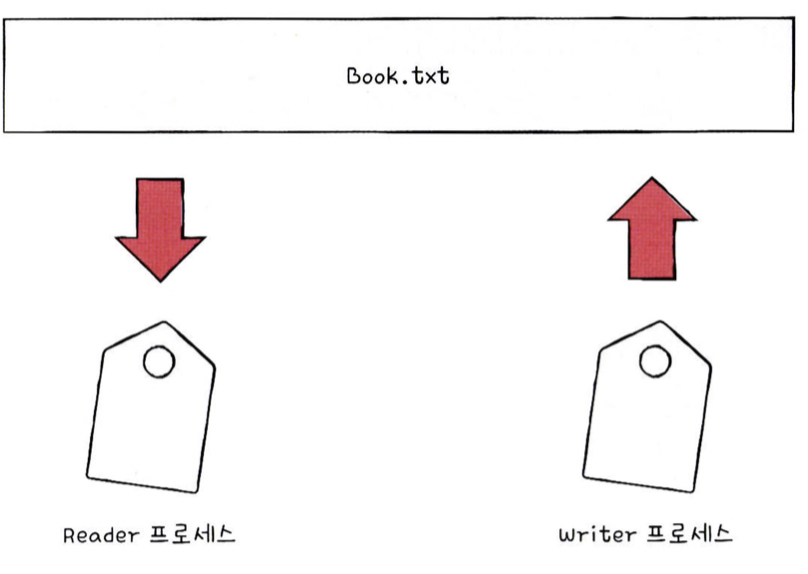
**Chapter 12 프로세스 동기화**

***12-1*** **동기화란**

**<동기화의 의미>**

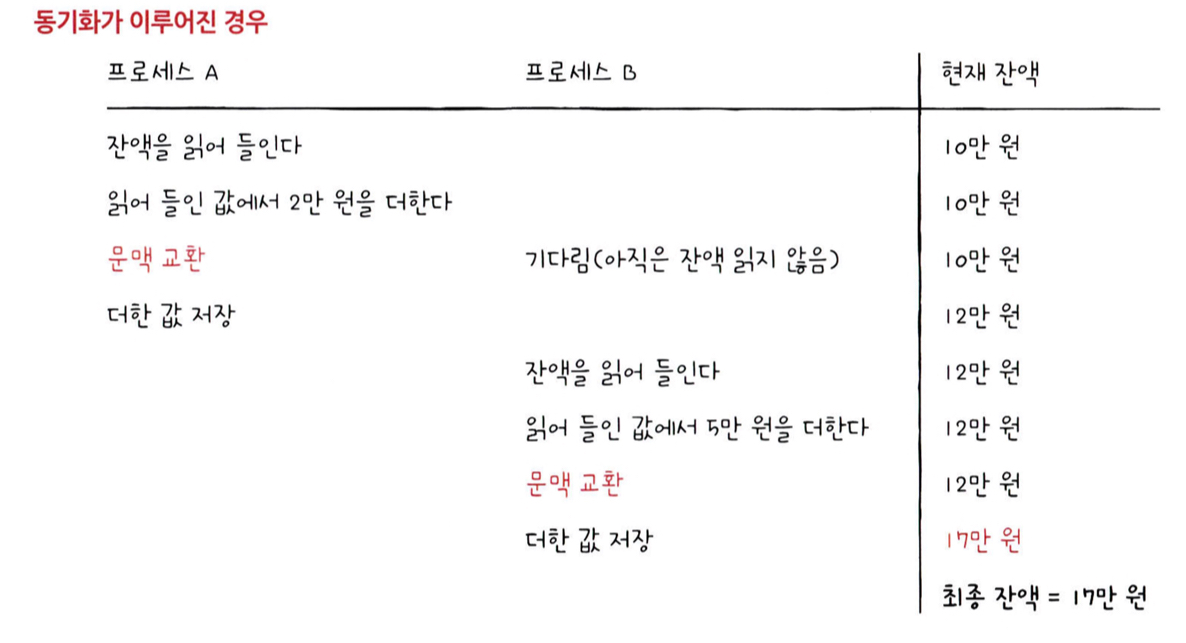
**프로세스 동기화:** 프로세스들 사이의 수행 시기를 맞추는 것

* 협력적으로 실행되는 프로세스들은 실행 순서와 자원의 일관성을 보장해야 함
* **실행 순서 제어**: 프로세스를 올바른 순서대로 실행
* **상호 배제**: 동시에 접근해서는 안 되는 자원에 하나의 프로세스만 접근하게 하기



