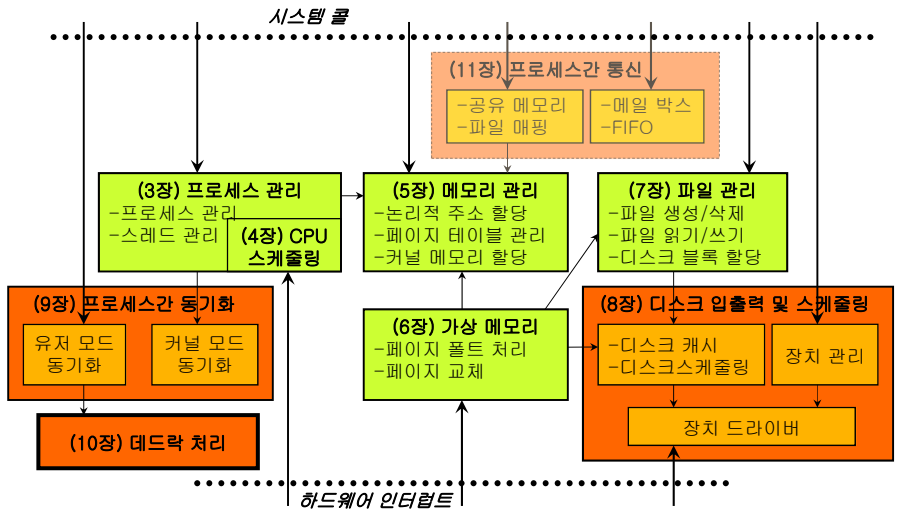
데드락 문제 및 그 해법

- ✓식사중인 철학자 문제
- ✓사거리의 데드락 문제
- ✓데드락 문제의 해결 방안들
- ✓데드락 문제의 모델링
- ✓데드락 방지 기법
- ✓데드락 회피 기법
- ✓데드락 탐지 및 복구 기법
- ✓picoKernel의 데드락 탐지 및 복구 구현

관련 운영체제 구성 모듈

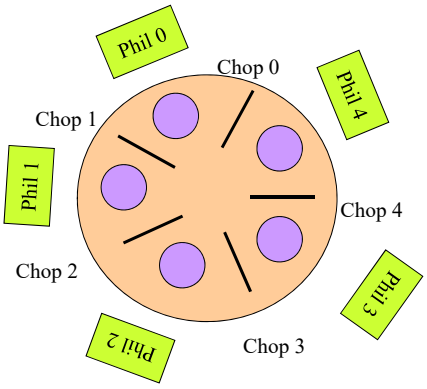


이해하고 넘어가야 할 내용들

- ✓ 공유 자원의 사용과 뮤텍스나 세마포와의 연관성에 대한 이해
- ✓ 프로세스들 간에 발생할 수 있는 데드락 상태에 대한 이해
- ✓ 데드락 방지 기법과 데드락 회피 기법의 차이점 및 장단점의 이해
- ✓ 데드락 검사 및 복구 기법의 장점 및 적용에 있어서의 제약성에 대한 이해
- ✓ 데드락 상태를 검사하는 방법 및 프로그램으로 구현하는 방법에 대한 이해

