

# 운영체제

2019.6.27



컴퓨터공학과 이병문



6/24 과목, 강의소개

25 운영체제 개요

26 Process, Thread

27 Concurrent Process

28 Concurrent Process

7/1 중간고사1

## 비동기 프로세스(상호배제)

- 임계영역(Critical Section)
- 상호배제(Mutual Exclusion)
- Producer/Consumer

## 상호배제 해결법

- 소프트웨어 알고리즘 1,2,3,4

## 상호배제/동기화

- Semaphore

# 병행프로세스(Concurrent Process)

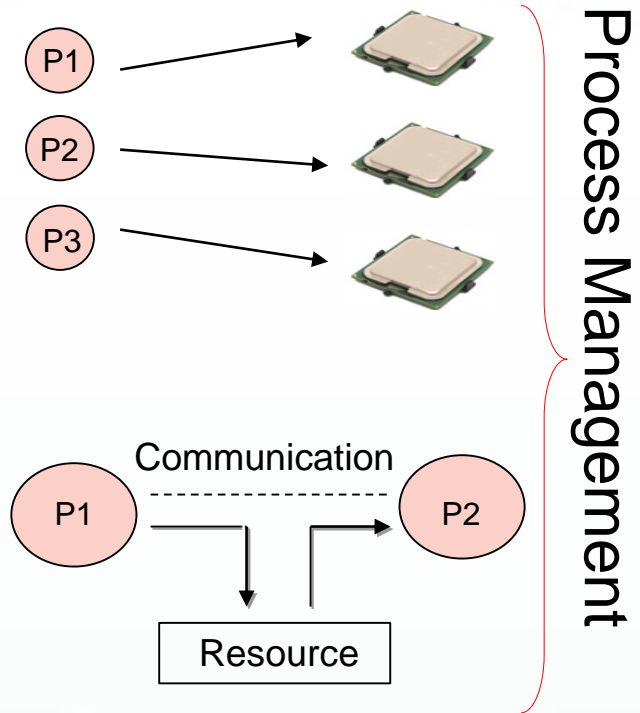
- **Concurrency (병행성) -> 성능향상 !**

- **Concurrent Process**

- 동시에 실행중인 2개 이상의 프로세스
- Co-Operation  
between two or more processes
- **Asynchronous Concurrent Processes**

- **Concurrent 가 가능 하려면 ...**

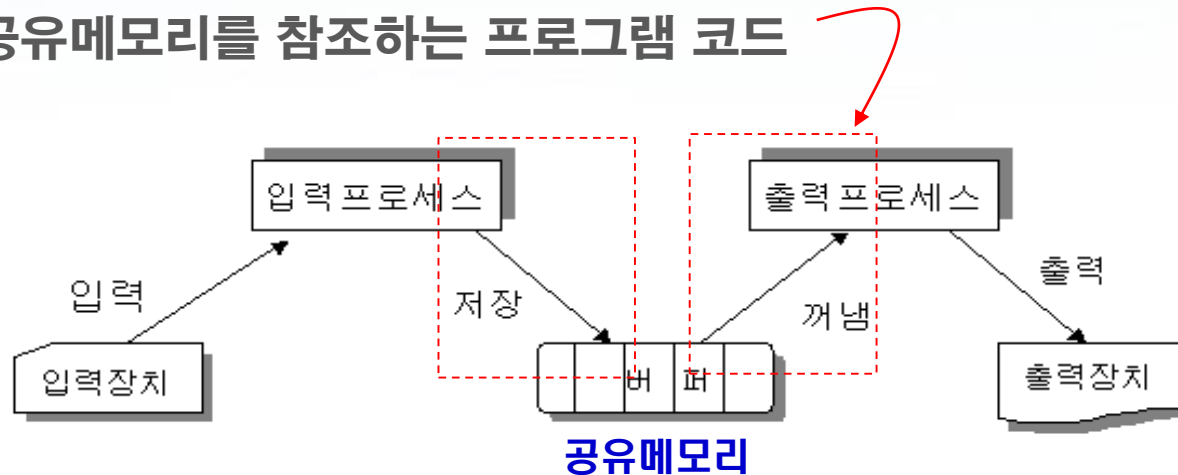
- 공유자원의 배타적 사용을 허용
- 상호 프로세스 협력/동기화 관리
- 프로세스간의 상호 통신방식이 필요
- 교착상태 해결방법 제시
- 상호 실행속도와 관계없이 일정한 실행결과가 나와야



