# 운영체제 핵심 개념 통합 가이드

*CPU* 스케줄링, 동기화 도구, 교착상태, 주 메모리 관리의 완전한 이해

## 목차



## 1. CPU 스케줄링

### 1.1 개요와 핵심 개념

CPU 스케줄링은 운영체제의 핵심 기능으로, 한정된 CPU 자원을 여러 프로세스가 효율적이고 공정하게 사용할 수 있도록 관리하는 기술입니다 . 다중프로그래밍 환경에서 CPU 활용률을 최대화하고 시스템 성능을 향상시키는 것이 주요 목표입니다 .[1] [1]

### 1.2 프로세스 실행 모델

#### CPU I/O 버스트 사이클

프로세스 실행은 CPU 실행과 I/O 대기 구간이 번갈아 나타나는 사이클로 구성됩니다 . 이를 CPU I/O 버스트 사이클이라고 하며, 대부분의 프로세스는 짧은 CPU 버스트가 많고 긴 CPU 버스트가 적은 패턴을 보입니다 .[1] [1]

#### 프로세스 유형별 특성

- **I/O 중심 프로세스**: 짧은 CPU 버스트, 빈번한 I/O 요청, 높은 사용자 상호작용 [1]
- **CPU 중심 프로세스**: 긴 CPU 버스트, 적은 I/O 요청, 높은 계산 처리 [1]

### 1.3 CPU 스케줄러와 디스패처

#### 스케줄러 기능

CPU 스케줄러는 준비 큐에서 프로세스를 선택하여 CPU 코어를 할당합니다 . 스케줄링 결정은 프로세스 상태 전환 시점에 발생하며, 선점형과 비선점형으로 구분됩니다 .[1] [1]

### 디스패처 역할

디스패처는 스케줄러가 선택한 프로세스에게 실제로 CPU 제어권을 전달하는 모듈로, 컨텍스트 스위칭, 모드 전환, 프로그램 카운터 점프 등을 수행합니다 .[1]

## 1.4 스케줄링 기준과 성능 지표

주요 성능 지표는 다음과 같습니다 :[1]

- **CPU 활용률**: CPU가 바쁜 시간의 비율 (최대화 목표)
- **처리량**: 단위 시간당 완료된 프로세스 수 (최대화 목표)
- **반환 시간**: 제출부터 완료까지의 총 시간 (최소화 목표)
- **대기 시간**: 준비 큐에서 보낸 시간 (최소화 목표)
- **응답 시간**: 요청부터 첫 응답까지의 시간 (최소화 목표)

## 1.5 스케줄링 알고리즘

### 선입선출 (FCFS)

가장 단순한 비선점형 알고리즘으로, 먼저 도착한 프로세스를 먼저 처리합니다 .[1] 호송 효과(convoy effect) 문제가 발생할 수 있어 짧은 프로세스들이 긴 프로세스 뒤에서 오래 기다리게 됩니다 .[1]

### 최단 작업 우선 (SJF)

평균 대기 시간이 최적인 알고리즘으로, 가장 짧은 CPU 버스트를 가진 프로세스를 먼저 실행합니다 .[1] 선점형 버전은 SRTF Shortest Remaining Time First)라고 합니다 .[1]

### 우선순위 스케줄링

각 프로세스에 우선순위를 부여하여 높은 우선순위 프로세스를 먼저 실행합니다 .[1] 기아 현상을 방지하기 위해 에이징 기법을 사용할 수 있습니다 .[1]

### 라운드 로빈 (RR)

시분할 시스템의 핵심 알고리즘으로, 각 프로세스에 동일한 시간 할당량을 부여하여 순환 실행합니다 .[1] 공정한 CPU 시간 분배를 보장하지만 컨텍스트 스위치 오버헤드가 발생합니다 .[1]

### 다단계 큐 스케줄링

다단계 큐 스케줄링

프로세스 유형별로 별도의 큐를 사용하며, 각 큐마다 다른 스케줄링 알고리즘을 적용할 수 있습니다 .[1] 시스템 프로세스, 대화형 프로세스, 배치 프로세스 등으로 분류합니다 .[1]

### 다단계 피드백 큐

프로세스가 큐 간 이동이 가능한 고급 알고리즘으로, 프로세스의 실행 특성에 따라 동적으로 우선순위를 조정합니다 .[1]

## 1.6 다중처리기 스케줄링

다중처리기 환경에서는 로드 밸런싱, 프로세서 친화성, 대칭 다중처리(SMP) 등을 고려해야 합니다 .[1] 각 프로세서는 자체 스케줄러를 가지거나 공통 스케줄러를 사용할 수 있습니다 .[1]

## 1.7 실시간 스케줄링

실시간 시스템에서는 데드라인 보장이 핵심이며, 하드 실시간과 소프트 실시간으로 구분됩니다 .[1] Rate Monotonic Scheduling과 Earliest Deadline First 등의 알고리즘이 사용됩니다 .[1]

# 2. 동기화 도구

## 2.1 배경 및 기본 개념

### 동시성과 데이터 일관성

프로세스는 동시에 실행될 수 있으며, 언제든지 실행이 부분적으로 완료된 상태로 중단될 수 있습니다 .[2] 공유 데이터에 대한 동시 접근은 데이터 불일치를 초래할 수 있어, 협력하는 프로세스의 순서 있는 실행을 보장하는 메커니즘이 필요합니다 .[2]

### 경합 조건(Race Condition)

여러 프로세스가 동시에 공유 데이터에 접근할 때, 실행 순서에 따라 결과가 달라지는 상황입니다 .[2] 예를 들어, `counter++` 연산이 세 개의 기계어 명령어로 구현될 때 중간에 다른 프로세스가 개입하면 예상과 다른 결과가 나올 수 있습니다 .[2]

## 2.2 임계 구역 문제

### 임계 구역 정의

n개의 프로세스가 공통 변수를 변경하거나 테이블을 업데이트하는 등의 작업을 수행하는 코드 세그먼트입니다 .[2] 한 프로세스가 임계 구역에서 실행 중일 때 다른 프로세스는 자신의 임계 구역에 들어갈 수 없어야 합니다 .[2]

### 해결책의 요구사항

- **상호 배제**: 프로세스가 임계 구역에서 실행 중이면 다른 프로세스는 진입 불가 [2]
- **진행**: 임계 구역이 비어있고 진입 요청이 있으면 선택 과정이 무한 연기되지 않아야 함 [2]
- **유한 대기**: 진입 요청 후 허가까지 다른 프로세스의 진입 횟수에 상한이 존재해야 함 [2]

### 2.3 피터슨의 해결책

두 프로세스를 위한 소프트웨어 기반 해결책으로, `turn` 변수와 `flag` 배열을 사용합니다 . 세 가지 요구사항을 모두 만족하지만, 현대 아키텍처에서는 명령어 재배열로 인해 올바르게 작동하지 않을 수 있습니다 .

### 2.4 하드웨어 지원

#### Test-and-Set 명령어

현재 값을 반환하고 새 값을 true로 설정하는 원자적 명령어입니다 . 이를 이용해 뮤텍스 락의 기본 구현이 가능하지만 바쁜 대기 문제가 있습니다 .

#### Compare-and-Swap 명령어

값이 예상값과 같으면 새 값으로 변경하고 이전 값을 반환하는 원자적 명령어입니다 .

### 2.5 뮤텍스 락

가장 간단한 동기화 도구로, `acquire()` 와 `release()` 함수를 제공합니다 . 락이 사용 가능한지 나타내는 부울 변수를 사용하며, 바쁜 대기를 사용하므로 스핀락이라고도 불립니다 .

### 2.6 세마포어

#### 기본 개념

정수 변수 S와 `wait()` P , `signal()` V 두 원자적 연산으로 구성되는 동기화 도구입니다 . 카운팅 세마포어와 이진 세마포어로 구분됩니다 .

#### 세마포어 구현

바쁜 대기 없는 구현에서는 각 세마포어마다 대기 큐를 유지하고, `block` 과 `wakeup` 연산을 사용합니다 . `wait()` 에서 값이 음수가 되면 프로세스를 대기 큐에 추가하고, `signal()` 에서 대기 중인 프로세스를 깨웁니다 .

#### 사용 예시

- 임계 구역 문제: 세마포어를 1로 초기화하여 상호 배제 보장
- 실행 순서 동기화: 세마포어를 0으로 초기화하여 특정 순서 보장

### 2.7 모니터

### 모니터 개념

프로세스 동기화를 위한 고수준 추상화 메커니즘으로, 내부 변수는 프로시저 내에서만 접근 가능하며 한 번에 하나의 프로세스만 활성화됩니다 . [2]

### 조건 변수

`x.wait()` 와 `x.signal()` 연산을 제공하여 프로세스의 중단과 재시작을 제어합니다 . 모니터와 함께 사용되어 복잡한 동기화 문제를 해결할 수 있습니다 . [2] [2]

## 2.8 고전적인 동기화 문제

### 유한 버퍼 문제

프로듀서와 소비자가 고정 크기 버퍼를 공유하는 문제로, `mutex`, `full`, `empty` 세마포어를 사용하여 해결합니다 . [2]

