1. **병행 프로세스1 개관**

병행 프로세스의 기본 개념과 상호배제와 동기화 문제 해결하기 위해 제공되는 장치 학습목표

1. **병행성과 병행 프로세스의 개념을 이해한다.**
2. **병행성 문제를 이해한다.**
3. **세마포어의 개념을 이해한다. 주요용어**

병행성

독립 프로세스 협력 프로세스 임계영역 상호배제 동기화 세마포어

* 1. **병행 프로세스의 개요**
     1. **병행성과 병행 프로세스 [51]**

# 병행성(concurrency)

여러개의 프로세스 또는 쓰레드가 동시 수행되는 시스템의 특성 병행 프로세스

동시에 수행되는 여러 개의 프로세스 또는 쓰레드 병행 프로세스가 실행되는 형태는 ()에 따라 차이가 있다.

**CPU의 개수**

하나의 CPU에서 병행 프로세스가 실행되는 경우 인터리빙 형식

각 프로세스가 짧은 시간간격으로 번갈아 가며 실행

정확히 한 순간에는 하나의 프로세스가 실행, 전체적으로 보면 여러 프로세스가 동시에 수행 여러 개의 CPU에서 병행 프로세스가 실행되는 경우

정확히 한순간에도 여러 프로세스가 실행되는 병렬처리 방식

**여러 개의 CPU, 즉 멀티프로세서(multiprocessor) 시스템에서 병행 프로세스가 실행되는 경우 메모리 구조**

**강결합(tighty-coupled) 시스템 여러 CPU가 하나의 기억장치 공유**

# 예) master-slave, SMP(Symmetric MultiProcessing; 대칭구조)

**마스터 슬레이브는 약결합 시스템이라는 의견도 있음 (RAM을 공유하지 않는 경우 대부분) 약결합(loosely-coupled) 시스템**

각 시스템은 자신의 운영체제와 기억장치를 가지고 있으며, 독립적으로 운영되고 필요할 때 통신한다. 2개 이상의 독립된 컴퓨터 시스템이 네트워크로 서로 연결

시스템 부하를 조절하기 위해 부하가 적은 시스템에 작업을 보내기도 함. 약결합 시스템 간 통신

메시지 전달

**원격 프로시저 호출 (Remote Procedure Calls: RPC)**

**예) 클러스터 구조 : Cluster Structure**

* 1. **병행 프로세스의 개요 (2)**
     1. **프로세스 간의 관계 [53]**

독립 프로세스

자신의 데이터, 상태 공유 x

수행 중인 다른 프로세스에 영향을 주지도 않고 받지도 않는 프로세스 독립 프로세스의 실행

# 결정적(deterministic)

재생가능

입력 데이터만 동일하다면 실행결과 항상 동일 다른 프로세스와 무관하게 중단/재시작 가능 협력 프로세스

다른 프로세스와 데이터 공유, 자신의 상태 공유

수행 중인 다른 프로세스와 영향을 주고받으며 동작하는 프로세스 협력 프로세스의 실행

**비결정적(nondeterministic) 재생 불가능**

입력 데이터가 동일하더라도 실행결과는 달라질 수 있음

예) 두 사람이 하나의 계좌를 이용하여 돈을 입금하거나 출금하는 경우

**잔고(자원): 10,000 프로세스A: 5,000 입급 프로세스B: 15,000 출금 협력 프로세스는 자원을 공유, 상태를 공유, 일 처리의 서순이 중요하다. => 멱등성 보장X**

* 1. **병행성 문제 [54]**

협력 프로세스는 병행성과 관련한 다양한 문제가 발생할 수 있다.