# 비동기 병행프로세스(상호배제)

## ■ Critical Section(임계영역)

### ■ 공유메모리를 참조하는 프로그램 코드



#### 프로세스1

```
x = 5;  
y = 4;  
a = x + y;
```

```
z = 2;  
b = z + 1;
```

**c := a - b;**

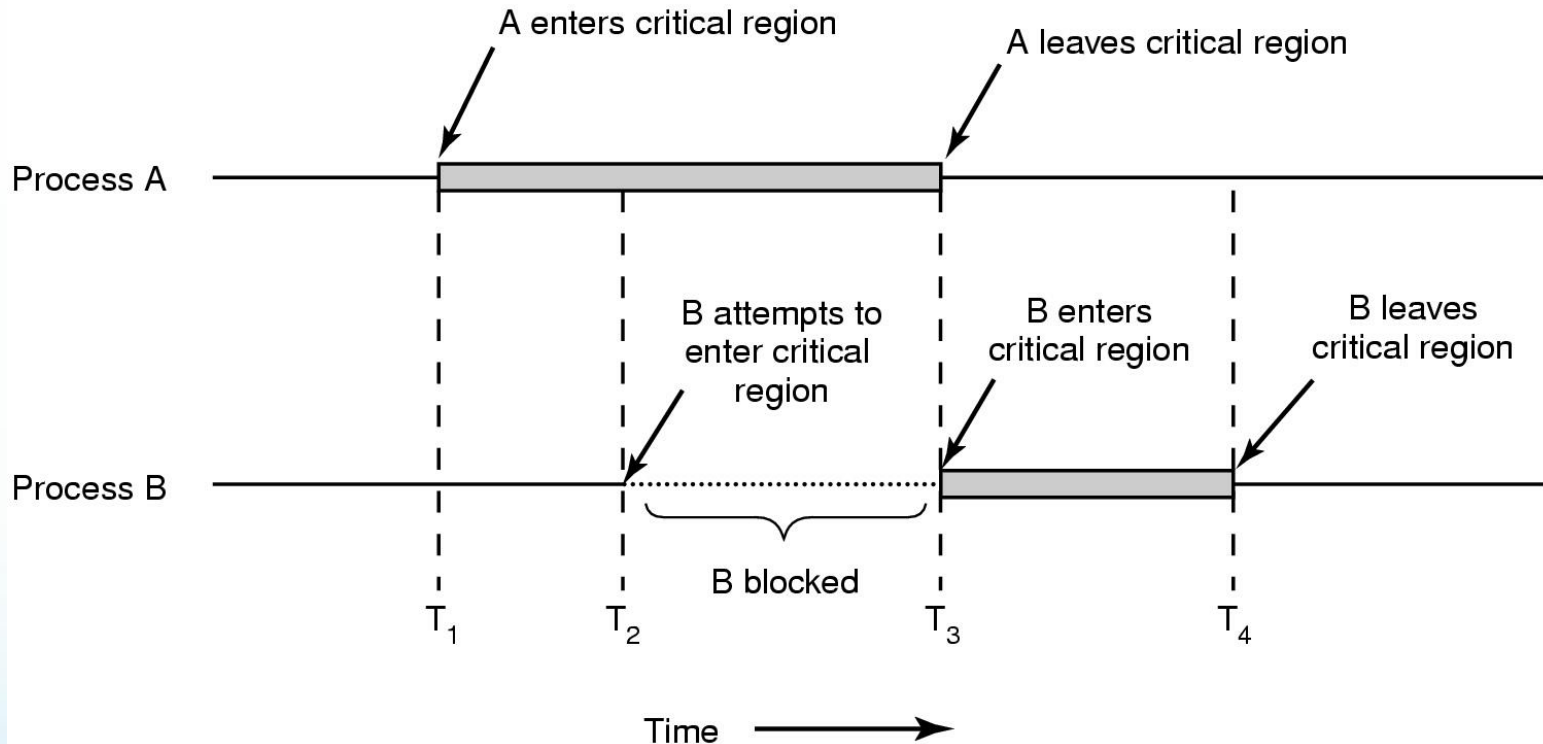
#### 프로세스2

- 프로세스가 공유메모리(버퍼)를 사용하는 코드를 임계영역이라 함
- 임계영역의 사용을 통제할 필요가 있음!

# 비동기 병행프로세스(상호배제)

## ■ Mutual Exclusion(상호배제)

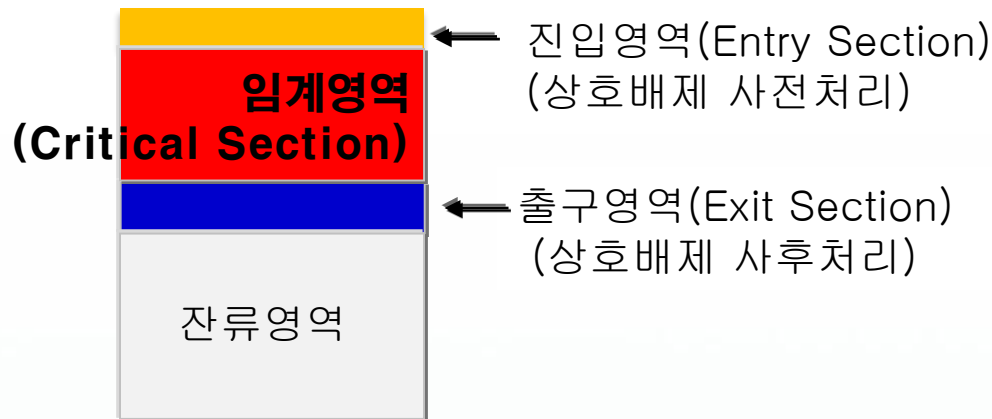
- 한 프로세스가 임계영역에 들어가면 다른 프로세스는 임계영역에 들어가지 못하도록 운영체제가 통제함.
- 임계영역 내에서는 빠른 처리! Block되지 않도록 처리!



