CMSC 321: Operating Systems

Lecture 8

More IPC Problems
Dining Philosophers

- Five philosophers at round table with plates of Beef Chow Mein from Lee Ho Fook's in Soho (imagine forks are chopsticks)*
- Each alternates between thinking & eating

- When hungry:
  - Try to acquire chopsticks (in either order)
  - Eat, then return chopsticks
- No deadlock, no starvation!

* I have taken some liberties with the traditional anecdote used to illustrate this problem.
An Obvious Solution

```java
philosopher(int myPos) {
    while (true) {
        think();
        takeFork(myPos);
        takeFork((myPos+1) % N);
        chowDown();
        chowDown();
        lickCleanAndReturnFork(myPos);
        lickCleanAndReturnFork((myPos+1) % N);
    }
}
```

• What is the problem and why?
Attempt to Correct

• Be polite:
  – after taking fork, check if other available
  – if not, return fork
  – problem: potential starvation

• Randomize:
  – wait a random amt of time before trying again
  – in practice, often sufficient
  – insufficient for a fail-safe solution
Try a Semaphore

```java
philosopher(int myPos) {
    while (true) {
        think();
        takeFork(myPos);
        takeFork((myPos + 1) % N);
        chowDown();
        lickCleanAndReturnFork((myPos + 1) % N);
    }
}
```
philosopher(int myPos) {
    while (true) {
        think();
P(mutex);
        takeFork(myPos);
        takeFork(myPos);
        takeFork((myPos+1) % N);
        chowDown();
        lickCleanAndReturnFork((myPos+1) % N);
lickCleanAndReturnFork((myPos+1) % N);
        V(mutex);
    }
}

• Solution works, but is lock-step
• Only one philosopher can eat at a time
  – but there are more than two forks!
A Better Solution

• Use array to track each philosopher’s state:
  – thinking, eating, or hungry

• Can eat only if neither neighbor is eating

• Use array of semaphores so hungry philosopher can block if fork unavailable

• Read and understand the solution on p. 242-244, 248-249
philosopher(int whichPhil) {
    while (true) {
        think();
        takeForks(whichPhil);
        chowDown();
        lickNReturn(whichPhil);
    }
}

takeForks(int whichPhil) {
    P(mutex);
    state[whichPhil] = HUNGRY;
    testAvail(whichPhil); // test if both can be had
    V(mutex);
    P(philSems[whichPhil]); // block if both not gotten
}
Better Solution

```c
int state[N];
Semaphore mutex = 1;
Semaphore
    philSems[N]
    = {0};

int testAvail(int whichPhil) {
    if (state[whichPhil] == HUNGRY &&
        state[TO_LEFT] != EATING &&
        state[TO_RIGHT] != EATING)
    {
        state[whichPhil] = EATING;
        V(philSems[whichPhil]);
    }
}

int takeForks(int whichPhil) {
    P(mutex);
    state[whichPhil] = HUNGRY;
    testAvail(whichPhil);  // test if both can be had
    V(mutex);
    P(philSems[whichPhil]);  // block if both not gotten
    ```
Better Solution

```c
int state[N];
Semaphore mutex = 1;
Semaphore philSems[N] = {0};

philosopher(int whichPhil) {
    while (true) {
        think();
        takeForks(whichPhil);
        chowDown();
        lickNReturn(whichPhil);
    }
}

lickNReturn(whichPhil) {
    P(mutex);
    state[whichPhil] = THINKING;
    testAvail(TO_LEFT);    // what does this do?
    testAvail(TO_RIGHT);   // "    "    "    "
    V(mutex);
}
```
Your Assignment

• Read Chapter 5: Scheduling
  – batch: FCFS, SJF, SRTN
  – interactive: RR, Priority, Multi-level Feedback
  – real-time
Next time: Deadlocks