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)

```
N = 1
M = . . .
queue
```

customers
capacity = 20
tellers

Wait random
time

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 ...
    }
}
```

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

- Ken Wong, 10/25/2005
Problem 1

- \(N \leq 20\) and \(M = 1\)

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 ...
    }
}

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

Problem 2

- \(M = 1, N > 20\) (wait outside if lobby is full)

Semaphore \(tRdy = 0, cRdy = 0, tDone = 0,\)
\(capacity = 20;\) // forms queue

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

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

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

Semaphore \(tRdy = 0, cRdy = 0, tDone = 0;\)
\(n = 0;\) // # in lobby
Semaphore \(nLock = 1;\) // protect \(n\)

Process customer (int i) {
    do forever {
        \(n\); // is in lobby
        do {
            ... Random delay ...
        } until (isRoom);
        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);
        ... Wander around ...
    }
}

Problem 3

- \(M = 1, N > 20\) (wait outside if lobby is full)

Semaphore \(tRdy = 0, cRdy = 0, tDone = 0;\)
\(n = 0;\) // # in lobby
Semaphore \(nLock = 1;\) // protect \(n\)

Process customer (int i) {
    do forever {
        \(n\); // is in lobby
        do {
            ... Random delay ...
        } until (isRoom);
        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);
        ... Wander around ...
    }
}

[[ ... ]] means:
- Wait(nLock); ... Signal(nLock);
Problem 4 (1)

Semaphore tRdy[M] = 0, cRdy[N] = 0, tDone[M] = 0;
int n = 0; // # in lobby
Semaphore nLock = 1; // protect n

Process customer (int i) {
  do forever {
    ... dcl and do from Prob 3 ...
    Wait(tRdy[i]);
    Signal(cRdy[i]);
    ... Get service ...
    Wait(tDone[i]);
    [n = n-1; ]
    ... Leave bank ...
  }
}

Problem 4 (2)

Semaphore next = 0, cDone[M] = 0;
int tRdy[M] = { 0, ... , 0 };
int cRdy[M] = { 0, ... , 0 };
int n = 0; // # in lobby
Semaphore nLock = 1; // protect n

Process customer (int i) {
  do forever {
    int myteller;
    ... declarations and do ...
    Wait(next); // ???
    myteller = get_teller( );
    cRdy[myteller] = 1;
    ... Serve customer ...cRdy[myteller] = 0;
    Wait(cDone[myteller]);  // ???
    ... Wander around ...
  }
}

Problem 4 (3)

Semaphore tLock[M] = { 1, ... , 1};
int get_teller (void) { // atomically get teller
  for (int i=0; i<M; i++) {
    Wait(tLock[i]);
    if (tRdy[i]) {
      tRdy[i] = 0; // ???
      Signal(tLock[i]);
      return i;
    }
    Signal(tLock[i]);
  }
}