CS 521: Systems Programming

Concurrency Bugs

Lecture 15

Today's Schedule

- Semaphores
- Concurrency Bugs

Today's Schedule

- **Semaphores**
- Concurrency Bugs

Semaphores [1/2]

- We discussed using condition variables to protect a shared, limited resource
  - Such as whiteboards
- In our setup, we needed to maintain a mutex, a condition variable, and a counter (number of students at the board)
- There is a higher-level abstraction for handling this situation: semaphores

Semaphores [2/2]

- Counting semaphores include the counter logic
- Two functions:
  - P – *proberen* – "to test"
  - V – *vrijgave* – "release"
- Invented by Edsger Dijkstra, a Dutch computer scientist

pthread Semaphores

- In pthreads, we have these functions:
  - `sem_wait`
  - `sem_post`
- Initialize with sem_init:
  - int `sem_init(sem_t *sem, int pshared, unsigned int value);`

Breaking Down P/V Functions

```
P(s):
    s = s - 1
    if (s < 0) , wait
```

```
V(s):
    s = s + 1
    Notify waiting threads
```

Demo: thread-limit.c

Today's Schedule

- Semaphores
- **Concurrency Bugs**

Dining Philosophers Problem

- Five silent philosophers sit around a table
- Each philosopher has two functions:
  - Think
  - Eat
- Five bowls of rice and five chopsticks are placed around the table
- A philosopher must have **two** chopsticks to begin eating

Dining Philosophers: Algorithm

- Think until left chopstick is available
  - Pick it up
- Think until right chopstick is available
  - Pick it up
- Eat until full
- Put the chopsticks down
- Repeat
- Is this algorithm safe?

Problem 1: Deadlock

- What will happen if all the philosophers pick up the chopstick on the left at the same time?
  - Everyone will have one chopstick
  - Everyone will wait
- Deadlock
  - See: `deadlock.c`

How Likely is Deadlock?

- In this situation, deadlock might not happen right away
- The philosophers will eat and think for eternity
  - It’s their job, after all!
- **Eventually**, deadlock will happen

Problem 2: Starvation

- Let's assume the system won't deadlock. We still have another problem!
- Two (or three) of the philosophers might be a bit quicker than the others
  - Always get the chopsticks first
- The other philosophers wait, wait, and wait
  - Never get a chance to eat
- This demonstrates **resource starvation**
  - See `starvation.c`

Problem 3: Livelock

- Let’s assume that we only let a philosopher hold onto a single chopstick for 1 minute
  - After the minute elapses, they have to put it back down
- This will solve the problem, right?
- Not necessarily: it is possible that all the philosophers put down the chopstick at the same time, and then pick them back up the same time
- Livelock: the system keeps moving but makes no progress
  - See `livelock.c`

Solution: A Waiter

- If we can introduce a third party arbiter (Waiter), then we can make sure the philosophers stay alive and get their thinking done
- How is this implemented in code?
  - With a mutex!
- To pick up a chopstick, you have to ask the waiter for permission
  - Only pick up chopsticks if you can take both
- You can put down a chopstick at any time