# 비동기 병행프로세스(상호배제)

## ■ Mutual Exclusion(상호배제)

- 임계영역 문제를 해결하기 위해서 상호배제가 필요함
- 상호배제를 하기 위한 기본 처리구조



```
while (1) {  
    // 임계 영역에 들어가기 전의 코드를  
    ...  
    BeginCriticalSection();  
    // 임계 영역 (critical section)  
    ...  
    EndCriticalSection();  
    // 나머지 코드  
    ...  
}
```

⇒ 프로세스1

```
x = 5;  
y = 4;  
진입영역  
a = x + y;  
출구영역
```

⇒ 프로세스2

```
진입영역  
c := a - b;  
출구영역
```

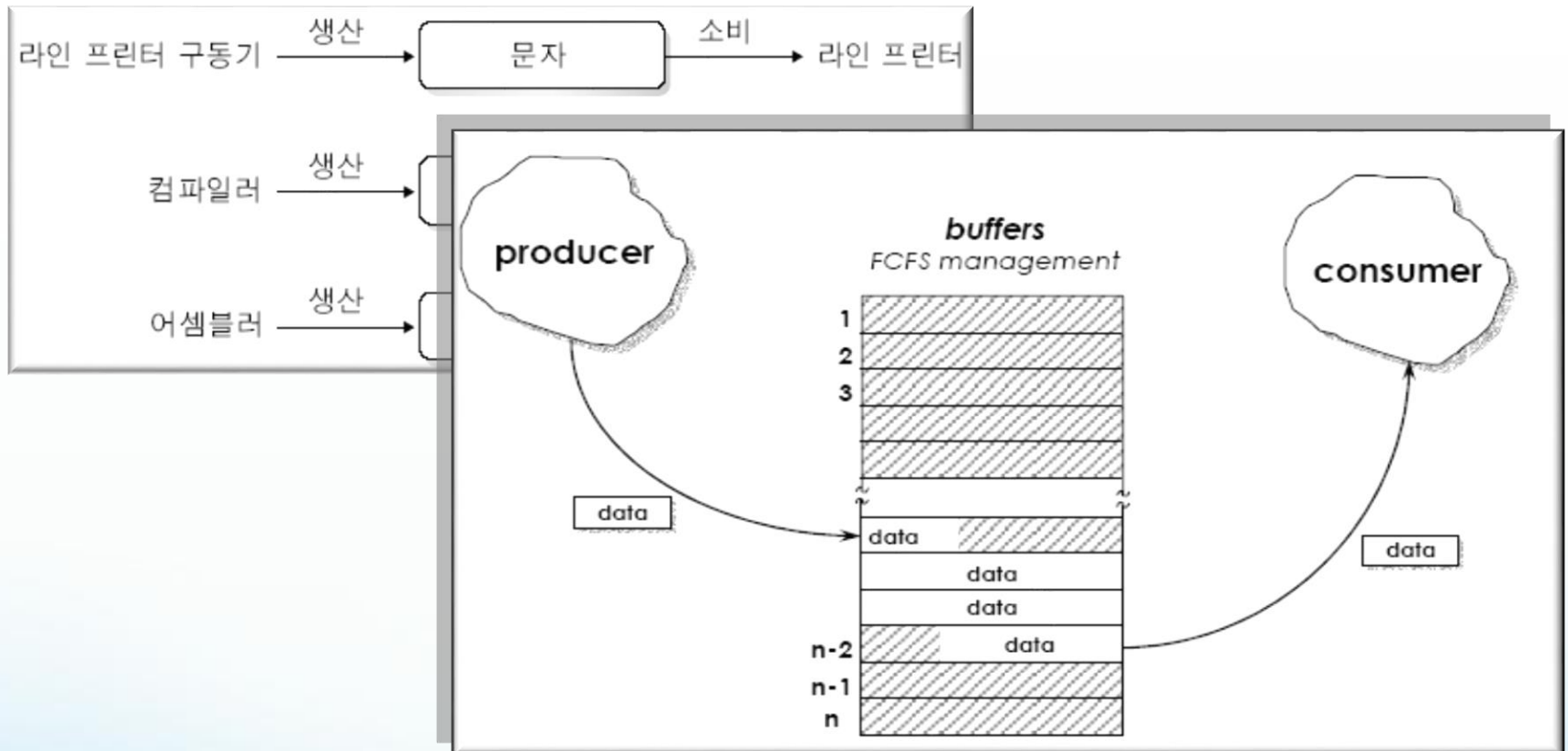


# 비동기 병행프로세스(상호배제)

## ■ Producer / Consumer Process

### ☑ 병행처리과정에서 발생하는 문제들

#### - 생산자/소비자 프로세스



# 비동기 병행프로세스(상호배제)

## ■ Example (limited buffer)

### ☑ Limited buffer

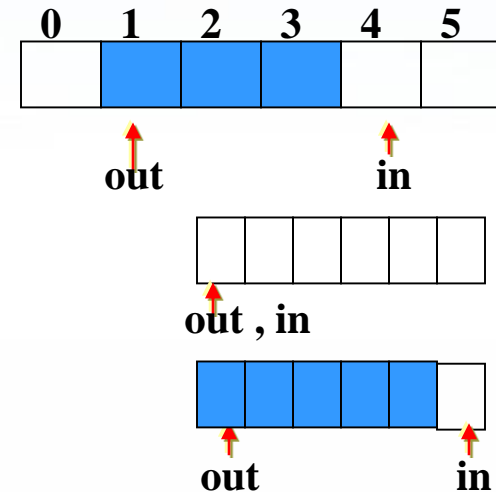
```
char  buffer[n];  
int   in, out, nextp, nextc;  
in = 0; out = 0;
```

