운영체제

2019.7.2

컴퓨터공학과 이병문
가천대학교 Gachon University

강의일정

- 7/2 Deadlock
- 3 CPU스케줄링1(온라인수행과제)
- 4 CPU스케줄링2
- 5 메모리관리1
- 7/8 중간고사2

교착상태(Deadlock) 개요
- Deadlock 정의
- Deadlock 발생
- 자원할당그래프

교착상태(Deadlock) 처리
- Deadlock Prevention(예방)
- Deadlock Avoidance(회피)
- Deadlock Detection(탐지)
- Deadlock Recovery(복구)

교착상태(Deadlock) 개요

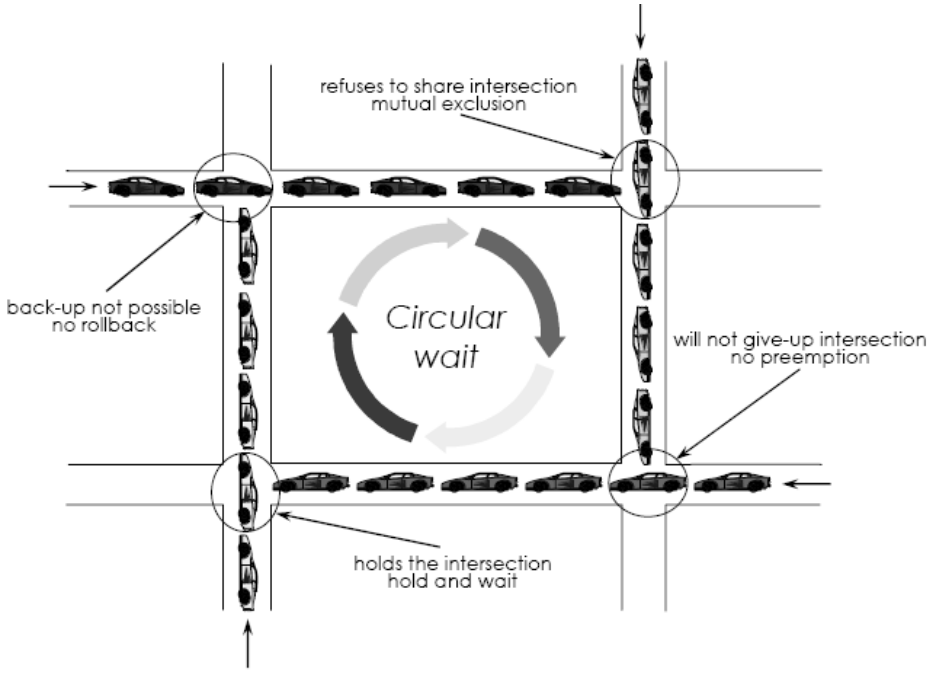
■ Deadlock(교착상태)의 정의

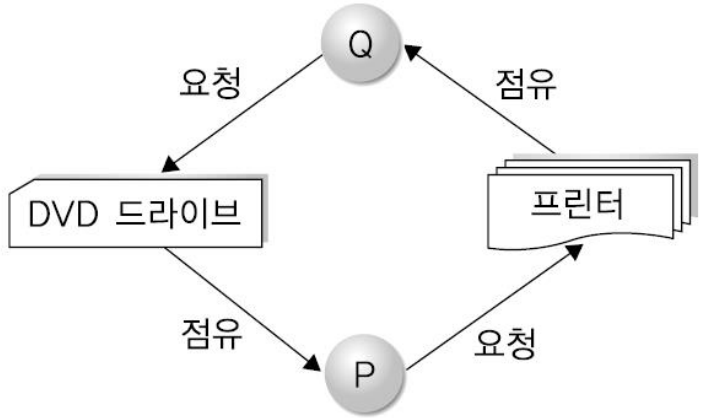
☑ **두 개 이상의 프로세스가 필요한 자원을 기다리면서 무한정 중지된 상태**

☑ 발생원인 : 부작용 (제한된 자원의 이용률▲, 시스템 효율성 ▲).

☑ 해결방안 : 프로세스 종료/교체, 외부에서 강제해제, CPU **사용률** -> 0

예)





