Theory

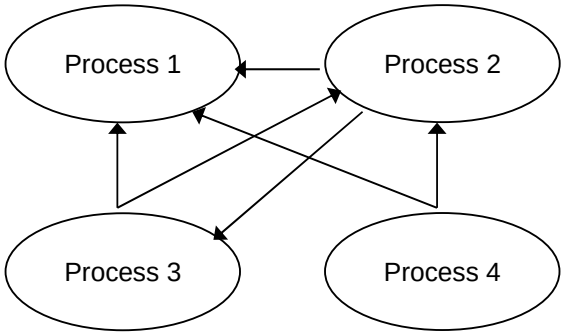
- Introduction to simulation
- Flow of simulation -- generating process realizations
  - Principles of discrete event simulation
  - Example: M/M/1 queue
  - Generic components of a simulation program
- Random number generation from given distribution
- Collection and analysis of simulation data
- Variance reduction techniques

Generating process realisations

- Assume that the system under consideration has been modelled as a stochastic process
- The next task is to generate process realisations (i.e. simulation in a narrow sense). It consists of two subtasks:
  - all random variables involved in the process have to be drawn a value (usually a real number) randomly from the respective distributions (taking into account the dependencies between the variables, i.e. using their joint distributions)
  - using these values one constructs a realisation of the process, i.e. the development of the process in time
- In a simulation these two tasks are not performed consecutively but they are **interleaved**:
  - at an instant of the simulation one draws the value of some random variable; this value is then used (together with previously drawn values) to construct the development of the system over a short interval of time from the current time onward until the value of a new random variable has to be drawn
  - Generation of the values of the random variables is based on the use of so called **(pseudo) random numbers**
- The generation of the process realisation is usually done in an **event driven** fashion (discrete event simulation)

Event driven simulation (1)

- When implementing a discrete event simulation model, the following must be identified:
  - What are the variables of interest from the point of view of the system state and observations?
  - Which are the important **events**?
  - Which are the event triggers and their timings?
- In practise, the system model consists of multiple parts (processes)
  - The functionality of each part can often be represented as a state machine
  - The different parts communicate with each other via events



Event driven simulation (2)

- Idea: simulation proceeds **from one event to the next one**
  - if nothing happens within an interval, one can skip over the interval
- An event corresponds (usually) to a change in the state of the system
  - for instance, in a simple traffic theoretical model events include at least arrivals and departures of customers into and from the system
  - additionally, the end of the process generation is an event
  - also the collection of data may generate some additional “events”
- An event is characterised by two parameters:
  - the time of the event (i.e. *when* the event is to be handled) and
  - the type of the event (i.e. *how* the event is to be handled)
- The events are usually arranged as an **event list** in the order of the event times; at the head of the list is the event with smallest time
  - The list is handled from event to event, at the same time generating new events as may be needed and adding them in the list at the appropriate location. When an event has been handled it is removed from the list.
  - The simulation clock tells the current time, i.e. the time of the event being handled
  - The time advances with discrete increments when moving to the next event.

Event driven simulation (3)

- 1. Initialisation
  - set the simulation clock to zero
  - set the system in a chosen initial state
  - generate the next event of each event type (as far as possible) and add these events in the event list
- 2. Time advancement
  - set the simulation time to the time of the next event (at the head of the event list)
- 3. Event handling
  - handle the event (generating new events which may be triggered by the event and add them in the event list to the location corresponding their times) and update the state of the system
  - remove the handled event from the list
- 4. Stopping condition
  - if the stopping condition is satisfied, stop the generation of the process realisation; otherwise return to point 2

Theory

- Introduction to simulation
- Flow of simulation -- generating process realizations
  - Principles of discrete event simulation
  - Example: M/M/1 queue
  - Generic components of a simulation program
- Random number generation from given distribution
- Collection and analysis of simulation data
- Variance reduction techniques

Example: simulation of M/M/1- FIFO queue

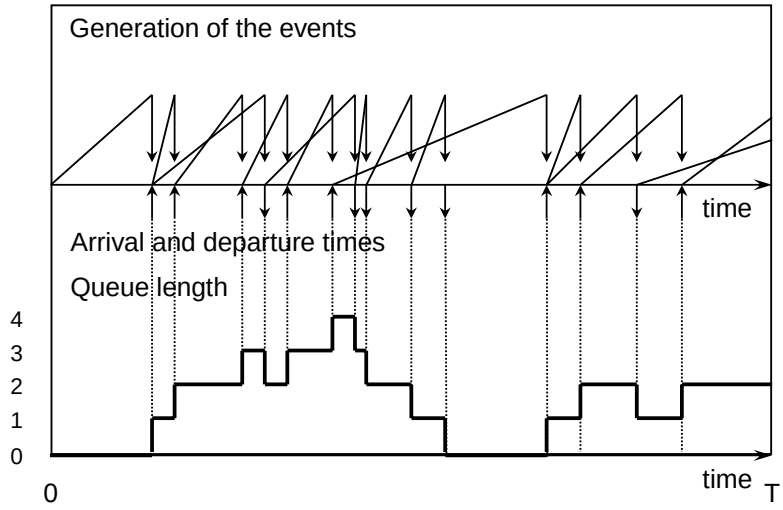
- M/M/1-FIFO queue
  - Customers arrive according to a Poisson process
  - The service times of the customers obey an exponential distribution
  - 1 server
  - Infinite number of waiting places
  - Arriving customers are served in order of arrivals (FIFO)