### ☑ 3가지 조건

- 1) Empty case ;  $in == out$
- 2) Full case ;  $((in + 1) \% n) == out$
- 3) Other case ;  $in > out$

```
parbegin  
  생산자 : begin  
    repeat  
      ...  
      nextp 에서 한 항목 생산  
      ...  
      while  $(in + 1) \% n = out$  do no-op ;  
      buffer[in] := nextp ;  
      in := in + 1 mod n ;  
    until false ;  
  end ;
```

buffer[6]

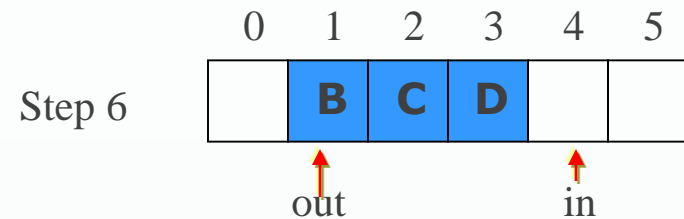
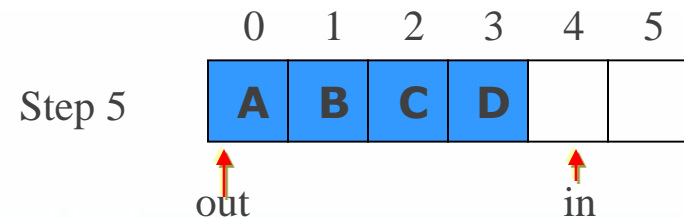
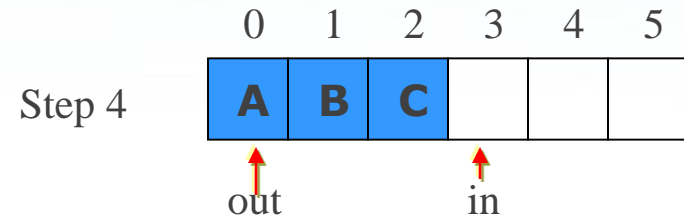
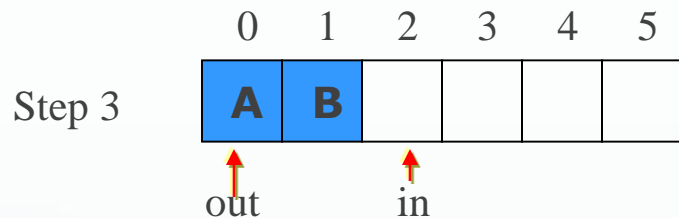
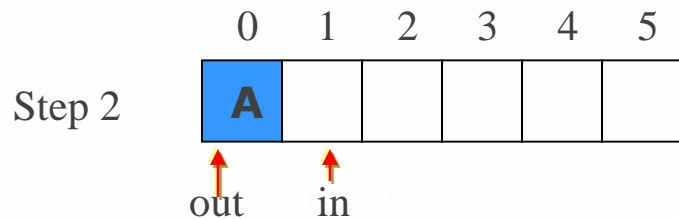
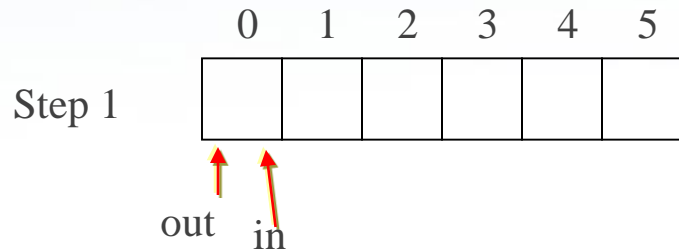


```
소비자 : begin  
  repeat  
    while in = out do no-op ;  
    nextc := buffer[out] ;  
    out := out + 1 mod n ;  
    ...  
    nextc 에서 한 항목 소비  
    ...  
  until false ;  
end ;  
parend ;
```