☑ **프로세스의 자원이용 순서**

① **요청 ; 자원요청 -> 자원할당**

② **사용 ; 할당된 자원의 사용**

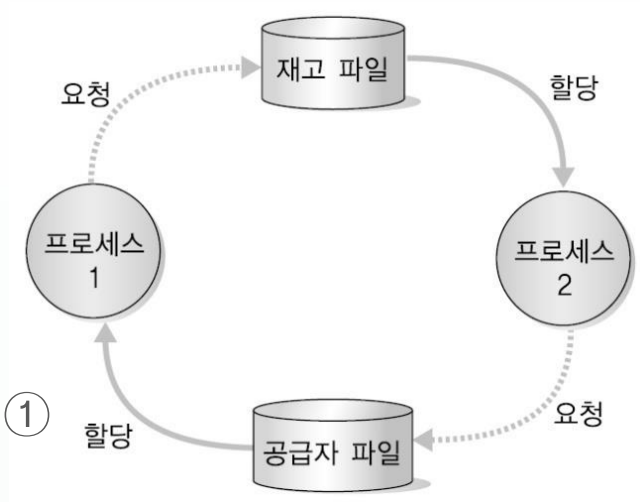
③ **해제 ; 할당자원을 반납 후 종료**

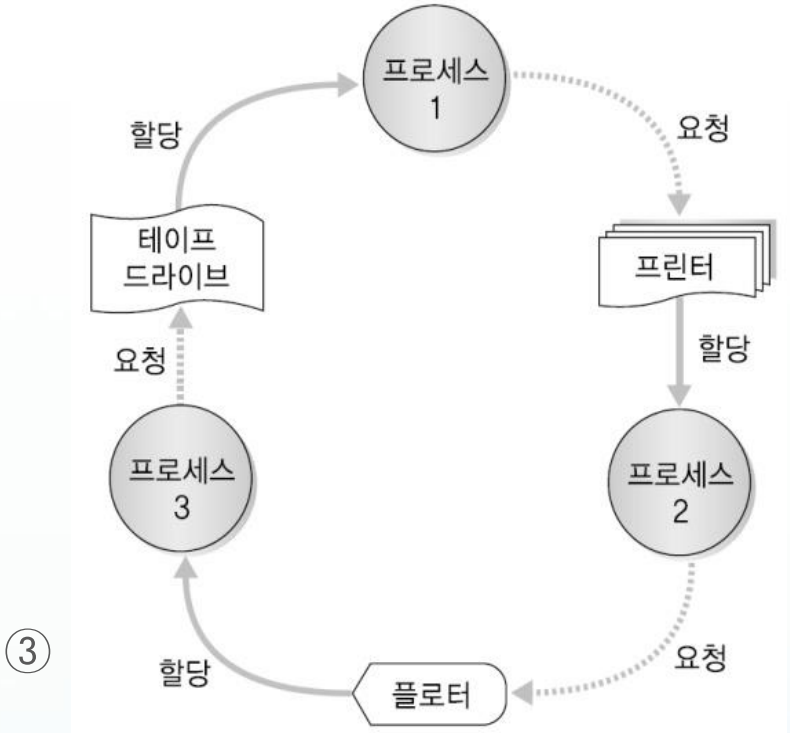
교착상태(Deadlock) 개요

■ Deadlock 의 발생 사례

① **파일을 요청할 때 발생**,
② **전용장치를 할당할 때 발생(예,CD복사)**
③ **다중 주변장치를 할당할 때 발생**,
④ **스풀링 시스템에서 발생**
⑤ **디스크를 공유할 때 발생**,
⑥ **네트워크에서 발생**

예)



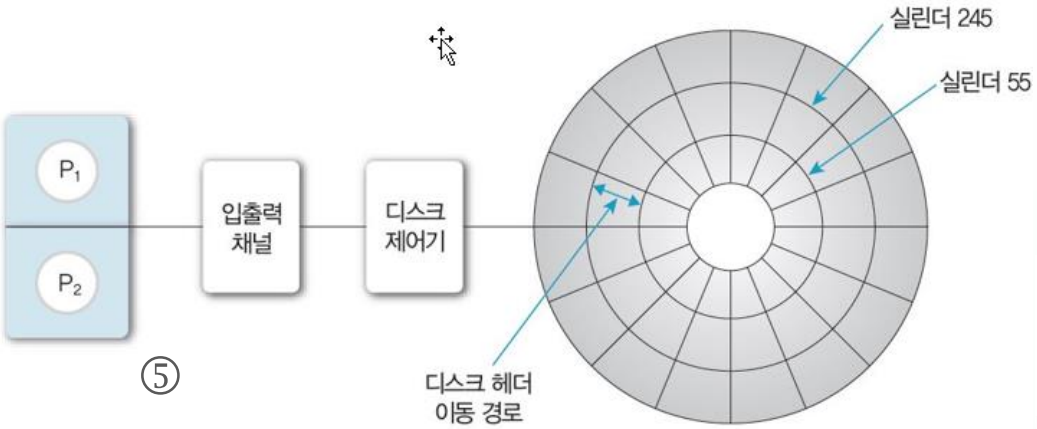


교착상태(Deadlock) 개요

■ Deadlock 의 발생 사례

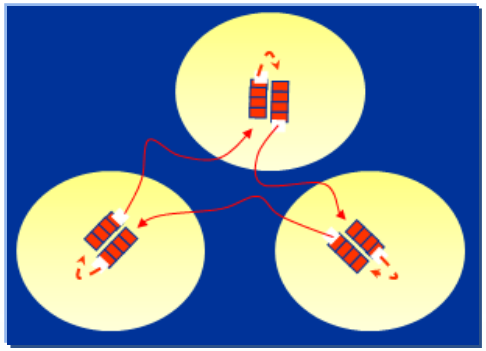
① 파일을 요청할 때 발생,
② 전용장치를 할당할 때 발생(예,CD복사)
③ 다중 주변장치를 할당할 때 발생,
④ 스풀링 시스템에서 발생
⑤ 디스크를 공유할 때 발생,
⑥네트워크에서 발생

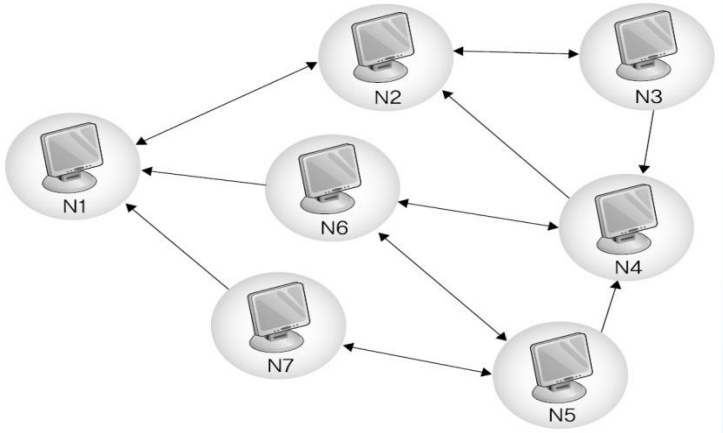
예)



⑤

⑥





Deadlock 발생 필요충분조건 !
상호배제 + 점유와 대기 + 비선점 + 순환대기

교착상태(Deadlock) 개요

■ 자원할당 그래프(Resource Allocation Graph)

☑ Deadlock ; **자원할당 그래프에서 사이클이 생기면** Deadlock **임**.

☑ **자원할당 그래프**, G = (V, E) **정점 집합**(V)

☑ **프로세스 집합**, P = {P1, P2, ... Pn}

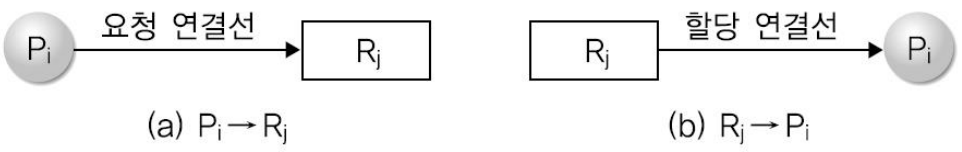
☑ **자원들로 구성된 간선 집합**(E) , R = {R1, R2, ... Rn}

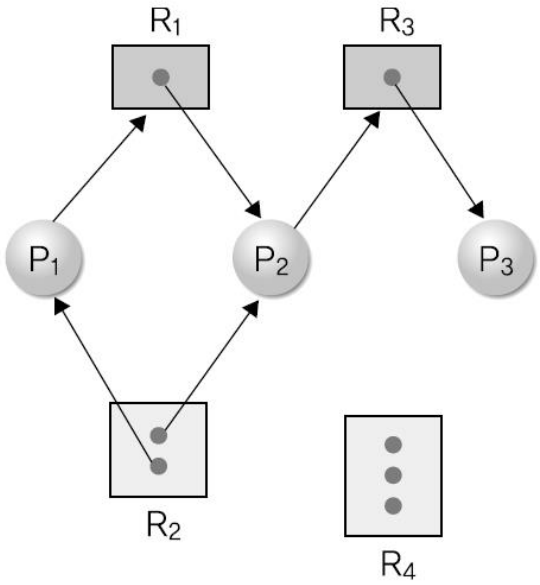
☑ **집합** P, R, E :