**실행 순서 제어**

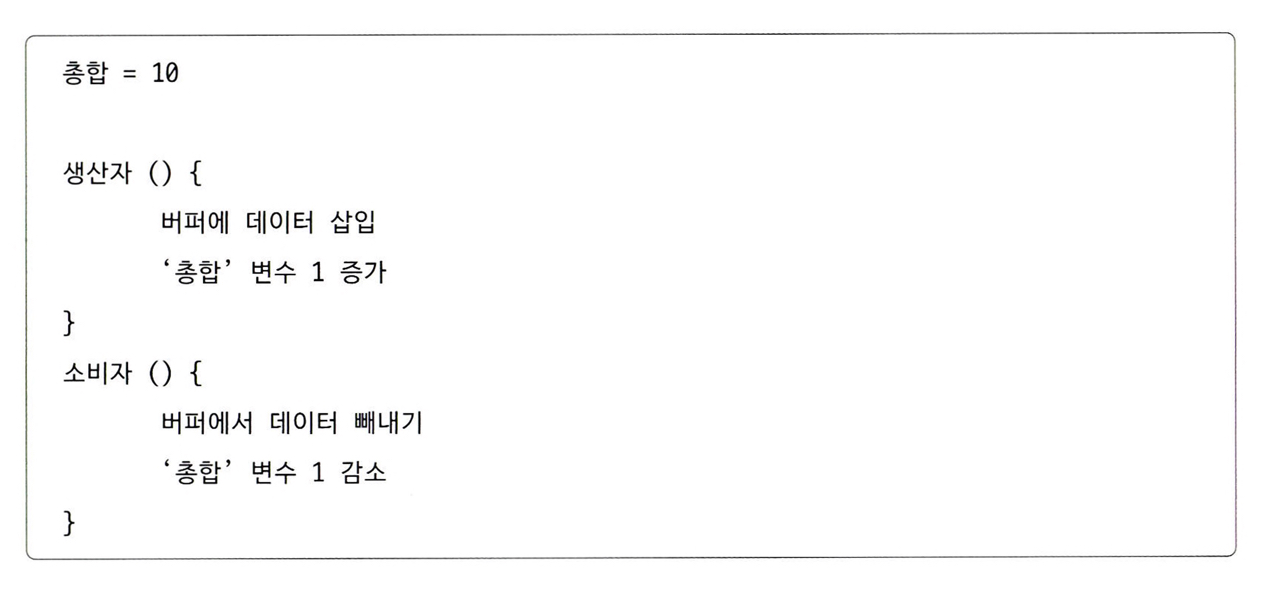
* 동시에 실행되는 프로세스를 올바른 순서대로 실행하는 것
* Reader 프로세스: 파일에 저장된 값을 읽어들임
* Writer 프로세스: 파일에 값을 저장
* Reader 프로세스는 Book.txt 안에 값이 존재한다는 특정 조건이 만족되어야만 실행을 이어나갈 수 있음



**상호 배제**

* 공유가 불가능한 자원의 동시 사용을 피하기 위해 사용하는 알고리즘
* **상호 배제를 위한 동기화**: 동시에 접근해서는 안 되는 자원에 동시에 접근하지 못하게 함

**<생산자와 소비자 문제>**

****

* 기대되는 총합의 변수는 10이지만 막상 코드 실행 시 다른 수가 되거나 오류가 날 가능성이 있음
* 생산자와 소비자는 총합이라는 데이터를 동시에 사용 -> **공유 자원**
* 소비자가 생산자의 작업이 끝나기도 전에 총합을 수정, 생산자가 소비자 작업이 끝나기 전에 총합을 수정

**<공유 자원과 임계 구역>**

**임계 구역**: 동시에 실행하면 문제가 발생하는 자원에 접근하는 코드 영역

* 두 개 이상의 프로세스가 임계 구역에 진입하고자 하면 둘 중 하나는 대기해야 함
* 임계 구역에 먼저 진입한 프로세스의 작업이 마무리되면 기다렸던 프로세스가 임계 구역에 진입 가능