* + 1. **상호배제**

**상호 배제 (mutual exclusion)**

**2개 이상의 프로세스가 동시에 임계영역 수행하지 못하도록 하는 것 임계영역 (critical section)**

2개 이상의 프로세스가 동시에 사용하면 안 되는 공유자원을 액세스하는 프로그램 코드 영역 예제 (갱신 손실)

**자원: 10,000원 프A: +5,000 입금 프B: +8,000 입금**

# 프A: 10,000 읽기 ---> +5,000 = 15,000원 프B: --> 10,000 읽기 ---> +8,000 = 18,000원

**===================================**

갱신 손실 발생 해결책

임계영역, 상호배제를 명확하게 설정

**최종 18,000원**

* + 1. **동기화**

임계영역: 계좌의 잔고를 읽고 입금액을 더한 후 더해진 액수를 계좌에 쓰는 일련의 입금과정에 해당하는 코드

상호배제: 한 프로세스가 임계영역에 해당하는 입금과정을 수행 중이라면, 다른 프로세스는 입금과정을 수행하지 못하 도록 하는 것

**프로세스 동기화(synchronization)**

2개 이상의 프로세스에 대한 "처리순서를 결정"하는 것

**예) 한 계좌에서 다른 계좌로 10, 000원을 송금한다는 가정**

프로세스 A는 출금을 담당하고 프로세스 B는 입금을 담당할 때, A가 10,000원을 출금 처리한 후 프로세스 B가 이어서

**10,000원 입금을 처리하도록 동기화 필요**

상호배제 또한 임계영역에 대한 동기화 문제

* + 1. **통신**

**프로세스 간 통신(InterProcess Communication: IPC)**

서로 다른 프로세스들이 데이터를 주고받거나 동기화하는 방법

운영체제에서 각 프로세스는 독립적인 주소 공간을 가지므로, 프로세스 간 데이터를 직접 공유할 수 없고, 특정한 IPC 기법을 사용해야 함

통신하는 방법에 다라 상호배제 및 동기화 문제를 포함하여 다양한 문제 발생할 수 있음=> 5장 IPC 주요 방식

**메시지 기반(Message-based) 파이프(Pipe)**

# 소켓(Socket)

**메시지 큐(Message Queue)**

**메모리 기반(Memory-based) 방식 공유 메모리 (Shared Memory)**

# 메모리 매핑(Memory-Mapped Files, MMF)

**동기화(Synchronization) 방식 세마포어 (Semaphore)**

* 1. **세마포어 [56]**

**뮤텍스 (Mutex)**

**이벤트 (Event)**

* + 1. **세마포어의 정의 세마포어(semaphore)**

상호배제와 동기화 문제를 해결하기 위해 다익스트라가 제안한 도구 세마포어 s

정수형 공용변수

**(1) 사용 가능한 자원의 수 또는 (2) 잠김이나 풀림 상태의 값**

처음 세마포어 s를 선언할 때 상황에 맞는 값(0 이상인 정수)으로 초기화 한다. 세마포어 s는 기본 연산 P와 V에 의해서만 사용될 수 있다.

**연산 P와 V를 통하지 않고는 세마포어 s의 값을 확인하거나 수정할 수 없다. 세마포어의 기본 연산 (primitive operation)**

기본연산 P

**down(감소) / wait(대기)**

기본연산 V

**up(증가) / signal(진행 신호)**

'두 연산 P와 V가 기본연산이다'의 의미 인터럽트되지 않고 하나의 단위로 처리된다

어떤 프로세스가 세마포어 s에 대해 연산 P나 V를 수행 중이라면 다른 어떤 프로세스도 세마포어 s에 대해 연산 P나 V를 수행할 수 없다.

예를 들어, P(s)의 경우 s>0임을 확인 후 s--를 하기 전에 다른 프로세스에 의해 세마포어 s의 값이 바뀌는 일은 발 생하지 않는다.

* + 1. **상호배제 해결**

세마포어는 임계영역에 대한 상호배제 문제를 해결할 수 있다.

한 프로세스가 임계영역을 수행 중이라면, 다른 프로세스는 접근X 임계영역을 수행 중이던 프로세스가 임계영역을 벗어나면 접근 허용O

상호 배제 해결 방법

**임계영역 시작 부분 (진입영역): 연산 P**

임계영역에 대한 수행을 해도 되는지 체크 임계영역 끝 부분(해제영역) : 연산 V

다른 프로세스가 임계영역 수행을 시작할 수 있도록 함

진입 영역에서 대기하는 프로세스들은 '적절한 시간' 내에 임계영역 수행을 시작할 수 있어야 한다. 세마포어 mutex

임계영역 수행의 시작 가능 여부 초기 값 1

임계영역을 수행할 수 있음

# 0

임계 영역을 수행할 수 없음

**진입영역을 P(mutex)로 두면 어떤 프로세스가 P(mutex)를 수행할 때**

**mutex가 1인 경우**

**mutex만 0으로 바뀌고 그 프로세스는임계영역 수행을 시작하게 된다. mutex가 0인 경우**

**현재 프로세스는 대기 상태로 전이되고 mutex의 대기 큐에 추가 된다. 이 때 mutex는 변함없이 0이다.**

**해제영역을 V(mutex)로 두면임계영역 수행을 마친 프로세스가 V(mutex)를 수행할 때**

**mutex의 대기 큐가 비어있다면**

**mutex만 1로 바뀐다.**

**mutex의 대기 큐가 비어있지 않다면**

가장 오래 기다린 프로세스를 큐에서 빼서 준비상태로 전이시킨다. 이 때 mutex는 계속 0이다.

준비상태로 전이된 프로세스가 디스패치되어 실행상태가 되면 연산 P가 완료되며, 임계영역 수행을 시작

세마포어와 뮤텍스는 모두 세마포어의 한 종류다

**4.3. 세마포어 (2) [60]**

* + 1. **동기화 해결**

세마포어는 프로세스 동기화 문제를 해결할 수 있다.

예) 프로세스 A가 특정 코드 S1을 수행한 후에 프로세스 B가 코드 S2를 수행하도록 동기화를 하려면? 코드 S2 앞에 연산 P를 두고 코드 S1 뒤에 연산 V를 두는 것

**이 때 세마포어 sync의 초기 값은 0이다.**

세마포어 변수의 초기 값을 0으로 두면, 연산 V로 시작하면 된다.

초기 값 0은 먼저 수행되어야 하는 코드 S1이 아직 수행되지 않았으므로 코드 S2를 수행할 수 없음을 의미한다.

요약

1. **병행성은 여러 개의 프로세스 또는 스레드가 동시에 수행되는 시스템의 특성을 의미하고, 동시에 수행되는 여러 개의 프로세스 또 는 스레드를 병행 프로세스라고 한다.**
2. **병행 프로세스는 프로세스 간의 관계에 따라 독립 프로세스 또는 협력 프로세스라고 한다.**
3. **협력 프로세스는 병행성과 관련하여 상호배제, 동기화, 통신과 관련한 다양한 문제가 발생할 수 있다.**
4. **임계영역은 2개 이상의 프로세스가 동시에 사용하면 안되는 공유자원을 액세스하는 프로그램 코드 영역을 의미한다.**
5. **상호배제는 2개 이상의 프로세스가 동시에 임계영역을 수행하지 못하도록 하는 것이다.**
6. **동기화는 2개 이상의 프로세스에 대한 처리순서를 결정하는 것이다.**
7. **세마포어는 상호배제와 동기화 문제를 해결하기 위한 정수형 공용변수로, 2개의 기본 연산 P와 V에 의해서만 사용될 수 있다.**
8. **병행 프로세스 2**

개관

개관

협력 프로세스에서 발생할 수 있는 문제 생산자-소비자 문제

판독기-기록기 문제 학습목표

1. **생산자-소비자 문제를 알아보고, 세마포어를 이용해 해결하기**
2. **판독기-기록기 문제를 알아보고, 세마포어를 이용하여 해결하기 주요 용어**

생산자-소비자 문제 판독기-기록기 문제 IPC

통신 링크 직접 통신 간접 통신

* 1. **생산자-소비자 문제 [65]**
     1. **생산자-소비자 문제의 정의 생산자-소비자 문제**

두 협력 프로세스 사이에 (공유) 버퍼를 두고, (생산자) 한쪽 프로세스는 데이터를 넣고 를 꺼내는 상황을 다루는 문제

**생산자-소비자 문제의 구체적인 조건(X2)**

버퍼에 여러 프로세스가 동시에 접근할 수 없다.

생산자-소비자 문제를 해결하기 위해 상호배제가 필요하다.

