Introduction to SimPy
Course: System Performance Evaluation

Tran, Van-Hoai (hoai@hcmut.edu.vn)
Nguyen, Phuong-Duy (pdnguyen@hcmut.edu.vn)
Nguyen, VM. Man (man.vm.nguyen@outlook.com)

Faculty of Computer Science & Engineering HCMC University of Technology
Faculty of Science - Mahidol University

2023-2024/Semester 1

Outline

1. Basic Simpy

2. Completed sample source codes

3. Use SimPy implement Queuing node M/M/1

Basic SimPy

1. SimPy is process-based
2. SimPy is discrete-event simulation framework

```
import simpy

def timer(env, duration=5):
    while True:
        print('Start␣timer␣at␣%d' % env.now)
        yield env.timeout(duration)
        print('End␣timer␣at␣%d' % env.now)


env = simpy.Environment()
env.process(timer(env, 3))
env.run(until=10)
```

The output:

```
Start timer at 0
End timer at 3
Start timer at 3
End timer at 6
Start timer at 6
End timer at 9
Start timer at 9

...
```

3. **Environment.Process** start new tasks (generate events)
4. **Environment.Timeout** schedule events at a given time
5. **Environment.Run** perform the simulation

Simulation control

1. Simpy is very flexible in execution, it runs simulation until there are no more events
2. The most important method is **Environment.run()**
3. To step through the simulation by event
   - peek() returns the time of the next scheduled event
   - step() processes the next scheduled event

```
until = 10
while env.peek() < until:
   env.step()
```

State access

1. The current simulation time via the **Environment.now**
2. The **simpy.Resources** can be used to manage multiple processes or model overlap points among processes.
3. In the process (function implementation), it calls request() to request a resource.
   - The method generates an event that lets you wait until the resource becomes available again.

```
import simpy

def timer(env, name, resource, duration=5):
    with resource.request() as req:
        yield req

        print('Start␣timer␣%s␣at␣%d' % (name,
             env.now))
        yield env.timeout(duration)
        print('End␣timer␣%s␣at␣%d' % (name, env.
             now))
env = simpy.Environment()
# capacity changes the number of generators in
     the system.
server1 = simpy.Resource(env, capacity=1)
```

```
for i in range(4):
    env.process(timer(env, '
         Timer␣%s' % i, server1
         , i+1))
env.run()

Start timer Timer 0 at 0
End timer Timer 0 at 1
Start timer Timer 1 at 1
End timer Timer 1 at 3
Start timer Timer 2 at 3
End timer Timer 2 at 6
Start timer Timer 3 at 6
End timer Timer 3 at 10
```

Event creattion

1. To create events, you normally import **simpy.events**
2. Shortcut.
   - **Environment.process**() as previous sample
   - **Environment.timeout**()
   - **Environment.all_of()**
   - **Environment.any_of()**

```
...
def final_msg(msg, list_of_processes):
    yield simpy.AllOf(env, list_of_processes)
    print(env.now, msg)

listp = []
env = simpy.Environment()
# capacity changes the number of generators in
     the system.
server1 = simpy.Resource(env, capacity=1)
for i in range(4):
    listp.append(env.process(timer(env, 'Timer␣%
         s' % i, server1, i+1)))
```

```
env.process(final_msg("All␣done"
     , listp))
env.run()




...
Start timer Timer 3 at 6
End timer Timer 3 at 10
10 All done
```

Outline

1 Basic Simpy

2 Completed sample source codes

3 Use SimPy implement Queuing node M/M/1

Sample 1 - The first timer

```
import simpy


def timer(env, duration=5):
    while True:
        print('Start␣timer␣at␣%d' % env.now)
        yield env.timeout(duration)
        print('End␣timer␣at␣%d' % env.now)


env = simpy.Environment()
env.process(timer(env, 3))
env.run(until=20)
```

Sample 2 - Timer with resource.request()

```python
import simpy


def timer(env, name, resource, duration=5):
    with resource.request() as req:
        yield req

        print('Start␣timer␣%s␣at␣%d' % (name, env.now))
        yield env.timeout(duration)
        print('End␣timer␣%s␣at␣%d' % (name, env.now))


env = simpy.Environment()
# capacity changes the number of generators in the system.
server1 = simpy.Resource(env, capacity=1)
for i in range(4):
    env.process(timer(env, 'Timer␣%s' % i, server1, i+1))
env.run()
```

