Thread Priorities

CSCI 201
Principles of Software Development

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

USC Viterbi
School of Engineering

Outline

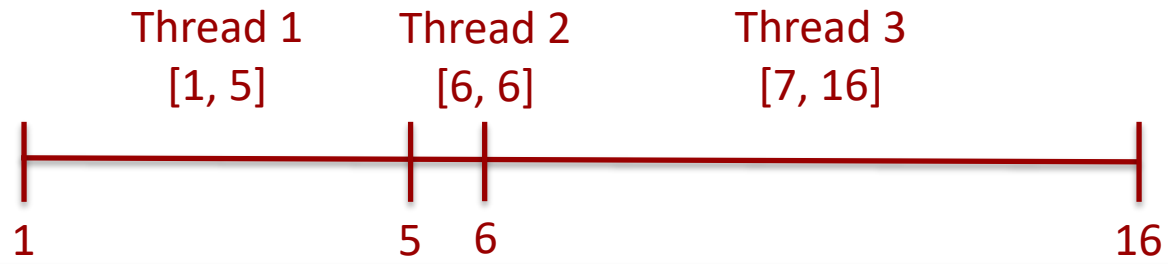
- Thread Priorities
- Program

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)



Priority Example

```
1   public class Test {
2     public static void main(String[] args ) {
3       System.out.println("First line");
4       TestThread ta = new TestThread('a');
5       TestThread tb = new TestThread('b');
6       TestThread tc = new TestThread('c');
7       tb.setPriority(Thread.MAX_PRIORITY);
8       ta.start();
9       tb.start();
10      tc.start();
11      System.out.println("Last line");
12    }
13  }
14
```

```
15  class TestThread extends Thread {
16    private char c;
17    public TestThread(char c) {
18      this.c = c;
19    }
20    public void run() {
21      for (int i=0; i < 20; i++) {
22       System.out.print(i + "" + c + " ");
23      }
24      System.out.println("");
25    }
26  }
```

```
First line
Last line
0b 1b 0a 2b 1a 3b 2a 4b 3a 5b 4a 6b 5a 7b 6a 8b 7a 9b 10b 11b 12b 13b 14b 15b 16b 17b 8a 18b 0c 19b 1c 2c 3c 4c 5c 6c 9a
7c 8c 10a 9c 11a 10c 12a 11c 13a 12c 14a 13c 15a 14c 16a 15c 17a 16c 18a 17c 19a
18c 19c
```

```
First line
Last line
0b 1b 2b 3b 4b 5b 6b 0c 0a 1c 2c 3c 7b 4c 1a 5c 8b 9b 6c 2a 7c 10b 8c 11b 3a 12b 9c 13b 4a 14b 10c 15b 5a 16b 11c 17b 6a 18b 12c 19b 7a
13c 14c 8a 15c 9a 16c 10a 17c 11a 18c 12a 19c
13a 14a 15a 16a 17a 18a 19a
```

