

Mutual Exclusion

Companion slides for
The Art of Multiprocessor
Programming
by Maurice Herlihy & Nir Shavit

(Abridged version. Original at <http://booksite.elsevier.com/9780123705914/?ISBN=9780123705914>)

Bakery Algorithm

- Provides First-Come-First-Served
- How?
 - Take a "number"
 - Wait until lower numbers have been served
- Lexicographic order
 - $(a,i) > (b,j)$
 - If $a > b$, or $a = b$ and $i > j$

Bakery Algorithm

```
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;  
        }  
    }  
    ...  
}
```

Bakery Algorithm

```
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;
```

```
        }
```

```
    }
```

```
    ...
```



f	f	t	f	f	t	f	f
0	0	4	0	0	5	0	0

CS

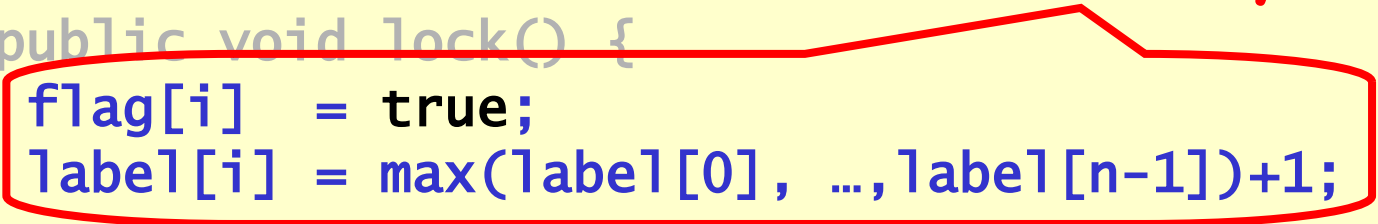
Bakery Algorithm

```
class Bakery implements Lock {  
    ...  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0], ..., label[n-1])+1;  
        while ( $\exists k$  flag[k]  
                && (label[i],i) > (label[k],k));  
    }
```

Bakery Algorithm

```
class Bakery implements Lock {  
    ...  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0], ..., label[n-1])+1;  
        while ( $\exists k$  flag[k]  
                && (label[i], i) > (label[k], k));  
    }  
}
```

Doorway



Bakery Algorithm

```
class Bakery implements Lock {
```

```
...
```

```
public void lock() {
```

```
    flag[i] = true;
```

```
    label[i] = max(label[0], ..., label[n-1])+1;
```

```
    while ( $\exists k$  flag[k]
```

```
           && (label[i], i) > (label[k], k));
```

```
}
```

I'm interested

Bakery Algorithm

Take increasing
label (read labels
in some arbitrary
order)

```
class Bakery implements Lock {  
    ...  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0], ..., label[n-1])+1;  
        while ( $\exists k$  flag[k]  
                && (label[i], i) > (label[k], k));  
    }
```


Bakery Algorithm

```
class Bakery implements Lock {  
    ...  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0], ..., label[n-1])+1;  
        while ( $\exists k$  flag[k]  
                && (label[i], i) > (label[k], k));  
    }
```

Someone is
interested



Bakery Algorithm

```
class Bakery implements Lock {  
    boolean flag[n];  
    int label[n];  
  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0], ..., label[n-1])+1;  
        while ( $\exists k$  flag[k]  
                && (label[i], i) > (label[k], k));  
    }
```

Someone is
interested

With lower (label,i) in
lexicographic order

Bakery Algorithm

```
class Bakery implements Lock {  
    ...  
  
    public void unlock() {  
        flag[i] = false;  
    }  
}
```

Bakery Algorithm

```
class Bakery implements Lock {
```

```
...
```

```
public void unlock() {
```

```
    flag[i] = false;
```

```
}
```

```
}
```

No longer
interested



labels are always increasing

No Deadlock

- There is always one thread with earliest label
- Ties are impossible (why?)

Mutual Exclusion

- Suppose A and B in CS together
- Suppose A has earlier label
- When B entered, it must have seen
 - flag[A] is *false*, or
 - label[A] > label[B]

```
class Bakery implements Lock {  
  
    public void lock() {  
        flag[i] = true;  
        label[i] = max(label[0],  
                        ..., label[n-1])+1;  
  
        while (∃k flag[k]  
                && (label[i], i) >  
                (label[k], k));  
    }  
}
```

Mutual Exclusion

- Labels are strictly increasing so
- B must have seen $\text{flag}[A] == \text{false}$

Mutual Exclusion

- Labels are strictly increasing so
- B must have seen $\text{flag}[A] == \text{false}$
- $\text{Labeling}_B \rightarrow \text{read}_B(\text{flag}[A]) \rightarrow \text{write}_A(\text{flag}[A]) \rightarrow \text{Labeling}_A$

Mutual Exclusion

- Labels are strictly increasing so
- B must have seen $\text{flag}[A] == \text{false}$
- $\text{Labeling}_B \rightarrow \text{read}_B(\text{flag}[A]) \rightarrow \text{write}_A(\text{flag}[A]) \rightarrow \text{Labeling}_A$
- Which contradicts the assumption that A has an earlier label

This work is licensed under a [Creative Commons Attribution-ShareAlike 2.5 License](https://creativecommons.org/licenses/by-sa/2.5/).

- **You are free:**
 - to Share — to copy, distribute and transmit the work
 - to Remix — to adapt the work
- **Under the following conditions:**
 - **Attribution.** You must attribute the work to “The Art of Multiprocessor Programming” (but not in any way that suggests that the authors endorse you or your use of the work).
 - **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.
- **For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to**
 - <http://creativecommons.org/licenses/by-sa/3.0/>.
- **Any of the above conditions can be waived if you get permission from the copyright holder.**
- **Nothing in this license impairs or restricts the author's moral rights.**