

Synchronization, Dining Philosophers, and Deadlock

The background of the slide features a classical statue, likely the 'Alma Mater' statue at the University of Illinois, rendered in a monochromatic red color. The statue is positioned in the center-right of the frame, with its arms outstretched. The overall image has a textured, slightly grainy appearance, suggesting it might be a scan of a physical document or a low-resolution digital image.

CS 240 - The University of Illinois

Wade Fagen-Ulmschneider

September 23, 2021

Synchronization

A photograph of a crowd gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The word "Synchronization" is written in large white text across the center of the image. The background shows a large group of people, mostly seen from the back, looking towards a central statue. The statue is a woman in a long, flowing dress, standing on a pedestal. The scene is outdoors, with trees and foliage visible in the background. The overall color scheme is dominated by the orange overlay.

Synchronization Technique #1

pthread_mutex_init: Creates a new lock in the “unlocked” state.

pthread_mutex_lock(pthread_mutex_t *mutex):

- When `mutex` is unlocked, change the lock to the “locked” state and advance to the next line of code.
- When `mutex` is locked, this function **blocks** execution until the lock can be acquired.

pthread_mutex_unlock: Moves the lock to the “unlocked” state.

pthread_mutex_destory: Destroys the lock; frees memory.

```
5 pthread_mutex_t lock;
6 int ct = 0;
7
8 void *thread_start(void *ptr) {
9     int countTo = *((int *)ptr);
10
11     int i;
12     for (i = 0; i < countTo; i++) {
13         pthread_mutex_lock(&lock);
14         ct = ct + 1;
15         pthread_mutex_unlock(&lock);
16     }
17
18     return NULL;
19 }
```

Synchronization Technique #2

pthread_cond_init: Create a new conditional variable.

pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex): Performs two different synchronization actions:

-
-

pthread_cond_signal(pthread_cond_t *cond): Unblocks “at least one thread” that is blocked on `cond` (if any threads are blocked; otherwise an effective “NO OP”).

pthread_cond_broadcast(pthread_cond_t *cond): Unblocks ALL threads blocked on `cond`.

pthread_mutex_destory: Destroys the lock; frees memory.

```
11 int things[THINGS_MAX];
12 int things_ct = 0;
13
14 void *producer(void *vptr) {
15     while (1) {
16         pthread_mutex_lock(&lock);
17
18         // Cannot produce until there's space:
19         while (things_ct >= THINGS_MAX) {
20             pthread_cond_wait(&cond, &lock);
21         }
22
23         // Produce a thing:
24         things[things_ct] = rand();
25         printf("Produced [%d]: %d\n", things_ct, things[things_ct]);
26         things_ct++;
27
28         // Signal any waiting consumers:
29         pthread_cond_broadcast(&cond);
30
31         pthread_mutex_unlock(&lock);
32     }
33 }
```


Synchronization Technique #3

sem_init: Creates a new semaphore with a specified “value”.

sem_wait: When the value is greater than zero, decreases the value and continues. Otherwise, **blocks** until the value is non-zero.

sem_post: Increments the value by one.

sem_destroy: Destroys the semaphore; frees memory.

Critical Sections

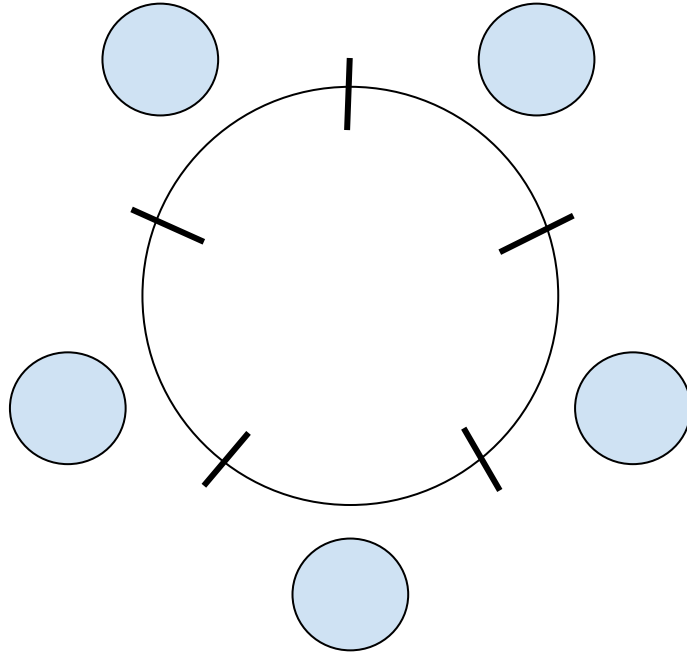


Critical Section

Dining Philosophers

A photograph of a group of people gathered around a statue of a philosopher, overlaid with a red tint. The text "Dining Philosophers" is written in white across the center. The background shows a crowd of people, some looking towards the statue, which is a large, classical-style figure. The scene is set outdoors with trees in the background.

Dining Philosophers

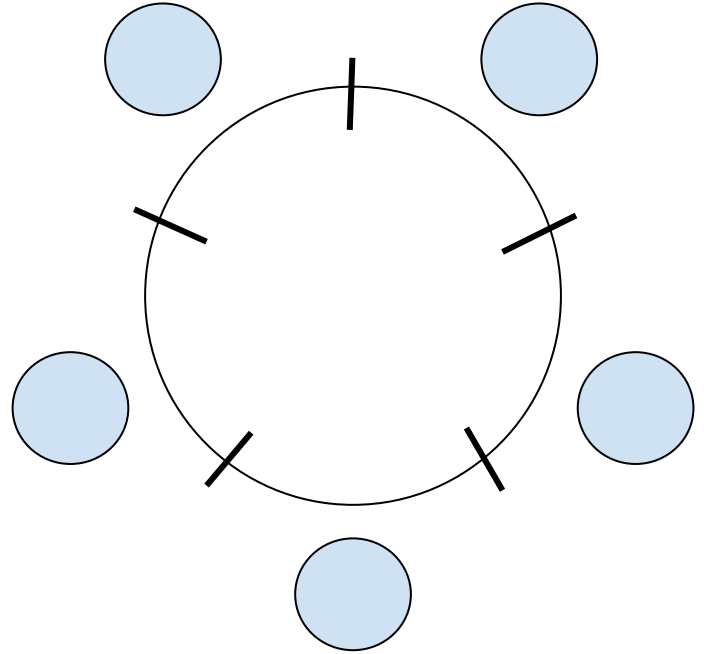


```
16 while (1) {
17     printf("%d is thinking...\n", id);
18
19     // Get left chopstick:
20     printf("%d is reaching for the left chopstick
        (chopstick=%d)...\n", id, left_chopstick_id);
21     pthread_mutex_lock(&locks[left_chopstick_id]);
22     printf("%d has the left chopstick
        (chopstick=%d).\n", id, left_chopstick_id);
23
24     // Get right chopstick:
25     printf("%d is reaching for the right chopstick
        (chopstick=%d)...\n", id, right_chopstick_id);
26     pthread_mutex_lock(&locks[right_chopstick_id]);
27     printf("%d has the right chopstick
        (chopstick=%d).\n", id, right_chopstick_id);
```

```
28
29 // Eat:
30 printf("%d is eating... 🍜🍴\n", id);
31
32 // Release chopsticks:
33 printf("%d is returning their chopsticks
      (chopsticks: %d, %d)... \n", id,
      left_chopstick_id, right_chopstick_id);
34 pthread_mutex_unlock(&locks[right_chopstick_id]);
35 pthread_mutex_unlock(&locks[left_chopstick_id]);
36 }
```