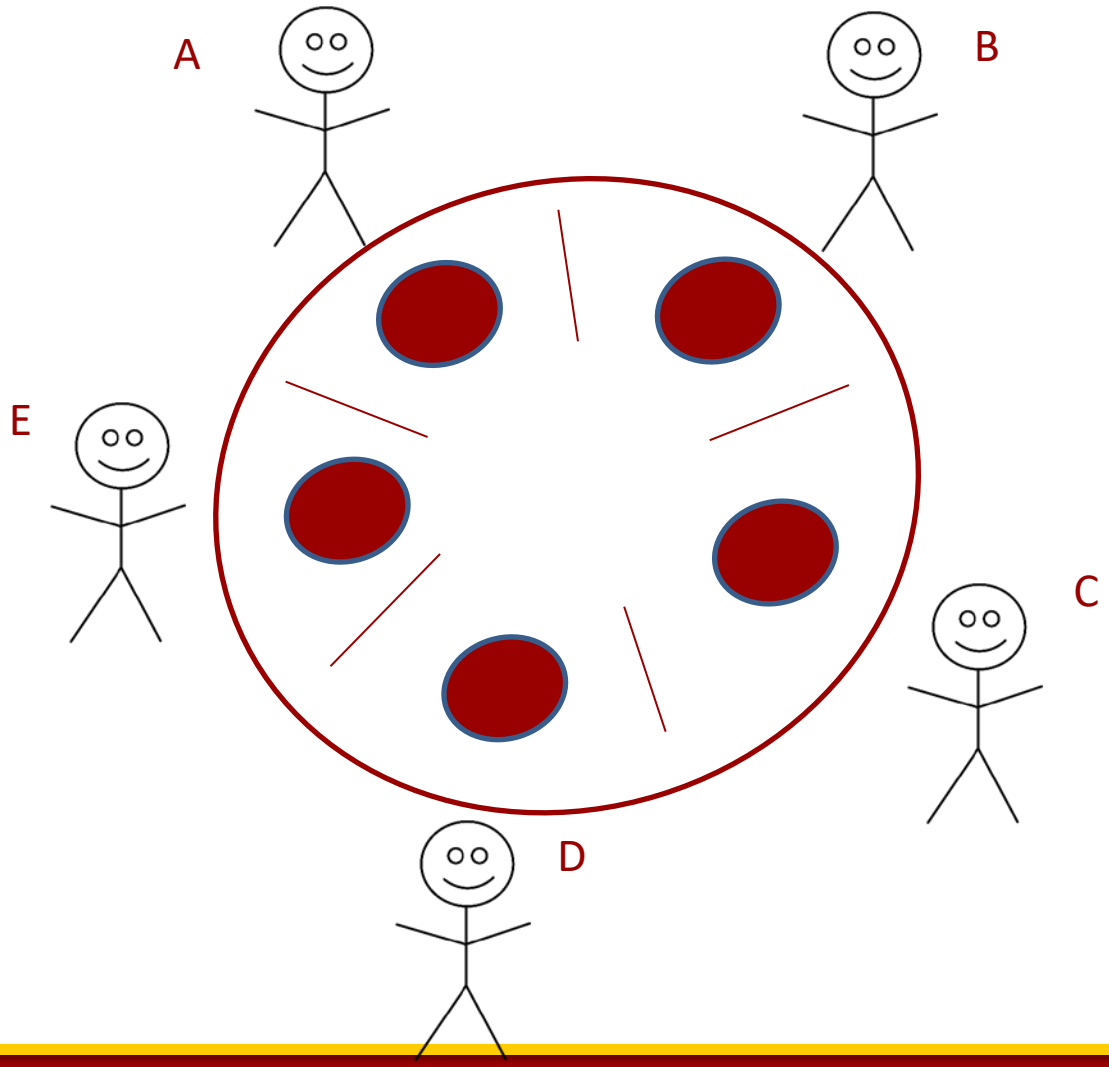
Starvation

- 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

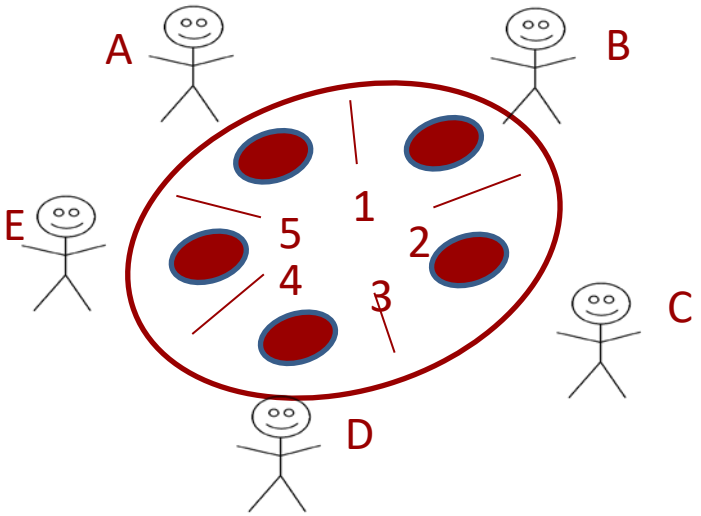
Dining Philosopher Problem

- 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



Dining Philosopher Solution




```
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.