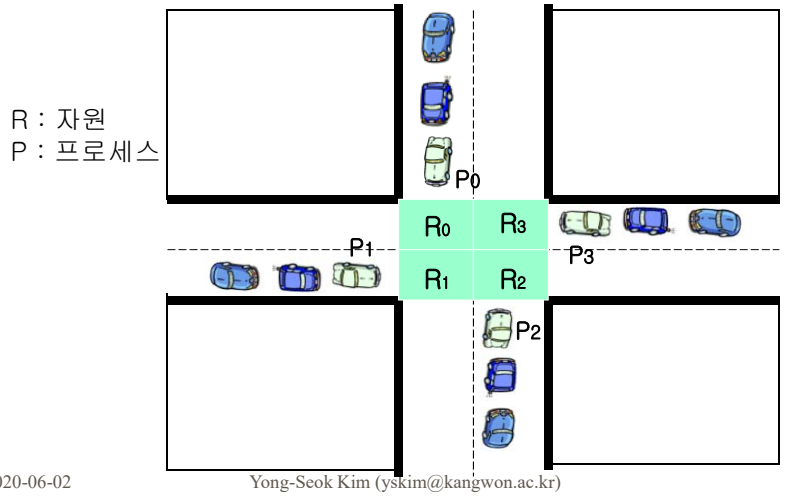
식사중인 철학자 문제

- ✓ **데드락 (deadlock)**
  - ✓ 하나의 프로세스 집합에 대해
  - ✓ 모든 프로세스가 대기상태이고
  - ✓ 이들이 기다리는 이벤트는 집합속의 다른 프로세스들에 의해서 발생
- ✓ **식사중인 철학자 문제**
  - ✓ 철학자 (프로세스)들 간에 젓가락 (자원) 공유
  - ✓ 뮤텍스를 이용한 프로그램 예 (프로그램 10.1)
- ✓ **데드락 시나리오**
  - ✓ 모든 철학자가 동시에 왼쪽 젓가락을 쥔 후 오른쪽 젓가락을 확보하기 위해 대기하고 있을 때
- ✓ **해결방안 들**



사거리 데드락의 모델링

- ✓ 차량: 프로세스
- ✓ 사거리의 각 사분면: 공유 자원



사거리 통과 알고리즘

- ✓ 오른쪽 방향으로 진행하는 차량 (P1)

P_1의 우회전	P_1의 직진	P_1의 좌회전

mutexLock(R_1);
사거리 진입;
우회전 및 통과;
mutexUnlock(R_1);
	mutexLock(R_1);
	사거리 진입;
	mutexLock(R_2);
	전진;
	mutexUnlock(R_1);
	사거리 통과;
	mutexUnlock(R_2);
		mutexLock(R_1);
		사거리 진입;
		mutexLock(R_2);
		전진;
		mutexUnlock(R_1);
		mutexLock(R_3);
		좌회전 및 전진;
		mutexUnlock(R_2);
		사거리 통과;
		mutexUnlock(R_3);

사거리의 데드락 해결 방안들

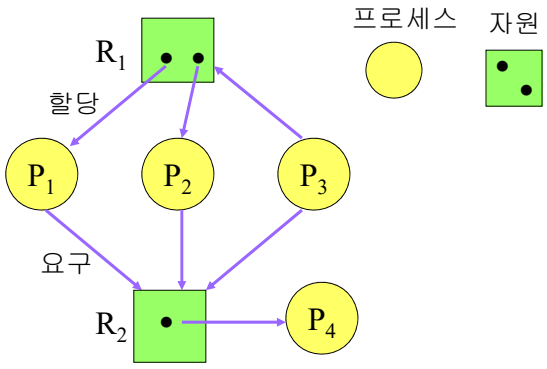
1. **사거리를 완전한 입체교차로로**
   - ✓ 너무 많은 비용 소요 (교통량이 적은 사거리에는 필요 없음)
2. **모든 운전자가 준수하는 사거리 통과 규칙 제정**
   - ✓ 예: 교통신호등 시스템
   - ➔ 데드락 방지 기법의 일종
3. **모든 운전자는 데드락이 발생할 가능성이 없을 때만 진입**
   - ➔ 데드락 회피 기법의 일종
4. **전방이 비어있으면 일단 진입 ➔ 데드락 상황이 발생하면 별도의 해결 조치**
   - ➔ 데드락 탐지 및 복구 기법의 일종
5. **아무 대책없이 방치**
   - ➔ 데드락 발생 확률이 아주 낮고, 데드락이 발생하더라도 심각한 사태를 유발하지는 않는 경우
   - ✓ 일반 운영체제에서는 데드락에 대한 별도 대책이 없음
   - ✓ Ostrich Algorithm