### 읽기-쓰기 문제

다수의 읽기 프로세스와 쓰기 프로세스가 데이터베이스를 공유하는 문제로, 읽기는 동시에 가능하지만 쓰기는 독점적이어야 합니다 . [2]

### 식사하는 철학자 문제

5명의 철학자가 5개의 젓가락을 공유하는 문제로, 교착상태 방지를 위한 다양한 해결책이 있습니다 . [2]

# 3. 교착상태

## 3.1 시스템 모델과 특성

### 시스템 구성

시스템은 자원 유형 $R_1$, $R_2$, ..., $R_m$으로 구성되며, 각 자원 유형은 여러 인스턴스를 가집니다 . 프로세스는 자원을 요청, 사용, 해제하는 순서로 활용합니다 . [3] [3]

### 교착상태의 4가지 필요조건

교착상태가 발생하려면 다음 네 조건이 동시에 만족되어야 합니다 : [3]

- **상호 배제**: 한 번에 하나의 프로세스만 자원 사용 가능 [3]
- **점유와 대기**: 자원을 보유한 채로 다른 자원을 대기 [3]
- **비선점**: 자원을 강제로 빼앗을 수 없음 [3]
- **순환 대기**: 프로세스들이 원형으로 서로의 자원을 대기

### 3.2 자원 할당 그래프

프로세스와 자원 간의 관계를 나타내는 방향 그래프로, 요청 간선과 할당 간선으로 구성됩니다 . 사이클이 없으면 교착상태가 없고, 사이클이 있으면 교착상태 가능성이 있습니다 .[3][3]

### 3.3 교착상태 처리 방법

#### 예방 (Prevention)

네 가지 필요조건 중 하나를 무효화하여 교착상태를 원천 차단합니다 :[3]

- **점유와 대기 무효화**: 모든 자원을 한 번에 요청하거나 자원을 보유하지 않은 상태에서만 요청[3]
- **비선점 무효화**: 추가 자원 요청 시 현재 자원을 모두 해제[3]
- **순환 대기 무효화**: 자원에 순서를 부여하여 오름차순으로만 요청[3]

#### 회피 (Avoidance)

시스템이 안전 상태를 유지하도록 동적으로 자원 할당을 제어합니다 . 은행가 알고리즘이 대표적인 회피 기법입니다 .[3][3]

#### 은행가 알고리즘

각 프로세스의 최대 자원 요구량을 미리 알고 있다고 가정하고, 자원 할당 요청 시 안전성을 검사합니다 . Available, Max, Allocation, Need 행렬을 사용하여 안전 순서의 존재 여부를 확인합니다 .[3][3]

#### 탐지 및 복구 (Detection & Recovery)

교착상태 발생을 허용하되 주기적으로 탐지하여 복구합니다 :[3]

- **탐지**: 대기 그래프의 사이클 검사 또는 다중 인스턴스용 탐지 알고리즘 사용[3]
- **복구**: 프로세스 종료 또는 자원 선점을 통해 교착상태 해결[3]

### 3.4 실제 시스템에서의 적용

대부분의 현대 운영체제(Windows, Linux, macOS)는 성능상의 이유로 교착상태를 무시하는 접근법을 사용합니다 . 교착상태 발생 빈도가 매우 낮고 예방/회피의 오버헤드가 크기 때문입니다 .[3][3]

## 4. 주 메모리 관리

### 4.1 개요와 메모리 계층 구조

주 메모리 관리는 한정된 메모리 자원을 효율적으로 관리하고 프로세스 간 메모리 보호를 제공하는 운영체제의 핵심 기능입니다 . 메모리 계층은 CPU 레지스터, 캐시, 주 메모리, 보조 저장소 순으로 구성되며 속도와 용량이 반비례 관계를 가집니다 .[4][4]

## 4.2 메모리 보호

### 베이스 레지스터와 리밋 레지스터

베이스 레지스터는 프로세스의 시작 주소를, 리밋 레지스터는 프로세스 크기를 저장하여 메모리 접근을 제한합니다[4]. MMU가 모든 메모리 접근을 검사하여 보호 위반 시 트랩을 발생시킵니다[4].