P = {P1, P2, P3} , R = {R1, R2, R3, R4},

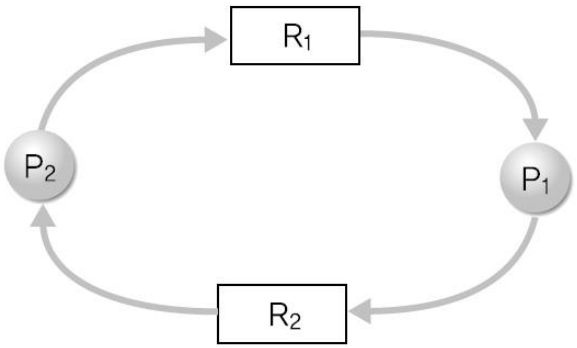
E = {P1→R1, P2→R3, R1→P2, R2→P2 ,

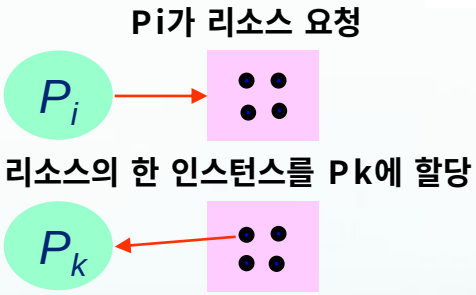
R2→P1, R3→P3, P3 → R2}





Cycle -> Deadlock





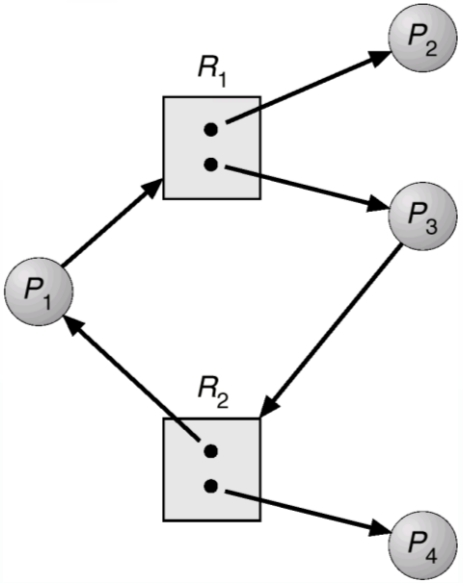
교착상태(Deadlock) 개요

■ 자원할당 그래프(Resource Allocation Graph)

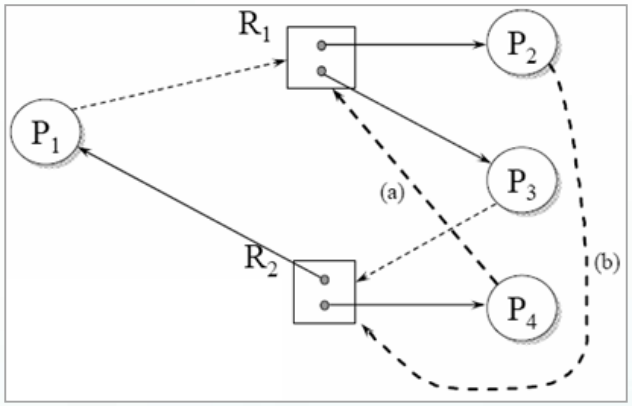
☑ Deadlock 이 없는 경우

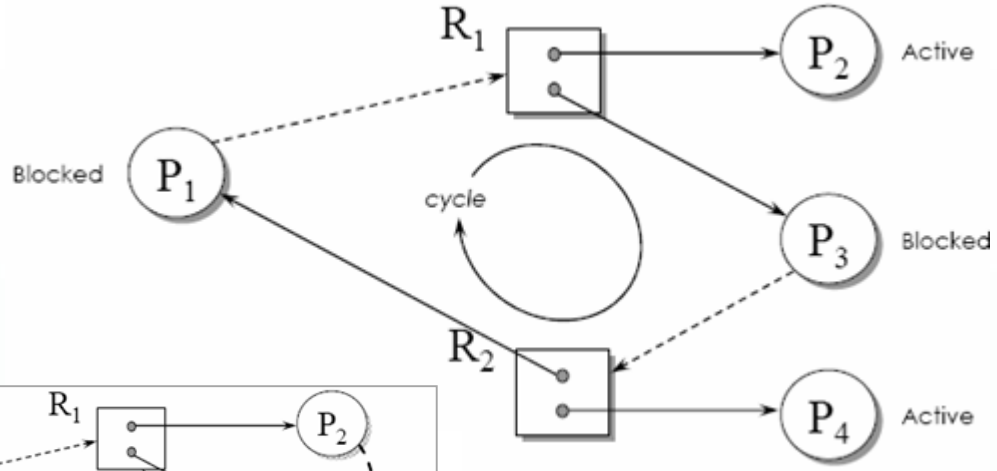
$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

프로세스 P_4가 자원 R_2를 해제할 수 있기 때문
이 자원이 P_3에 할당되면, 주기가 없어짐



? ▶





▲ ?

교착상태(Deadlock) 처리 – 발생조건

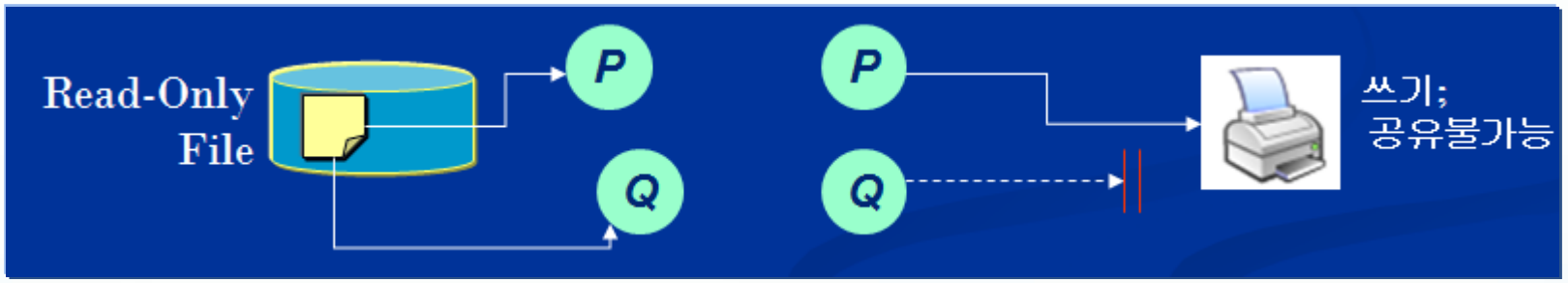
■ Deadlock **처리기법**

☑ Deadlock Prevention(**예방**)
☑ Deadlock Avoidance(**회피**)
☑ Deadlock Detection(**탐지**)
☑ Deadlock Recovery(**복구**)