- Interesting quantities:
  - number of waiting customers
  - waiting time of a customer
  - number of customers in system (including the one in the service)
  - the total time of a customer in the system (sojourn time)

Example: simulation of M/M/1- FIFO queue

- The task is to simulate the development of the queue length process in the M/M/1 queue from time 0 to time T assuming the system is empty at time 0
- The state of the system at time t is defined by the queue length X_t.
- Events: arrivals and departures of customers and the end of the simulation
- Initialisation:
  - set $X_0 = 0$
  - draw the arrival time of the first customer from the distribution Exp(λ)
- Event handling upon the arrival of a new customer (at time t)
  - the system state, i.e. the queue length is incremented by one : $X_t = X_t + 1$
  - if the system is empty upon the arrival, generate the departure time of the customer, t + S, where S is the service time of the customer (drawn from Exp(μ) distribution)
  - generate the arrival instant of the next customer, t + I, where I is the interarrival time (drawn from Exp(λ) distribution)
- Event handling upon the departure of a customer (at time t)
  - the system state, i.e. the queue length is decremented by one: $X_t = X_t - 1$
  - if there are customers left in the system, generate the departure time of the customer to be served next, t + S, where S is the service time of the customer (Exp(μ) distribution)
- Stopping condition: t > T

Queue length in an M/M/1-FIFO queue



Theory

- Introduction to simulation
- Flow of simulation -- generating process realizations
  - Principles of discrete event simulation
  - Example: M/M/1 queue
  - Generic components of a simulation program
- Random number generation from given distribution
- Collection and analysis of simulation data
- Variance reduction techniques

The components of a simulation program

1. Event scheduler (event list manager)
   - maintains a linked list of the future events
   - scheduler may modify the event list, e.g.
     - schedules an event e to occur at time t
     - cancels a previously scheduled event e (removes from the list)
     - holds an event e by a time interval Δt
     - puts event e on hold (until another event reschedules it)
     - schedules an event being held
   - constitutes the core of a simulation program, the most frequently executed part of the program
   - is always executed before an event
   - the event scheduler may be called many times during event handling

The components of a simulation program (continued)

2. Simulation clock and time advance mechanism
   - every simulation program has a global variable representing the current time in the system
   - time advance mechanism may be based on two different approaches

     a) advance by constant increments
     - time is divided in small increments
     - time is advanced by one increment at a time
     - all the events whose scheduled time is within the interval are handled

     b) event based advance
     - time is set to point to the next event
   - the mechanism b) is more general and dynamic
     - one avoids the problems caused by several events occurring within the same interval
     - also one avoids advancing time without anything happening (which may occur frequently in the method a) if the increment is chosen to be very short in order to avoid simultaneous events)

The components of a simulation program (continued)

3. The state variables of the system
   - global variables which define the state of the system
   - for instance, the number of customers in system
     - differs from local variables such as the service time of a customer, which can be saved in a data structure presenting a customer

4. Event handlers
   - events of each type are handled by specific routines
   - handling routines update the values of the state variables and schedule new events (call the event list handler)
   - e.g. event handlers for customer arrivals and customer departures

5. Input routines
   - routines for inputting the values of different parameters
   - asks the user for the values
   - ranges and increments when exploring the effect of some parameter

The components of a simulation program (continued)

6. Report generator
   - collects data on the interesting variables during the course of the simulation
   - makes statistical analysis of data
   - outputs the results in a desired format

7. Initialization routines
   - sets the initial values of the state variables and the seeds for the random number streams
     - a separate routine for initializing the system
     - a separate routine for initializing the system in the beginning of repeated simulations using the same parameters (resetting the counters etc)

8. Trace-routine
   - outputs intermediate values during the simulation
   - facilitates debugging the program
   - on/off switch: can be switched off for the production runs

The components of a simulation program (continued)

9. Dynamic memory management
   - the system ”lives” in the course of simulation
   - new objects arrive and depart
   - when using C/C++ memory management is done by the user
   - when using special languages (or tools) the programmer does not need to take care of such things (not either with many of the general purpose languages)

10. Main program
    - binds different parts of the program together
    - calls input routines and initialization
    - starts the simulation loop
    - finally calls the report generator