Simulation: introduction

Tran, Van Hoai (hoai@hcmut.edu.vn)
Le, Hong Trang (lhtrang@hcmut.edu.vn)

Faculty of Computer Science & Engineering
HCMC University of Technology

2021-2022/Semester 2

Outline

1. Introduction and classification
   - Types of simulation
2. Discrete event simulation (DES)
   - SimPy
3. Single queue

Outline

1. Introduction and classification
   - Types of simulation
2. Discrete event simulation (DES)
   - SimPy
3. Single queue

What is simulation?

Definition (Wikipedia)

A simulation is an approximate imitation of the operation of a process or system.

- A widely-used performance analysis method used for any stages
  - System is not available: to predict and compare alternatives
  - System is available: to compare under wider variety of workloads

What is simulation?

Definition (Wikipedia)

A simulation is an approximate imitation of the operation of a process or system.

- A widely-used performance analysis method used for any stages
  - System is not available: to predict and compare alternatives
  - System is available: to compare under wider variety of workloads

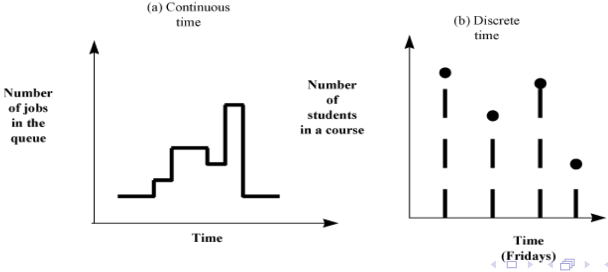
Model developers must be proficient in

- software development
- statistical techniques

Terminology (1)

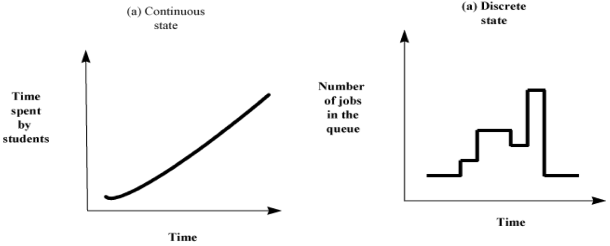
CPU scheduling is used as examples.

- **State variables**: values to define the state of the system
  E.g., length of job queue
- **Event**: something happens to change the system state
  E.g., arrival of a job; beginning of a new execution
- **Continuous-time** and **discrete-time**: whether the system state defined at all times or not.



Terminology (2)

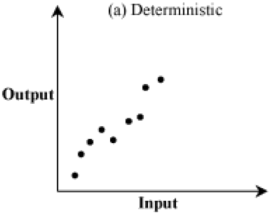
- **Continuous-state** and **discrete-state**: whether the state variables are continuous or discrete.

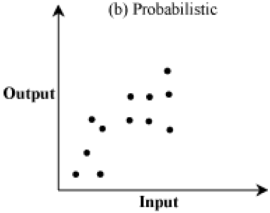


  - Continuous-state model = **Continuous-event model**
  - Discrete-state model = **Discrete-event model**

Terminology (3)

- **Deterministic** and **probabilistic/stochastic** models
  - Deterministic model: output is predicted with certainty
  - Probabilistic model: different output on repetitions for the same set of input parameters



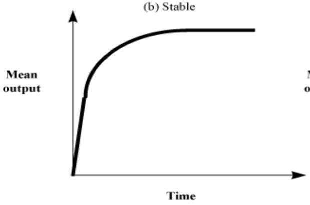


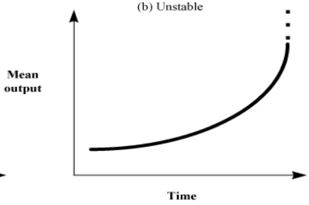
- **Static** and **dynamic** models
  - Static model: time is not variable
  - Dynamic model: system state changes with time

E.g., master-to-energy transformation $E = mc^2$.

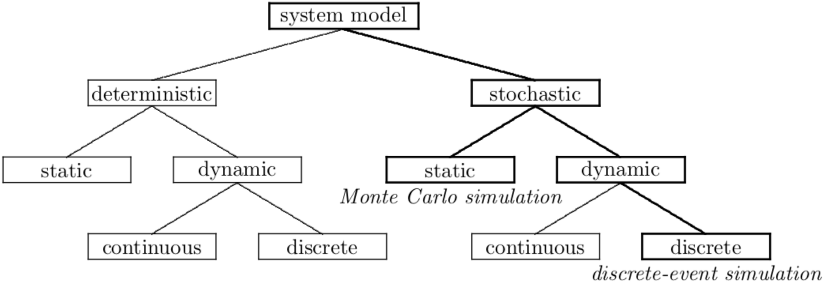
Terminology (4)

- **Open** and **closed** models: *read lecture #5 "queueing networks"*.
- **Stable** and **unstable** models
  - Stable model: dynamic behaviour settles down to a steady state.
  - Unstable model: behaviour is continuously changing.