} **Most OS (Windows, Unix, Linux,…)**

■ Deadlock Prevention (4**가지 필요충분 조건이 발생하지 않도록**)

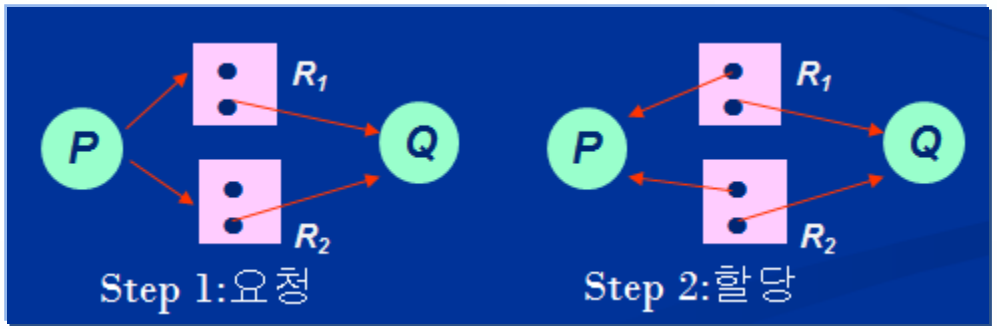
☑ **상호배제** : 자원에 따라 다르므로 상호배제는 **유지**

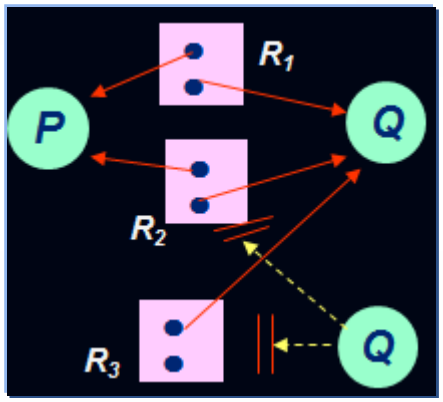


교착상태(Deadlock) 처리 – 발생조건

■ Deadlock Prevention (4**가지 필요충분 조건이 발생하지 않도록**)

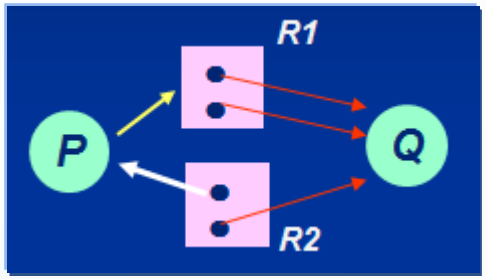
☑ **점유와 대기조건**(Hold and Wait) **방지** !!





필요한 자원을 한꺼번에 요청하고 동시에 허용될때까지 프로세스 보류

☑ **비선점**(No Preemption) **방지** !!



P = {P, Q} , R = {R1, R2},

T1 ; { R1→Q, R1->Q, R2->Q, R2->P }
T2 ; { P->R1 }
강제 회수!! → T3 ; { R1→Q, R1->Q, R2->Q, ~~R2->P~~ }
T4 ; { R1→Q, R1->Q, R2->Q }
...
T8 ; { R1→Q, R2->Q } ; Q가 R1 해제 !!
T9 ; {R1→Q, R2->Q, P->R1 , P->R2}
T10 ; {R1→Q, R2->Q, R1->P , R2->P }

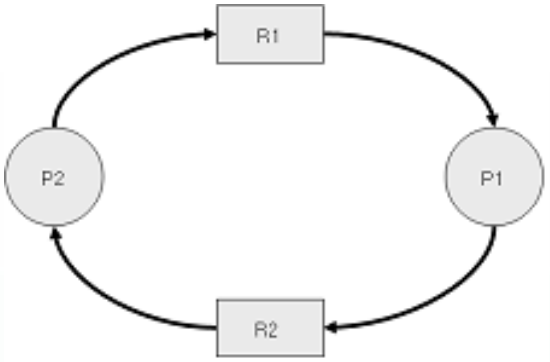
교착상태(Deadlock) 처리 – 발생조건

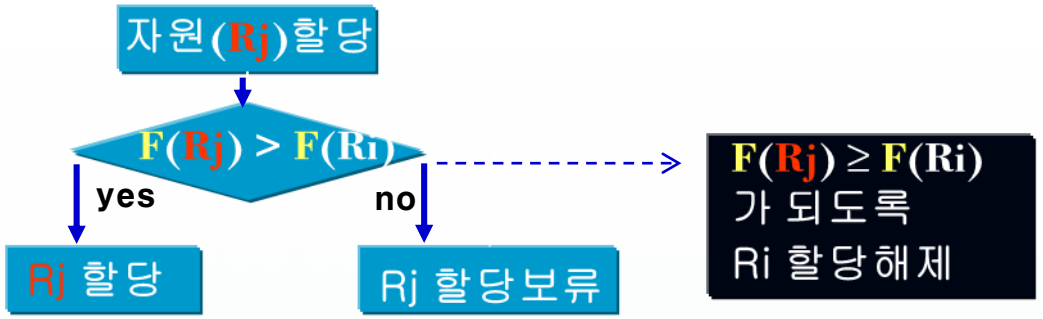
■ Deadlock Prevention (4**가지 필요충분 조건이 발생하지 않도록**)

☑ **순환대기**(Circular Wait) **방지** !!

- **모든 자원에 순서**(F)**를 부여** -> **순서**(F)**순으로만 자원을 할당함**(**예**, 1->5->12->…)

F(Tape Drive) = 1, F(Disk Drive) = 5, F(Printer) = 12, …





☑ 4**가지 조건중 3가지를 방지하면 교착상태**(Deadlock)**이 발생되지 않음**
-> **교착상태 예방됨**!

