1 교착 상태 해결기법

1 교착 상태 해결기법

1 해결방법 3가지

- 예방
- 회피
- 탐지·회복

1 교착 상태 해결기법

2 하벤더의 교착 상태 예방 방법

- 각 프로세스는 필요한 자원 한 번에 모두 요청해야 하며, 요청한 자원을 모두 제공받기 전까지는 작업 진행 불가
- 어떤 자원을 점유하고 있는 프로세스의 요청을 더 이상 허용하지 않으면 점유한 자원을 모두 반납하고, 필요할 때 다시 자원 요청
- 모든 프로세스에 자원 순서대로 할당

1 교착 상태 해결기법

3 교착 상태 예방 방법

- 자원의 상호 배제 조건 방지
- 점유와 대기 조건 방지
- 비선점 조건 방지
- 순환(환형) 대기 조건 방지

1 교착 상태 해결기법

3 교착 상태 예방 방법

자원의 상호배제 조건 방지

- 상호배제는 **자원의 비공유**가 전제 되어야 함
- 일반적으로 상호배제 조건을 만족하지 않으면 교착 상태 예방 불가능

1 교착 상태 해결기법

3 교착 상태 예방 방법

점유와 대기 조건 방지

- 프로세스가 작업 수행 전에 필요한 자원 모두 요청하고, 획득해야 함
- 보류 상태에서는 프로세스가 자원 점유 불가능하므로 대기 조건 성립 안됨(최대 자원 할당)

3 교착 상태 예방 방법

점유와 대기 조건 방지

- 점유와 대기 조건 방지 방법
  - 자원 할당 시 시스템 호출된 프로세스 하나를 실행하는 데 필요한 모든 자원 먼저 할당하며, 실행 후 다른 시스템 호출에 자원 할당
  - 프로세스가 자원을 전혀 갖고 있지 않을 때만 자원 요청할 수 있도록 허용
  - 프로세스가 자원을 더 요청하려면 자신에게 할당된 자원을 모두 해제

3 교착 상태 예방 방법

점유와 대기 조건 방지

- 단점
  - 자원 효율성 너무 낮음
  - 기아 상태 발생 가능(대화식 시스템에서 사용 불가)

3 교착 상태 예방 방법

비선점 조건 방지

- 이미 할당된 자원에 선점권이 없어야 한다는 전제 조건 필요
- 어떤 자원을 가진 프로세스가 다른 자원 요청할 때 요청한 자원을 즉시 할당 받을 수 없어 대기해야 한다면 프로세스는 현재 가진 자원 모두 해제

1 교착 상태 해결기법

3 교착 상태 예방 방법

비선점 조건 방지

- 프로세스가 작업 시작할 때는 요청한 새로운 자원과 해제한 자원 확보
  - 비선점 조건을 효과적으로 무효화시키지만 이미 실행한 작업의 상태를 잃을 수도 있음
- 작업 상태를 쉽게 저장, 복구할 수 있을 때나 빈번하게 발생하지 않을 때만 좋은 방법
- 전용 입출력장치 등을 빼앗아 다른 프로세스에 할당 후 복구하는 과정 간단하지 않음

3 교착 상태 예방 방법

비선점 조건 방지

- 대안 방안
  - 프로세스가 어떤 자원을 요청할 때 요청한 자원이 사용 가능한지 검사
    - 사용할 수 있다면 자원 할당,
      사용할 수 없다면 대기

3 교착 상태 예방 방법

비선점 조건 방지

대안 방안

- 프로세스가 요청한 자원을 점유하고 있는지 검사

  요청한 자원을 대기 프로세스가 점유하고 있다면 자원을 해제하고 요청 프로세스에 할당, 요청한 자원을 사용할 수 없거나 실행 중인 프로세스가 점유하고 있다면 요청 프로세스는 대기

