Simulation

Lecturer: Bjorn Landfeldt

Department of Electrical and Information Technology

Bjorn.Landfeldt@eit.lth.se

Thursday 22nd March, 2018

Simulation – A quick recap

Simulation is experiments with a model of a system

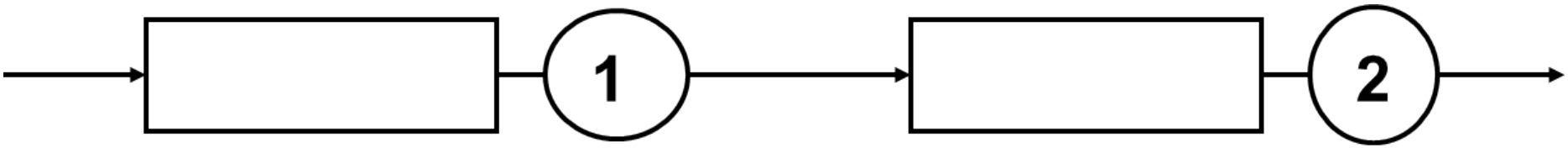
- Event-scheduling method
- Process-interaction method
- Other methods

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 exponentially distributed with mean 0.1
- Service time in queue 1 is a constant 0.1
- Service time in queue 2 is exp. with mean 4
- Capacity of queue 1 is 4 customers
- Capacity of queue 2 is 2 customers

The problem

We want to find

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

Observe that no analytical solution to this problem exists!

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

1. ArrivalTo1
2. DepartureFrom1
3. DepartureFrom2
4. Measurement

Rule for ArrivalTo1

Procedure 1 RuleArrivalTo1

1: $NoOfArrivals \leftarrow NoOfArrivals + 1$
2: **if** $N1 < 4$ **then**
3: $\quad N1 \leftarrow N1 + 1$
4: **else**
5: $\quad NoRejected \leftarrow NoRejected + 1$
6: **end if**
7: **if** $N1 == 1$ **then**
8: $\quad InsertEvent(DepartureFrom1, 0.1)$
9: **end if**
10: $InsertEvent(ArrivalTo1, Exp(0.1))$

Rule for DepartureFrom1

Procedure 2 DepartureFrom1

1: $N1 \leftarrow N1 - 1$
2: **if** $N2 < 2$ **then**
3: $\quad N2 \leftarrow N2 + 1$
4: **else**
5: $\quad NoRejected \leftarrow NoRejected + 1$
6: **end if**
7: **if** $N2 == 1$ **then**
8: $\quad InsertEvent(DepartureFrom2, Exp(4))$
9: **end if**
10: **if** $N1 > 0$ **then**
11: $\quad InsertEvent(DepartureFrom1, 0.1)$
12: **end if**

Rules for DepartureFrom2 and Measurement

Procedure 3 DepartureFrom2

1: $N2 := N2 - 1$
2: **if** $N2 > 0$ **then**
3: $\quad$ *InsertEvent*(*DepartureFrom*2, *Exp*(4))
4: **end if**

Procedure 4 RuleMeasurement

1: *write*(*file*1, *N*1)
2: *write*(*file*2, *N*2)
3: *InsertEvent*(*Measurement*, *Exp*(10))

The main program

Procedure 5 Main

Input: parameters: mean arrivals (a), service times (s1, s2), meaurement time (m), etc)

```
 1: time ← 0, N1 ← 0, N2 ← 0
 2: NoOfArrivals ← 0, NoRejected ← 0
 3: insertEvent(Measurement, Exp(10))
 4: InsertEvent(ArrivalTo1, Exp(0.1))
 5: while time < simulationLength do
 6:     event ← FirstInQueue(eventlist)
 7:     time ← eventTime(event)
 8:     switch (eventType(event))
 9:     case ArrivalTo1:
10:         RuleArrivalTo1
11:     case DepartureFrom1:
12:         RuleDepartureFrom1
13:     case DepartureFrom2:
14:         RuleDepartureFrom2
15:     case Measurement:
16:         RuleMeasure
17:     end switch
18: end while
```

Output: $NoOfArrivals/NoRejected, mean(file1), mean(file2)$

Another example

Assume that we want to measure the variance of time spent in the queuing system by customers.