교착상태(Deadlock) 처리 – 교착상태 회피

■ Deadlock Avoidance (**회피**)

☑ Deadlock **의 예방은 상호배제, 점유대기, 비선점, 순환대기의 조건을 제거하여 문제를 해결할 수 있더라도 장치의 비효율적 사용, 시스템 처리성능 저하의 문제발생**

→ Deadlock **인정하고 회피대책**

☑ **프로세스 시작거부** ;

시작거부 ; 최대요구량(현재프로세스) + 최대요구량(새로시작할 프로세스) > 최대값

☑ **자원할당의 거부**

자원을 할당시 Deadlock **가능성이 높으면 할당거부** ; banker's algorithm

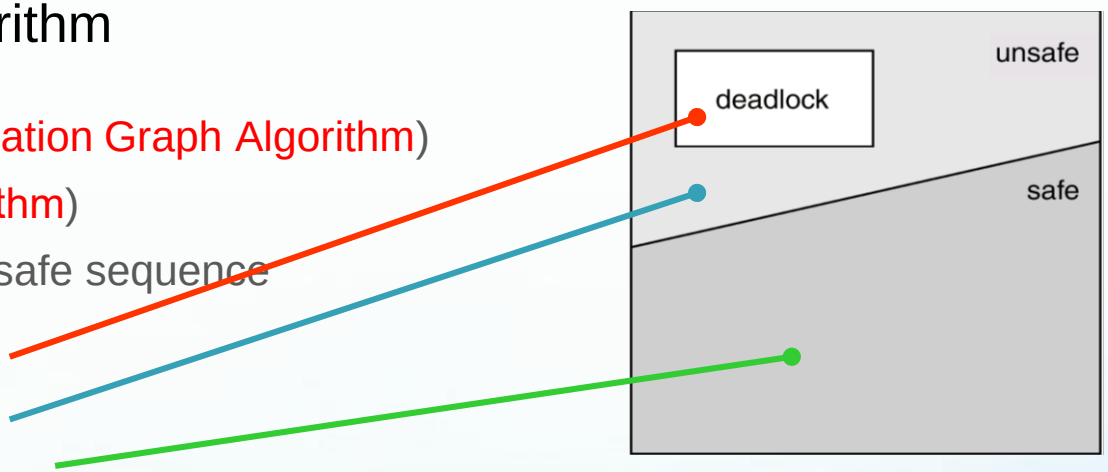
■ Deadlock Avoidance Algorithm

☑ **자원할당그래프**(Resource Allocation Graph Algorithm)

☑ **은행가알고리즘**(Banker's Algorithm)

☑ Safe state only if there exists a safe sequence

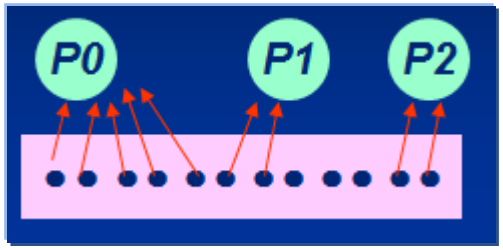
- Deadlock state is an unsafe state
- Not all unsafe sate are deadlocks
- Safe state is not a deadlocked state



교착상태(Deadlock) 처리 – 교착상태 회피

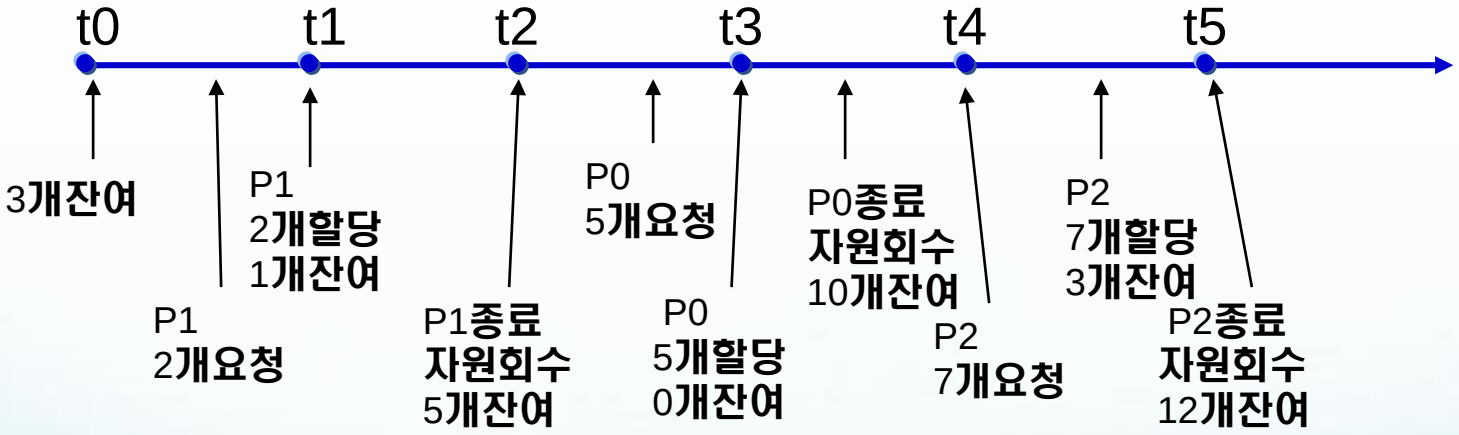
■ Safe State

☑ P={P0,P1,P2} R = { R0,R1,R2,R3,R5,R6,R7,R8,R9,R10,R11 }



	최대가능한 사용량	현재사용량(t0)
P0	10	5
P1	4	2
P2	9	2
잔여량	3	

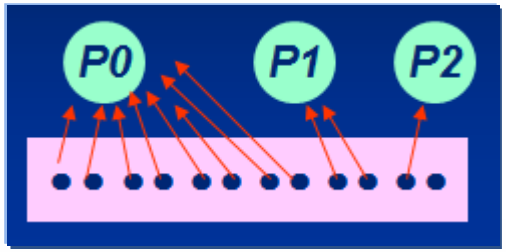
Safe Sequence : P1->P0->P2 ; safe **한 상태로 유지됨**,