데드락 문제의 모델링

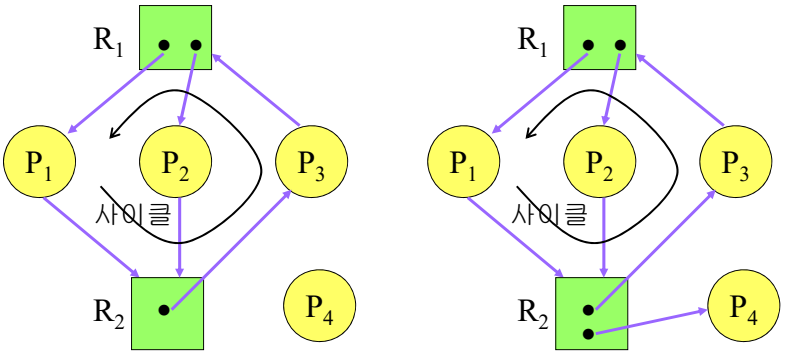
- ✓ **데드락 정의 (프로세스 집합 S)**
  - ✓ 집합 S의 모든 프로세스들이 waiting 상태이고
  - ✓ 이들이 기다리는 event는 S의 다른 프로세스들에 의해 발생해야 하는 상태
  - ✓ (참고) 시스템 내의 모든 프로세스는 아님
- ✓ **시스템 모델**
  - ✓ 프로세스: P_1, P_2, …, P_n
  - ✓ 자원: R_1, R_2, …, R_m
  - ✓ 자원 R_i는 N_i개가 존재
  - ✓ 프로세스는 각 자원에 대하여 확보, 사용, 해제의 절차 준수

자원할당 그래프

✓ 자원할당 그래프 (resource allocation graph)



사이클의 존재와 데드락



(a) Deadlock 발생 (b) Deadlock 아님

데드락 방지 기법

- ✓ **데드락 방지 (deadlock prevention)**
  - ✓ 대드락이 발생할 조건을 사전에 제거
- ✓ **데드락 발생 조건**
  - ✓ 상호 배제 (mutual exclusion)
  - ✓ 선점 불가 (no preemption)
  - ✓ 보유 및 대기 (hold and wait)
  - ✓ 원형 대기 (circular wait)
- ✓ **데드락 방지 방안들**
  - ✓ 상호배제 및 선점불가 조건은 자원의 특성에 의해 결정

  ➔ 보유 및 대기 조건이나 원형대기 조건을 피하는 알고리즘

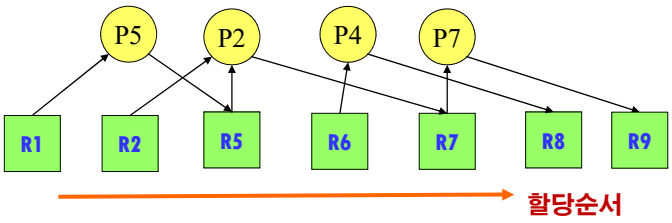
  1. 기확보 자원의 반납 ← 보유 및 대기 조건
  2. 동시에 사용할 자원들은 일괄 확보 ←보유 및 대기 조건
     - ✓ 여러 뮤텍스를 한꺼번에 확보하는 기능이 필요함
     - ✓ 사거리 통과 문제에 적용한 예 (프로그램 10.2)
  3. 자원들의 사용순서를 (사이클이 없도록) 고정 ←원형 대기 조건
     - ✓ 사거리 통과 문제에 적용한 예 (프로그램 10.3)
     - ✓ 데드락이 발생하지 않는 근거는?

POSIX 세마포를 이용한 일괄 확보

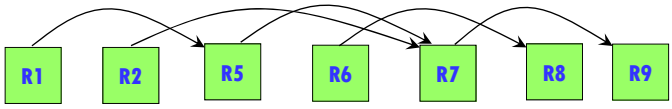
```
mutexLock2(semid, id1, id2)
{  sembuf sop[2];

   sop[0].sem_num = id1;
   sop[0].sem_op = -1;
   sop[0].sem_flg = 0;
   sop[1].sem_num = id2;
   sop[1].sem_op = -1;
   sop[1].sem_flg = 0;

   semop(semid, &sop, 2);
}
```

