Software Solutions for the Critical Section Problem with *N* Processes

- Eisenberg and McGuire's Algorithm for *N* processes, where $N \geq 2$.
- Bakery Algorithm for *N* Processes, where $N \geq 2$.
- Note: Peterson's solution (given in our text book) can be used only for two cooperating processes, like a producer process and a consumer process.

Eisenberg and McGuire's Algorithm for *N* processes

- Shared variables:

  enum pstate {idle, want_in, in_cs};

  pstate flag[n]; // initialized to idle

  int turn; // initialized to any value in [0, n-1]
- The first contending process in the cyclic order (P_{turn}, P_{turn+1}, …, P_{n-1}, P_o, …, P_{turn-1}) will enter the critical section.

Eisenberg and McGuire's Algorithm (cont'd)

- **Structure of Process P_i**

```
do {
    while (true) {
        flag[ i ] = want_in;
        int j = turn;

        while (j != i) {
            if (flag[ j ] != idle)
                j = turn;  /* to repeat scan from P_turn to P_i */
            else  j = (j +1) % n;
        }

        flag[i] = in_cs; /* flag[ i ] == in_cs doesn't imply
                            turn == i or only P_i is in in_cs */
        j = 0;
        while ( (j < n) && (j == i || flag[ j ] != in_cs)) j++;
        if ( (j >= n) && (turn == i || flag[turn] == idle))
            break;
    }

    turn = i;
```

```
        // critical section
    j= (turn +1) % n;
    while (flag[ j ] == idle)
        j = (j + 1) % n;
    turn = j:
    flag [ i ] = idle;
        // remainder section
} while(true);
```

Eisenberg and McGuire's Algorithm (cont'd)

- It is possible that more than one process has *in_cs* state.
  - Suppose there are 5 processes (P_0, P_1, P_2, P_3, P_4) and *turn is* 1. Initially all the processes are in *idle* state.
  - P_4 executes the 2nd while statement and its state becomes *in_cs*.
  - Shortly after that, P_3 executes the 2nd while statement and its state also becomes *in_cs*.
  - Both P_4 and P_3 execute their 3rd while statement and find each other in *in_cs* state. So, they will escape their 3rd while statement (with j < n) and start their entry sections all over again.
  - In this 2nd iteration of the entry section, P_4 will find that P_3 is not idle, so P_4 cannot change its state from *want_in* to *in_cs*. On the other hand, P3 will change its state from *want_in* to *in_cs*, then breaks out of the 1st while statement and set turn=3.
    - Before P_3 breaks out of the 1st while statement, it checks whether it has the *turn* or P_{turn} is still idle.
- After a process, say P_i, finishes it critical section, it will scan the states of other processes, starting from process P_{i+1} (in cyclic order), and the turn will be given to the first non-idle process.
  - If all other processes are idle when P_i finishes its critical section, the *turn* will remain as i.

Bakery Algorithm for *N* Processes

- Before entering its critical section, each process picks up a ticket number, that is (1 + the largest ticket number assigned to some other process). A process that holds the smallest positive number will enter the critical section.
- The numbers picked up by the processes that want to enter their critical sections are monotonically increasing; e.g., 1, 2, 3, 3, 3, 3, 4, 5, ... That means, two or more processes may pick up the same number.
- Suppose that processes P_i and P_j have picked up the same number. In that case, their process ids are used as a tie-breaker:
  - If i < j, then P_i enters its critical section first; otherwise P_j enters its critical section first.

Bakery Algorithm (cont'd)

- Notation **<** is used for lexicographical ordering of different pairs of (ticket #, process id).
  - $(a, b) < (c, d)$ if $a < c$, or if $a = c$ and $b < d$
- max $(a_0, \ldots, a_{n-1})$ is a number k, such that $k \geq a_i$ for $i = 0, \ldots, n-1$.
- Shared variables:

  boolean choosing[n]; // initialized to false

  int number[n]; // initialized to 0

  - choosing[i] is true when process P_i wants to enter its critical section but has not picked up a number yet.

Bakery Algorithm (cont'd)

- **Structure of Process P_i**

```
do {
     choosing[i] = true;
     number[i] = max(number[0], number[1], … ,
                          number [n–1]) +1;
     choosing[i] = false;
     for (j = 0; j < n; j++) {
               while (choosing[ j]) ; /* busy waiting until Pj
                                         picks up a ticket number */
               while ((number[ j] != 0) && ((number[ j], j) <
                          number[ i], i))) ; /* busy waiting until
                                    Pi has higher priority than Pj */
      }
          // critical section
     number[ i] = 0;
          // remainder section
} while (true);
```