Lamport's Bakery Algorithm

Devised as a solution to the n-process mutual exclusion problem.

Analogy

- Bakery with a numbering machine.
- Each customer gets a unique number.
- Global counter displays the customer being served currently.
- After baker is done, next number is displayed; served customer leaves.

Numbering scheme always generates numbers in increasing order. (i.e.: 1, 2, 2, 3, 3, 4, 4, 4, ...)

If process P_i & P_j receive the same number,

if $i < j$, P_i is served first.

else, P_j is served first.

Assumptions:

1) Processes numbered 0 to N-1
2) num is an integer array of size N.
3) entry is a boolean array of size N.
4) pair : (number, ID)
5) $(a, b) < (c, d) \equiv$
   $(a < c)$ or $(a == c)$ && $(b < d)$

(priority to lower number, break tie with pid)

Algorithm :

Lock(i)

```
entry[i] = true
num[i] = 1 + max(num[0], ..., num[N-1])
entry[i] = false.

for (j = 1 ; j < N ; j++) { if(j == i) continue.
    // Wait until thread j receives its
       number.
    while (entry[j]) { // do nothing }

    // wait for all threads w/ smaller
    // numbers or same number (but higher
    // priority) to finish their execution.

    while (num[i] != 0 &&
           (num[j], j) < (num[i], i))
      { // do nothing }
}
```

Unlock(i)

```
num[i] = 0
```

Thread(i)

```
while (true) {
    lock(i)
    /* critical section */
    unlock(i)
    /* non critical section */
}
```

The algorithm guarantees:

1) Mutual exclusion
2) Bounded waiting
3) Progress

Space complexity : $O(2n)$ — cannot improve beyond $O(n)$.
Time complexity : $O(n)$ — can be improved to $O(1)$ with fast mutual exclusion.

Example :

	P_1	P_2	P_3	P_4	P_5
1	$n_0 = 0$	$n_1 = 0$	$n_2 = 0$	$n_3 = 0$	$n_4 = 0$
2		$n_1 = 1$			
3	$n_0 = 2$				
4			$n_2 = 3$		
5				$n_3 = 3$	
6					$n_4 = 4$
7		exec CS			
8		$n_1 = 0$			
9	exec CS				
10	$n_0 = 0$				
11			exec CS		
12			$n_2 = 0$		
13				exec CS	
14			tie is resolved by pid.	$n_3 = 0$	
15					exec CS
16					$n_4 = 0$

n_i : num[i]