**레이스 컨디션**: 잘못된 실행으로 인해 여러 프로세스가 동시 다발적으로 임계 구역의 코드를 실행하여 문제가 발생하는 경우 -> 데이터의 일관성이 깨짐

* 레이스 컨디션이 발생하는 근본적인 이유는 컴퓨터가 여러 줄의 저급 언어로 변환된 고급 언어 한 줄을 실행하는 과정에서 문맥 교환이 일어나기 때문

**상호 배제를 위한 동기화 원칙**

* **상호 배제**: 한 프로세스가 임계 구역에 진입했다면 다른 프로세스는 임계 구역에 들어올 수 없다.
* **진행**: 임계 구역에 어떤 프로세스도 진입하지 않았다면 임계 구역에 진입하고자 하는 프로세스는 들어갈 수 있어야 한다.
* **유한 대기**: 한 프로세스가 임계 구역에 진입하고 싶다면 그 프로세스는 언젠가는 임계 구역에 들어올 수 있어야 한다(무한정 대기x)

다른 프로세스의 기아를 방지하기 위해 한 번 임계 구역에 들어간 프로세스는 그 다음에 임계 구역에 들어갈 때 제한을 두어야 함.

***12-2*** **동기화 기법**

동기화를 위한 대표적인 도구로는 뮤텍스락, 세마포, 모니터가 있음

**<뮤텍스 락>**

* 상호 배제를 위한 동기화 도구로 자물쇠와 같은 기능을 코드로 구현한 것
* 자물쇠 역할: 프로세스들이 공유하는 **전역 변수 lock**
* 임계 구역을 잠그는 역할: **acquire 함수**
* 임계 구역의 잠금을 해제하는 역할: **release 함수**
* acquire 함수: 프로세스가 임계 구역에 진입하기 전에 호출.

임계 구역이 **잠겨** 있다면(lock이 true인 상태) 임계 구역이 열릴때까지 임계 구역을 **반복적**으로 확인 -> **바쁜 대기**(busy wait)

임계 구역이 **열려** 있다면(lock이 false인 상태) 임계 구역을 **잠금**(lock을 true로 변경)

* release 함수 : 임계 구역에서의 작업이 끝나고 호출.

현재 잠긴 임계 구역을 **열어주는** 함수(lock을 false로 변경)

**<세마포>**

* 뮤텍스 락과 비슷하지만 조금 더 일반화된 방식의 동기화 도구
* 공유 자원이 **여러 개** 있는 상황에서도 적용이 가능
* STOP, GO 신호를 받는 것에 따라 기다리거나 임계 구역에 들어감
* 임계 구역에 진입할 수 있는 프로세스의 개수: **전역 변수 S**
* 임계 구역에 들어갈지, 기다릴지 알려주는 역할: **wait 함수**
* 기다리는 프로세스에게 임계 구역에 들어가도록 신호를 주는 역할: **signal 함수**
* wait 함수: 임계 구역 진입 전 호출

1. 임계 구역에 진입할 수 있는 프로세스 개수가 0 이라면(S <= 0)
2. 사용할 수 있는 자원이 있는지 반복적으로 확인 -> 바쁜 대기
3. 임계 구역에 진입할 수 있는 프로세스 개수가 1 이상이면(S >=1) S를 1 감소시키고 임계 구역 진입

* signal 함수: 임계 구역 진입 후 호출

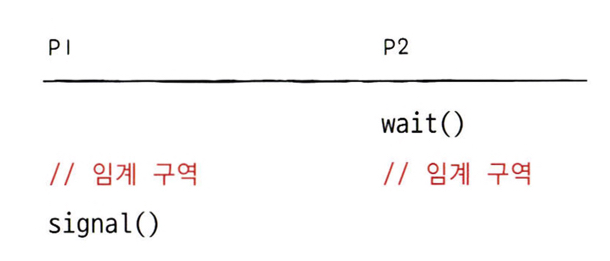
1. 임계 구역에서의 작업을 마친 뒤 S를 1 증가시킴

* 세마포는 바쁜 대기를 해결하기 위해 더 좋은 방법을 사용함

1. wait 함수는 사용할 수 있는 자원이 없을 경우 해당 프로세스 상태를 대기로 만듦
2. 그 프로세스의 PCB를 세마포를 위한 대기 큐에 집어 넣음
3. 다른 프로세스가 임계 구역에서 작업 후 signal 함수를 호출
4. signal 함수는 대기 중인 프로세스를 대기 큐에서 제거
5. 프로세스 상태를 준비 상태로 변경한 뒤 준비 큐로 옮김