자원의 사용순서를 고정하면

자원할당 그래프



자원간의 그래프로 변경하면



화살표는 오른쪽 방향으로만 존재하므로 사이클이 생길 수 없음

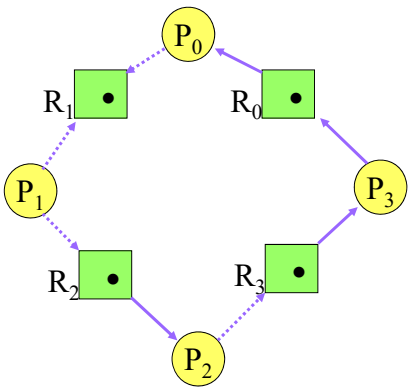
데드락 회피 기법

- ✓ **데드락 회피 (deadlock avoidance)**
  - ✓ 자원 할당을 요청한 시점에 할당 여부 결정
  - ✓ 이미 할당된 자원이면 요청한 프로세스는 대기
  - ✓ 사용 가능한 자원이라도 할당해 주면 이후에 데드락 발생 가능성이 있으면 요청한 프로세스는 대기
  - ➔ 프로세스들이 미래에 사용할 자원들에 대한 정보 필요
  - ✓ 그렇지 않을 경우에만 요청대로 할당
- ✓ **데드락 방지 기법과의 비교**
  - ✓ 데드락 방지 기법: 당장 사용하지 않을 자원도 미리 확보해 두는 경우 존재 ➔ 자원의 활용률 저하
  - ✓ 데드락 회피 기법: 자원 할당을 요청받을 때마다 데드락 가능성을 검사하는 오버헤드
  - ➔ 자원의 활용율 향상과 검사 오버헤드의 타협점

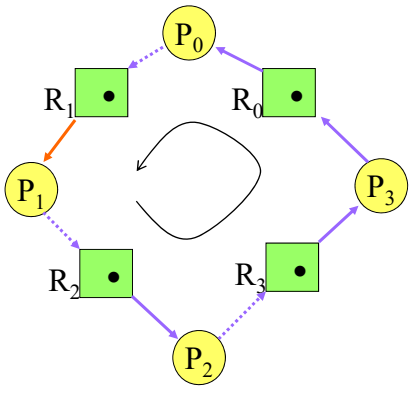
데드락 발생 가능성 검사

- ✓ **Dijkstra의 알고리즘 (1965)**
  - ✓ 미래의 가능성 검사를 위해서는 프로세스 별로 사용할 자원들의 정보를 미리 운영체제에 통보
  - ✓ 미리 통보된 자원들에 대해 클레임 에지 (claim edge) 추가
  - ✓ 자원 할당을 요청한 시점에 클레임 에지를 요구 에지 (request edge)로 변경
  - ✓ 자원을 할당하면 요구 에지를 할당 에지 (assignment edge)로 변경
- ✓ **데드락 발생 가능성 판단**
  - ✓ 자원당 1개씩인 경우: 사이클의 존재 여부로 판단
  - ✓ 자원당 여러 개 일 경우: 은행원 알고리즘 (banker's algorithm)

P1이 R1을 요청하면?

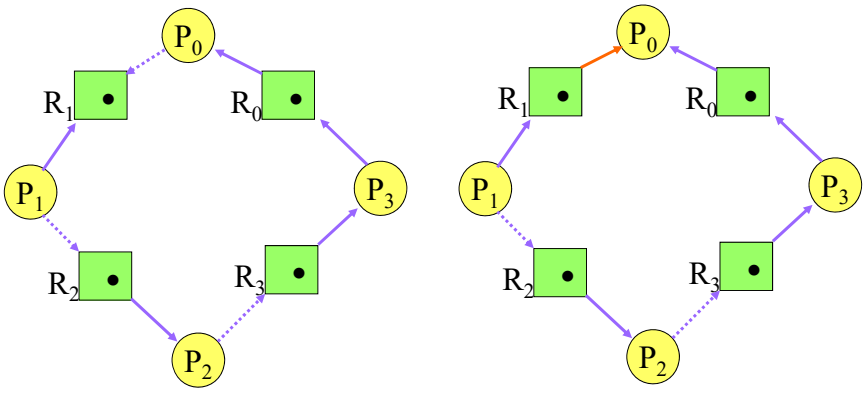


현상태



➔ 할당하지 않음

P0 이 R1을 요청하면?



