{ · 이론상으로 중요하고 처리해내야 함.
· 현실적으로 처리 비용이 너무 비싸서 "안 생기길" ... 기도 메타.

교착 상태.

· 자원 소유한 채 상대방의 자원 기다리며 무한대기

*교착상태의 필요충분조건 (w. 식사하는 철학자 문제).

· Circular Wait (순환 요청 · 순환 대기).

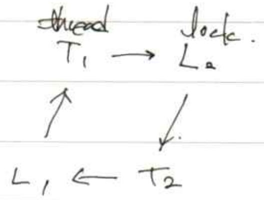
5명 모두 왼손에 포크를 가지고 오른쪽 포크를 요청하는 순환 고리.

⇒ 순환 상태로 요청 생기지 X도록 해야함.

* 철학자는 대화 못함: 각 스레드 IPC X. ← 가상메모리 기술.

But.

· Mutex로 구현 ...



· vs 기아.

기아: 잘못된 정책 사용 ⇒ 특정 작업 지연

교착: 여러 작업 진행 ⇒ 자연히 발생.)

교착상태의 잠재적 요인들.

· 자원: 교착상태의 발생지.

멀티스레드가 자원 동시 사용하려는 충돌 ... 요인.

· 자원-스레드.

· 자원-운영체제.

· 자원 비선점. 할당된 자원 자발적 반납 전에 강제로 뺏지 못함.

*자주 발생하면서도 덜 발생함. · 같은 파일 동시 참조할 확률. ↓

NO.

#교착상태의 해결 · 직접 해결은 비용 ↑ : 대부분 고려 ×. ⟷ Mission Critical한 작업에만 고려.

· 의심되는 프로세스 종료하여 해소 모색.

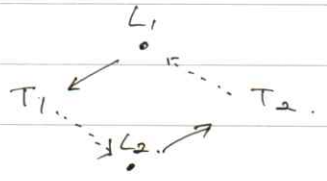
#교착상태의 모델링. Resource Allocation Graph.

* (P1) ← [R1] 할당. (P1) ⋯⋯> [R1] 대기.

· T1 ⟵ out ⟵⋯⋯ T2.

· T1 ⟵ out ⟶ T2.

· T1 ⟵ out.



#교착상태의 필요충분조건. Coffman Condition.

필요충분조건. 4개 중 하나라도 성립 ×. ⇒ 교착상태 ×.

자원적 특성:
· Mutual Exclusion. 상호 배제.
· No Preemption 비선점.
선점 A 데이터 캐릴추트.

행위적 특성:
· Hold & Wait 점유와 대기.
* · Circular Wait 환형 대기.

#교착상태의 해결.

· 교착상태 예방.
→발생 구조 A 자원 요청 후시.

· 교착상태 회피. "발생 여지 ×"
→발생 가능성 검사 … 발생 위험 요청 시 할당.

자원적 특성은 제한하기 어려움.
· 상호배제 조건.
→상호배제를 제거할 수 있을까? ⇒ 어려움.
· 비선점 조건.
→선점 허용? ⇒ 어려움
· 점유와 대기 조건.
→기다리게 하기.

쓰레드에게 자원 할당 전, 필요한 모든 자원 파악 후 한 번에 할당.

· 새로운 자원 요청은 할당받은 모든 자원 반환 후.

자원할당 상태 A ⟵ 이미 소유하고 있는 자원도 반납.

NO.

교착상태의 해결.

- 점유와 대기 조건.

문제점: · 자신이 사용하는 모든 자원, 계속 보기 어려움.
· 당장 사용 x 자원 묶음: 활용률 ↓.
· 자원 ↑ 필요한 작업: 기아 상태 가능성.
… 다중 프로세싱 자체가 깨집니다.

- 환형대기 조건.

→ 환형대기 제거 · 모든 자원에 숫자 부여. 숫자가 큰 방향으로 할당.
i.e. 마우스(1) HDD(2) 프린터(3).
1 → 2 → 3 V. / 1 → 3 ↛ 2 X.

문제점: · 작업 진행 유연성 ↓.
· 숫자 어떻게 붙이지?

· 점유와 대기, 환형대기: 프로세스 작업 방식 제한·남비.

*교착상태 회피. · 자원 할당 시 … 환형대기 발생할지 결정 ⇒ 할당 X.

· Safe state / Unsafe state. 판단? banker's 알고리즘. (대표 회피기법).

total = 14 available = 2.

P	Max	Allocation	Expect.
av = 8. ② ← P_1	5	2	3
av = 6. ① ← P_2	6	4	2
③ ← P_3.	10	6	4.

자원 할당 시 미래에 작업 종료 가능.

문제점: · 실행 전, 사용할 자원 개수 어떻게 예측?
· 프로세스 개수 동적 변화. → 미리 개수 정적 고정 불가 (버퍼에서 만들어야 함).

교착상태의 해결.

· 예방 전략 · 회피전략 모두 비현실적. 비일반적.

⇒ 특수 목적 환경에서 사용.

* 교착상태 감지 및 복구.

· 교착상태 감지하는 프로그램 운영.

↳ 자원 할당 그래프 이용. (그래프 중 사이클 존재 시).

↳ Time Out 이용. i.e. (응답없음) 창.

↳ 은 좀있으면 풀리기도..

· 감지 시 복구 방법.

1. 자원 강제 선점. (w. 희생대기).
2. 롤백.
3. 프로세스 강제 종료.. (w. 희생대기).

① 연대책임: 모두 종료.

② 하나를 골라 종료.

- 우선순위 순.
- 진행된 작업시간 짧은 순 (재작업 덜 해도 되니까-)
- 자원 많이 사용하는 순.

* Ostrich Algorithm.

· 교착을 해결할 필요가 있을까? ⇒ 없다면. ... 대부분 OS에서 채택.

· 데이터 손실 가능성 : 하지만 확률적으로는 크지 않은 손실.

+) 주기적으로 데이터 스냅샷 생성 (백업).