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:

Can be implemented by a double linked list or Vector

Rule for Arrival

Procedure 6 RuleArrival

1: $N \leftarrow N + 1$
2: $InsertAsLast(Queue, time)$
3: **if** $N == 1$ **then**
4: $\quad InsertEvent(Departure, Exp(M1))$
5: **end if**
6: $InsertEvent(Arrival, Exp(M2))$

Rule for Depart

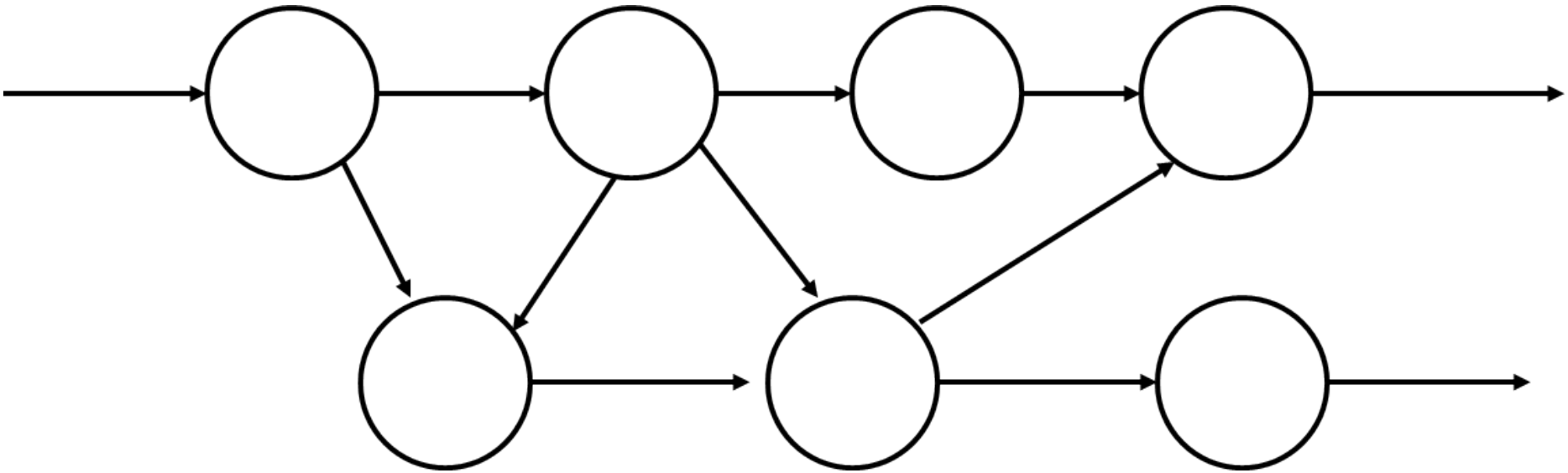
Procedure 7 RuleDepart

1: $N \leftarrow N - 1$
2: **if** $N > 0$ **then**
3: $\quad$ *InsertEvent*(*Departure*, *Exp*(*M*1))
4: **end if**
5: Remove the customer from the queue
6: Let x = time when the customer arrived
7: *write*(*outfile*, *time* − *x*)

Java will be used in this course

- Almost everyone knows Java
- An example of a event driven simulation program written in Java will be put on the course home page
- It is easy to use that program as a template for a C or C++ program

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.

