

Wim Couwenberg



Printing for
Professionals

What is this talk about?

- Copier/printer is built up of many parts
 - Scan engine
 - Print engine
 - Print file interpreters (PCL, PS, ...)
 - Local user interface
 - Controller
- Different hardware platforms
 - Intel
 - ARM
 - ASIC/FPGA
- Slow (cheap!) connections between these parts
- How to predict behaviour?

A quick experiment with Lua...

- Use Lua to simulate everything
- All processing is done by “scriptlets”
 - Scriptlets are just functions or script files executing in separate coroutines
 - The main simulator loop runs a scriptlet scheduler
- Scriptlets can post timed events
- Scriptlets can wait for events (yield)
- Shared resources are modeled on top of events
 - Semaphore
 - Processor (process for X secs. with Y% load)
 - I/O (limited bandwidth)
- Time is “virtual”: a numeric property (ordering) of events

A simple example

- Spawn server and client scriptlets
- Scriptlets produce csv logging (easy visualization)

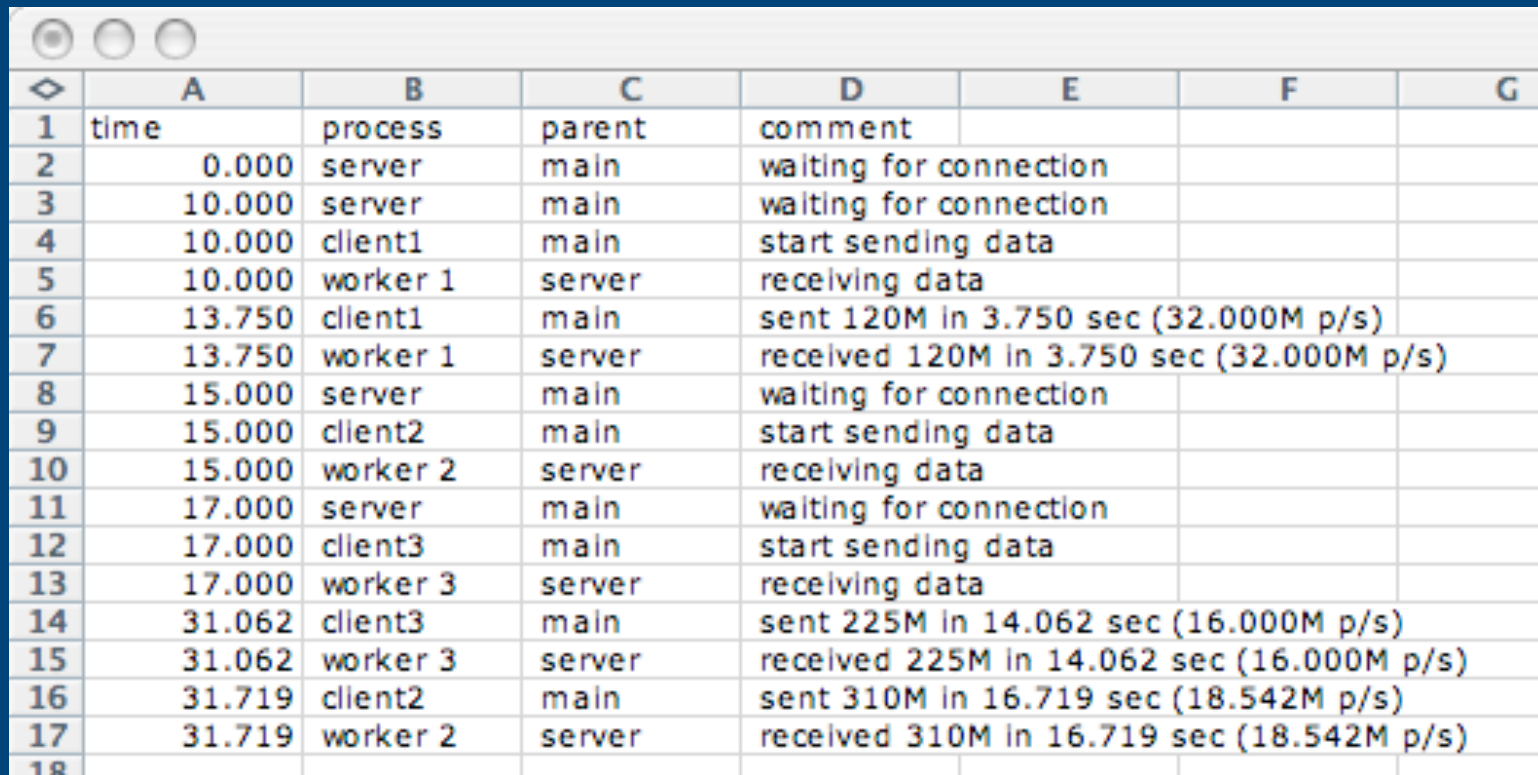
```
-- load simulation module
local sim = require "sim"

-- schedule server scriptlet to run at time 0
sim.spawn(0, "server", "server.lua")

-- schedule 3 clients with different arguments
sim.spawn(10, "client1", "client.lua", 120)
sim.spawn(15, "client2", "client.lua", 310)
sim.spawn(17, "client3", "client.lua", 225)

-- run the simulation
sim.run()
```