3 교착 상태 예방 방법

비선점 조건 방지

대안 방안

- 프로세스가 대기하는 동안 다른 프로세스가 점유한 자원을 요청하면 자원을 해제할 수 있음

3 교착 상태 예방 방법

비선점 조건 방지

대안 방안

- 두 프로세스에 우선순위 부여하고 높은 우선순위의 프로세스가 그보다 낮은 우선 순위의 프로세스가 점유한 자원 선점하여 해결
- 이 방법은 프로세서 레지스터나 기억장치 레지스터와 같이 쉽게 저장되고 이후에 다시 복원하기 쉬운 자원에 사용
- 프린터, 카드 판독기, 테이프 드라이버 같은 자원에는 적용 않음

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 모든 자원에 일련의 순서 부여, 각 프로세스가 오름차순으로만 자원을 요청할 수 있게 함
- 이는 계층적 요청 방법으로 순환 대기circular wait의 가능성 제거하여 교착 상태 예방
- 예상된 순서와 다르게 자원을 요청하는 작업은 실제로 자원을 사용하기 전부터 오랫동안 자원 할당 받은 상태로 있어야하므로 상당한 자원 낭비 초래

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 자원 집합을 R= {R1, R2, ……, Rn}이라고 할때, 각 자원에 고유 숫자 부여 어느 자원의 순서가 빠른지 알 수 있게 함
- 각 프로세스는 오름차순으로만 자원 요청 가능
  - 즉, 프로세스는 임의의 자원 Ri 요청할 수 있지만 그 다음부터는 F(Rj) > F(Ri) 일 때만 자원 Rj 요청 가능

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 데이터 형태 자원이 여러 개 필요하다면, 우선 요청할 형태 자원 하나 정해야 함

  ↳ 앞서 정의한 함수 이용하면, CD 드라이브와 프린터를 동시에 사용해야 하는 프로세스는 CD 드라이브 먼저 요청 후 프린터 요청

- 프로세스가 자원 Rj 요청 때마다 $F(Ri) \geq F(Rj)$가 되도록 Rj의 모든 자원 해제하는 것으로 이럴 경우 순환 대기 조건 막을 수 있음

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 프로세스가 자원 Rj 요청 때마다 F(Ri)≥F(Rj)가 되도록 Rj의 모든 자원 해제하는 것으로 이럴 경우 순환 대기 조건 막을 수 있음
- 순환 대기 성립한다는 가정하에서 사실의 성립 (모순에 의한 증명)

3 교착 상태 예방 방법

순환 대기 조건 방지

- 순환 대기의 프로세스 집합을 {P0, P1, ……, Pn}라 할 때,

  ↳ Pi는 프로세스 Pi +1이 점유 한 자원 Ri 대기
  (첨자는 모듈러 n. Pn은 P0이 점유한 자원 Rn을 대기하고, 모듈러 관계이므로 한 바퀴 뒤로 돌면 Pn은 Rn을 기다리며, P0이 Rn을 갖게 됨)

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 순환 대기의 프로세스 집합을 {P0, P1, ……, Pn}라 할 때,
  - 프로세스 Pi +1은 자원 Ri+1을 요청하는 동안 자원 Ri를 점유하므로 모든 i에 대하여 F(Ri) < F(Ri +1)이 성립 되도록 해야 함
  - 이는 F(R0) < F(R1) < …… < (Rn) < F(R0)이 성립함 의미

1 교착 상태 해결기법

3 교착 상태 예방 방법

순환 대기 조건 방지

- 순환 대기의 프로세스 집합을 {P0, P1, ……, Pn}라 할 때,
  - 즉, F(R0) ＜F(R0)은 불가능하므로 여기서는 순환 대기 불가능
  - 함수 F는 시스템에 있는 자원의 정상적인 이용 순서에 따라서 정의해야 함