교착상태(Deadlock) 처리 – 교착상태 회피

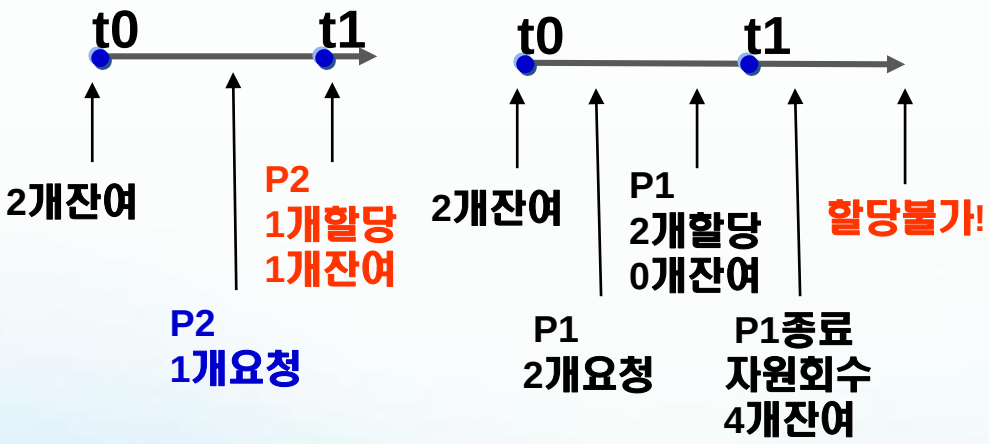
■ Unsafe State

☑ P={P0,P1,P2} R = { R0,R1,R2,R3,R5,R6,R7,R8,R9,R10,R11 }



	최대가능한 사용량	현재사용량(t0)
P0	10	8
P1	4	2
P2	9	1
잔여량	1	

Safe State -> Unsafe state

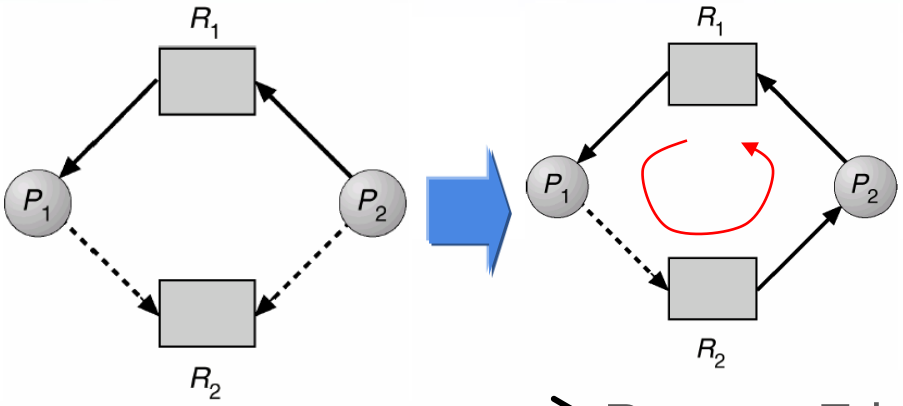


	최대가능한 사용량	현재사용량(t0)
P0	10	5
P1	4	2
P2	9	3
잔여량	2	

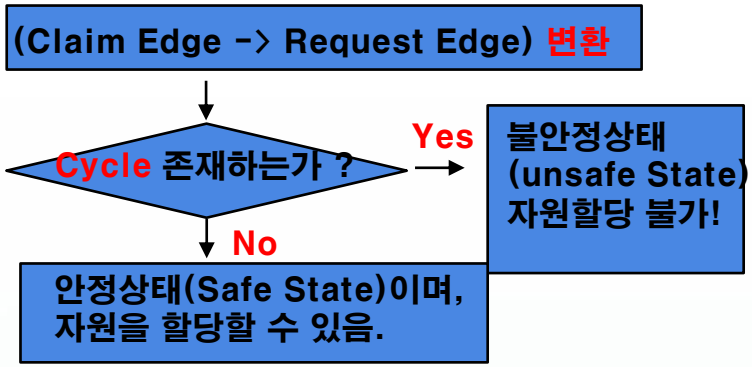
교착상태(Deadlock) 처리 – 교착상태 회피

■ Resource Allocation Graph Algorithm

☑ Only one Instance of resource



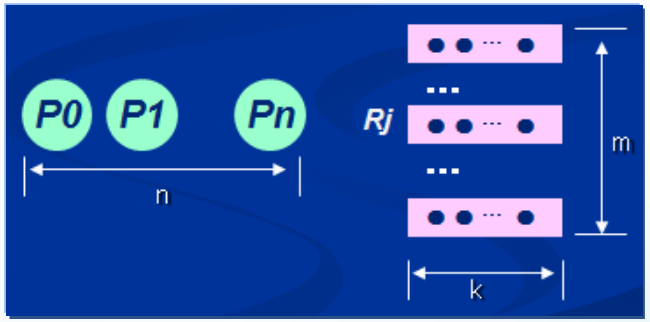
claim edge ; dashed line
= request edge in the future



■ Banker's Algorithm

☑ multiple instances of resource

잔여량 …… Available[j] = k,
최대요구량 … Max[i,j] = k
할당량 …… Allocation[i,j] = k
필요량 …… Need[i,j] = Max[i,j] – Allocation[i,j]



