CS 240
Computer Systems

#9: Sync, Dining Philosophers, and Deadlock

Sept. 23, 2021 · Wade Fagen-Ulmschneider

Synchronization: Three Techniques

For C-level synchronization, there are three constructs that we have available to help us synchronize access to **critical sections**:

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.

09/count-with-lock.c

```c
pthread_mutex_t lock;
int ct = 0;

void *thread_start(void *ptr) {
  int countTo = *((int *)ptr);

  int i;
  for (i = 0; i < countTo; i++) {
    pthread_mutex_lock(&lock);
    ct = ct + 1;
    pthread_mutex_unlock(&lock);
  }

  return NULL;
}
```

Q: What happens when we run this code now?

...and the performance?

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.

09/producer-consumer.c

```c
int things[THINGS_MAX];
int things_ct = 0;

void *producer(void *vptr) {
  while (1) {
    pthread_mutex_lock(&lock);

    // Cannot produce until there's space:
    while (things_ct >= THINGS_MAX) {
      pthread_cond_wait(&cond, &lock);
    }

    // Produce a thing:
    things[things_ct] = rand();
    printf("Produced [%d]: %d\n", things_ct, things[things_ct]);
    things_ct++;

    // Signal any waiting consumers:
    pthread_cond_broadcast(&cond);

    pthread_mutex_unlock(&lock);
  }
}
```

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.

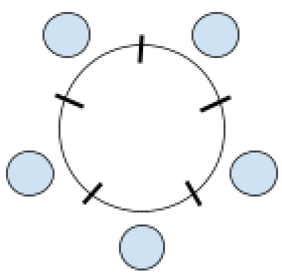
Critical Sections

We know that critical sections require exclusive access to a resource. We also know locking a resource is computationally expensive. However, are there other concerns?

The Dining Philosophers

Imagine five philosophers and five chopsticks at a circular table. Each philosopher has two states: **eating** and **thinking**:

- When a philosopher is thinking, she holds no chopsticks.
- When a philosopher starts the process of eating, she must take the chopstick to her left, then her right, and then begin eating.



Q: Using the strategy described above (take left, take right, then eat), what happens over a long period of time?

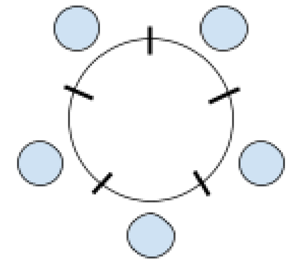
`See Lecture Code: 09/dinning-philosophers.c`

Deadlock:

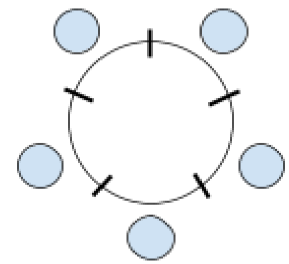
- Definition:

- Four **necessary** conditions of deadlock:

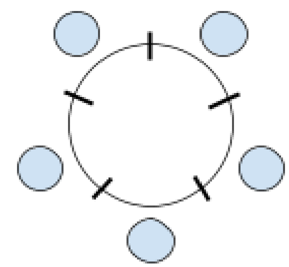
1)



2)



3)



4)