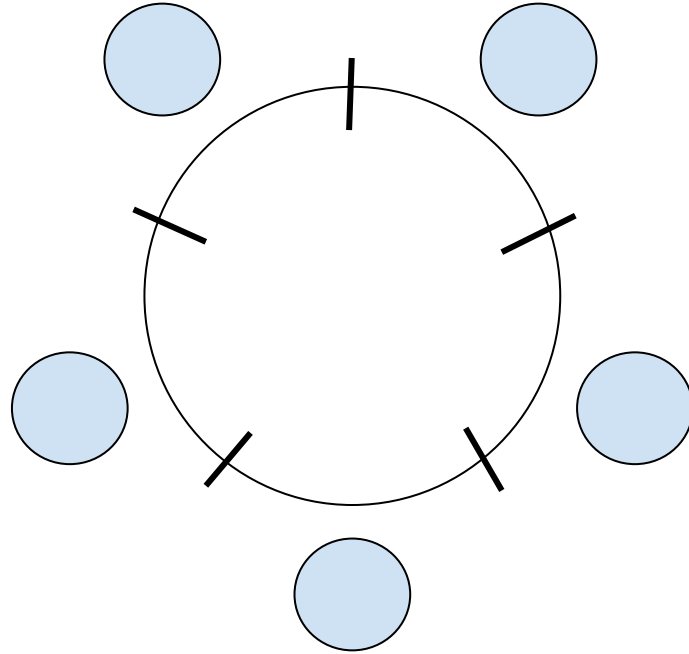


```
while (1) {  
    think();  
    lock_left(&mutex);  
    lock_right(&mutex);  
    eat();  
    release_right(&mutex);  
    release_left(&mutex);  
}
```

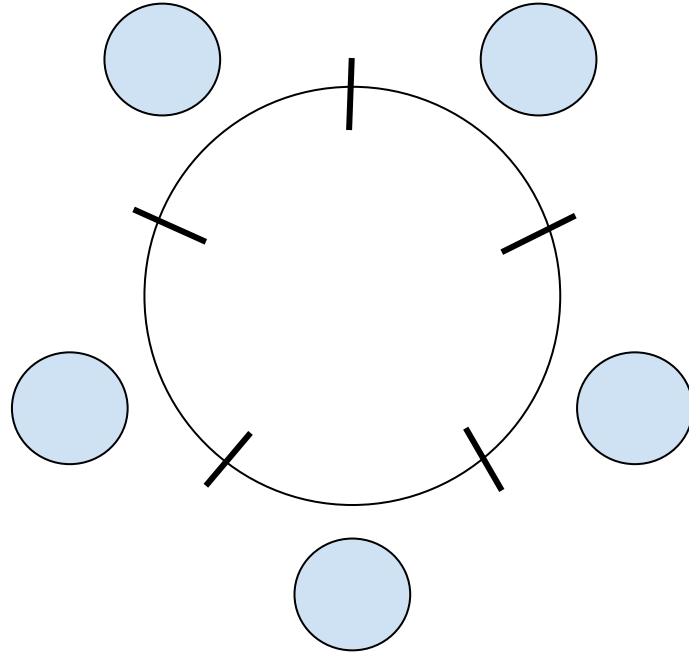


Four Necessary Conditions of Deadlock (“Hoffman Conditions”):

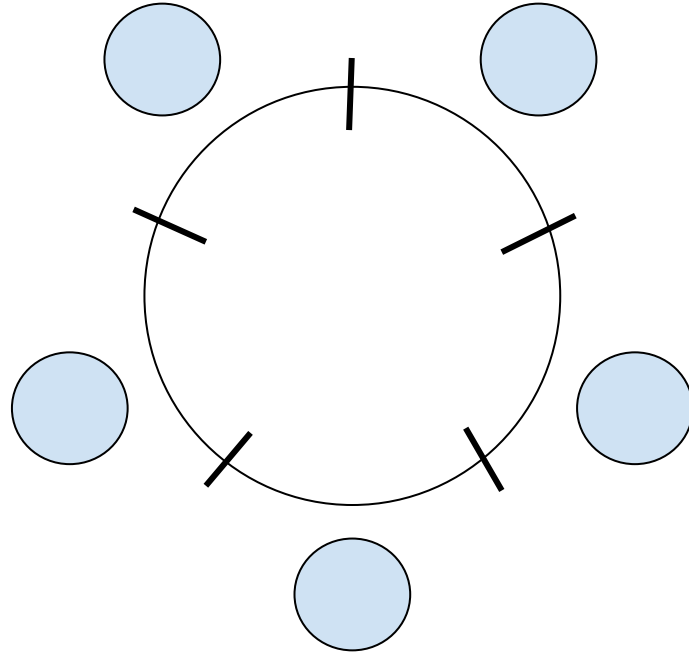
Mutual Exclusion



Circular Wait



Hold and Wait



No Preemption

