**Lamport's Bakery Algorithm**

- Each process receives a number before entering its critical section
- Smallest number has highest precedence
- If two processes i and j have the same number and i < j, then i is served first; else j is served
- Numbers are generated in non-decreasing order (e.g., 1, 2, 3, 3, 4, 5)
- Semantics of (a,b) < (c,d)
  - if (a < c) or (a = c) and (b < d)

```java
// Lamport's Bakery Algorithm
shared int choosing[N] = {0,...};
shared int num[N] = {0,...};
process(i) {
    choosing[i] = 1;
    num[i] = 1 + max(num[0], ..., num[N-1]);
    choosing[i] = 0;
    for j=0 to N-1 {
        while (choosing[j]) { ... do nothing ... }
        while ((num[j] != 0) and ((num[j], j) < (num[i], i))
            { ... do nothing ... }
    }
    ... critical section ...
    num[i] = 0;
}
```

**Binaray Semaphore (Atomic)**

```java
BinWait(S):    BinSigna(S):
   if (S.flag == 1)  if (S.queue is empty)
      S.flag = 0;   S.flag = 1;
   else {
      Enter process into    Remove process from
      S.queue;             S.queue;
      Suspend;             Place process on run queue;
   }
```

**Counting Semaphore Implementation**

- Implement a counting semaphore using binary semaphores
- $S1, S2, S3$: Binary semaphores
- $C$: Integer counter
- Initialization
  - $S1 = 1$: // protect critical section protecting $C$
  - $S2 = 0$: // signal with 1 that $C > 0$
  - $C =$ initial count value of semaphore;