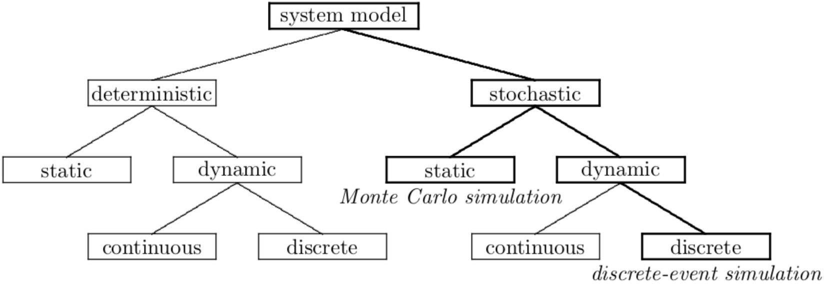




System model taxanomy (phân loại)



System model taxanomy (phân loại)



Computer system models

- Continuous time
- Discrete state
- Probabilistic
- Dynamic
- Nonlinear
- Open and closed
- Stable and unstable

Types of simulation

There are different ways to classify simulation types. Below is only one of them.

Types of simulation

There are different ways to classify simulation types. Below is only one of them.

1. Emulation: Using hardware or firmware.
   - E.g., Terminal emulator, processor emulator
   - Mostly hardware design issues.

Types of simulation

There are different ways to classify simulation types. Below is only one of them.

1. Emulation: Using hardware or firmware.
   - E.g., Terminal emulator, processor emulator
   - Mostly hardware design issues.
2. Monte Carlo Simulation.
   - Static simulation without time axis
   - Used for evaluating nonprobabilistic expressions using probabilistic methods

     E.g., π computation

Types of simulation

There are different ways to classify simulation types. Below is only one of them.

1. Emulation: Using hardware or firmware.
   - E.g., Terminal emulator, processor emulator
   - Mostly hardware design issues.
2. Monte Carlo Simulation.
   - Static simulation without time axis
   - Used for evaluating nonprobabilistic expressions using probabilistic methods

     E.g., π computation
3. Trace-Driven Simulation.
   - A simulation using a trace as its input

     E.g., using trace-driven simulation to analyse paging algorithms, cache analysis, CPU scheduling algorithms, deadlock prevention algorithms, etc.

Types of simulation

There are different ways to classify simulation types. Below is only one of them.

1. Emulation: Using hardware or firmware.
   - E.g., Terminal emulator, processor emulator
   - Mostly hardware design issues.
2. Monte Carlo Simulation.
   - Static simulation without time axis
   - Used for evaluating nonprobabilistic expressions using probabilistic methods

     E.g., π computation
3. Trace-Driven Simulation.
   - A simulation using a trace as its input

     E.g., using trace-driven simulation to analyse paging algorithms, cache analysis, CPU scheduling algorithms, deadlock prevention algorithms, etc.
4. Discrete Event Simulation.

Outline

1. Introduction and classification
   - Types of simulation
2. Discrete event simulation (DES)
   - SimPy
3. Single queue

Definition

A simulation using a discrete-state model of the system.

What is Discrete-Event Simulation (DES) ?

Discrete-Event Simulation is

- Discrete (in state)
- Dynamic (in time)
- Stochastic

DES mostly applied to queueing systems (but not limited to)

- Factory workflow
- Freeway traffic simulation
- Network traffic simulation
- ...

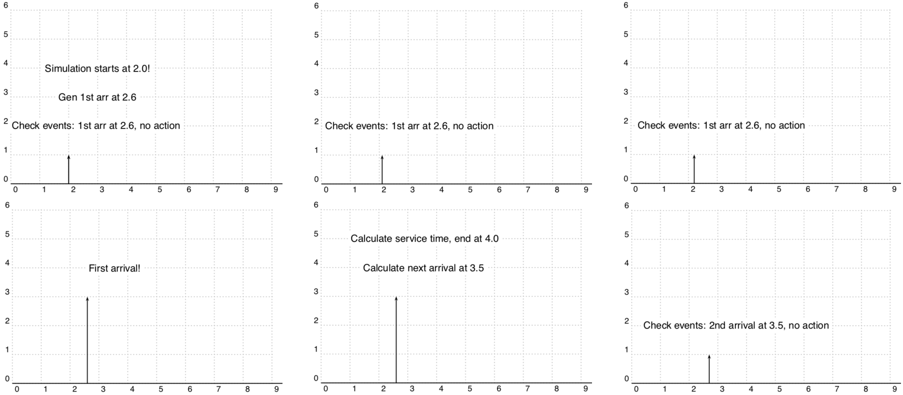
DES categories

- Activity-oriented
  - fixed increment of time
  - time-consuming
- Event-oriented
  - on each event, generate next event and put into event queue and sort
  - simulation time advances to next closed event
  - faster than activity-oriented
- Process-oriented
  - abstract one object into a process
  - easier to maintain as object-oriented approach

