Semaphore

A semaphore S is an integer variable that can be accessed only through two standard atomic operations P and V,

where:

```
        P(S): While S ≤ 0  do skip;
                S = S -1;
and
        V(S):  S = S + 1;
```

Semaphore

Examples:

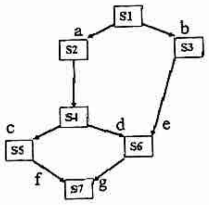
1- Process Pi and Pj are made of s1 and s2 statements, respectively. Use a semaphore to synchronize Pi and Pj in such a way that s1 be executed before s2.

Semaphore

```
        synch = 0;
                            P(sync)
S1                          S2
v(sync)
```

Semaphore

2- Use semaphores to synchronize the following precedence graph.



```
            a = b = c = d = e = f = 0
Parbegin:
        s1; V(a); V(b);
        P(a); s2; s4; V(c); V(d);
        P(b); s3; V(e);
        P(c); s5; V(f);
        P(d); p(e); s6; V(g);
        P(f); p(g); s7;
Parend;
```

Semaphore

3- Synchronize n-cooperating processes using a semaphore.

Semaphore

Solution for problem # 3:

```
  Repeat
      P(mutex)
          Critical section
      V(mutex)
          Remainder
Until false;
```

Semaphore

Implementation problem:
Busy-waiting
Solution
Blocking operation and
Wake Up operation

Semaphore

Modified definition of atomic operations P and V:

```
#define MAX = 100;
Struct{
        int value;
        int L[MAX];
} S;
```

```
Wait(s): s.value - -;
            If (S.valu<0)
            {add the process to the list ,S.L;
             block (process);
            }
Signal(s): S.value + +;
            If (S.value ≤ 0)
            {Remove process from S.L;
              Wake up(process)
            }
```

Classical process coordination problems

1. **Bounded buffer**
2. **Readers/Writers**
3. **Dinning philosophers**

Bounded Buffer

empty = n;   full = 0;   mutex = 1;

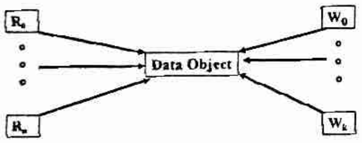
```
Repeat
    .
    .
    Produce an item nextp
    P(empty)
    P(mutex)
    .
    .
    Add nextp to buffer
    .
    V(mutex)
    V(full)
Until false
```

Producer

```
Repeat
      .
      .
      P(full)
      P(mutex)
       .
       .
       .
       remove an item from buffer →nextc
      .
      V(mutex)
      V(empty)
            consume the item in nextc
Until false
```

Consumer

Readers/Writers



Readers/Writers

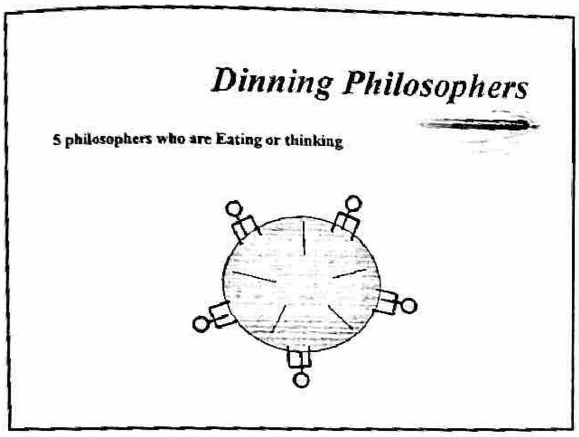
readcount = 0;   wrt = 1;   mutex = 1;

```
Repeat
    P(mutex);
      readcount = readcount +1;
      If (readcount = 1) then {P(wrt)}
    V(mutex)
    .
    .
    Reading is performed
    .
    P(mutex);
      readcount = readcount -1;
      If (readcount = 0) then {V(wrt)}
    V(mutex)
Until false
```

Reader

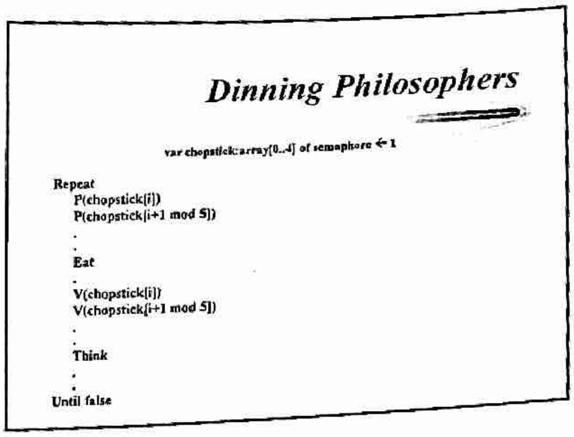
```
Repeat
        P(wrt);
        .
        .
        Writing is performed
         V(wrt)
Until false
```

Writer



Dinning Philosophers

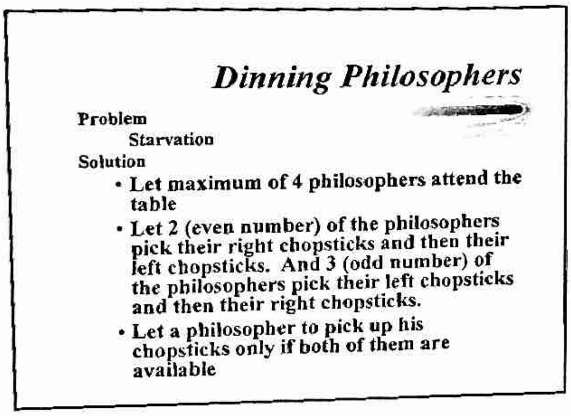
5 philosophers who are Eating or thinking



Dinning Philosophers

var chopstick:array[0..4] of semaphore ← 1

```
Repeat
    P(chopstick[i])
    P(chopstick[i+1 mod 5])
    .
    .
    Eat
    .
    V(chopstick[i])
    V(chopstick[i+1 mod 5])
    .
    .
    Think
    .
    .
Until false
```



Dinning Philosophers

Problem

Starvation

Solution

- Let maximum of 4 philosophers attend the table
- Let 2 (even number) of the philosophers pick their right chopsticks and then their left chopsticks. And 3 (odd number) of the philosophers pick their left chopsticks and then their right chopsticks.
- Let a philosopher to pick up his chopsticks only if both of them are available