* 세마포를 이용해서 동시에 실행되는 프로세스의 실행 순서도 제어 가능

1. 세마포의 변수 S를 0으로 두고 먼저 실행할 프로세스 뒤에 signal 함수 호출
2. 다음에 실행할 프로세스 앞에 wait 함수를 호출

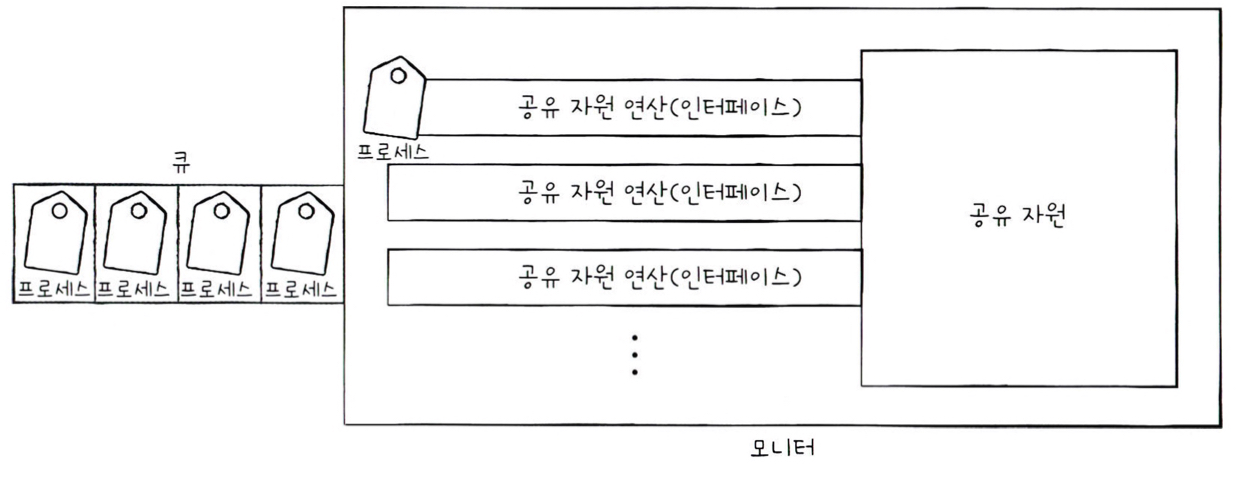


**<모니터>**

세마포의 단점: 매번 임계 구역 앞뒤로 일일이 wait와 signal 함수를 명시해야 함

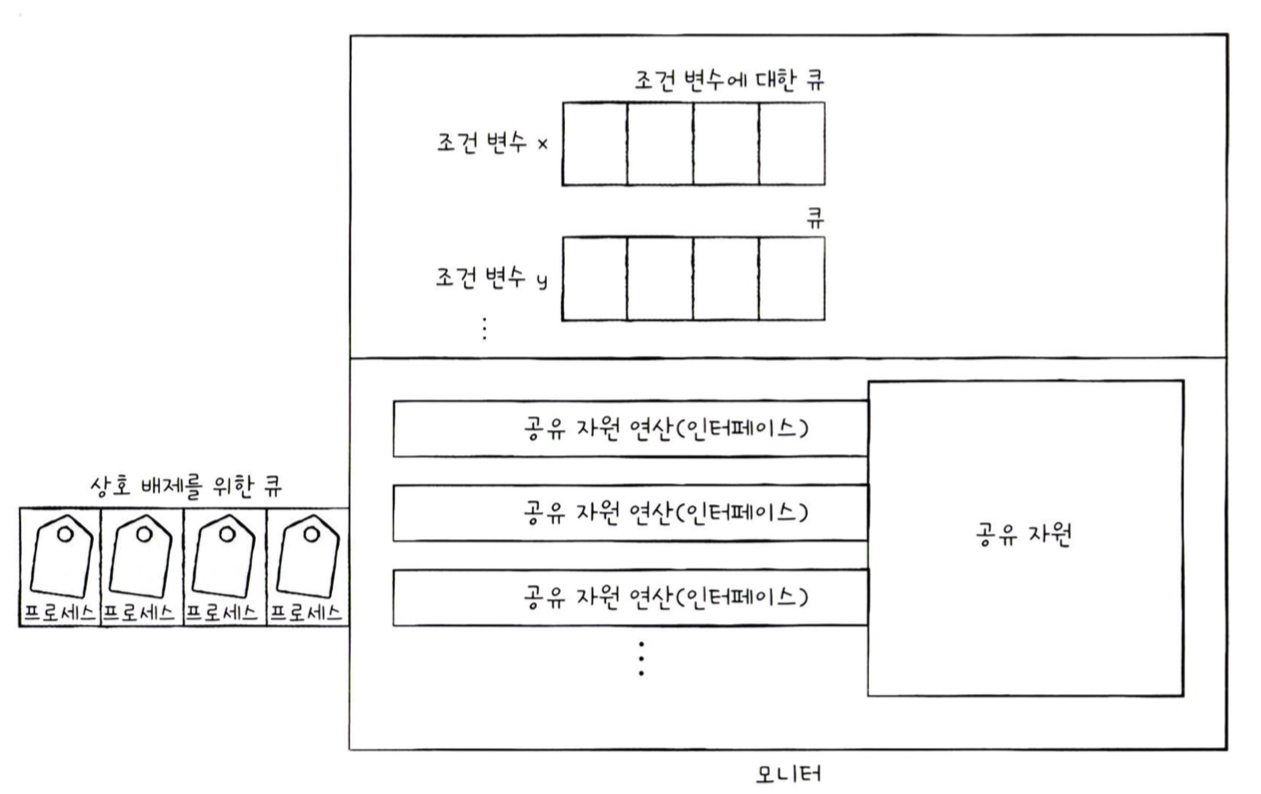
-> 세마포에 비해 사용자가 사용하기에 훨씬 편리한 도구: **모니터**

* 공유 자원과 공유 자원에 접근하기 위한 인터페이스를 묶어 관리
* 프로세스는 반드시 인터페이스를 통해서만 공유 자원에 접근



1. 공유 자원에 접근하려면 모니터 안의 인터페이스를 통해야 함
2. 공유 자원에 접근하고자 하는 프로세스를 큐에 삽입
3. 인터페이스에 하나의 프로세스만 들어오도록 상호 배제를 위한 동기화 제공

* 모니터는 실행 순서 제어를 위한 동기화도 제공