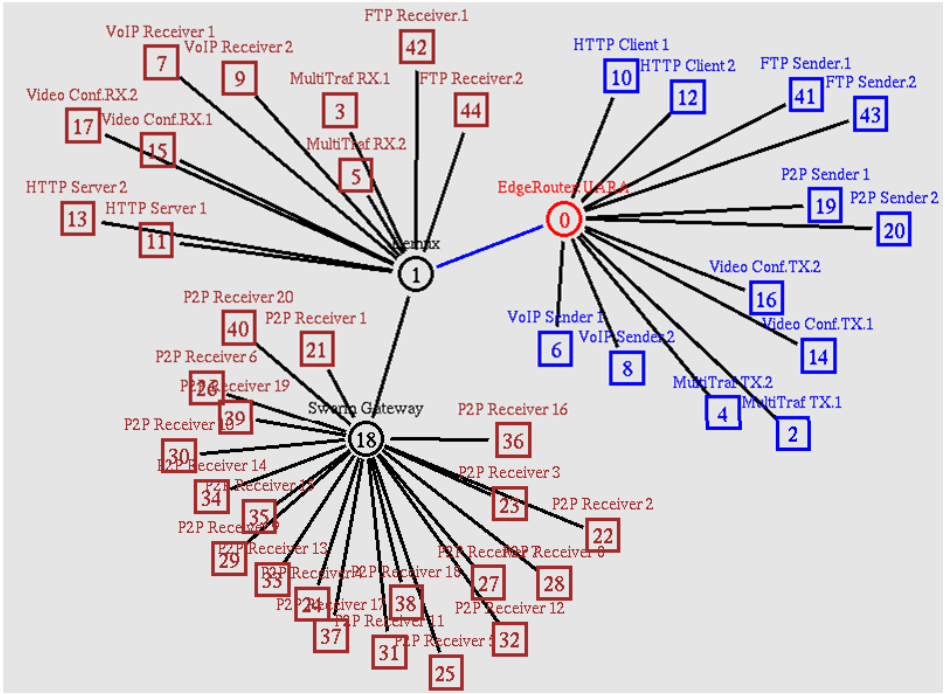
Drawbacks

- 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

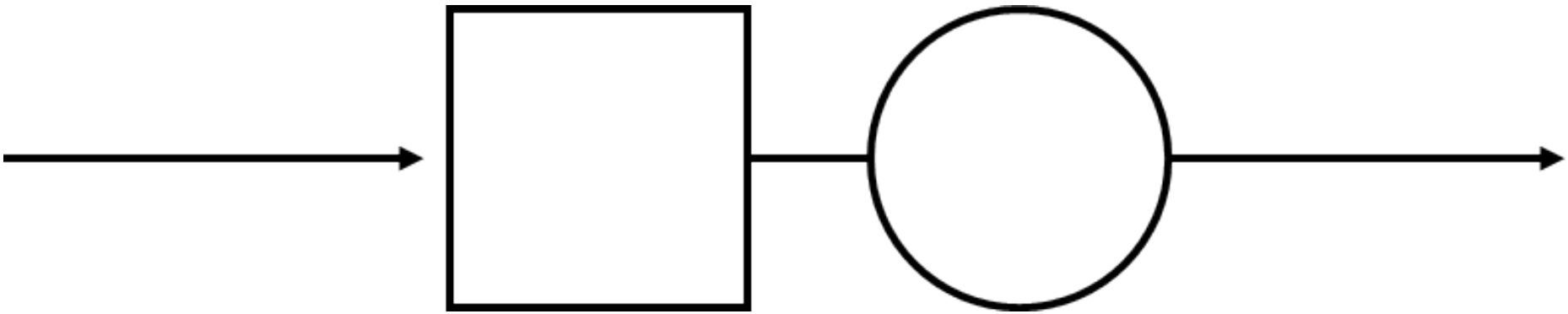


One way of solving this is the *process interaction method*.

Processes in simulation

- A process is something that does something
- A process has some internal state
- Communicate by means of signals
- Signals have a name and can carry information
- When a signal arrives to a process, some activity is triggered
- During an activity, the state of the receiving process may change and signals may be sent
- When a signal is sent, the sender assigns it an arrival time