2 교착 상태 회피

2 교착 상태 회피

1 회피 개념

목적 : 덜 엄격한 조건 요구하여
자원 좀 더 **효율적 사용**

- 교착 상태 발생할 가능성 인정하고,
  교착 상태가 발생하려고 할 때 적절히 회피하는 것
- 예방보다는 회피가 더 병행성 허용

2 교착 상태 회피

1 회피 개념

- 교착 상태의 회피 방법

프로세스의 시작 중단

- 프로세스의 요구가 교착 상태 발생시킬 수 있다면 프로세스 시작 중단

자원 할당 거부(알고리즘Banker's algorithm)

- 프로세스가 요청한 자원 할당했을 때 교착 상태가 발생할 수 있다면 요청한 자원을 할당하지 않음

2 프로세스의 시작 중단

- 교착 상태 회피를 위해 자원을 언제 요청하는지 **추가 정보 필요**
  - 각 프로세스마다 요청과 해제에서 정확한 순서 파악하고 있다면, 요청에 따른 프로세스 대기 여부 결정 가능
- 프로세스의 요청 수락 여부는 현재 사용 가능한 자원
- 프로세스에 할당된 자원 등 각 프로세스에 대한 자원의 요청과 해제를 미리 알고 있어야 결정 가능

2 교착 상태 회피

2 프로세스의 시작 중단

- 다양한 교착 상태 회피 알고리즘 중에서 가장 단순하고 유용한 알고리즘은 각 프로세스가 필요 한 자원의 최대치(할당 가능한 자원 수)를 선언하는 것

- 프로세스가 요청할 자원별로 최대치 정보를 미리 파악할 수 있으면 시스템이 교착 상태가 되지 않을 확실한 알고리즘 만들 수 있음

2 프로세스의 시작 중단

- 교착 상태 회피 알고리즘은 시스템이 순환 대기 조건이 발생 않도록 **자원 할당 상태** 검사
  - 사용 가능한 자원 수, 할당된 자원 수, 프로세스들 최대 요청 수로 정의
  - 각 프로세스에 (최대치까지) 자원을 할당할 수 있고, 교착 상태를 예방할 수 있으면 안정된 상태
  - 시스템에 안정 순서가 있으면 그 시스템은 안정, 순서가 없으면 불안정

2 교착 상태 회피

3 시스템의 상태

- 안정 상태와 불안정 상태로 구분
- 교착 상태는 불안정 상태에서 발생하나 쉬울 뿐이지 모든 상태가 교착 상태인 것은 아님
- 모든 사용자가 일정 기간 안에 작업을 끝낼 수 있도록 운영체제가 할 수 있으면 현재 시스템의 상태가 안정, 그렇지 않으면 불안정

2 교착 상태 회피

3 시스템의 상태

- 안정 상태의 경우, 불안정 상태와 교착 상태 예방 가능
- 불안정 상태의 경우, 교착 상태 발생시키는 프로세스의 자원 요청 방지 불가

2 교착 상태 회피

4 자원 할당 거부(은행가 알고리즘)

다익스트라의 은행가 알고리즘 이용

- 자원의 할당 허용 여부 결정 전에 미리 결정된 모든 자원의 최대 가능한 할당량 Maximum을 시뮬레이션하여 안전 여부 검사
- 그런 다음 대기 중인 다른 모든 활동의 교착 상태 가능성 조사하여 '안전 상태' 여부 검사·확인

3 교착 상태 회복

3 교착 상태 회복

1 교착 상태 회복에 필요한 알고리즘

- 시스템 상태 검사하는 교착 상태 **탐지** 알고리즘
- 교착 상태에서 **회복**시키는 알고리즘

3 교착 상태 회복

2 교착 상태 회복의 특징

- 교착 상태 파악 위해 교착 상태 탐지 알고리즘을 언제 수행해야 하는지 결정하기 어려움
- 교착 상태 탐지 알고리즘 자주 실행하면 시스템의 성능 떨어지지만, 교착 상태에 빠진 프로세스 빨리 발견하여 자원의 유휴 상태 방지 가능
  - ↳ 자주 실행하지 않으면 반대 상황 발생
- 탐지와 회복 방법은 필요한 정보를 유지하고 탐지 알고리즘을 실행시키는 비용뿐 아니라 교착 상태 회복에 필요한 부담까지 요청

