Introduction to Parallel & Distributed Programming

Lec 13— Mutual Exclusion

Subodh Sharma | Feb 13, 2026

Bakery Algorithm

An n thread ME Solution

- Doorway:
  - $flag[i] := true$ — I'am interested
  - $label[i]\ldots$ — Take increasing labels
- Bounded Wait:
  - Someone is interested **AND** has earlier lex-ordered label
  - $D_A \rightarrow D_B$ then $A's$ label is smaller
  - B is locked out while $flag[A]$ is true!

```
class Bakery implements Lock {
  boolean[] flag;
  Label[] label;
 public Bakery (int n) {
   flag  = new boolean[n];
   label = new Label[n];
   for (int i = 0; i < n; i++) {
      flag[i] = false; label[i] = 0;
   }
 }
```

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```

Prove that Bakery Alg is ME-safe!

Bakery Algorithm

Does it guarantee ME?

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```

T_0

T_1

$label[0] = 1$

$label[1] = 1$

$flag[1] \wedge (label[1],1) < (label[0],0)$

Enter CS

$flag[0] = false$

$flag[0] \wedge (label[0],0) < (1,1)$

Loop

$flag[0] \wedge (label[0],0) < (1,1)$

Enter CS

Bakery Algorithm

Does it guarantee ME?

- The label assignment is concurrent
  - Say P_1 gets the label 1, P_2 has not seen it yet and also gets the label 1.
  - In tie-breaking (1,1) < (1,2)
    - Even if threads race, lex-ordering ensures a total order!
- Say P_1 gets in to CS_1, then $(k_1,1) < (k_2,2)$
- For P_2: $(m_2,2) < (m_1,1)$
- The values of k and m need not be same!

```
class Bakery implements Lock {
  boolean[] flag;
  Label[] label;
 public Bakery (int n) {
   flag  = new boolean[n];
   label = new Label[n];
   for (int i = 0; i < n; i++) {
      flag[i] = false; label[i] = 0;
   }
 }
```

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```

Bakery Algorithm

Does it guarantee ME?

- **Lemma 1**: if P_i has label L_i at T_1 and P_j sets its label to L_j at T_2 and $T_2 > T_1$ then
  - $(L_j, j) > (L_i, i)$
- Temporal ordering implies label ordering
  - Because of SC contraints

```
class Bakery implements Lock {
  boolean[] flag;
  Label[] label;
 public Bakery (int n) {
   flag  = new boolean[n];
   label = new Label[n];
   for (int i = 0; i < n; i++) {
      flag[i] = false; label[i] = 0;
   }
 }
```

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```

Bakery Algorithm

Does it guarantee ME?

- Thm1: ME is guaranteed!
  - Proof by contradiction: At T both P_1, P_2 are in the CS
    - Establish total order between P_i, P_j label updates, ie $(L_i, i) < (L_j, j)$ assuming $T_i < T_j$
    - For P_i in CS at T_{ij}: $\neg F_j \vee (L_j, j) > (L_i, i)$
    - For P_j in CS at T_{ji}: $\neg F_i \vee (L_i, i) > (L_j, j)$
    - And $T_{ij} < T \wedge T_{ji} < T$
    - Use the above information to arrive at an inconsistent predicate:
      $(L_j, j) > (L_i, i) \wedge (L_j, j) < (L_i, i)$

```
class Bakery implements Lock {
  boolean[] flag;
  Label[] label;
 public Bakery (int n) {
   flag  = new boolean[n];
   label = new Label[n];
   for (int i = 0; i < n; i++) {
      flag[i] = false; label[i] = 0;
   }
 }
```

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```