Topic 12: Deadlock

Reading: Section 6.5

Next reading: Sections 8.1-8.3

- Deadlock is one area where there is a strong theory, but it's almost completely ignored in practice. Reason: solutions are expensive and/or require predicting the future.
    - We will focus on practice.
- Deadlock example: x=1, y=1. Process A does P(x) followed by P(y), while Process B at the same time does P(y) followed by P(x).
    - Example: in one code path a process gets a mutex on its own PCB followed by a mutex on its parent's. In another path it gets its own followed by a child's.
- Deadlock definition: a situation in which each process in a set is waiting for an event that only another process in the set (including itself) can cause.
- Dining philosophers:

```
Philosopher(int i) {
    while(1) {
        Think();
        TakeLeftFork(i);
        TakeRightFork(i);
        Eat();
        PutLeftFork(i);
        PutRightFork(i);
    }
}
```

- It is possible for all philosophers to get their left forks, but wait forever for their right forks.

- Conditions required for deadlock:

  1. Mutual exclusion: resources cannot be shared.

  2. Hold and wait: processes request resources incrementally, and hold on to what they've got.

  3. No preemption: resources cannot be forcibly taken from processes.

  4. Circular wait: circular chain of waiting, in which each process is waiting for a resource held by the next process in the chain.

- *Deadlock cannot occur if any of these conditions does not hold.*

- Why the dining philosophers deadlock:

1. Mutual exclusion: only one philosopher can use a fork at a time.

2. Hold and wait: once you have your left fork, you hold onto it while you wait for your right fork.

3. No preemption: you can't wrestle a fork away from your neighbor.

4. Circular wait: it is possible for each philosopher to wait for the philosopher on his/her right. Note that this could not happen if the table were not circular.

- One possible solution -- eliminate hold and wait. Put down your left fork if you can't get your right fork. Add new routine `TestRightFork` that returns TRUE if the fork was acquired, FALSE otherwise, but does not wait for the fork.

```
Philosopher(int i) {
    while(1) {
        Think();
        successful = FALSE;
        while (successful == FALSE) {
            TakeLeftFork(i);
            // got left fork
            if (TestRightFork(i) == FALSE) {
                // sleep and try again
                PutLeftFork();
                Sleep(10);
            } else {
                // got right fork
                successful == TRUE;
            }
        }
        Eat();
        PutLeftFork(i);
        PutRightFork(i);
    }
}
```

give up progress

- What happens if the philosophers all start at the same time? Starvation (no pun intended) -- no one ever eats, but they are not deadlocked either.
- Could sleep a random amount of time, but that is not guaranteed to work. Ok if you don't mind playing the odds.
- A solution that works: change one (and only one) philosopher to do the following: