Simulation

Simulation is experiments with a model of a system

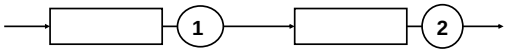
- Event-scheduling method
- Process-interaction method

Event scheduling approach

What is needed:

- A state description
- Events
- Rules telling what will happen when an event occurs
- Parameters (statistical data etc.)

A more complicated example



Assume the following

•Time between arrivals is has mean 0.1
•Service time in queue 1 is constant and equal to 0.1
•Service time in queue 2 is constant and equal to 4
•There is place for at most 4 customers in queue 1
•There is place for at most 2 customers in queue 2

The problem

We want to find

- The mean number of customers in queuing system 1 and 2
- The probability that a customer is rejected in when it arrives to queuing system 1

It does not exist any analytical solutions to this problem!

State description

N1 = number of customers in queuing system 1

N2 = number of customers in queuing system 2

Measuring variables:

NoOfArrivals (is just what you think!)

NoRejected (is just what you think!)

This is not state variables in a strict sense but they also have to be updated at certain events!

Events needed

- ArrivalTo1
- DepartureFrom1
- DepartureFrom2
- Measurement

Rule for ArrivalTo1

```
Procedure RuleArrivalTo1;
Begin
   NoOfArrivals := NoOfArrivals + 1;
   If N1 < 4 then
      N1 := N1 + 1
   else
      NoRejected := NoRejected + 1;
   If N1 = 1 then
      InsertEvent(DepartureFrom1, 0.1);
   InsertEvent(ArrivalTo1, Exp(0.1));
end;
```

Rule for DepartureFrom1

```
Procedure RuleDepartureFrom1;
Begin
   N1 := N1 - 1;
   If N2 < 2 then
      N2 := N2 + 1;
   If N2 = 1 then
      InsertEvent(DepartureFrom2, Exp(4));
   If N1 > 0 then
      InsertEvent(DepartureFrom1, 0.1);
end;
```

Rule for DepartureFrom2

```
Procedure RuleDepartureFrom2;
Begin
   N2 := N2 - 1;
   If N2 > 0 then
      InsertEvent(DepartureFrom2, Exp(4));
end;
```

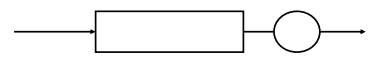
Rule for Measurement

```
Procedure RuleMeasurement;
Begin
   write(file1, N1);
   write(file2, N2);
   InsertEvent(Measurement, Exp(10));
end;
```

The main program

```
begin
   Read parameters
   N1 := 0;
   N2 := 0;
   time := 0;
   insert_event(Measurement,Exp(10));
   insert_event(ArrivalTo1, Exp(0.1));
   while time < simulationlength do
   begin
      dummy := FirstInQueue(eventlist);
      time := dummy.eventtime;
      case dummy.eventkind of
         ArrivalTo1: RuleArrivalTo1;
         DepartureFrom1: RuleDepartureFrom1;
         DepartureFrom2: RuleDepartureFrom2;
         Measurement: RuleMeasure;
      end;
   end;
end.
```

Another example



Assume that we want to measure the probability that A customer spends more than 5 seconds in the system.

Then it is not enough to keep track of the number of customers in the queuing system!

Events here are Arrival and Departure.

The state of this system

In this case the state can be a list where we can store customers and mark them with their arrival time:

⟶ 8.3 6.2 5.3 4.4 2.4 ⟶

Can be implemented by a double linked list or Vector

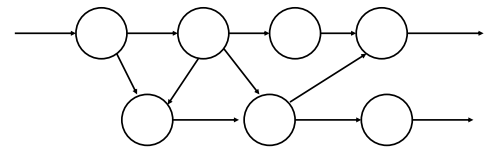
Rule for Arrive

```
Procedure RuleArrival;
Begin
   N := N + 1;
   InsertAsLast(Queue,time);
   If N = 1 then
      InsertEvent(Departure, Exp(3));
   InsertEvent(ArrivalTo1, Exp(7));
end;
```

Rule for Depart

```
Procedure RuleDepart;
Begin
   N := N - 1;
   If N > 0 then
      InsertEvent(Departure, Exp(3));
   Remove the customer from the list;
   Let x := time when the customer arrived;
   write(utfile, time - x);
end;
```

Drawback of event scheduling



Assume that we have a complicated network with many nodes. The network can model e.g a computer Network, material flow or luggage handling. The nodes are similar.

Drawbacks

- Many different events or events with attributes are needed
- It is difficult to change the system, a change in one of the nodes affects the programs global variables and rules
- It is more natural to think of such a problem as entities flowing through the network than to think about events

What we would like

- We would like to create a template for the nodes and customers
- When the program executes we would like to create instances of the nodes and customers
- We would like to set parameters to the instances when they are created

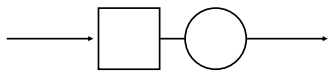
The solution

One way of solving this is the *process interaction method*.