(소비자) 다른 쪽 프로세스는 데이터

* + - 1. **생산자가 버퍼에 데이터를 넣는 동안 소비자가 버퍼에서 데이터를 꺼낼 수 없다.**
      2. **소비자가 버퍼에서 데이터를 꺼내는 동안 생산자는 버퍼에 데이터를 넣을 수 없다. 유한 버퍼(bounded buffer) : 버퍼의 크기가 유한하다.**

동기화가 필요하다.

(버퍼는 유한하므로) 버퍼가 가득 찬 경우

생산자는 소비자가 버퍼에서 데이터를 꺼낼 때까지 기다려야 한다. 버퍼가 빈 경우

소비자는 생산자가 버퍼에 데이터를 넣을 때까지 기다려야 한다.

* + 1. **세마포어를 이용한 해결**

생산자-소비자 문제는 "세마포어"를 이용하여 해결이 가능함

**mutex, empty, full을 이용한 생산자와 소비자 코드 세마포어의 초기 값**

# mutex : 1

상호배제

**empty : n (버퍼의 크기)**

버퍼에 존재하는 빈 공간의 개수

# full : 0

버퍼에 존재하는 데이터의 개수 상호배제를 위해 세마포어 mutex 이용

임계영역

생산자가 버퍼에 데이터를 넣는 부분 + 소비자가 버퍼에 데이터를 꺼내는 부분 임계영역 앞에 P(mutex)를 두고 임계영역 뒤에 V(mutex)를 두어 상호배제를 해결

동기화

버퍼가 가득 찬 경우 동기화 세마포어 empty를 이용

**생산자는 버퍼에 데이터를 넣는 코드 앞에 P(empty) 소비자는 버퍼에서 데이터를 꺼내는 코드 뒤 V(empty)**

버퍼가 빈 경우의 동기화 세마포어 full을 이용

**소비자는 버퍼에서 데이터를 꺼내는 코드 앞에 P(full) 생산자는 버퍼에 데이터를 넣는 코드 뒤에 V(full)**

**그림 5-2 [66]**

**empty = n, full = 0인 경우 소비자는?**

생산자는?

대기 중인 소비자가 없다면?

* 1. **판독기-기록기 문제**
     1. **판독기-기록기 문제의 정의 판독기-기록기 문제**

**여러 개의 판독기(Reader)와 하나 이상의 기록기(Writer)가 하나의 자원을 공유할 때 발생하는 동기화 문제 원칙**

하나의 기록기가 공유자원에 데이터를 쓰는 중에는 다른 기록기나 판독기는 공유자원에 접근할 수 없다. 상호배제가 필요하다.

기록기 공유자원에 데이터 쓰는 동안 누구도 공유자원에 접근 X

판독기 공유자원에 데이터 읽는 동안 기록기 공유자원에 접근 X

여러 판독기는 동시에 공유자원에서 데이터를 읽을 수 있다. (기록기는 이 동안 공유자원에 접근 불가)

만약 판독기가 공유자원의 데이터를 읽는 중이어서 기록기가 공유자원에 접근하지 못하고 대기하고 있는데 새로운 판독 기가 공유자원에 접근하려 한다면?

같은 판독기 우선?

대기 중인 기록기 우선?

* + - 1. **제1판독기-기록기 문제**

판독기가 공유자원에 접근 중이라면 기록기보다 판독기에 우선순위를 주는 것 장점

**제2원칙(조건)을 충실히 따른다. 단점**

**기록기가 기아상태에 빠질 수 있다. 기아상태(starvation)**

프로세스가 필요한 자원을 할당받지 못하고 계속적으로 대기하게 되는 상황

* + - 1. **제2판독기-기록기 문제**

판독기가 공유자원에 접근 중이라면 판독기보다 기록기에 우선순위를 주는 것

대기 중인 기록기가 있다면 새로운 판독기는 두 번째 조건에도 불구하고 공유자원에 접근할 수 없다. 장점

기록기가 기아상태에 빠지는 것 방지 단점

판독기의 병행성이 떨어진다.

기록기에 우선순위를 주는 방식에 따라 판독기가 기아상태에 빠질 수도 있다.