## 4.3 주소 바인딩

### 바인딩 시점

- **컴파일 시간**: 메모리 위치가 컴파일 시점에 확정[4]
- **로드 시간**: 프로그램이 메모리에 로드될 때 주소 결정[4]
- **실행 시간**: 실행 중 동적으로 주소 변환 (MMU 필요)[4]

### 논리 주소와 물리 주소

- **논리 주소**: 프로그램이 생성하는 가상 주소[4]
- **물리 주소**: 실제 메모리 하드웨어의 주소[4]
- **MMU**: 논리 주소를 물리 주소로 변환하는 하드웨어[4]

## 4.4 동적 로딩과 연결

### 동적 로딩

프로그램 실행 중 필요한 루틴만 메모리에 로드하여 메모리 공간을 효율적으로 활용합니다[4]. 사용되지 않는 루틴은 로드되지 않아 큰 프로그램에 적합합니다[4].

### 동적 연결

실행 시점에 라이브러리와 연결하여 실행 파일 크기를 줄이고 공유 라이브러리를 사용할 수 있습니다[4]. 스텁 메커니즘을 통해 실제 라이브러리 루틴 위치를 찾습니다[4].

## 4.5 연속 메모리 할당

### 메모리 분할

주 메모리는 운영체제 영역과 사용자 프로세스 영역으로 분할됩니다[4]. 가변 분할 방식에서는 홀(빈 공간)과 할당된 영역이 동적으로 변화합니다[4].

### 동적 저장소 할당 알고리즘

- **First Fit**: 첫 번째 적합한 홀에 할당 (빠름)[4]
- **Best Fit**: 가장 적합한 홀에 할당 (공간 효율적)[4]
- **Worst Fit**: 가장 큰 홀에 할당 (비효율적)[4]

### 단편화 문제

단편화 문제

- **외부 단편화**: 총 여유 공간은 충분하지만 연속되지 않아 할당 불가 [4]
- **내부 단편화**: 할당된 공간이 요청보다 커서 일부가 낭비됨 [4]
- **압축**: 외부 단편화 해결을 위해 메모리 재배치 (비용 높음) [4]

## 4.6 페이징

### 페이징 개념

페이징 개념

물리 메모리를 고정 크기 프레임으로, 논리 메모리를 같은 크기의 페이지로 나누어 외부 단편화를 해결합니다 [4]. 페이지 테이블을 통해 논리 주소를 물리 주소로 변환합니다 [4].

### 주소 변환

주소 변환

논리 주소는 페이지 번호(p)와 페이지 오프셋(d)으로 구성되며, 페이지 테이블에서 해당 프레임 번호를 찾아 물리 주소를 계산합니다 [4].

### TLB  Translation Lookaside Buffer)

페이지 테이블 접근 횟수를 줄이기 위한 고속 캐시로, 최근 사용된 페이지 변환 정보를 저장합니다 [4].
TLB 히트율이 성능에 큰 영향을 미칩니다 [4].

### 메모리 보호

메모리 보호

페이지 테이블 항목에 유효-무효 비트와 보호 비트를 포함하여 페이지별 접근 권한을 제어합니다 [4].
공유 페이지를 통해 메모리 효율성을 높일 수 있습니다 [4].

## 4.7 페이지 테이블 구조

### 계층적 페이지 테이블

계층적 페이지 테이블

페이지 테이블 크기 문제를 해결하기 위해 다단계 페이지 테이블을 사용합니다 [4]. 64비트 시스템에서는 3단계 이상의 페이징이 필요할 수 있습니다 [4].

### 해시 페이지 테이블

해시 페이지 테이블

희소 주소 공간에서 효과적인 방법으로, 가상 페이지 번호를 해시하여 물리 프레임을 찾습니다 [4]. 클러스터 페이지 테이블로 연속된 페이지 접근을 최적화할 수 있습니다 [4].

### 역 페이지 테이블

역 페이지 테이블

물리 메모리 기준으로 구성되어 시스템당 하나만 존재하므로 메모리를 절약합니다 [4]. 각 프레임이 어떤 프로세스의 어떤 페이지에 할당되었는지 기록합니다 [4].

### 4.8 스와핑

메모리 부족 시 프로세스를 일시적으로 보조 저장소로 이동시키는 기법입니다[4]. 스와핑 시간은 전송률과 프로세스 크기에 따라 결정되며, 현대 시스템에서는 페이지 단위 스와핑이 일반적입니다[4].

## 결론

이 통합 문서는 운영체제의 핵심 개념인 CPU 스케줄링, 동기화 도구, 교착상태, 주 메모리 관리를 포괄적으로 다루었습니다. 각 영역은 서로 밀접하게 연관되어 있으며, 현대 운영체제의 효율적인 동작을 위해 모두 중요한 역할을 합니다. 이러한 개념들의 깊이 있는 이해는 시스템 성능 최적화와 안정성 보장에 필수적입니다.

⁂

chapter05-cpu-scheduling.md

chapter06-synchronization-tools.md

chapter08-deadlocks.md

chapter09-main-memory.md