3 교착 상태 회복

3 교착 상태 탐지 알고리즘

- 쇼사니(Shoshani)와 코프만(Coffman)이 제안
- 은행가 알고리즘에서 사용한 자료구조들과 비슷

Available	Allocation	Request
▪ 자원마다 사용 가능한 자원 수를 표시하는 길이m 인 벡터	▪ 각 프로세스에 현재 할당된 각 형태들의 자원 수 표시하는n ×m 행렬	▪ 각 프로세스의 현재 요청 표시하는 n×m 행렬 ▪ Request[i, j]일 때 프로세스 Pi에 필요한 자원 수가 k개라면, 프로세스 Pi는 자원 Ri의 자원 k개 더 요청

3 교착 상태 회복

4 교착 상태 회복 방법

프로세스 중단

교착 상태 프로세스 모두 중단

- 교착 상태의 순환 대기를 확실히 해결하지만 자원 사용과 시간면에서 비용 많이 듦
- 오래 연산했을 가능성이 있는 프로세스의 부분 결과를 폐기하여 다시 연산해야 함

3 교착 상태 회복

4 교착 상태 회복 방법

프로세스 중단

한 프로세스씩 중단

- 한 프로세스 중단할 때마다 교착 상태 탐지 알고리즘 호출하여 프로세스가 교착 상태에 있는지 확인
- 교착 상태 탐지 알고리즘을 호출하는 부담이 매우 큼

4 교착 상태 회복 방법

프로세스 중단

- 프로세스 중단이 쉽지 않을 수도 있음
- 프로세스가 파일 업데이트하다가 중단된다면 해당 파일은 부정확한 상태

3 교착 상태 회복

4 교착 상태 회복 방법

최소 비용으로 프로세스들을 중단하는 우선 순위 선정

- 프로세스가 수행된 시간과 앞으로 종료하는 데 필요한 시간
- 프로세스가 사용한 자원 형태와 수 (자원을 선점할 수 있는지 여부)
- 프로세스를 종료하는 데 필요한 자원 수
- 프로세스를 종료하는 데 필요한 프로세스 수
- 프로세스가 대화식인지, 일괄식인지 여부

4 교착 상태 회복 방법

자원 선점

선점 자원 선택

- 프로세스를 종료할 때 비용을 최소화하려면 적절한 선점 순서 결정
- 비용 요인에는 교착 상태 프로세스가 점유한 자원 수, 교착 상태 프로세스가 지금까지 실행하는데 소요한 시간 등 매개변수 포함

3 교착 상태 회복

4 교착 상태 회복 방법

자원 선점

복귀

- 필요한 자원을 잃은 프로세스는 정상적으로 실행 불가
- 일반적으로 안전 상태 결정이 곤란하므로 프로세스를 안전상태로 복귀 시키고 다시 시작해야 함
- 프로세스를 교착 상태에서 벗어날 정도로만 복귀시키는 것이 더 효과적이나 시스템이 실행하는 모든 프로세스의 상태 정보를 유지해야 하는 부담

4 교착 상태 회복 방법

자원 선점

기아

- 동일한 프로세스가 자원들을 항상 선점하지 않도록 보장하려고 할 때 비용이 기반인 시스템에서는 동일한 프로세스를 희생자로 선택하며, 이 프로세스는 작업 완료하지 못하는 기아 상태가 되어 시스템 조치 요청
- 프로세스가 짧은 시간 동안만 희생자로 지정되도록 보장해야 함
- 가장 일반적인 해결 방법은 비용 요소에 복귀 횟수를 포함시키는 것