**5.2. 판독기-기록기 문제 (2)**

* + 1. **세마포어를 이용한 해결 제1판독기-기록기 문제해결**

기록기에 대한 상호배제 세마포어 wrt를 이용

**임계영역1: 기록기가 공유자원에 데이터를 쓰는 부분 임계영역 앞에 P(wrt) 임계영역 뒤에 V(wrt)**

**임계영역2: 판독기가 공유자원에서 데이터를 읽는 부분 임계영역 앞에 P(wrt) 임계영역 뒤에 V(wrt)**

단, 판독기가 임계영역을 수행 중일 때 다른 판독기도 임계영역 수행을 시작할 수 있어야 한다. → 일반변수

**rcount를 둔다.**

**일반변수 rcount**

동시에 공유자원의 데이터를 읽는 판독기의 개수

**rcount가 1보다 크다면?**

**임계영역 수행이 끝나 rcount를 1만큼 줄였을 때 여전히 0보다 크다면?**

**그림 5-4**

**제2판독기-기록기 문제 해결 그림 5-5**

내용

기록기가 대기 중일 때 이후에 새로운 판독기가 들어오면 임계영역을 수행하지 못하도록 세마포어 rd가 제1판독기-기록기 코드에 추가

기록기에 우선순위를 주기 위해 세마포어 mutex3도 추가

잘 이해 안돼요..

기록기에 우선순위를 준다라...

**5.3. 프로세스 간 통신[72]**

**프로세스 간 통신(InterProcess Communication: IPC)**

공유 메모리 이용 메시지 이용 특징

하나의 운영체제에서 공유 메모리와 메시지가 함께 사용될 수 있음

1. **공유 메모리 방법 공유 메모리 방법**

협력 프로세스가 동일한 변수를 사용함으로써 데이터를 서로 공유하는 방법

프로세스 A가 데이터를 변수 c에 대입하면 프로세스 B는 변수 c의 값을 읽으면 된다. 장점

빠른 속도로 통신: 각 프로세스가 메모리에 직접 접근하여 대량의 데이터를 쓰고 읽기 때문 단점

상호배제나 동기화 등 통신상 발생할 수 있는 문제를 응용 프로그래머가 책임지고 해결해야 함

1. **메시지 전달방법 메시지 전달방법**

협력 프로세스가 메시지를 주고받으면서 데이터를 서로 공유

**메시지를 주고받기 위해 커널이 제공하는 연산 "send" 연산 "recieve" 이용**

* 1. **프로세스 A는 메시지 M을 시스템 호출 send(M)을 이용하여 프로세스 B에 보내기**
  2. **프로세스 B는 메시지 M을 시스템 호출 recieve(M)을 이용하여 받기 이 때 연산 send와 recieve는 모두 커널 모드에서 동작한다.**

장점

통신상 발생할 수 있는 상호 배제, 동기화 문제를 운영체제가 책임진다 단점

매번 시스템 호출을 통해야 하므로 대량의 데이터보다는 소량의 데이터를 주고받는 데 적합한 방법이다.

1. **통신 링크**

두 프로세스가 서로 메시지를 주고받을 때, 두 프로세스 사이에 메시지가 전달되는 경로 통신 링크가 존재함. 특징

* 1. **다양한 성질을 가질 수 있음 그에 따라 다양한 형태로 구현 가능**
  2. **링크의 방향성을 단방향 또는 양방향으로 설정 가능**
  3. **링크의 용량을 여러 형태로 설정 가능 통신 링크의 용량이란?**

링크가 자체적으로 갖는 큐(버퍼)에 보관할 수 있는 메시지의 수 큐의 크기가 무한하다면

큐의 크기가 유한하다면 큐가 아예 없는 경우라면?

송신자는 수신자가 직접 메시지를 받을 수 있을 때까지 대기해야 한다.

