ECS150 Discussion Section

Sophie Engle
(February 18/20 2004)
Dining Philosophers Problem
- Solve using mutual exclusion
- Solve using semaphores
- Solve using monitors

Resources:
- Section 2.2 in book (Tanenbaum)
**Situation:**
- Five philosophers at table
- Want to eat spaghetti
- Must have 2 forks to eat
- Only 5 forks at the table
Each philosopher:
- Must grab fork one at a time
- Can act at any time (concurrently with other philosophers)
- Is either thinking, hungry, or eating
Spaghetti Situation 1

- Everyone follows:
  - Grab left fork
  - Grab right fork
  - Eat spaghetti
- If fork not available, wait until available
- Release fork when done eating
- What if all grab left fork at same time?
#define N 5

```c
void philosopher( int i ) {
    while( TRUE ) {
        think();
        take_fork( i );
        take_fork( (i+1) % N ); // left fork
        eat();
        put_fork(i);
        eat();
        put_fork( (i+1) % N ); // right fork
    }
}
```

Spaghetti Situation 1

DEADLOCK!
- One person follows:
  - Grab right fork
  - Grab left fork
  - Eat spaghetti

- Everyone else follows:
  - Grab left fork
  - Grab right fork
  - Eat spaghetti

- Avoids deadlock!
  - Uses hierarchical allocation
  - Must grab lower numbered fork first!
Spaghetti Situation 2

grabs right fork
Spaghetti Situation 2

waits for left fork
grabs left fork
Spaghetti Situation 2

grabs left fork
Spaghetti Situation 2

grabs left fork
Spaghetti Situation 2

DEADLOCK AVOIDED

grabs left fork
Must avoid:
- Deadlock
- Starvation

Some tools:
- Mutual exclusion
- Semaphores
- Monitors
void philosopher( int i ) {
  semaphore mutex = 1;

  while( TRUE ) {
    think();
    down( &mutex );    // enter critical region
    take_fork( i );
    take_fork( (i+1) % N );
    eat();
    put_fork(i);
    put_fork( (i+1) % N );
    up( &mutex );      // exit critical region
  }
}
Semaphores: Example 1

becomes hungry
Semaphore: Example 1

becomes hungry

acquires semaphore
**Semaphore: Example 1**

1. **grabs forks (one at a time)**
2. waits for semaphore
Semaphore: Example 1

1 becomes hungry

2 eats

3 waits for semaphore
Semaphore: Example 1

1. Releas**s** forks

2. acquires semaphore

3. Semaphore

4. Waiting for semaphore

5. Semaphore: acquires semaphore

1. Releasing forks and semaphore
Semaphore: Example 1

- Prevents deadlock
- Only allows one philosopher to eat at a time
- Can improve efficiency?
Semaphore: Example 2

- Each person is either:
  - Thinking
  - Hungry -or-
  - Eating
- Can eat only if neither neighbor is eating
  - Uses semaphores
  - See book p78 for code
Semaphore: Example 2

<table>
<thead>
<tr>
<th>#</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hungry</td>
</tr>
<tr>
<td>2</td>
<td>thinking</td>
</tr>
<tr>
<td>3</td>
<td>thinking</td>
</tr>
<tr>
<td>4</td>
<td>thinking</td>
</tr>
<tr>
<td>5</td>
<td>thinking</td>
</tr>
</tbody>
</table>

becomes hungry
Semaphore: Example 2

- 1: eating
- 2: thinking
- 3: hungry
- 4: thinking
- 5: thinking

- 1 grabs forks, eats
- 3 becomes hungry
Semaphore: Example 2

1. eating
2. hungry
3. eating
4. thinking
5. thinking

grabs forks, eats

becomes hungry
Semaphore: Example 2

<table>
<thead>
<tr>
<th>#</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>eating</td>
</tr>
<tr>
<td>2</td>
<td>hungry</td>
</tr>
<tr>
<td>3</td>
<td>eating</td>
</tr>
<tr>
<td>4</td>
<td>thinking</td>
</tr>
<tr>
<td>5</td>
<td>thinking</td>
</tr>
</tbody>
</table>

waits until neighbors done
cobegin( i:= 0 to N - 1 )
  do true →
      # think
      ...
      # get forks
      dp.getforks( i );

      # eat
      ...
      ...
      # release forks
      dp.relforks( i );
  od
coend
monitor dp
    # N - 1 forks, initialized to 2
    var forks[ 0 : N - 1 ] : int := ( [N] 2 )

    # condition variable (indicates both forks free)
    var both_free[ 0 : N - 1 ] : condition

    # right[i] is philosopher i’s right neighbor
    var right[ 0 : N - 1 ] : int := ( N - 1, 0, 1, ..., N - 2 )

    # left[i] is philosopher i’s left neighbor
    var left[ 0 : N - 1 ] : int := ( 1, ..., N - 1, 0 )

    ...
end

[ discussion section · ecs150 operating systems · winter quarter 2004 ]
Monitor: Example 1

```plaintext
monitor dp
... 

# grabs both forks, or waits until both free
proc getforks( i : int )
  if forks[i] < 2 → wait( both_free[i] ) fi
  forks[ right[i] ]--
  forks[ left[i] ]--
end

... 
end
```
Monitor: Example 1

monitor dp
...

# release both forks, signals if both free
proc relforks( i : int )
  forks[ right[i] ]++
  forks[ left[i] ]++

  if forks[ right[i] ] = 2
      \rightarrow signal( both_free[ right[i] ] ) \fi

  if forks[ left[i] ] = 2
      \rightarrow signal( both_free[ left[i] ] ) \fi
end
end
Monitor: Example 1

- Starvation still possible!
- Rest on chalkboard...