Bank Teller Problem (CSE 422S)

Ken Wong
Washington University

kenw@wustl.edu
www.arl.wustl.edu/~kenw

Bank Teller Problem (1)

- N customer processes and M teller processes, N > M
- Customers
  - Repeatedly arrive at random times to the bank
  - Leave (not enter the lobby) if the lobby is full
    - The lobby can hold at most 20 customers
  - Return a random time later (after service or full lobby)
- Tellers
  - Each serves one customer at a time
  - Teller should signal he/she is ready before customer can come to the teller window
  - There is exactly one line
- Use semaphores to show the synchronization between the customers and tellers

Bank Teller Problem (2)

- Limited resources
  - M tellers
  - Lobby capacity = 20
- Simplifications
  - N ≤ 20 ➔
    - No lobby capacity constraint or wait outside of lobby
  - M = 1

Bank Teller Problem (3)

- Process customer (int i) {
  do forever {
    ... Random delay ... 
    ... Do banking ... 
  }
}

- Process teller (int i) {
  do forever {
    ... Wait for customer ... 
    ... Serve customer ... 
  }
}
Problem 1

- N ≤ 20 and M = 1

```java
Semaphore tRdy = 0;
cRdy = 0;
tDone = 0;

Process customer (int i) {
    do forever {
        ... Random delay ...
        Wait(tRdy);
        Signal(cRdy);
        ... Get service ...
        Wait(tDone);
        ... Leave bank ...
    }
}
```

```java
Process teller (int i) {
    do forever {
        ... Random delay ...
        Signal(tRdy);
        Wait(cRdy);
        ... Serve customer ...
        Signal(tDone);
    }
}
```

Problem 2

- M = 1, wait outside if lobby is full

```java
Semaphore tRdy = 0, cRdy = 0, tDone = 0; // forms queue
capacity = 20;
```

```java
Process customer (int i) {
    do forever {
        ... Random delay ...
        Wait(capacity);
        Wait(tRdy);
        Signal(cRdy);
        ... Serve customer ...
        Signal(tDone);
        Signal(capacity);
        ... Leave bank ...
    }
}
```

```java
Process teller (int i) {
    do forever {
        ... Random delay ...
        Signal(tRdy);
        Wait(cRdy);
        ... Serve customer ...
        Signal(tDone);
    }
}
```

Problem 3 and 4

- M = 1, go away for random time if lobby is full
  - Don't use 'capacity' semaphore because customer can't get out of capacity queue
  - Replace 'capacity' semaphore with a protected counter
- M > 1, go away for random time if lobby is full
  - Need to handle how teller/customer selects customer/teller

```java
int n; // # in lobby
Semaphore nLock = 0; // protect n

Process customer (int i) {
    do forever {
        ... Random delay ...
        until \[ n < 20 \]
        \[ n = n+1; \]
        Wait(tRdy);
        Signal(cRdy);
        ... Get service ...
        Wait(tDone);
        \[ n = n-1; \]
        ... Leave bank ...
    }
}
```

```java
Process teller (int i) {
    do forever {
        ... Random delay ...
        Signal(tRdy);
        Wait(cRdy);
        ... Serve customer ...
        Signal(tDone);
    }
}
```
Problem 4

Semaphore tRdy = 0, cRdy = 0, tDone = 0;
int n; // # in lobby
Semaphore nLock = 0; // protect n

Process customer (int i) {
    do forever {
        do {
            ... Random delay ...
        } until (n < 20)
        [[ n = n+1; ]]
        Wait(tRdy);
        Signal(cRdy);
        ... Get service ...
        Wait(tDone);
        [[ n = n-1; ]]
        ... Leave bank ...
    }
}

Process teller (int i) {
    do forever {
        Signal(tRdy);
        Wait(cRdy);
        ... Serve customer ...
        Signal(tDone);
    }
}

[[ ... ]] means:
Wait(nLock); ... Signal(nLock);

The End