* 1. **프로세스 간 통신 (2) [73]**

**2. 메시지 전달방법 (2)**

1. **직접 통신**

두 프로세스가 직접 서로를 지정하여 메시지를 주고 받는 방법

**송신자는 연산 send에 수신자를 직접 명시, 수신자는 연산 recieve에 송신자를 직접 명시**

**send(B,M) : 프로세스 A가 프로세스 B로 메시지 M을 보낸다 recieve(A,M) : 프로세스 B가 프로세스 A로부터 메시 지 M을 받는다**

특징

서로를 지정한 두 프로세스 사이에 "오직 하나의 통신 링크" 양방향

두 프로세스는 서로 메시지 전달 연산을 바꿔서 사용할 수 있음 대칭형 주소 지정

송신자와 수신자가 각각 상대를 직접 명시 비대칭형 주소 지정

수신자가 송신자를 미리 지정하지 않고 메시지를 받을 때 송신자 이름도 받는 경우

# recieve(id,m)

메시지 m을 받으며 송신자 이름을 id에 받는 형태

수신자가 여러 송신자와의 통신 링크를 갖는 경우에 사용될 수 있음

1. **간접 통신**

통신을 원하는 프로세스들 사이에 우편함(mailbox)를 두고 이를 통해 메시지를 주고받는 방법

**송신자: 연산 send에 수신자 대신 우편함을 명시 수신자: 연산 recieve에 송신자 대신 우편함을 명시 특징/ 시나리오**

두 프로세스 사이의 통신 링크는 두 프로세스가 같은 우편함을 이용하는 경우 설정됨. 한 프로세스가 송신자이면 다른 프로세스는 수신자여야 함.

두 프로세스가 여러 개의 우편함을 이용하여 서로 메시지를 주고 받는다면? 여러 개의 통신 링크

여러 프로세스가 하나의 우편함을 이용 하나의 링크는 여러 프로세스와 연관

송신자가 여럿이고 수신자가 하나 통신 링크 단방향

대개 우편함이 수신 프로세스에 소속되는 경우, 수신 프로세스가 종료하면 우편함도 같이 사라짐. 수신자가 여럿

우편함이 운영체제에 소속되는 경우

우편함에 있는 메시지를 누가 받을 것인지 정할 필요가 있음

한순간에 하나의 수신자만 우편함에서 메시지를 받을 수 있도록 운영체제가 수신자를 관리하는 방법이 있음 통신 링크는 양방향

요약

1. **생산자-소비자 문제는 상호배제와 동기화가 필요한 문제로 세마포어를 이용하여 구현할 수 있다.**
2. **판독기-기록기 문제는 기록기는 상호배제가 필요하나, 판독기는 상황에 따라 다르게 처리할 필요가 있어 세마포어를 이용하여 구 현할 수 있다.**
3. **판독기-기록기 문제는 특정 상황에서 판독기에 우선순위를 주는 형태의 문제와 기록기에 우선순위를 주는 형태의 문제로 정의할 수 있다.**
4. **프로세스 간 통신 방법에는 공유 메모리 방법과 메시지 전달 방법이 있다.**
5. **공유 메모리 방법은 협력 프로세스가 공유 메모리를 이용하여 동일한 변수를 사용함으로써 데이터를 서로 공유하는 방법이다.**
6. **메시지 전달 방법은 협력 프로세스가 메시지를 주고받으면서 데이터를 공유하는 방법이다.**
7. **메시지 전달 방법에는 송신자와 수신자가 직접 서로를 지정하여 메시지를 주고받는 "직접 통신 방식"과 운영체제의 우편함을 활 용해 메시지를 주고받는 "간접 통신 방식"이 있다.**
8. **교착상태1 개관**

개관

컴퓨터 시스템의 제한된 자원을 사용하기 위해 서로 경쟁할 수 있음 교착상태(deadlock) 발생 가능

운영체제는 교착상태를 예방하거나 제거함으로써 프로세스의 동작이 원활하게 이루어지도록 해야 함 학습 목표

1. **교착상태의 개념을 이해한다.**
2. **교착상태가 발생하기 위한 필요조건을 이해한다.**
3. **자원할당 그래프의 개념을 이해한다.**
4. **교착상태를 예방하는 방법을 이해한다. 주요 용어**

교착상태 점유대기 환형대기 자원할당 그래프 교착상태 예방

* 1. **교착상태의 개요[81] 프로세스가 자원을 사용하는 절차**

프로세스는 필요한 자원이 있으면 1) 요구과정(시스템 콜 등)을 통해 2) 자원을 획득한 후 사용하고, 3) 사용이 끝나면 획득했던 자원을 해제하여 반납함