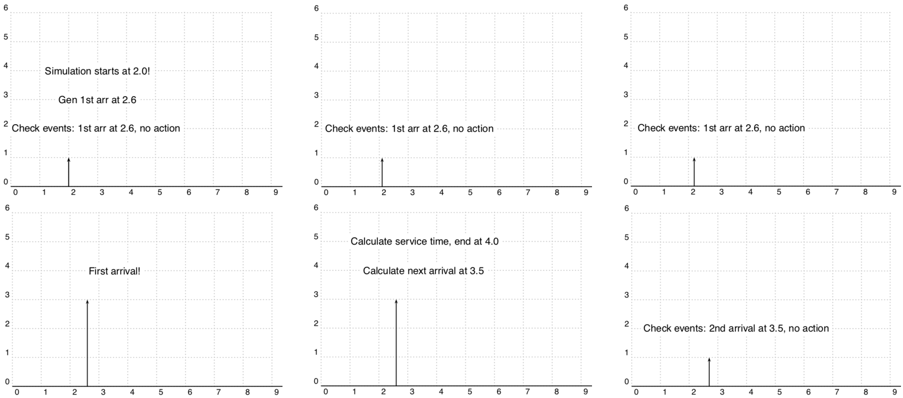
DES categories

- Activity-oriented
  - fixed increment of time
  - time-consuming
- Event-oriented
  - on each event, generate next event and put into event queue and sort
  - simulation time advances to next closed event
  - faster than activity-oriented
- Process-oriented
  - abstract one object into a process
  - easier to maintain as object-oriented approach
  - **SimPy is here**

Activity-oriented

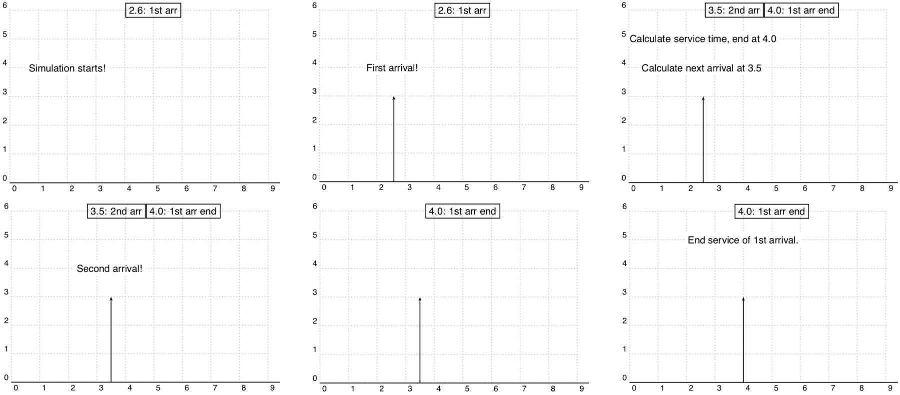


Activity-oriented

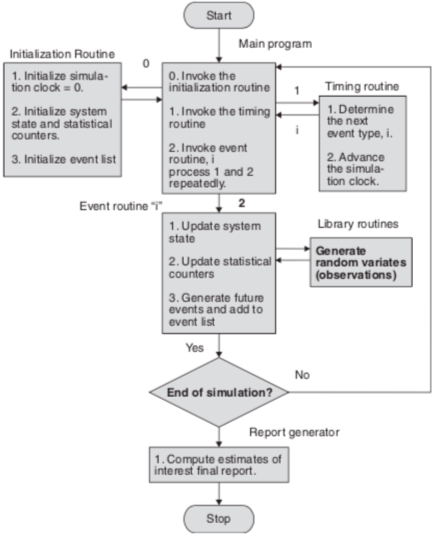


Much time for redundant checks of the unchanged state

Event-oriented



Event-oriented DES architecture



Process-oriented (1)