Sample 3 - Timer with finalizing all_of events

```
import simpy


def timer(env, name, resource, duration=5):
    with resource.request() as req:
        yield req

        print('Start␣timer␣%s␣at␣%d' % (name, env.now))
        yield env.timeout(duration)
        print('End␣timer␣%s␣at␣%d' % (name, env.now))


def final_msg(msg, list_of_processes):
    yield simpy.AllOf(env, list_of_processes)
    print(env.now, msg)


listp = []
env = simpy.Environment()
# capacity changes the number of generators in the system.
server1 = simpy.Resource(env, capacity=1)
for i in range(4):
    listp.append(env.process(timer(env, 'Timer␣%s' % i, server1, i+1)))
env.process(final_msg("All␣done", listp))
env.run()
```

Outline

1 Basic Simpy

2 Completed sample source codes

3 Use SimPy implement Queuing node M/M/1

Queuing node M/M/1 - Define Generater and Customer

1. Generator creates event to generate a/many customer(s).
2. Customer arrives the system, waits for servicing time and leaves
3. Customer is created with inter-arrival time is exponential
4. Customer is serviced with service time is exponential

```
import random
import simpy
import numpy as np


new_customers = 10000  # Total number of customers in the system
interarrival = np.random.poisson(6, size=None)  #
```

(*Notice: this implementation is mainly for illustrating purpose, code performance is bad, buggy and no guarantee, try at your own risk*)

Queuing node M/M/1 - Generator and customer

```
def generator(env, number, interval, server, service_time):
    """generator generates customers randomly"""
    for i in range(number):
        c = customer(env, 'Customer%02d' % i, server, service_time=random.
             expovariate(service_time))
        env.process(c)
        t = random.expovariate(1.0 / interval)
        yield env.timeout(t)  # adds time to the counter, does not delete from
             the memory

def customer(env, name, server, service_time):
    # customer arrives to the system, waits and leaves
    arrive = env.now
    # print('%7.4f : Arrival time of %s' % (arrive, name))
    with server.request() as req:
        results = yield req | env.timeout(arrive)

        if req in results:
            servertime = service_time
            yield env.timeout(servertime)
            serviceTimes.append(servertime)
       else:
            waiting_time = env.now - arrive
            waitingTimes.append(waiting_time)
```

Queuing node M/M/1 - Execute the simulator

1. Declare the environment and execute the simulation

```
from random import seed


seed(29384)  # for seed of randint function
random_seed = 42  # for seed of other random generators

random.seed(random_seed)
env1 = simpy.Environment()
server1 = simpy.Resource(env1, capacity=1)  # capacity changes the number of
      generators in the system.
env1.process(generator(env1, new_customers, interarrival, server1, 0.15))
env1.run()
```

```
import statistics

waitingTimes = []
serviceTimes = []
interarrivalTimes = []

interarrivalTimes.append(interarrival)
average_interarrival = statistics.mean(interarrivalTimes)
average_serviceTime = statistics.mean(serviceTimes)
print("Average␣Interarrival␣Time␣Is␣:␣%7.4f" % average_interarrival)
print("Average␣Service␣Time␣Is␣:␣%7.4f" % average_serviceTime)
if len(waitingTimes) > 0:
    average_waitingTime = statistics.mean(waitingTimes)
    print("Average␣Waiting␣Time␣Is␣:␣%7.4f" % average_waitingTime)
```

Queuing node M/M/1 - Extends to Multiple Queuing system

1. Multiple Queuing system has 2 Parallel M/M/1 Queue

```
from random import seed
env = simpy. Resource (...)
server1 = simpy.PreemptiveResource(env,capacity=1)
env.process(generator(...)
server2 = simpy.PreemptiveResource(env,capacity=1)
env.process(generator(...)}
env.run
```

2. Multiple Queuing system has 2 Nested M/M/1 Queue

```

def arrivalcustomer1(...):
...
     c2 = arrivalcustomer2(...)
     env.process(c2)
...
```

(The provided sample code is only for initialized illustration and is lack of reliability.)