Chair for Network Architectures and Services—Prof. Carle
Department of Computer Science
TU München

Übungsblatt 2

Some of today's slides/figures borrowed from:

James Kurose, Keith W. Ross
Oliver Rose
Averill Law, David Kelton

What's inside a DES? (1/2: data)

- ***Simulated time:*** internal (to simulation program) variable that keeps track of simulated time
  - May progress in huge jumps (e.g., 1ms, then 20s, then 2ms,…)
  - *Not* related to real time or CPU time in any way!
- ***System state:*** variables maintained by simulation program define system state, e.g.: number of packets in queue, current routing table of a router, TCP timeout timers, …
- ***Events:*** points in time when system changes state
  - Each event has an associate *event time*
    - e.g., arrival of packet at a router, departure from the router
    - precisely at these points in time, the simulation must take action (i.e., change state and maybe come up with new future events)
  - Model for time between events (probabilistic) caused by external environment
- ***Event list:*** dynamic list of events (→later slides)
- ***Statistical counters:*** used for observing the system

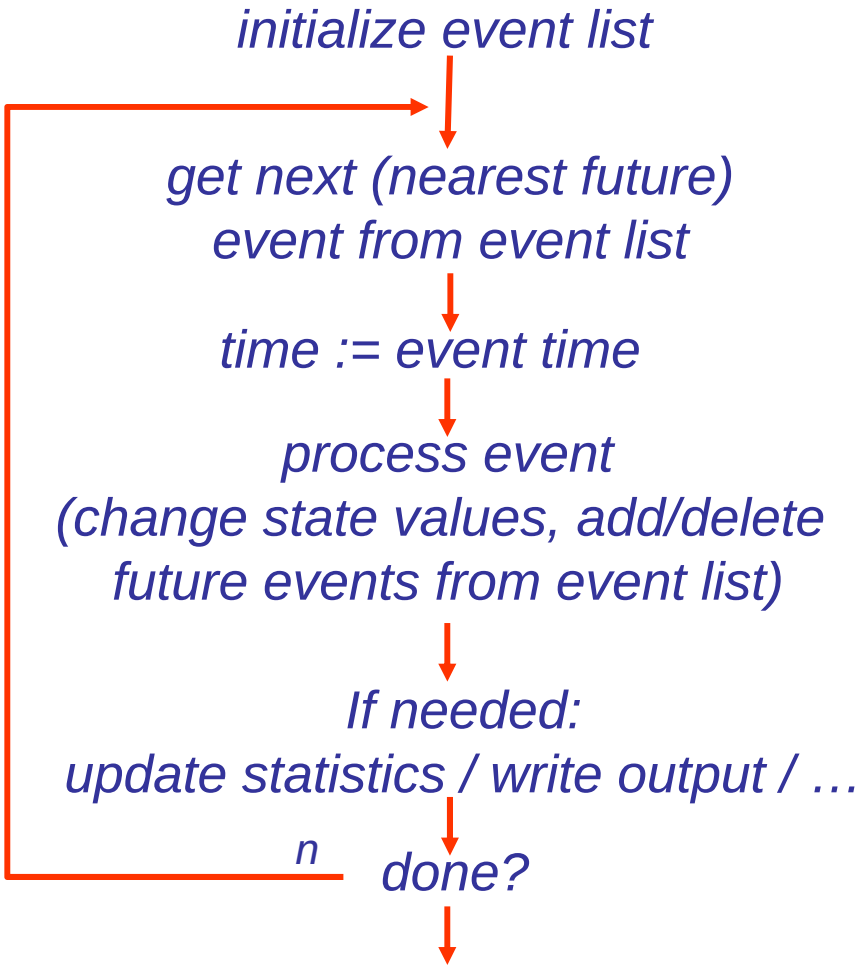
What's inside a DES? (2/2: program code)

- ***Timing routine:***
  - determines the next event and
  - moves the simulation clock to the next event time
- ***Event routine:*** "process the event", i.e., change the system state when an event happens (one subroutine per event *type*)
- ***[P]RNG library routines:*** generate random numbers
- ***Report generators:*** compute performance parameters from statistical counters and generate a report. Runs at simulation end, at interesting events, and/or or at specific pseudo-events
- ***Main program:***

```
while(simulation_time < end_time) {
      next_event = timing_routine();
      next_event.process();
}
```

Discrete event simulator: What's inside? (2/2)

Control view:



Event list: Data structure

- Must be sorted by event time
- Operations:
  - `insert_event( )`: arbitrary time
  - `get_next_event( )`: newest time
- What kind of data structure to use?
- Answer: **Priority queue.**
- Algorithms for this data structure (selection):
  - Array or linked list which we keep sorted? — *bad idea!*
  - Binary heap
  - van Emde Boas tree (vEB tree)
  - Binomial heap
  - …actually, all kinds of search trees that allow efficient execution of `insert( )` and `extract_min( )`

`≡ extract_min( )`

Class Event

Variables and methods:

- `eventTime, getEventTime()`
  - What data type?
  - Access rights?
- `process()`
  - An abstract method
  - Makes the entire class abstract
- `compareTo()`
  - Be able to sort Event objects by their time
  - Warning: consistency with `.equals()` is important

Class `EventChain`

- Represents our event queue
- Variables and methods:
  - Some storage data structure (list? tree? …)
  - `getNextEvent()` – look up `Event` object with lowest event time, remove it from storage, and return it

Class `Simulator`

- Main class
- Read simulation end time from command line parameter (or from a configuration file)
- Schedule a `SimulationTerminationEvent`
- Start the simulation: schedule an event that kicks off things, e.g., a customer arrival
- Run the main loop (cf. previous slides), i.e., pop events from event queue and execute them

Class `SimulationTerminationEvent`

- `process()` will terminate the entire simulator and write some final statistical reports
- N.B. Two variants for ending a simulation:

1.
```
while(simulationTime < endTime) {
        … process events …
  }
```

2.
```
while(true) {
        … process events;
          termination event will stop …
  }
```

Class `CustomerArrival`

- Simulates the event that a new customer enters the system
- Distinguish:
  - Customer can be served immediately (service unit unoccupied)
  - Customer needs to be queued (service unit currently occupied)
- Regardless of above distinction, schedule the next `CustomerArrival` event:
  - `nextEventTime = now()` + random number for customer interarrival time
  - Create new object (or bad hack: re-use current one…) and insert into event queue

Class `ServiceDeparture`

- Simulates the event that the service unit has completed a job
- Distinguish:
  - Queue is empty: nothing to do
  - Queue non-empty:
    - Pop next job
    - Schedule new ServiceDeparture event, `nextEventTime = now()` + random number for service time

Further classes

- Optional: class representing the customer queue
  - Waiting jobs, FIFO
  - Statistics (e.g., queue length)
  - `isEmpty()`
- Optional: class representing the service unit
  - Statistics (e.g., utilisation etc.)
  - `isOccupied()`
- A class representing individual jobs?
  - Question of design / personal taste
  - Overhead in our scenario, but might be helpful for gathering further statistics