인덕대학교

교착 및 기아 상태
- 예방 및 해결

컴퓨터소프트웨어학과

김병국 교수

학습목표

- 교착 상태에 대한 해결방법을 안다.
- 교착 상태에 대한 진입을 사전에 방지할 수 있다.
- 교착 상태에 대한 복구를 할 수 있다.

목차

1. 교착 해결 방법

❑종류

- 예방 기법(Prevention Method)
- 회피 기법(Avoidance Method)
- 탐지 및 복구(Detection and Recovery Method)

2. 예방 기법 (1/3)

❑4개의 deadlock 발생 필요 조건 중 하나를 제거

- 자원 사용의 상호 배제(Mutual Exclusion)
- 비선점형 자원(Non-preemptive)
- 점유와 대기(Hold-and-Wait)
- 원형 대기(Circular Wait)

❑위 사항 중 하나라도 없으면,

절대 Deadlock이 발생하지 않음!!

2. 예방 기법 (2/3)

❑ 자원의 공유를 허용

- 상호 배제(Mutual Exclusion) 조건을 제거
- 현실적으로 불가능
- 원하는 결과물이 나올 수 없음

❑ 선점을 허용

- 비선점(Non-preemptive) 조건을 제거
- 현실적으로 불가능
- 재진입 시 다시 처음부터 수행해야 하기 때문에, 심각한 자원을 낭비

2. 예방 기법 (3/3)

❑ 모두 점유

- 점유와 대기(Hold-and-Wait) 조건을 제거
- 자원 낭비 발생
  - 자원이 필요가 없음에도 할당
- 무한 대기 현상 발생이 가능

❑ 원형 대기 조건 제거

- 자원들이 할당될 프로세스들에게 순서를 부여
- 프로세스의 순서대로 자원 요청이 가능
- 자원 낭비 발생

3. 교착 회피 방법 (1/2)

❑ 방법

- 시스템의 상태를 계속 감시
- 자원의 요청이 있는 프로세스에 대하여 Dead Lock 상태가 될 가능성이 있는 경우 요청을 보류
- 시스템을 항상 Safe State로 유지
- 상태:
  - Safe State
    - 모든 프로세스가 **정상적으로 종료가 가능**한 상태
  - Unsafe State
    - Dead Lock 상태가 될 가능성이 있음
    - *꼭 발생한다는 의미는 아님*

3. 교착 회피 방법 (2/2)

❑가정

- 프로세스의 수가 고정됨
  - 중간에 새로운 프로세스가 생성되지 않음
- 자원의 종류와 수가 고정됨
- 프로세스가 요구하는 자원과 최대 수량을 알고 있음
- 프로세스는 자원을 사용 후 반납

※ 현실적이지 못함(Not Practical)

4. Dijkstra's Algorithm (1/3)

❑은행원 알고리즘

- Dead Lock 회피(avoidance)를 위한 간단한 이론적 기법
- 다익스트라(Dijkstra)가 제안한 알고리즘
- 프로세스가 **요구한 자원의 수**가 현재 **가용한(남은) 자원의 수보다 작을 때 할당**
- 가정
  - 한 종류(type)의 자원이 여러 개(unit)
- 시스템을 항상 safe state로 유지

4. Dijkstra's Algorithm (2/3)

❑은행원 알고리즘 예 1

- 자원 종류: 1
- 자원 개수: 100
- 프로세스 개수 : 3

프로세스	필요 자원	현재 자원	요구 자원
P1	30	10	20
P2	90	50	40
P3	50	20	30

- 할당 순서: **P1→P3→P2**
- 상태: **Safe State**

4. Dijkstra's Algorithm (3/3)

❑은행원 알고리즘 예 2

- 자원 종류: 1
- 자원 개수: 100
- 프로세스 개수 : 3

프로세스	필요 자원	현재 자원	요구 자원
P1	60	10	50
P2	90	50	40
P3	50	20	30

•할당 순서: 불가
•상태: **Unsafe State**

5. Habermann's Algorithm (1/3)

❑ Habermann's Algorithm

- Dijkstra's Algorithm의 확장
- 여러 종류의 자원을 고려
- 각 자원의 종류에 여러 개의 자원을 고려
- 시스템을 항상 safe state로 유지

5. Habermann's Algorithm(2/3)

❑ Habermann's Algorithm 예 1

- 자원의 종류: R1, R2, R3
- 자원별 개수: 100, 50, 70

프로세스	필요자원 R1	필요자원 R2	필요자원 R3
P1	10	10	10
P2	50	40	20
P3	60	50	50

프로세스	필요자원 R1, R2, R3	현재 자원 R1, R2, R3	요구 R1, R2, R3
P1	10, 10, 10	0, 10, 0	10, 0, 10
P2	50, 40, 20	40, 0, 10	10, 40, 10
P3	60, 50, 50	30, 20, 0	30, 30, 50

5. Habermann's Algorithm(3/3)

❑ Habermann's Algorithm 예 2

- 자원의 종류: R1, R2, R3
- 각 자원별 개수: 100, 50, 70

프로세스	필요자원 R1	필요자원 R2	필요자원 R3
P1	20	30	30
P2	50	40	60
P3	70	50	50

프로세스	필요자원 R1, R2, R3	현재 자원 R1, R2, R3	요구 R1, R2, R3
P1	20, 30, 30	10, 10, 0	10, 20, 30
P2	50, 40, 60	40, 0, 40	10, 40, 20
P3	70, 50, 50	30, 20, 0	40, 30, 50

6. 교착 회피 정리

❑ 정리

- Deadlock의 발생을 막을 수 있음

- High Overhead
  - 항상 시스템을 감시하고 있어야 함
  - 시스템 사용 효율이 떨어짐

- Low resource utilization
  - 사용되지 않는 여유 자원이 있어야 함

- 실현 불가능(Not Practical)
  - 프로세스 수, 자원의 수가 고정
  - 시스템은 필요한 최대 자원의 수를 알고 있어야 함

7. 탐지 및 복구

❑ 교착 상태에 대한 탐지

- 자원 할당 그래프(RAG: Resource Allocation Graph)를 통해 판별

❑ 교착 상태에 대한 복구

- 프로세스 종료(Process Termination)
  - Deadlock 상태에 있는 프로세스를 종료 시킴
  - 강제 종료된 프로세스는 나중에 다시시작하여 처리

- 자원 선점(Resource Preemption)
  - Deadlock 상태 해결을 위해 선점할 자원을 선택
  - 프로세스에게 할당된 자원을 빼앗음
    - 대상 프로세스는 오류발생으로 강제 종료되는 상황 발생

8. 복구 (1/2)

❑ 프로세스 종료

- Deadlock 상태인 프로세스 중 일부를 종료 시킴

- 종료할 프로세스 선택 방식
  - 우선순위(Process Priority)
  - 종류(Type)
  - 총 수행 시간(Total Execution Time)
  - 남은 수행 시간(Remaining Time)
  - 종료 비용(Accounting Cost)
  - 복구 비용(Recovery Cost)
  - 기타

8. 복구 (2/2)

❑자원을 빼앗음

- Deadlock 상태를 해결하기 위한 선점된 자원을 빼앗음
- 선점한 프로세스를 종료 시킴
  - 추후 해당 프로세스를 다시 시작
- 선점 방식
  - 선점 비용 모델 적용(Preemption Cost Model)
  - Minimum Cost Recovery Method 등이 채택

수고하셨습니다.