Processes in simulation

- In simulation a *process* is something that does something
- A process has some *internal state*
- Processes communicate by sending *signals* to each other
- Signals have a name and can carry information
- When a signal arrives to a process some *activity* is trigged
- During an activity the state of the receiving process might be changed and signals may be sent
- When a signal is sent the sender assigns it an *arrival time*

An example

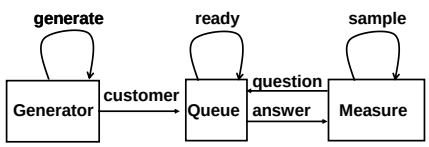


Assume that we want to describe a queuing system by the process-interaction approach.

The processes we need

- A process representing the queuing system
- A process that generates customers
- A process that measures the number of customers in the queuing system

The processes and signals



Generate, ready and sample are delayed. Answer has a parameter, the number of customers.

The internal state of the processes

Generator: no internal state needed
Queue: N = number of customers
Measure: no internal state needed

Activity of Generator

```
If received signal = generate then
begin
   SendSignal(customer, Queue, time);
   SendSignal(generate, Generator, time + Exp(4));
end
else
   write('Generator received illegal signal!');
```

Activity of Queue on next slide!

```
If received signal = customer then
begin
   N := N + 1;
   if N = 1 then
      SendSignal(ready, Queue, time + Exp(2));
end
else if received signal = ready then
begin
   N := N - 1;
   if N > 0 then
      SendSignal(ready, Queue, time + Exp(2));
end
else if received signal =  question then
begin
   SendSignal(answer(N), Measure, time);
end
else
   write('Queue received illegal signal!');
```

Activity of measure

```
If received signal = sample then
begin
   SendSignal(question, Queue, time);
   SendSignal(sample, Measure, time + Exp(10));
end
else if received signal = answer then
begin
   Extract N from signal answer;
   write(outfile, N);
end
else
   write('Generator received illegal signal!');
```

Some problems we must solve

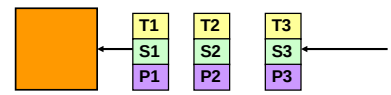
- How to keep track of time in the system
- How to make sure that signals arrive at the right time

Observe that it is not a question of real time!

Time is just updated when a signal arrives. It does not have any values in between.

Signal list

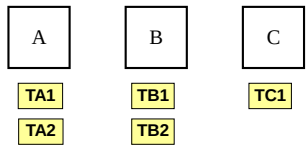
Each process has a signal list. It is very similar to the event list in the event scheduling approach.



Ti = arrival time of signal
Si = what kind of signal this is (the name of the signal)
Pi = parameters of the signals (if any)
T1 < T2 < T3 < etc

Process list

Processes with signals in their signal lists are organized in a process list. Only the arrival times of the signals are shown here.



The process list is sorted so that **TA1 < TB1 < TC1< etc**

How a process interaction simulation is done

1. **Remove the first process from the process list (call it A)**
2. **Remove the first signal in A:s signal list**
3. **Process the activities**
4. **If there are any signals left in A:s signal list, sort it into the process list again**
5. **If simulation shall continue, go to 1**

What to do when a process gets a signal

Assume that process B gets a signal.

1. Sort the signal into process B:s signal list.
2. If the signal list was empty before the signal arrived, B shall be sorted into the process list.
3. If the signal list was not empty, B is already in the process list. If the signal is put first in B:s signal list, B might have to change its place in the process list.

An example, the queuing system (1)

Time = 0
Generator: (3,arrival)
Measure: (10, sample)

Queue: [N=0] () Queue is not in the process list!

⇩

Time = 3

Queue: [N=0]
(3,customer)
Measure: (10, sample)
Generator: (11, generate)

An example, the queuing system (2)

Time = 3

Queue: [N=0] (3,customer)
Measure: (10, sample)
Generator: (11, generate)

⇩

Time = 3

Measure: (10, sample)
Generator: (11, generate)
Queue: [N=1] (12,ready)

An example, the queuing system (3)

Time = 3

Measure: (10, sample)
Generator: (11, generate)
Queue: [N=1] (12,ready)

⇩

Time = 10

Queue: [N=1](10,question)(12,ready)
Generator: (11, generate)
Measure: (20, sample)

An example, the queuing system (4)

Time = 10

Queue: [N=1](10,question) (12,ready)
Generator: (11, generate)
Measure: (20, sample)

⇩

Time = 10

Measure: (10, answer) (20, sample)
Generator: (11, generate)
Queue: [N=1](12,ready)

An example, the queuing system (5)

Time = 10

Measure: (10, answer) (20, sample)
Generator: (11, generate)
Queue: [N=1](12,ready)

⇩

Time = 10

Generator: (11, generate)
Queue: [N=1](12,ready)
Measure: (20, sample)

The steps in constructing a process interaction simulation program

- What processes are needed?
- What variables are need to describe the state of the processes?
- What signals are needed?
- What information (besides its name) shall a signal carry?
- What shall happen when a signal reaches a process?

When these questions are answered, it is not difficult to write a process interaction simulation program! Time spent thinking on these questions will save a lot of time later!

A further wish

We would like to define process types, e.g. generator and queue. When we start a program we would like to create as many instances of these types as we need.

In this way we can create a library of processes that can be reused. This is one more advantage of the process interaction approach.

Just one signal list

- It is possible to use just one signal list in a program
- In that case the implementation of a process interaction simulation program is very similar to a event scheduling program

The template program is written like that.