Discrete event simulation

- A type of simulation where
  - time advances in discrete steps and
  - *events* cause stepwise changes to the system state
- Applied to problems where the timing of activities is the main interest
- The system is divided into *entities* rather than trying to model it as one big finite state machine.
  - Temporary *entities* flow through the system
    - e.g. parts, customers or messages that arrive according to a stochastic distrubution
  - Permanent entities stay in the system during the simulation
    - e.g. machines, servers or routers, processing the temporary entities with stochastically distributed processing times
  - *Attributes* are used for defining the states and properties of individual entities.

Discrete event simulation scheduler

- The *scheduler* (timer) maintains the simulation time and sends timer events to the entities
  - contains the *simulation clock* and
  - a list of scheduled future events
- Operation:
  - seeks the scheduled event that has the smallest time stamp
  - advances the simulation time
  - sends a timer event to the corresponding entity

Different implementations of discrete event simulation

- Event-oriented simulation (event-scheduling approach)
  - concentrates on handling and sending events
  - the activity following each event is implemented as an event-routine
  - the event-routine may schedule new events and re-schedule existing event
- Process-oriented simulation (process approach)
  - concentrates on tracking the behaviour of permanent (and temporary) entities
  - the sequence of activities of each entity are written as a co-routine
  - the coroutine may schedule and re-scedule itself and other processes

Event-oriented simulation

```
void Event1() {
    // do what event 1 involves
    // schedule or re-schedule events
    event_queue.ScheduleAt(Event2, time);
    return;                 // return to scheduler
    }
void Scheduler() {
    while(!event_queue.Empty()) {
        e= event_queue.GetFirst();   // remove first event from queue
        simtime= e->schedtime;       // advance simulated time
        e.proc();                    // run event procedure
    }}
```

Process oriented simulation

```
void Process1::body() { // process 1
    // do what process 1 does
    ACTIVATE(proc2, time); // schedule proc2 at time
    HOLD(delay);        // schedule self after delay, resume Scheduler
    ...
    PASSIVATE();        // passivate self, resume Scheduler
    }
void Scheduler() {      // co-routine scheduler
    while(!process_queue.Empty()) {
      p= process_queue.GetFirst();      // remove first from queue
      simtime= p->schedtime;            // advance simulated time
      resume(p);                        // resume co-routine
    }}
```

R. Lahdelma 5

Process oriented simulation primitives

Process::ACTIVATE(delay)

- schedule process to become active after delay (at simtime+d)
- process is inserted into process_queue and state becomes SUSPENDED

Process::CANCEL()

- cancel scheduling of SUSPENDED or ACTIVE process
- process is removed from process queue and state becomes PASSIVE

HOLD(delay)

- schedule *current_process* to become active after delay
- state becomes SUSPENDED

PASSIVATE()

- interrupt the execution of *current_process* and resume next in process_queue
- state becomes PASSIVE

R. Lahdelma 6

Process oriented simulation

- Implementation without co-routines
  - process BODY is implemented as a finite state machine that will execute one event at a time
  - the Scheduler calls the state machines according to their schedtime
  - DETACH is implemented as return from the BODY
- Implementation with co-routines (threads)’
  - the Scheduler and Process instances run as a separate co-routines (threads)
  - DETACH is implemented as resumption of the Scheduler

Example: queue with single server

- Arrival process (AProc):
  - creates customers with some interval and inserts them into a server queue
  - if the server is idle, it is activated
  - stops after a given time, number of customers or continues forever
- Server process (SProc)
  - checks if there are customers awaiting service in the queue, becomes idle if not
  - takes the first customer from the queue and initiates service
  - stops after a given time, number of served customers or continues forever

Implementing process oriented simulation

- With co-routines

```
AProc::BODY() {
    while(count<CMAX) {
      c-> new Customer();
      squeue.Insert(c);
      if(server->Idle())
        server->ACTIVATE(0.0);
      HOLD(delay);
    }}
SProc::BODY() {
    while(1) {
      if(squeue.Empty())
        PASSIVATE();
      c= squeue->GetFirst();
      HOLD(sertime);
    }}
```

- Without co-rotines

```
AProc::BODY() {
    if(count<CMAX) {
      c-> new Customer();
      squeue.Insert(c);
      if(server->Idle())
        server->ACTIVATE(0.0);
      HOLD(delay);
    }}
SProc::BODY() {
    if(squeue.Empty())
      return;
    c= squeue->GetFirst();
    HOLD(sertime);
}
```