Thread Priorities

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
Outline

• Thread Priorities
• Program
Thread Priorities

- Threads have priorities and can get set using `setPriority(int)` and can be retrieved with `getPriority()`
- Priorities are given values from 1 to 10
  - `Thread.MIN_PRIORITY` is 1
  - `Thread.NORM_PRIORITY` is 5
    - Default priority for main thread
  - `Thread.MAX_PRIORITY` is 10
- When the JVM decides on the next thread to execute from the ready state, it follows a probabilistic algorithm
  - Higher priority threads will be chosen based on a probabilistic algorithm
    - A thread with a priority of 10 is twice as likely to be chosen as a thread with a priority of 5
- There is still a chance that lower priority threads will execute even when a higher priority thread is ready because the higher priority thread does get time-sliced out of the CPU while it is running
  - Another thread then can be switched into the CPU
Priority Example

- Assume you have three threads
  - Thread 1 has a priority of 5
  - Thread 2 has a priority of 1
  - Thread 3 has a priority of 10

- Assume they are all in the ready state at the same time and the JVM is about to swap one of them into the CPU
  - The JVM will choose a random number between 1 and the sum of the priorities (16 in this case)
public class Test {
    public static void main(String[] args) {
        System.out.println("First line");
        TestThread ta = new TestThread('a');
        TestThread tb = new TestThread('b');
        TestThread tc = new TestThread('c');
        tb.setPriority(Thread.MAX_PRIORITY);
        ta.start();
        tb.start();
        tc.start();
        System.out.println("Last line");
    }
}

class TestThread extends Thread {
    private char c;
    public TestThread(char c) {
        this.c = c;
    }
    public void run() {
        for (int i = 0; i < 20; i++) {
            System.out.print(i + " " + c + " ");
        }
        System.out.println();
    }
}
Starvation

- With certain scheduling algorithms, there may be a higher priority thread that never yields or sleeps, causing a lower priority thread to never execute
  - This is called **starvation** or **contention**

- This can be seen in the infamous Dining Philosopher problem
Dining Philosopher Problem

- Originally proposed by Dijkstra in 1965
- There are five philosophers sitting around a table, each with a plate in front of him
- There is one chopstick placed between each of the plates, so there are only five chopsticks on the table
- Philosophers only do two things – think and eat
  - When they aren’t thinking, they are eating
  - When they aren’t eating, they are thinking
- To eat, a philosopher needs both of the chopsticks surrounding his plate
- With each philosopher operating independently and not communicating, how can you make sure the philosophers don’t starve?
Dining Philosopher Problem
When a philosopher is ready to eat:
- try to pick up left chopstick
  - if left available
    - try to pick up right chopstick
    - if right available
      - eat
    - else if right not available
      - if priority(right) < priority(left)
        - release left chopstick
      - else if priority(right) > priority(left)
        - wait for right chopstick
Outline

• Thread Priorities
• Program
Program

- Create an instance of the Dining Philosopher problem with five threads all sharing five static variables. Implement one of the solutions discussed in class.