**요구과정에서 가용한 자원이 없으면 자원을 획득할 때까지 대기 교착상태(deadlock)**

여러 개의 프로세스가 서로 상대방의 작업이 끝나기만 기다리고 있기 때문에 결과적으로 어느 쪽도 영원히 진행하지 못하는 상 태

**두 개 이상의 프로세스가 (자원을 점유한 상태에서) 교착상태 vs 기아상태(starvation)**

교착상태

상대방이 작업을 끝내고 자원을 해제하기만 무한히 기다리는 교착상태

관련된 모든 프로세스가 자원획득의 가능성 없이 무한히 대기상태 기아상태

관련된 프로세스의 일부가 자원획득의 가능성은 있으나 계속적으로 대기 상태

(b)는 가로 방향 차들이 일정 기간 없는 경우가 생긴다면 세로 방향의 차들이 진행할 수 있기 때문에 교착상태가 아님

* 1. **교착상태의 특성 [82]**
     1. **교착상태의 필요조건 (X4)**

**교착상태는 (X4)조건이 "동시에" 만족될 때 발생"할 수 있다." (필요조건)**

**상호배제(mutual exclusion) 조건**

프로세스가 자원에 대한 배타적 통제권을 요구

**적어도 하나 이상의 자원은 여러 프로세스에 의해 동시에 사용될 수 없음 점유대기(hold and wait) 조건**

1) 프로세스가 이미 자원을 할당받아 점유하고 있는 상황에서 2) 다른 프로세스가 점유하고 있는 또 다른 자원을 요청하여 3)

해제되기를 기다리는 상황

**비선점(no preemption) 조건**

**프로세스에 할당된 자원이 그 프로세스가 사용을 마치고 스스로 반환하기 전에 해제되지 않음 환형대기(circular wait) 조건**

프로세스의 자원 점유 및 점유된 자원의 요구 관계가 환형을 이루며 대기하는 상황이 존재

* + 1. **자원할당 그래프**

교착상태를 명확하게 표현하기 위한 그래프 자원할당 그래프 G=(V, E)는 방향 그래프

**V: 정점의 집합**

E: 방향이 있는 간선의 집합 요구간선

할당간선 그래프 표현

각 프로세스 Pi는 원, 자원 r은 사각형으로 표기

자원은 프린터나 디스크처럼 자원의 유형을 나타내기에 각 자원 r... 마다 하나 이상의 단위자원을 가질 수 있다. 단위자원은 자원 사각형 위에 단위자원의 개수를 숫자로 표시한다.

그래프 표현2

프로세스 p(i)가 r(j)의 단위자원 하나를 요구할 때 그래프에는 요구간선이 하나 추가된다.

이 요구가 만족되면, 즉 가용한 단위자원이 존재하면 요구간선은 할당간선으로 바뀌고 사용이 끝나서 자원이 해제될 때 할당 간선이 삭제된다.

* 1. **교착상태의 특성 [82]**
     1. **자원할당 그래프(2)**

자원할당 그래프에서의 교착상태의 필요 조건 상호배제 조건

하나의 단위자원은 하나의 프로세스에만 할당되므로, 하나의 할당간선으로 표현 점유대기 조건

한 프로세스에 할당간선과 요구간선이 연결된 것으로 표현 비선점 조건

가용한 단위자원이 없는 경우 요구간선으로 표현 환형대기 조건

**할당간선과 요구간선으로 만들어지는 사이클(cycle)**

자원할당 그래프에 사이클이 없으면 교착상태가 존재하지 않고, 사이클이 있으면 교착상태가 발생할 수 있다. 물론 필요조건이 모두 충족되었다고 해서 항상 교착상태가 발생하는 것은 아님

**[그림 6-6]의 위 그림과 같이 자원 r1이 가진 두 개의 단위자원이 프로세스 p1과 p4에 할당된 경우, 지금은 r1이 가용**

한 단위자원이 없어 사이클

**'r1 -> p1 -> r2 -> p2 -> r1'이 존재하지만,**

프로세스 p4가 자원 r1을 다 쓰고 해제하면

**가운데 그림 처럼 r1에 가용한 단위자원이 하나 생기고, 요구 간선 (p2, r1)이 할당간선 (r1, p2)로 바뀌어 자원할당 그**

래프에서 사이클이 사라지게 된다.

* + 1. **교착상태의 처리 기법 교착상태 예방(prevention)**

따라서 교착상태가 발생하지 않는다.

**교착상태의 네 가지 필요조건이 동시에 만족되는 것을 피함 교착상태 회피(avoidance)**

**프로세스에 필요한 자원의 최대량에 대한 정보를 이용하여 교착상태가 발생하지 않도록 하는 방법 교착상태 탐지 및 복구(detection and recovery)**

교착상태의 발생 여부를 조사하여 발생한 경우에는 적절한 조치를 취해 정상상태로 복구

* 1. **교착상태 예방**

네 가지 필요조건 중 어느 하나라도 만족되지 않으면 교착상태는 발생하지 않는다.

1. **상호배제 조건 제거**

공유할 수 있는 자원은 상호배제가 필요하지 않으므로 교착상태를 유발하지 않는다. 읽기 전용 파일 -> n개의 프로세스 동시 접근 가능

1. **점유대기 조건 제거**

자원을 점유했을 때 대기하지 않도록 하기

프로세스가 앞으로 필요한 모든 자원을 처음에 한꺼번에 요구하여 할당 받는다 단점

자원 이용률이 낮아질 수 있다 기아상태에 빠질 수 있다

대기할 때 자원을 점유하고 있지 않도록 하기

새로운 자원을 요구할 때는 먼저 할당받았던 자원을 모두 해제하면 된다. 단점

프린터처럼 점유 도중 해제할 수 없는 자원에는 적용할 수 없다. => 1번의 해결책을 사용해야하는데 같은 문제가 발생 한다.

1. **비선점 조건 제거**

선점이 가능하도록 한다. 단점

프린터 같은 자원은 어떤 프로세스가 사용중에 다른 프로세스가 선점하면 문제가 발생할 수 있다. => 선점 불가능한 자원

자원을 점유한 프로세스가 다른 자원을 요구할 때 대기가 발생한다면, 할당받았던 자원을 모두 해제하여 다른 프로세스가 비선 점 조건으로 대기할 가능성을 줄인다.

단점

프린터 같은 자원에 적용할 수 없음. (마찬가지)

1. **환형대기 조건 제거**
   1. **모든 자원에 일련번호 지정**
   2. **프로세스는 자원을 요구할 때 일련번호 기준으로 항상 오름차순이 되도록 요구**
   3. **프로세스가 자원을 요구할 때 그보다 일련번호가 큰 자원을 점유하고 있지 않도록 해제 특징**

자원의 일련번호가 가급적 프로세스가 자원을 요구하는 순서에 맞도록 설정되어야 프로세스 입장에서 큰 무리 없이 수행 가능 프로세스마다 자원의 요구순서가 다를 수 있으므로 자원의 일련번호 설정에 어려움이 존재

프로세스의 자원 요구순서가 설정된 일련번호의 오름차순을 지키지 못하면 일련번호가 큰 점유 중인 자원을 해제해야 하는 데, 적용이 불가능한 자원도 있을 수 있다.

요약

1. **교착상태는 여러 개의 프로세스가 서로 상대방의 작업이 끝나기만 기다리고 있기 때문에 결과적으로 어느 쪽도 영원히 진행하지 못하는 상태**
2. **교착상태의 네 가지 필요조건은 상호배제, 점유대기, 비선점, 환형대기로, 있다.**

이 조건들이 모두 만족될 경우 교착상태가 발생할 수

1. **자원할당 그래프는 프로세스와 자원 사이에 요구와 할당 관계를 나타내는 그래프로, 사이클의 존재 유무를 통해 교착상태의 발생 가능성을 확인할 수 있다.**
2. **교착상태를 처리하는 기법에는 교착상태 예방, 교착상태 회피, 교착상태 탐지 및 복구 가 잇다.**
3. **교착상태 예방은 교착상태의 네 가지 필요조건 중 어느 하나라도 발생할 수 없도록 막는 방법**