* 특정 조건을 바탕으로 프로세스를 실행하고 일시 중단하기 위해 **조건 변수**를 사용
* 조건 변수로는 wait와 signal 연산을 수행 가능
* wait: 호출한 프로세스의 상태를 대기 상태로 전환하고 일시적으로 조건 변수에 대한 대기 큐에 삽입하는 연산



**상호 배제를 위한 큐**: 모니터에 한 번에 하나의 프로세스만 진입하도록 하기 위해 만들어진 큐

**조건 변수에 대한 큐**: 모니터에 이미 진입한 프로세스의 실행 조건이 만족될 때까지 잠시 실행이 중단되어 기다리기 위해 만들어진 큐

* signal: wait를 호출하여 큐에 삽입된 프로세스의 실행을 재개하는 연산

뮤텍스와 모니터의 차이점

* 뮤텍스는 다른 프로세스 간에 동기화를 할 때 사용한다. 보통 **운영체제 커널**에 의해 제공되며 무겁고 느리다.
* **운영체제 커널** : 항상 메모리에 올라가 있는 운영체제의 핵심 부분
* 모니터는 하나의 프로세스 내 다른 스레드 간 동기화를 할 때 사용한다. 프레임워크나 라이브러리 그 자체에서 제공되며 가볍고 빠르다.