CS 571 Materials

February 19, 2002

4:30pm - 7:10pm

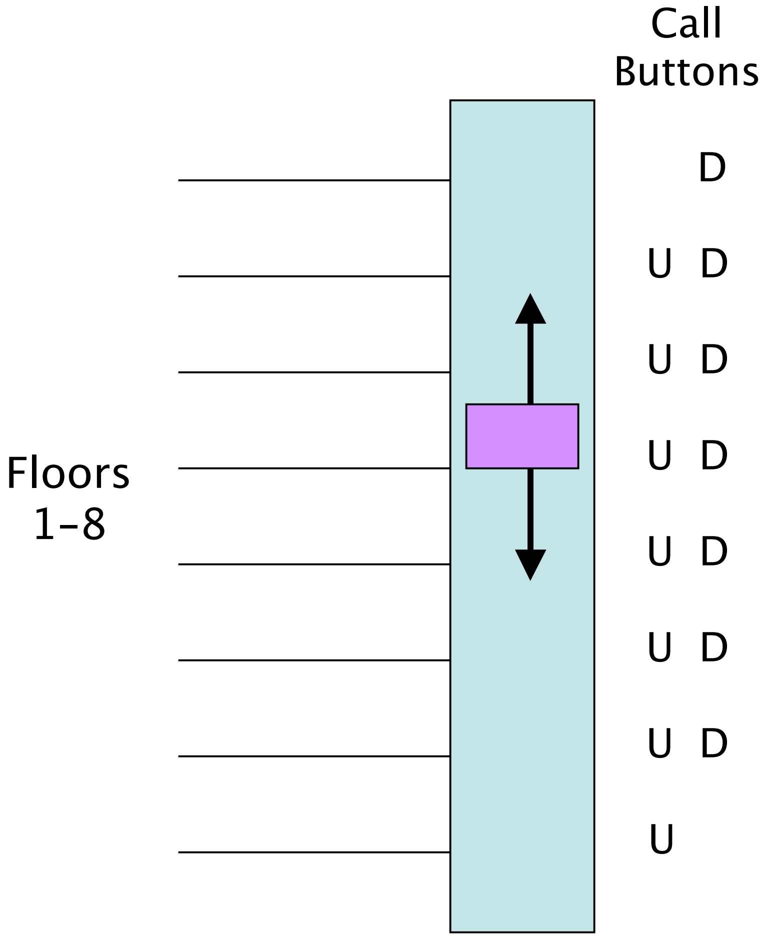
Peter J. Denning

AGENDA

- Q&A
- Review of A2 and P1
- Virtual Machines
- Info Objects
- Handles and Directories

Review of A2

- Elevator controller
- Passenger threads
- Car (elevator) thread
- Monitor to synchronize



System state

U	D	S
	2	2
		1
	1	
2	1	
		1

request matrix

floor, direction

car

Passenger:

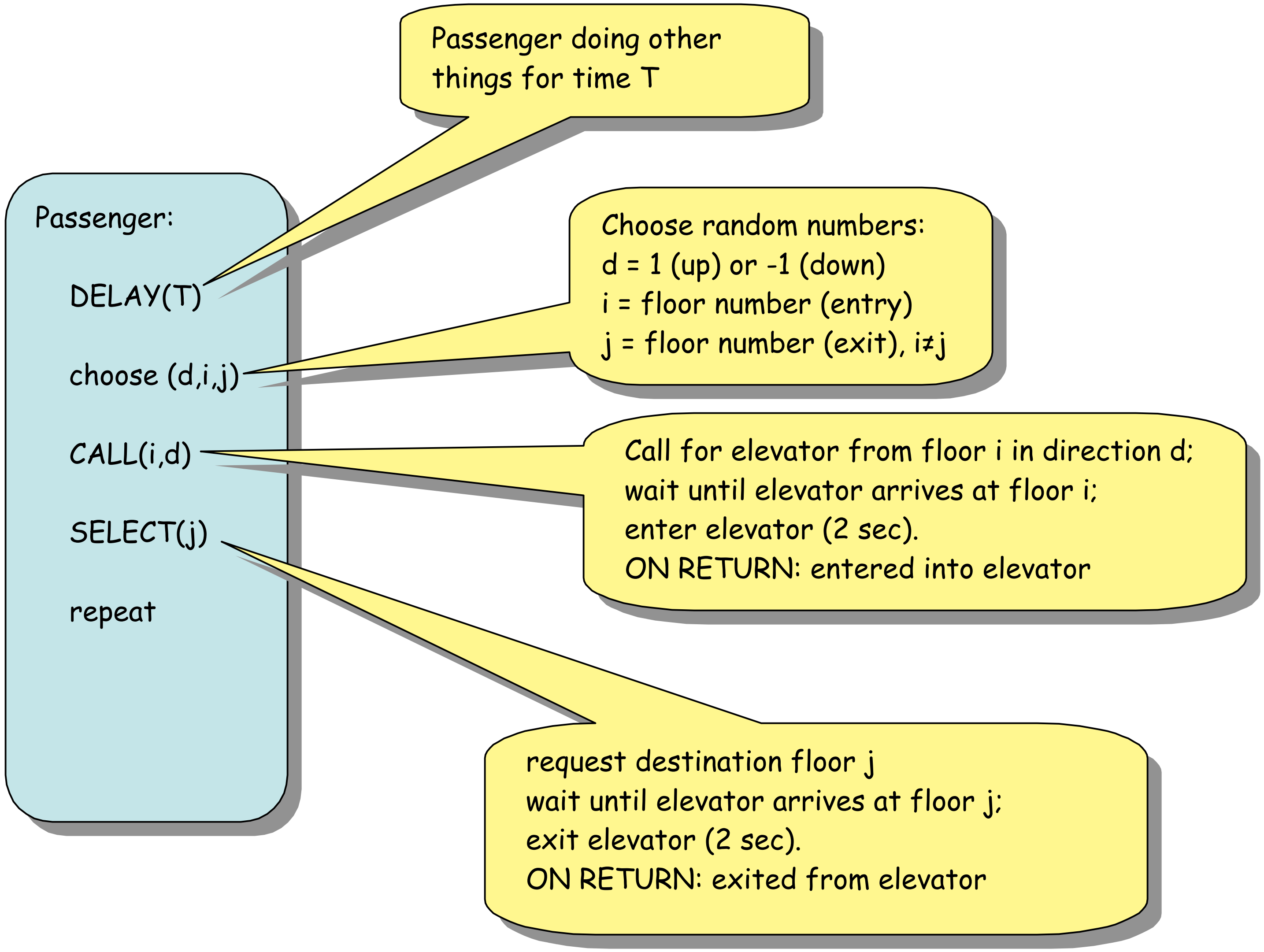
DELAY(T)

choose (d,i,j)

CALL(i,d)

SELECT(j)

repeat


Passenger:
DELAY(T)
choose (d,i,j)
CALL(i,d)
SELECT(j)
repeat
Passenger doing other things for time T
Choose random numbers:
d = 1 (up) or -1 (down)
i = floor number (entry)
j = floor number (exit), i≠j
Call for elevator from floor i in direction d;
wait until elevator arrives at floor i;
enter elevator (2 sec).
ON RETURN: entered into elevator
request destination floor j
wait until elevator arrives at floor j;
exit elevator (2 sec).
ON RETURN: exited from elevator

CONDITION VARIABLES:

timetoEnter[i,d] -- true when passenger on floor i requesting direction d can now enter the elevator car, which has arrived and opened its door.

timetoExit[j] -- true when elevator has arrived at floor j and has now opened door to allow passengers to exit.

timetoMove -- true when elevator has a request to move to another floor.

selectionsMade -- true when current group of new passengers have all made their floor selections.