# 비동기 병행프로세스(상호배제)

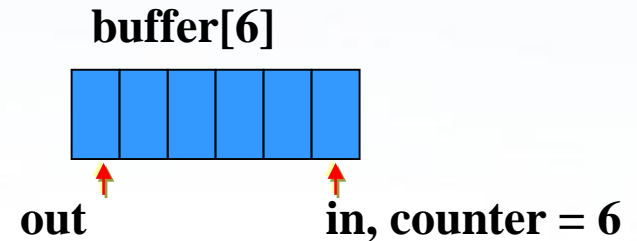
## ■ Example



# 비동기 병행프로세스(상호배제)

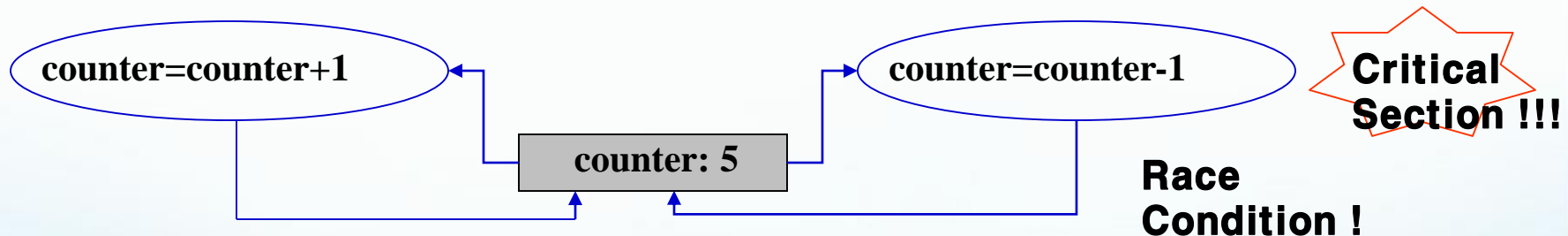
## ■ 임계영역의 예

```
char buffer[n];  
int in, out, nextp, nextc, counter=0;  
in = 0; out = 0;
```



```
repeat /* 생산자 프로세스 */  
...  
nextp 에서 한 항목 생산  
...  
while counter = n do no-op ;  
buffer[in] := nextp ;  
in := in + 1 mod n ;  
counter := counter + 1 ;  
until false ;
```

```
repeat /* 소비자 프로세스 */  
while counter = 0 do no-op ;  
nextc := buffer[out] ;  
out := out + 1 mod n ;  
counter := counter - 1 ;  
...  
nextc 에서 한 항목 소비  
...  
until false ;
```



# 상호배제 해결법(SW에 의한 해결방법1)


## ■ Algorithm 1, 특징

☑ 교대로 임계영역에 들어갈 수 있다. (한 개의 변수사용, processnumber)

### ☑ Algorithm 1

```
01 program versionone;
02   var processnumber : integer;
03   procedure p1;
04     begin
05       while true do
06         begin
07           while processnumber = 2 do ;
08             critical_sectionone;
09             processnumber := 2 ;
10             otherstuffone;
11         end;
12     end;
```

```
13 procedure p2;
14   begin
15     while true do
16       begin
17         while processnumber = 1 do ;
18           critical_sectiontwo;
19           processnumber := 1;
20           otherstufftwo;
21         end;
22     end;
```




```
23   begin
24     processnumber:=1;
25     parbegin
26       p1;
27       p2
28     parend
29   end
```

# 상호배제 해결법(SW에 의한 해결방법1)

## ■ Algorithm 1의 분석

- 두 프로세스 중 하나가 비정상적으로 중지되면 다른 프로세스는 진행불가능  
( Lockstep Synchronization Problem)



Time	Process, p1	Process, p2	비고
T1	23, 24 (processnumber = 1), 25		
T2	03 (p1)	13 (p2)	
T3	04, 05, 06	14, 15, 16	
T4	07	17	
T5	08	17	p1은 임계영역 p2는 바쁜대기
T6	08	17	
T7	08	17	
T8	09 (processnumber=2)	17	
T9	10	18	p2은 임계영역 p1는 바쁜대기
T10	10	18	
T11	05, 06,07	18	
T12	07	19 (processnumber=1)	
T13	08	20,21,15,16,17	

# 상호배제 해결법(SW에 의한 해결방법2)


## ■ Algorithm 2

☑ 2개의 변수를 사용 (p1inside, p2inside)

☑ Algorithm 2

```
01  program versiontwo
02  var  plinside, p2inside : boolean;
03  procedure p1;
04      begin
05          while true do
06              begin
07                  while p2inside do ;
08                  plinside := true;
09                  critical_sectionone;
10                  plinside := false;
11                  otherstuffone
12              end;
13      end;
```

```
14  procedure p2;
15      begin
16          while true do
17              begin
18                  while p1inside do ;
19                  p2inside := true;
20                  critical_sectiontwo;
21                  p2inside := false;
22                  otherstufftwo
23              end;
24      end;
```




```
25  begin
26      plinside:=false;
27      p2inside:=false;
28      parbegin;
29          p1;
30          p2
31      parend
32  end
```

# 상호배제 해결법(SW에 의한 해결방법2)

## ■ Algorithm 2의 분석

-두개의 변수사용( P1inside, P2inside )

-두 프로세스가 동시에 임계영역에 들어간다면 상호배제가 보장되지 않음



Time	Process, p1	Process, p2	비고
T1	25 (p1inside=false), 24 (p2inside=false), 28		
T2	3,4,5,6	14,15,16,17	
T3	07 (wait if p2inside)	18 (wait if p1inside)	
T4	08 (p1inside=true)	19 (p2inside=true)	
T5	09	20	오류발생!!! 상호배제가 되지않음!!!
T6	09	20	
T7	09	20	
T8	10 (p1inside=false)	21 (p2inside=false)	
T9	11,12, 5,6	22,23,16,17	
T10	07 (wait if p2inside)	18 (wait if p1inside)	
T11	08 (p1inside=true)	19 (p2inside=true)	
T12	09	20	오류발생!!!
T13	09	20	