데드락 회피 기법의 적용

- ✓ **운영체제에 사용할 자원들의 통보 및 해제**
  - ✓ mutexClaim: 동시에 사용할 자원들을 운영체제에 통보
  - ✓ mutexDeclaim: 사용 통보 내용을 해제
- ✓ **사거리의 데드락 문제에 적용한 예**
  - ✓ 프로그램 10.4

데드락 탐지 및 복구 기법

- ✓ **데드락 탐지 및 복구 (deadlock detection and recovery)**
  - ✓ 자원의 할당 요청시 자원이 이미 할당되지 않았으면 무조건 할당
  - ✓ 데드락이 실제로 발생했는지는 적절한 시점마다 검사
  - ✓ 데드락이 발생했으면 적절히 복구
- ✓ **장단점**
  - ✓ 자원의 활용율 극대화 ← 데드락 회피 기법은 데드락 가능성이 있으면 자원할당 보류
  - ✓ 데드락 발생 가능성 검사 오버헤드 감소 ← 데드락 회피 기법은 자원할당 요청시 마다 검사하는 데 비해 탐지 및 복구 기법은 검사하는 빈도를 적절히 조절 가능
  - ✓ 적용에 제한이 있음 ← 데드락 복구 과정이 복잡하고 실제 적용할 수 없는 경우도 존재

데드락 검사 시점 및 복구

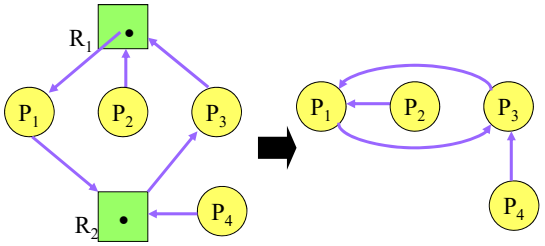
- ✓ **검사 시점**
  1. 요청 자원이 이미 사용 중일 때마다 ➔ 검사 오버헤드가 큼
  2. 주기적으로 한번씩 ➔ 발생즉시 검사되지 않으므로 데드락에 연루된 프로세스/자원들이 시간을 허비, 검사주기가 길 수록 연루된 프로세스 수 증가
  3. CPU 활용률이 낮은 시점 ➔ 2번과 유사
- ✓ **검사 주체**
  - ✓ 커널이 직접
  - ✓ 커널 모드에서 실행되는 전담 프로세스
- ✓ **데드락의 복구**
  1. 연루된 프로세스 중 일부(또는 전부)를 강제로 종료하고 사용중이던 자원들을 회수
  2. 프로세스별로 체크포인트 (check point) 시점에 모든 정보를 저장했다가 데드락 판정시 이전 체크포인트로 롤백 (rollback)

데드락 처리 방안들의 비교

항목	데드락 방지 (일괄확보)	데드락 방지 (사용순서)	데드락 회피	데드락 탐지 및 복구	무대책
자원의 활용률	낮음	약간개선	중간	높음	높음
실행 오버헤드	낮음	낮음	높음	중간	없음
데드락 복구 주체	불필요	불필요	불필요	전담 프로세스 또는 운영체제	시스템 관리자
데드락 유지 기간	없음	없음	없음	중간	장기간

데드락의 검사

- ✓ **사이클의 검사**
  - ✓ 자원당 1개씩일 때: 사이클이 존재하는지로 검사
  - ✓ 자원할당 그래프를 직접 검사하기보다 Wait-for 그래프로 단순화 시켜서 검사 ← 사이클 검사: $O(n^2)$
  - ✓ 그래프의 사이클 검사 알고리즘 적용 (프로그램 10.5 참조)
- ✓ **Wait-for 그래프**



자원할당 그래프 Wait-For 그래프

picoKernel의 데드락 탐지 및 복구

- ✓ 프로그램 10.5
- ✓ 모든 스레드들이 대기 상태일 때 null 스레드가 실행
- ✓ 데드락 복구는 시스템 재부팅으로 처리
- ✓ 자원할당 그래프는 소유 스레드가 기록되는 뮤텍스들에 한해서 적용
- ✓ 자원할당 그래프를 Wait-for 그래프로 변환하여 사이클 검사
- ✓ 사이클 검사 알고리즘 실행 예