Process Interaction

The `Process` instance that is returned by the constructor `Process(env, func)` can be utilized for process interactions. The two most common examples for this are to wait for another process to finish and to Interrupt another process while it is waiting for an event.

Waiting for a Process

As it happens, a SimJulia `Process` can be used like an event (technically, a `Process` is a subtype of `AbstractEvent`). If you yield it, you are resumed once the process has finished. Imagine a car-wash simulation where cars enter the car-wash and wait for the washing process to finish. Or an airport simulation where passengers have to wait until a security check finishes.

Assume that the car from the last example magically became an electric vehicle. Electric vehicles usually take a lot of time charging their batteries after a trip. They have to wait until their battery is charged before they can start driving again.

This can be modeled with an additional charge process. Therefore, two process functions are created: `car(env)` and `charge(env, duration)`.

A new charge process is started every time the vehicle starts parking. By yielding the `Process` instance that `Process(env, func, args...)` returns, the run process starts waiting for it to finish:

```
julia> using SimJulia

julia> function car(env::Environment)
         while true
           println("Start parking and charging at $(now(env))")
           charge_duration = 5.0
           charge_proc = Process(env, charge, charge_duration)
           yield(charge_proc)
           println("Start driving at $(now(env))")
           trip_duration = 2.0
           yield(Timeout(env, trip_duration))
         end
       end
car (generic function with 1 method)

julia> function charge(env::Environment, duration::Float64)
         yield(Timeout(env, duration))
       end
charge (generic function with 1 method)
```

Starting the simulation is straightforward again: create an environment, one (or more) cars and finally call `run(env, 15.0)`:

```
julia> env = Environment()
Environment(0.0,PriorityQueue{BaseEvent,EventKey}(),0,0,Nullable{Process}())

julia> Process(env, car)
SimJulia.Process 1: car

julia> run(env, 15.0)
Start parking and charging at 0.0
Start driving at 5.0
Start parking and charging at 7.0
Start driving at 12.0
Start parking and charging at 14.0
```

Interrupting Another Process

Imagine, you don't want to wait until your electric vehicle is fully charged but want to interrupt the charging process and just start driving instead.

SimJulia allows you to interrupt a running process by calling the constructor `Interrupt(proc)` that returns an interrupt event.

An interrupt is thrown into process functions as an `InterruptException` that can (should) be handled by the interrupted process. The process can than decide what to do next (e.g., continuing to wait for the original event or yielding a new event):

```
julia> using SimJulia

julia> function driver(env::Environment, car_proc::Process)
         yield(Timeout(env, 3.0))
         yield(Interrupt(car_proc))
       end
driver (generic function with 1 method)

julia> function car(env::Environment)
         while true
           println("Start parking and charging at $(now(env))")
           charge_duration = 5.0
           charge_proc = Process(env, charge, charge_duration)
           try
             yield(charge_proc)
           catch exc
             println("Was interrupted. Hopefully, the battery is full enough ...")
           end
           println("Start driving at $(now(env))")
           trip_duration = 2.0
           yield(Timeout(env, trip_duration))
         end
       end
car (generic function with 1 method)

julia> function charge(env::Environment, duration::Float64)
         yield(Timeout(env, duration))
       end
charge (generic function with 1 method)
```

When you compare the output of this simulation with the previous example, you'll notice that the car now starts driving at time `3` instead of `5`:

```
julia> env = Environment()
Environment(0.0,PriorityQueue{BaseEvent,EventKey}(),0,0,Nullable{Process}())

julia> proc = Process(env, car)
SimJulia.Process 1: car

julia> Process(env, driver, proc)
SimJulia.Process 3: driver

julia> run(env, 15.0)
Start parking and charging at 0.0
Was interrupted. Hopefully, the battery is full enough ...
Start driving at 3.0
Start parking and charging at 5.0
Start driving at 10.0
Start parking and charging at 12.0
```