**프로세스 동기화 & 상호 배제**

**Process synchronization**

-다중 프로그램 시스템: 여러 프로세스들이 독립적으로 동시에 동작, 공유 자원 데이터가 있을 때 문제 발생

-동기화(synchronization)- 프로세스간의 대화: 동작을 맞추고 정보를 공유하는 행위

**Asynchronous and concurrent p’s**

-비동기적: 프로세스들이 서로 모름

-병행적: 여러 프로세스들이 동시에 시스템에 존재

- 병행 수행적인 비동기적 프로세스들이 공유할 때 문제 발생- 대화가 없음

Shared data-여러 프로세서들이 공유하는 데이터

Critical data-공유 데이터를 접근하는 코드 영역

**Mutual exclusion**- 둘 이상의 프로세서가 동시에 critical section에 진입하는 것을 막는 것

-methods: 1. Mutual exclusion primitives (기본 연산) – enterCS() primitive-진입 전 검사

- exitCS() primitive- 후 처리 과정 알림

Requirement of ME primitives

-mutual exclusion

-progress –프로세스가 CS에 진입하는 것을 방해하면 안됨

-bounded waiting- 프로세스는 유한 시간 내에 허용 되어야함

**SW solutions**

1. Dekker’s algorithm-two process ME를 보장하는 최고의 알고리즘
2. Dijkstra’s algorithm-

속도가 느림 구현이 느림 busy waiting

**HW solutions**

1. TestAndSet instruction- Test와 Set을 한번에 수행하는 기계어, interrupt를 받지 않음

Busy waiting

**OS supported SW solution**

1. Spinlock: 정수 변수

-멀티 프로세스만 가능

- busy waiting

2. Semaphore: 음이 아닌 정수형 변수(S) 임의의 s변수 하나에 ready queue 하나가 할당됨

-상호 배제 문제

-프로세스 동기화 문제

-reader-writer 문제

-producer-consumer problem

-no busy waiting

- wake-up 순서는 비결정적 starvation problem

3. Eventcount/sequence: 은행의 번호표와 비슷한 개념 sequencer, Eventcount: 정수형 변수 tickets(S) individual operation

- no busy waiting

- no starvation

- semaphore 보다 더 low-level control이 가능

**Language-Level solution (High-level mechanism)**

1. Monitor(책방): language 보장

* 자원 할당 문제
* Producer-Consumer problem
* Reader-writer problem
* Dining philosopher problem

사용이 쉽다 deadlock error 발생 가능성이 낮음

지원하는 언어에서만 사용 가능 컴파일러가 OS를 이해하고 있어야 함