교착상태(Deadlock) 처리 – 교착상태 회피

- Banker's Algorithm

Available 1000

600

할당

Max 500
Need$_1$ 100

Finish$_1$
False

Allocation 400

Max 500
Need$_2$ 300

Finish$_2$
False

Allocation 0

Max 200
Need$_3$ 200

Finish$_3$
False

Allocation 0

Available 600

300

할당

Max 500
Need$_1$ 100

Allocation 400

Max 500
Need$_2$ 200

Allocation 300

Max 200
Need$_3$ 200

Allocation 0

교착상태(Deadlock) 처리 – 교착상태 회피

- Banker's Algorithm

Available ~~300~~ 200

할당

Max 500
Need$_1$ 0

Allocation 500

Max 500
Need$_2$ 200

Allocation 0

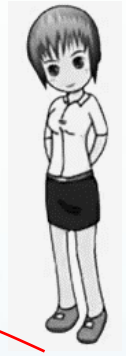
Max 200
Need$_3$ 200

Allocation 0

Available ~~200~~ 700

종료 및 반납

Max 0
Need$_1$ 0



Finish$_1$ True

Allocation ~~500~~

Max 500
Need$_2$ 200

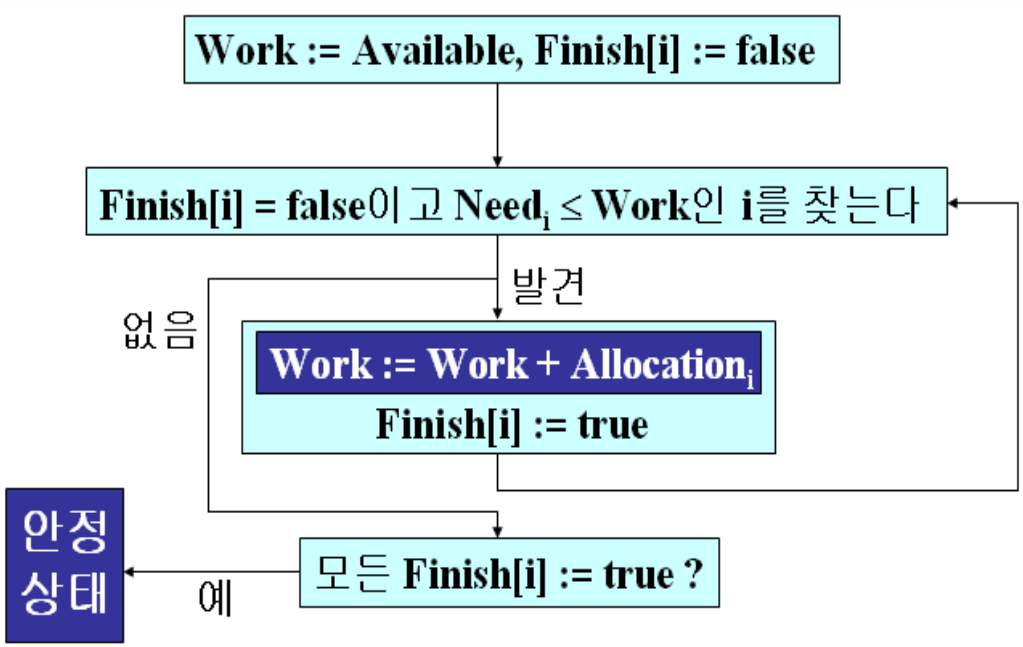
Allocation 300

Max 200
Need$_3$ 200

Allocation 0

교착상태(Deadlock) 처리 – 교착상태 회피

- Banker's Algorithm



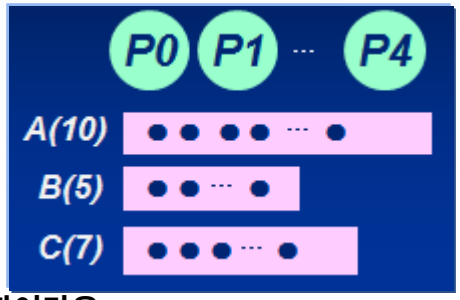
잔여량..... Available[j] = k,
최대요구량... Max[i,j] = k
할당량..... Allocation[i,j] = k
필요량..... Need[i,j] =
Max[i,j] – Allocation[i,j]

Finish[i] : all true …….. **처리완료**
: all false …... **모두 처리**X
: other case .. **일부처리중**

교착상태(Deadlock) 처리 – 교착상태 회피

■ Banker's Algorithm **예제**

	Allocation ABC	Max ABC	Need ABC	Available ABC
P0	010	753	743	332
P1	200 **322**	322	122	
P2	302	902	600	
P3	211	222	011	
P4	002	433	431	



종료되고 322반납 - 잔여량은

불안정 상태

취 소

$Request_0$ => (0, 2, 0)

$<P_1, P_3, P_4, P_2, P_0>$

안정 상태

$Request_1$ => (1, 0, 2)

	Allocation ABC	Need ABC	Available ABC
P0	010	743	230
P1	302	020	
P2	302	600	
P3	211	011	
P4	002	431	

$Request_4$ (3,3,0) ?

교착상태(Deadlock) 처리 – 교착상태 탐지

■ Deadlock Detection(**교착상태 탐지**)

☑ **교착상태가 발생하도록 허용하되, 교착상태의 발생을 탐지하고 복구(사후처리)**

☑ **2가지 알고리즘의 필요**

- Deadlock Detection Algorithm
- Deadlock Recovery Algorithm

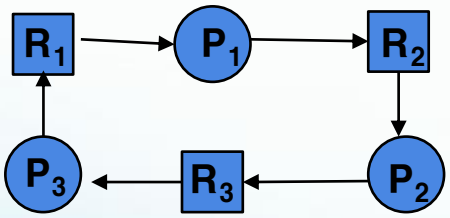
■ Deadlock Detection Algorithm

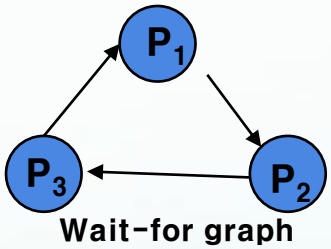
☑ Multiple instances of resource

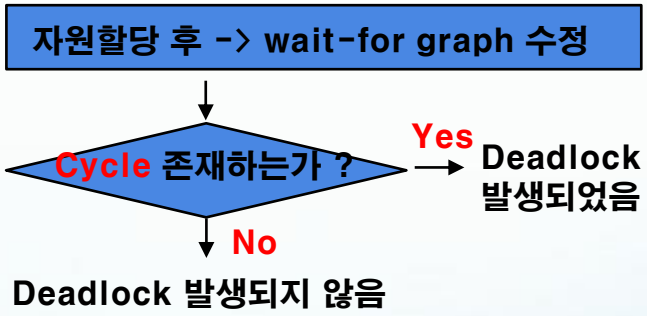
Banker's algorithm**의 변형알고리즘을 활용**

☑ Single instance of resource

wait-for graph **활용알고리즘**







교착상태(Deadlock) 처리 – 교착상태 탐지

교착상태 탐지 알고리즘

- 쇼사니(Shoshani)와 포크만(Coffman)이 제안.
- 다음과 같은 자료구조들을 사용함.

  Available : 자원 형태마다 사용 가능한 자원 수를 표시하는 길이가 m인 벡터.

  Allocation : 각 프로세스에 현재 할당된 각 형태들의 자원 수를 표시하는 n x m 행렬.

  ~~Need~~ Request : 각 프로세스의 현재 요청을 표시하는 n x m 행렬.

  Request[i, j] : 프로세스 P_i가 필요한 자원 수가 k 개라면 프로세스 P_i는 자원 형태 R_j의 자원을 k개 더 요청함. (프로세스 i 가 자원 j 를 요청함)

- 알고리즘 : 남아있는 프로세스들에 대한 할당 가능 순서를 모두 찾음.

  1단계 : Work과 Finish는 각각 길이가 m과 n인 벡터로, 'Work := Available' 로 초기화 함.

