CSCI 4061 Discussion 11

4/9/18

Overview

- Deadlock
  - Definition
  - Conditions
- Livelock
- Non-blocking Synchronization
- Exercise

Deadlock

- Multiple threads mutually exclude each other from resources that they both require.

Required Conditions

- Mutual Exclusion
- Hold-and-wait
- No preemption
- Circular (unbounded) wait

The Dining Philosophers Problem

- A group of N philosophers are eating at a table with N utensils.
- In order to eat, a philosopher must possess 2 utensils simultaneously.
- If every philosopher grabs their left or right utensil first, then a deadlock occurs and they could all starve.

Livelock

- Threads/Processes are ‘alive’ in the sense they are not constantly blocked (removed from ready queue).
- Still, no progress is made due to interference between them.

Livelock Example

- An OS detects deadlocks by having a maximum blocking time for threads.
- When exceeded, the OS wakes the threads (spurious wake-up), and drops their locks.
- If threads blocking each other are awoken simultaneously, the deadlock will resume when each thread simply grabs their same lock back.

Tie-breaking Mechanisms

- To prevent livelock, ‘random backoff’ times are often used.
- These, and other deadlock/livelock breaking ideas are often referred to as ‘tie-breaking mechanisms’.
- Often, some random value is involved.

Non-blocking Synchronization

```
// Attempts to grab lock. Returns 0 if success.
pthread_mutex_trylock(pthread_mutex_t* mutex);
pthread_mutex_timedlock(pthread_mutex_t* mutex, const struct
timespec* timeout);

// Attempts to decrement semaphore. Returns 0 if success.
sem_trywait(sem_t* sem);

// Timed wait for semaphore. Returns 0 if decremented.
sem_timedwait(sem_t* sem, const struct timespec* timeout);
```

Exercise

- The file rec11.c contains an implementation of the dining philosophers problem, which may encounter deadlock.
- Develop a solution to this problem.
- You may only alter the ‘dine’ function.
- Your solution may run up to 2x slower than the one provided.