A simple example (2)



	A	B	C	D	E	F	G
1	time	process	parent	comment			
2	0.000	server	main	waiting for connection			
3	10.000	server	main	waiting for connection			
4	10.000	client1	main	start sending data			
5	10.000	worker 1	server	receiving data			
6	13.750	client1	main	sent 120M in 3.750 sec (32.000M p/s)			
7	13.750	worker 1	server	received 120M in 3.750 sec (32.000M p/s)			
8	15.000	server	main	waiting for connection			
9	15.000	client2	main	start sending data			
10	15.000	worker 2	server	receiving data			
11	17.000	server	main	waiting for connection			
12	17.000	client3	main	start sending data			
13	17.000	worker 3	server	receiving data			
14	31.062	client3	main	sent 225M in 14.062 sec (16.000M p/s)			
15	31.062	worker 3	server	received 225M in 14.062 sec (16.000M p/s)			
16	31.719	client2	main	sent 310M in 16.719 sec (18.542M p/s)			
17	31.719	worker 2	server	received 310M in 16.719 sec (18.542M p/s)			
18							

Events & scheduling

```
event = {  
    time = scheduled time (can be infinite),  
    thread = scriptlet that posted event,  
}  
  
events = least time in first out queue (heap)  
  
function sim.getevent()  
    return coroutine.yield()  
end  
  
function sim.run()  
    for event in pop(events) do  
        sim.time = max(sim.time, event.time)  
        coroutine.resume(event.thread, event)  
    end  
end
```

Applications of event scheduling

- `sim.time == infinite` indicates a deadlock!
- Time of scheduled event can be changed in queue
- Example: “semaphore”
 - “Release” up to n events when semaphore count increases by n
 - Release means: set event time from infinite (blocking) to current simulator time
- Example: scriptlets sharing an “I/O channel”
 - The more traffic, the longer I/O events take
 - Event times are updated when I/O events start or expire
- Example: “processor” load shared among scriptlets
 - Events take longer to “complete” when total load $> 100\%$
 - Event times are updated when load changes

Example: semaphore

```
function sem:lock()
  while self.count == 0 do
    local event = sim.schedule(infinite)
    self:pushevent(event)
    sim.getevent()
  end
  self.count = self.count - 1
end

function sem:unlock(n)
  self.count = self.count + n
  for event in self:popevents(n) do
    sim.reschedule(event, sim.time)
  end
end
```


Did it work?

- Colleagues unfamiliar with Lua programmed scriptlets in a matter of hours (this is also a statement about people at Océ...) 😊
- Different disciplines (embedded, scanner, controller) “explained” their fields of expertise by developing scriptlet code together
- We were able to check theoretical discussions and consequences for overall system timing using a number of tiny but clever scriptlets
- And... we had a lot of fun doing it!