STATE VARIABLES:

U[1..8]:
counts of number of up requests
waiting at floor i

D[1..8]:
counts of number of down requests
waiting at floor i

S[1..8]:
counts of number of boarded passengers
requesting exit at floor j

floor: current floor of elevator car

dir: current direction of elevator car
+1 = up, -1 = down, 0 = stopped

sel: count of how many recently admitted
passengers have not made selections

MONITOR FUNCTIONS (for passengers):

```
CALL(i,d):
    if d=1 then U[i]++ else D[i]++
    timetoMove.signal
    timetoEnter[i,d].wait
    DELAY(2)
    return

SELECT(j):
    S[j]++
    sel--; if sel=0 then selectionsMade.signal
    timetoMove.signal
    timetoExit[j].wait
    DELAY(2)
    return
```

MONITOR FUNCTIONS (for elevator car):

dir = CHECKFLOOR:
  release passengers wanting to exit at current floor
  determine new value for direction (dir)
    continue (dir unchanged)
    reversed (dir = -dir)
    stop (dir = 0)
  admit new passengers waiting at current floor in new direction
  return dir

bool = RU:
  (Boolean) true if there are requests above current floor or
  up requests at current floor

bool = RD:
  (Boolean) true if there are requests below current floor or
  down requests at current floor

```
CHECKFLOOR:
  release exiting passengers
  if (dir=1 & RU) then {admit waiting up requests}
  else if (dir=1 & RD) then {admit waiting down requests; dir=-1}
  else if (dir=-1 & RD) then {admit waiting down requests}
  else if (dir=-1 & RU) then {admit waiting up requests; dir=1}
  else dir=0
  selectionsMade.wait
  return dir
```

```
CHECKFLOOR:

  release exiting passengers

  if (dir=1 & RU) then {admit waiting up requests}

  else if (dir=1 & RD) then {admit waiting down requests; dir=-1}

  else if (dir=-1 & RD) then {admit waiting down requests}

  else if (dir=-1 & RU) then {admit waiting up requests; dir=1}

  else dir=0

  selectionsMade.wait

  return dir
```

```
while S[floor]>0 do {
   timetoExit[floor].signal
   S[floor]--
   }
```

```
sel=0
while U[floor]>0 do {
   timetoEnter[floor,1].signal
   U[floor]--
   sel++
   }
```

```
sel=0
while D[floor]>0 do {
   timetoEnter[floor,-1].signal
   D[floor]--
   sel++
   }
```

```
elevator:
  timetoMove.wait
  dir = CHECKFLOOR
  if dir≠0 then DELAY(15)
  floor = floor+dir
  repeat
```

P1

- Group project
- Objective: implement in Java a simulation of threads (representing people) using the elevator controller of A2. Simulate elevator use with different usage scenarios.
  - Experience in multi-threaded programming
  - Prepare engineering report on your approach, findings, and conclusions.

Engineering Report Components

- Statement of the problem, approach to solution, and main claims of the report
- Overview of architecture investigated as a solution to this problem (includes diagrams, data flows, data structure, algorithm sketches)
- Overview of the experiments used to test the architecture
- Results of the individual experiments (including graphs and plots)
- Findings and conclusions
- Appendices: simulator source code; raw data outputs

Data Collection

- Insert statements to gather data at key event points
- Use these event records to calculate samples of the metric of interest.
- Get a distribution and averages of the samples.

Data Collection Example

- Average time for passenger to travel on elevator (from moment of call to exit)
- Passenger identifier pid
- Insert "data(arrival,time,pid)" before CALL(i,d) -- records time in arrival[pid]
- Insert "data(departure,time,pid)" after SELECT(j) -- computes sample = time - arrival[pid]
- Aggregates
  - Total of samples
  - Count of number of samples
- Compute average = Total/Count

Data Collection Example

- Note that one "sample" is actually measured in customer-seconds.
- The "Total" is total number of customer-seconds accumulated by waiting customers.
- The "Count" C is total number of customers.
- The "Observation Period" T is the total time to track a given number of customers through the system.
- Then "Average Waiting Time" = Total/C
- And "Average Queue Length" = Total/T