  - (i = 1, 2, …, n)일 때 'Allocation$_i \neq 0$' 이면 'Finish[i] := false' 이고, 아니면 'Finish[i] := true'.

  2단계 : 조건을 만족하는 색인 i를 찾으며, 조건에 맞는 i가 없으면 4단계로 이동.

  - Finish[i] = false, Request$_i \leq$ Work (작업할 프로세스가 남아있고, 자원 잔여량이 남아 있다면)

  3단계 : 다음이 일치하는지 여부를 판단하여 2단계로 이동.

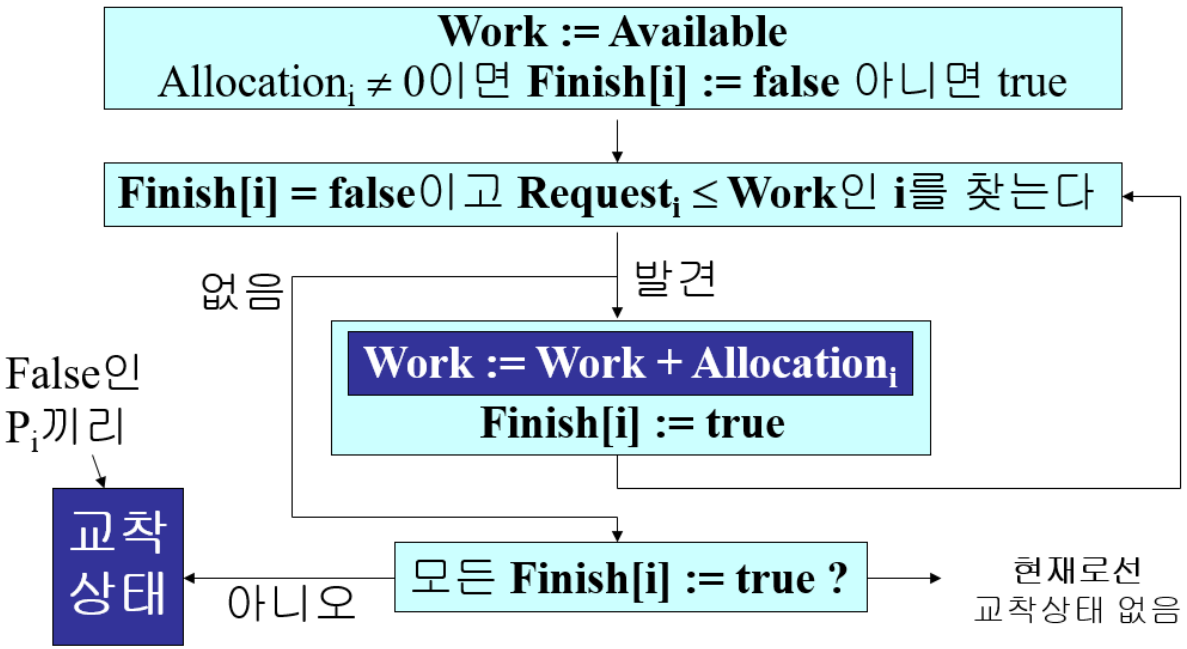
  할당처리 (Work := Work + Allocation$_i$ Finish[i] := true)

  4단계 : Finish[i] = false라면, 1 ≤ i ≤ n인 범위에서 시스템과 프로세스 P_i는 교착상태임.

교착상태(Deadlock) 처리 – 교착상태 탐지

■ 교착상태 탐지 알고리즘

- **쇼사니(Shoshani)와 포크만(Coffman)이 제안.**



Available
자원 형태 마다
사용 가능한 자원 수

Allocation
각 프로세스에 현재
할당된 각 형태들의 자원 수

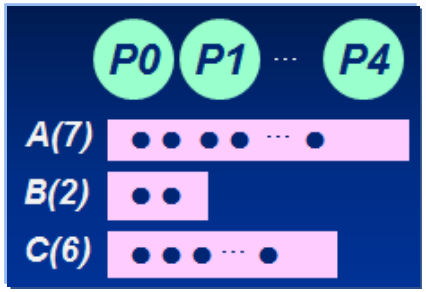
Request
각 프로세스의
현재 요청을 표시

Request[i, j]
프로세스 i 가
자원 j 를 요청함

교착상태(Deadlock) 처리 – 교착상태 탐지

표1

	Allocation ABC	Request ABC	Available ABC
P0	010	000	000
P1	200	202	
P2	303	000	
P3	211	100	
P4	002	002	



Sequence <P0, P2, P3, P1, P4> will result in Finish[i] = true for all i.

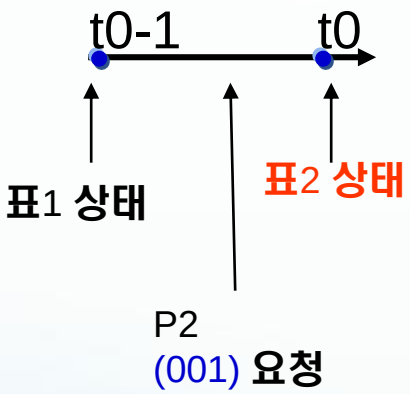


표2

	Request ABC
P0	000
P1	202
P2	001
P3	100
P4	002

P2 가 001 로 Request 할 경우에는 교착상태가 발생한다

즉, <P1, P2, P3, P4> 는 Deadlock 임.

교착상태(Deadlock) 처리 – 교착상태 복구

■ Deadlock Recovery(**교착상태 복구**)

☑ **복구 방법**

- **프로세스 중단**(Process Termination)
- **자원 선점**(Resource Preemption)

☑ **프로세스 중지**

방법1) 교착상태와 관련된 모든 프로세스를 중단

방법2) 교착상태가 해결될 때 까지 한 프로세스씩 중지

이때, 중지할 프로세스의 선택방법

- **프로세스들의 우선순위**
- **프로세스가 수행된 시간과 수행될 시간**
- **프로세스가 사용한 자원 형태와 수**
  (ex: 자원을 선점할 수 있는지의 여부)
- **프로세스 종료를 위해 필요한 자원 수**
- **프로세스를 종료하는 데 필요한 프로세스의 수**
- **프로세스가 대화식인지 일괄식인지 여부**

- Cost effective issue ?

☑ **자원선점(강제회수)**

- **희생자의 선택** ;
- **복귀**(Rollback) ; roll back to safe state and restart the process
- **기아**(Starvation) ; **해법**(include # of roll back in the cost factor.)

Q&A