# 상호배제 해결법(SW에 의한 해결방법3)

## ■ Algorithm 3

☑ 2개의 변수를 사용 (p1enter, p2enter)

☑ Algorithm 3

```
01  program versionthree;  
02  var p1enter, p2enter : boolean;  
03  procedure p1;  
04      begin  
05          while true do  
06              begin  
07                  p1enter :=true;  
08                  while p2enter do ;  
09                      criticalsectionone;  
10                      p1enter := false;  
11                      otherstuffone  
12              end;  
13  end;
```


```
14  procedure p2;  
15      begin  
16          while true do  
17              begin  
18                  p2enter :=true;  
19                  while p1enter do ;  
20                      criticalsectiontwo;  
21                      p2enter :=false;  
22                      otherstufftwo  
23              end;  
24  end;
```

```
25  begin  
26      p1enter :=false;  
27      p2enter :=false;  
28      parbegin  
29          p1;  
30          p2  
31      parend  
32  end
```

# 상호배제 해결법(SW에 의한 해결방법3)

## ■ Algorithm 3의 분석

- while 검사하기전에 검사시작 사실을 Flag 로 처리
- 두 프로세스가 while 검사전에 각각 자기 Flag 를 동시에 바꾸면 Deadlock 발생



Time	Process, p1	Process, p2	비고
T1	25, 26 (p1enter=false), 27 (p2enter=false), 28		
T2	3,4,5,6	14,15,16,17	
T3	07 (p1enter=true)	18 (p2enter=true)	동시에 셋팅!!
T4	08 (wait if p2enter)	19 (wait if p1enter)	while 검사!! Deadlock !!!
T5			
T6			
T7			
T8			
T9			
T10			

- 해결방법: 상대적인 시간차를 두고 셋팅하면 해소할 수도 있다. !!

# 상호배제 해결법(SW에 의한 해결방법4)

## ■ Algorithm 4

- ☑ 두 개의 변수사용 ( p1enter, p2enter )
- ☑ 상대적인 시간차, delay(random, fewcycle) 함수를 도입

```
program versionfour;  
var p1enter, p2enter : boolean;  
procedure p1;  
begin  
  while true do  
    begin  
      p1enter := true;  
      while p2enter do ;  
      begin  
        p1enter := false;  
        delay(random, fewcycles);  
        p1enter := true  
      end;  
      critical_sectionone;  
      p1enter := false;  
      otherstffone  
    end;  
  end;  
end;
```


```
procedure p2;  
begin  
  while true do  
    begin  
      p2enter := true;  
      while p1enter do ;  
      begin  
        p2enter := false;  
        delay(random, fewcycles);  
        p2enter := true  
      end;  
      critical_sectiontwo;  
      p2enter := false;  
      otherstufftwo  
    end;  
  end;  
end;
```

```
begin  
  p1enter := false;  
  p2enter := false;  
  parbegin  
    p1;  
    p2  
  parend  
end
```

# 상호배제 해결법(SW에 의한 해결방법4)

## ■ Algorithm 4의 분석

- 상대적인 시간차, delay(random, fewcycle) 함수를 도입,
- 각 프로세스의 상대적인 속도를 예측할 수 없어 무한연기 문제가 발생



Time	Process, p1	Process, p2	비고
T1	35, 36 (p1enter=false), 37 (p2enter=false), 38		
T2	3,4,5,6	19,20,21,22	
T3	07 (p1enter=true)	23 (p2enter=true)	
T4	08 (while)	24 (while)	
T5	09, 10(p1enter=false)	25, 26(p2enter=false)	
T6	11 (delay(random, few))	27 (delay(random, few))	지연시간
T7	11 (delay(random, few))	28 (p2enter=true), 29	
T8	11 (delay(random, few))	24	
T9	12 (p1enter=true), 13	30	
T10	08	30	
T11	09, 10(p1enter=false)	31 (p2enter=false)	
T12	11 (delay(random, few))	32,33,21,22,23	
T13	12 (p1enter=true), 13	24 (while)	
T14	08	25, 26(p2enter=false)	
T15	14	27 (delay(random, few))	

# 상호배제 해결법(Dekker 알고리즘)

## ■ 2개 프로세스 상호배제 문제해결

```
program dekkersalgorithm;  
var fprocess: (first, second);  
    p1enter, p2enter : boolean;  
procedure p1;
```

```
begin
```

```
while true do
```

```
begin
```

```
    p1enter := true;
```

```
    while p2enter do ;
```

```
    if fprocess = second then
```

```
        begin
```

```
            p1enter := false;
```

```
            while fprocess = second do ;
```

```
            p1enter := true
```

```
        end;
```

```
        critical_sectionone;
```

```
        fprocess := second;
```

```
        p1enter := false;
```

```
        otherstuffone
```

```
    end;
```

```
end;
```

```
procedure p2;
```

```
begin
```

```
while true do
```

```
begin
```

```
    p2enter := true;
```

```
    while p1enter do ;
```

```
    if fprocess = first then
```

```
        begin
```

```
            p2enter := false;
```

```
            while fprocess = first do ;
```

```
            p2enter := true;
```

```
        end;
```

```
        critical_sectiontwo;
```

```
        fprocess := first;
```

```
        p2enter := false;
```

```
        otherstufftwo
```

```
    end;
```

```
end;
```