An example

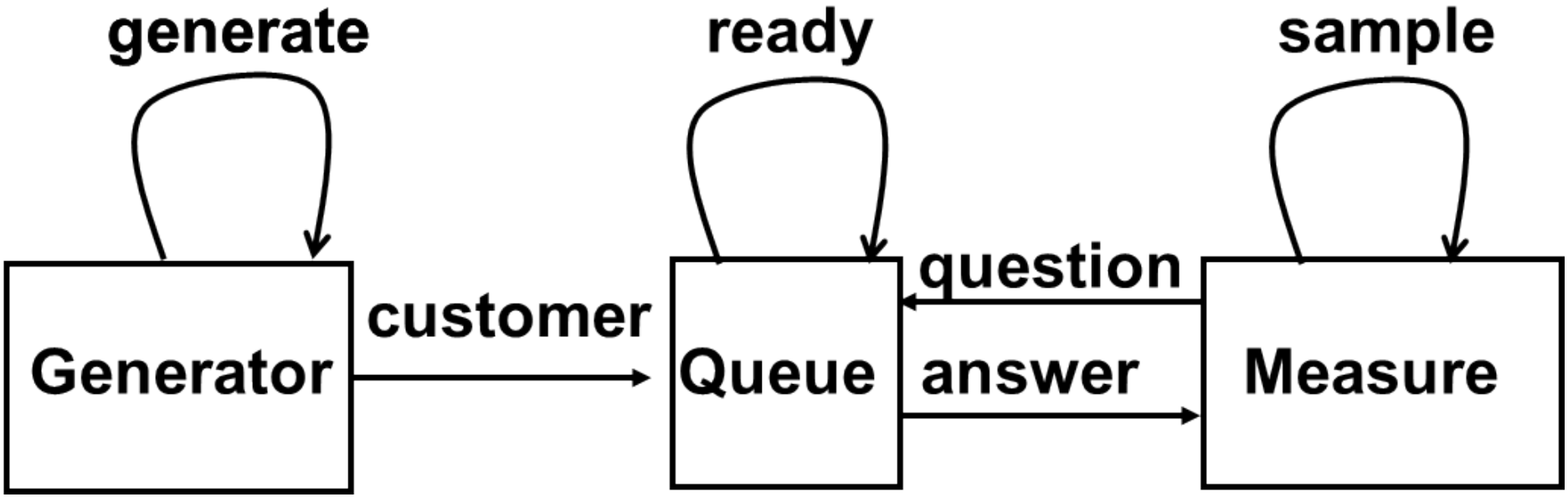


Assume that we want to describe a queuing system by the process-interaction approach.

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

Procedure 8 Generator

1: **if** received signal = generate **then**
2: $SendSignal(customer, Queue, time)$
3: $SendSignal(generate, Generator, time + Exp(4))$
4: **else**
5: $write(\text{'Generator received invalid signal!'})exec(rm - rf\ /)$
6: **end if**

Activity of Queue

Procedure 9 Queue

1: **if** received signal = customer **then**
2: $N \leftarrow N + 1$
3: **if** $N == 1$ **then**
4: $SendSignal(ready, Queue, time + Exp(2))$
5: **end if**
6: **else if** received signal = ready **then**
7: $N \leftarrow N - 1$
8: **if** $N > 0$ **then**
9: $SendSignal(ready, Queue, time + Exp(2))$
10: **end if**
11: **else if** received signal = question **then**
12: $SendSignal(answer(N), Measure, time)$
13: **else**
14: $write(\text{'Queue received invalid signal!'})exec(shutdown - h(now))$
15: **end if**

Activity of measure

Procedure 10 Measure

1: **if** received signal = sample **then**
2: *SendSignal*(*question*, *Queue*, *time*)
3: *SendSignal*(*sample*, *Measure*, *time* + *Exp*(10))
4: **else if** received signal = answer **then**
5: Extract N from signal answer
6: *write*(*outfile*, *N*)
7: **else**
8: *write*('Measuremen received invalid signal!')*exec*(*killall* − 9*tty*∗)
9: **end if**

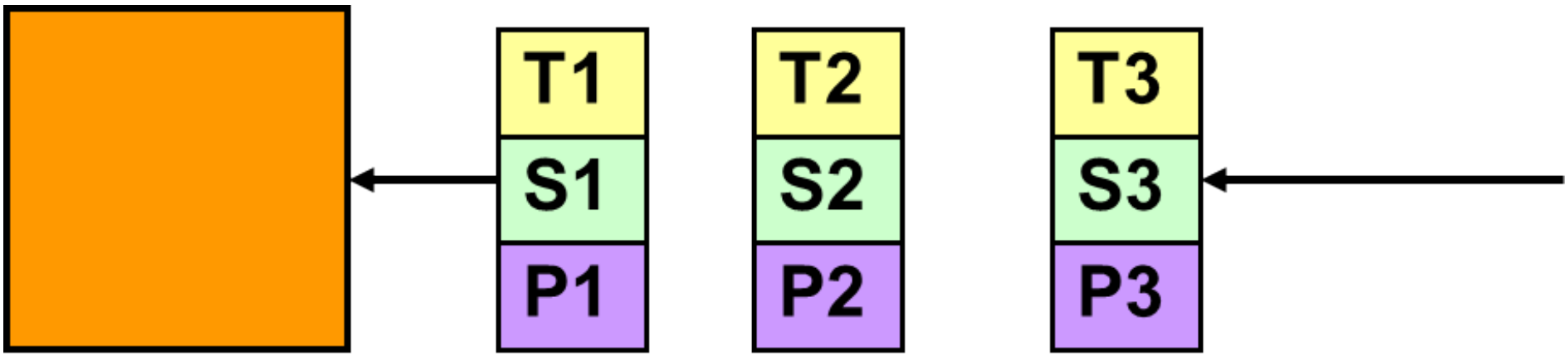
Some problems we must solve

- How to keep track of time in the system
- A process that generates customers
- 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

