 방지하기 위한 에이징 기법처럼, 스케줄링이 단순한 기술적 문제가 아니라 공정성과 형평성의 가치를 담고 있다는 점이 인상 깊었다.

운영체제가 단순히 컴퓨터를 관리하는 것을 넘어, 제한된 자원을 효율적이면서도 공정하게 배분하는 철학을 담고 있음을 배울 수 있었다.