임계영역 진입 시도

상대방이 먼저  
진입했다면 기다린다

내가 먼저  
진입했더라도 상대방  
차례면 진입취소하고  
재진입 시도

```
begin  
    p1enter := false;  
    p2enter := false;  
    fprocess := first;  
    parbegin  
        p1;  
        p2  
    parend  
end
```

# 상호배제 해결법(Dekker 알고리즘)

## ■ Dekker Algorithm 의 분석

☑ 3개 변수(p1enter, p2enter, fprocess)

☑ 동시에 2개의 프로세스가 임계영역에 들어가려 할때 fprocess 변수로 순서정함

Time	Process, p1	Process, p2	비고
T1	40, 41 (p1enter=false), 42 (p2enter=false), 43 (fprocess=first)		
T2	4,5,6,7	22,23,24,25	
T3	08 (p1enter=true)	26 (p2enter=true)	
T4	09 (wait if p2enter)	27 (wait if p1enter)	
T5	10 (if fprocess = second)	28 (if fprocess = first)	
T6	16	29,30 (p2enter=false)	
T7	16	31 (wait if first)	
T8	17 (fprocess=second)	31 (wait if first)	
T9	18 (p1enter=false)	32 (p2enter=true), 33	
T10	19	34	
T11	20, 6, 7, 8 (p1enter=true)	34	
T12	09 (wait if p2enter)	35 (fprocess=second)	
T13		36 (p1enter=false)	
T14		37	
T15		38,24,25,26, (p1enter=true)	



# 세마포어

# 상호배제 / 동기화

## ■ Semaphore, S

### ☑ 개념

- 기존 상호배제 해법(Algorithm1,2,3,4 등)으로는 임계영역 문제를 일반화시키기 어려움
- 다익스트라(Dijkstra)는 세마포어(Semaphore) 개념을 제시

### ☑ 정의

```
P(S) : wait(S) {  
    while (S <= 0)  
        no-op;    /* busy waiting */  
    S--;  
}
```

```
V(S) : signal(S) {  
    S++;  
}
```

$P_1: a=12+21$

$P_2: b=23-1$

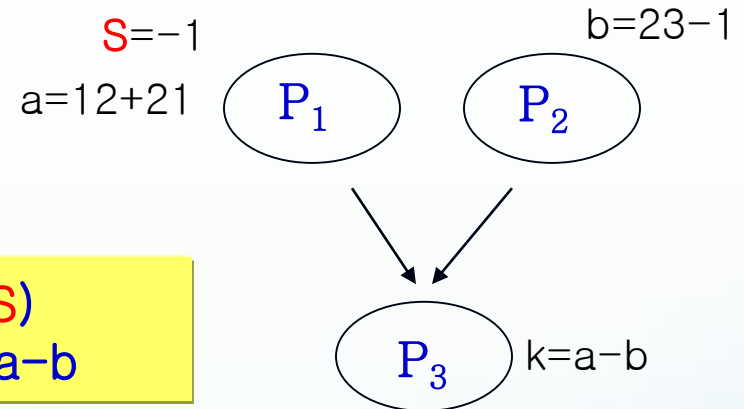
$P_3: k=a-b$

### ☑ 사용

$P_1$   $a=12+21$   
 $V(S)$

$P_2$   $b=23-1$   
 $V(S)$

$P_3$   $P(S)$   
 $k=a-b$



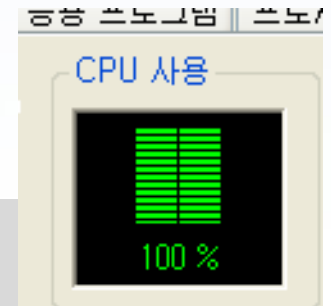
# 상호배제 / 동기화

## ■ 구현

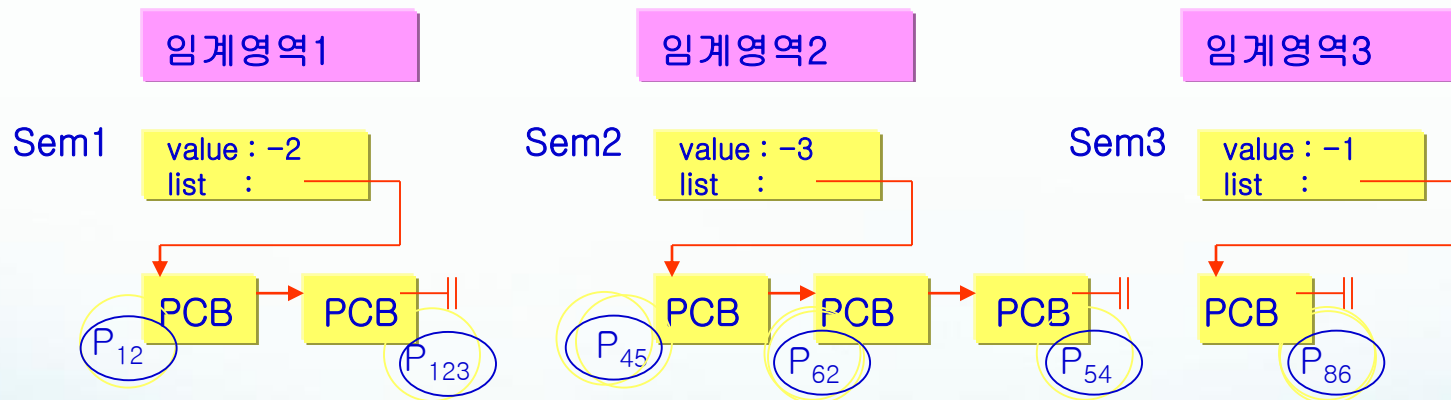
- ☑ Busy waiting 문제 -> semaphore 를 수정해서 해결

```
P(S) {  
    S--;  
    if (S < 0) {  
        프로세스를 대기(보류)상태로 설정  
        프로세스번호를 대기 큐에 추가  
    }  
    else  
        임계영역 수행  
}
```

```
V(S) {  
    S++;  
    if (S <= 0) {  
        대기 큐에서 프로세스 제거  
        제거한 프로세스를 준비 큐에 추가  
    }  
}
```