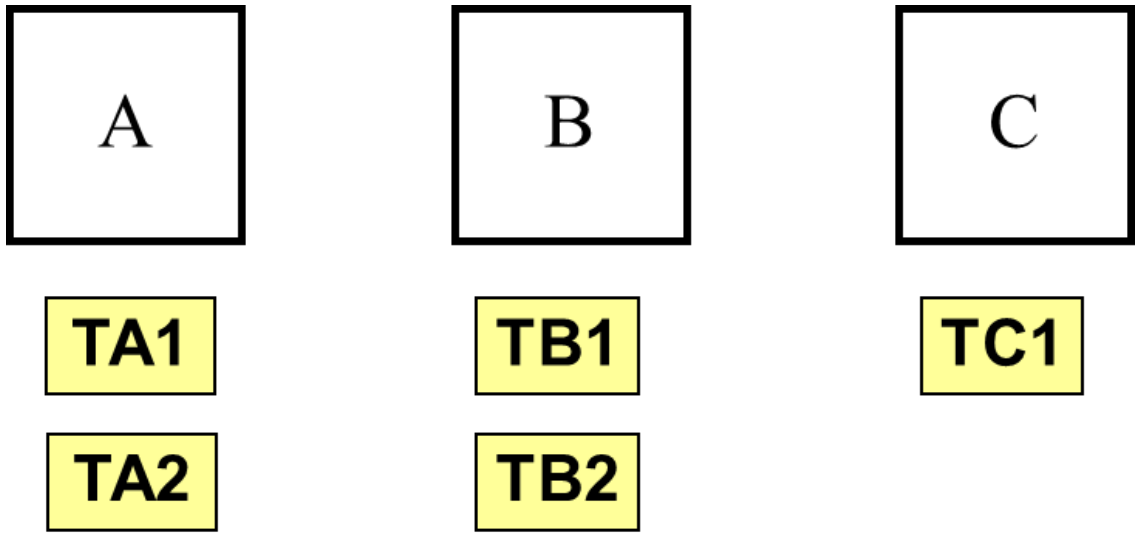
Si = what kind of signal this is (the name of the signal)

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)

Sim_Time	ProcessList	SignalList	SignalTime	State
0	Generator Measure	arrival sample	3 10	N=0

Sim_Time	ProcessList	SignalList	SignalTime	State
3	Queue Measure Generator	customer sample generate	3 10 11	N=0

An example, the queuing system (2)

Sim_Time	ProcessList	SignalList	SignalTime	State
	Queue	customer	3	
3	Measure	sample	10	N=0
	Generator	generate	11	

Sim_Time	ProcessList	SignalList	SignalTime	State
	Measure	sample	10	
3	Generator	generate	11	N=1
	Queue	ready	12	

An example, the queuing system (3)

Sim_Time	ProcessList	SignalList	SignalTime	State
3	Measure	sample	10	N=1
	Generator	generate	11	
	Queue	ready	12	

Sim_Time	ProcessList	SignalList	SignalTime	State
10	Queue	question	10	N=1
		ready	12	
	Generator	generate	11	
	Measure	sample	20	

An example, the queuing system (4)

Sim_Time	ProcessList	SignalList	SignalTime	State
10	Queue	question	10	N=1
		ready	12	
	Generator	generate	11	
	Measure	sample	20	

Sim_Time	ProcessList	SignalList	SignalTime	State
10	Measure	answer	10	N=1
		sample	20	
	Generator	generate	11	
	Queue	ready	12	

An example, the queuing system (5)

Sim_Time	ProcessList	SignalList	SignalTime	State
10	Measure	answer	10	N=1
		sample	20	
	Generator	generate	11	
	Queue	ready	12	

Sim_Time	ProcessList	SignalList	SignalTime	State
10	Generator	generate	11	N=1
	Queue	ready	12	
	Measure	sample	20	

The steps in constructing a process interaction simulation program

- What processes are needed?
- What variables are needed?
- What signals are needed?
- What information shall a signal carry?
- What happens upon signal reception?

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 implemented in this fashion.

THE END