- ☑ 예



# 상호배제 / 동기화

## ■ Semaphore 특성

### ☑ 원자적(Atomic) 명령

- 세마포어를 실행하는 동안에는 인터럽트나 이벤트에 의해 중단되지 않아야 함

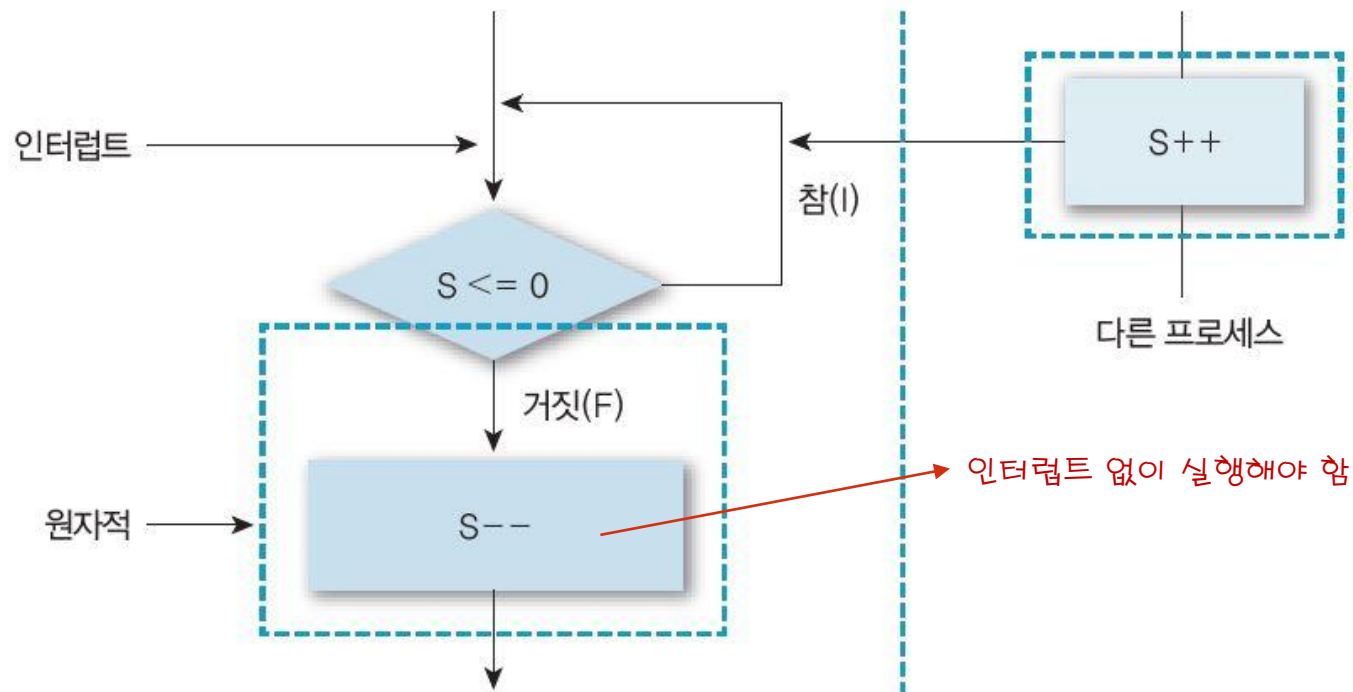


그림 4-24 세마포의  $S \leq 0$ 과  $S--$ 의 원자적 실행, 인터럽트

# 상호배제 / 동기화

## ■ Semaphore 특성

- ✓ 세마포어를 사용하면 N개의 임계영역 문제를 해결할 수 있음
- ✓ 세마포어를 사용하여 N개 프로세스의 상호배제를 구현할 수 있음

```
semaphore s(1);                // 변수 선언 및 초기화
while (1) {
    ...
    s.wait();    P(s);
    // 임계 영역 (critical section)
    ...
    s.signal();  V(s);
    ...
}
```

```
semaphore s;                    // 공유 변수
s.init(1);                      // 초기화
```

또는

```
semaphore s(1)
```

**Q&A**