- Based on event-oriented
- Designed into separate processes

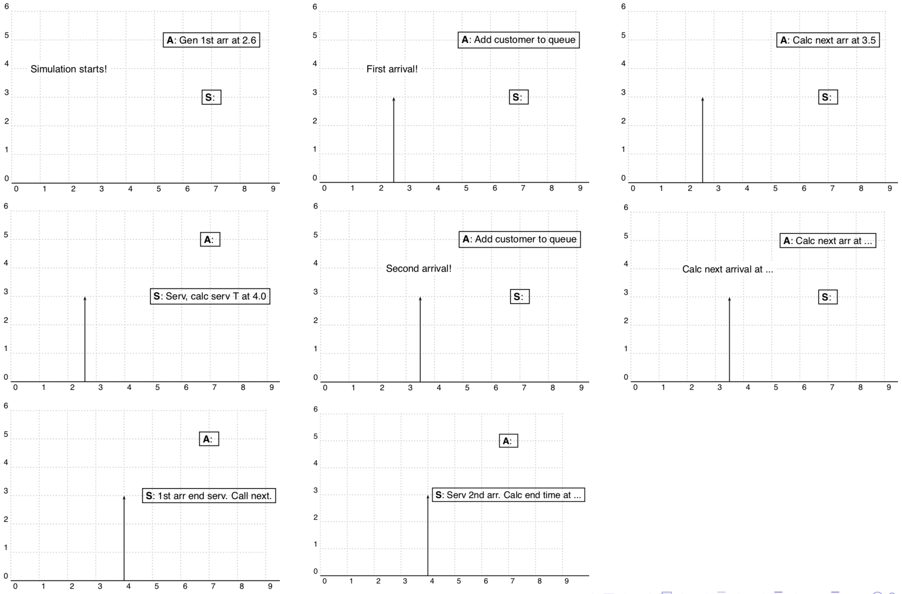
Process-oriented (1)

- Based on event-oriented
- Designed into separate processes

A single queue in process-oriented DES

- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue
- **Service process**: an infinite loop of the following
  - Sleep until waken by new jobs
  - Serve the queued jobs on waken until there is no job in queue

Process-oriented (2)



- Environment
- Process
- Event
- Resource

```
>>> import simpy
>>>
>>> def clock(env, name, tick):
...     while True:
...         print(name, env.now)
...         yield env.timeout(tick)
...
>>> env = simpy.Environment()
>>>
>>> env.process(clock(env, 'fast', 0.5))
<Process(clock) object at 0x...>
>>> env.process(clock(env, 'slow', 1))
<Process(clock) object at 0x...>
>>>
>>> env.run(until=2)
fast 0
slow 0
fast 0.5
slow 1
fast 1.0
fast 1.5
```

Start simpy environment

Register processes with their events

Run simulation until time of 2

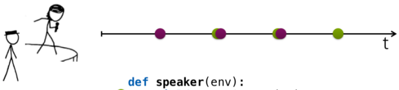
Timeout and processes

(based on slides of Stefan Scherfke)

Timeout is event


```python
def speaker(env, start):
    until_start = start - env.now
    yield env.timeout(until_start)

    yield env.timeout(30)
```

Process is event, too




```python
def speaker(env):
    yield env.timeout(30)
    return 'handout'


def moderator(env):
    for i in range(3):
        val = yield env.process(speaker(env))
        print(val)
```

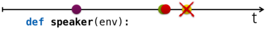
Synchronization
(based on slides of Stefan Scherfke)

Asynchronous interrupt



```python
def speaker(env):
    try:
        yield env.timeout(randint(25, 35))
    except simpy.Interrupt as interrupt:
        print(interrupt.cause)


def moderator(env):
    for i in range(3):
        speaker_proc = env.process(speaker(env))
        yield env.timeout(30)
        speaker_proc.interrupt('No time left')
```

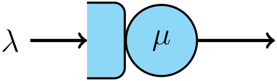
Condition event

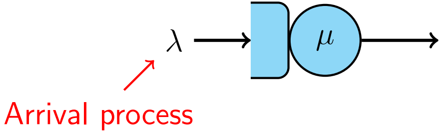

```python
def speaker(env):
    try:
        yield env.timeout(randint(25, 35))
    except simpy.Interrupt as interrupt:
        print(interrupt.cause)


def moderator(env):
    for i in range(3):
        speaker_proc = env.process(speaker(env))
        results = yield speaker_proc | env.timeout(30)

        if speaker_proc not in results:
            speaker_proc.interrupt('No time left')
```

Outline

FIFO queue

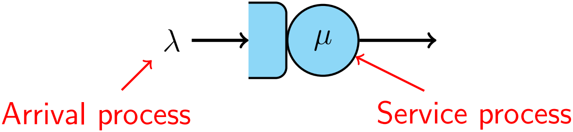


FIFO queue



- **Job**: an object of a class
- **Queue of jobs**: list of objects
- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue

FIFO queue



- **Job**: an object of a class
- **Queue of jobs**: list of objects
- **Arrival process**: an infinite loop of the following
  - Calculate next arrival time
  - Sleep until next arrival
  - Add new job to queue
- **Service process**: an infinite loop of the following
  - Sleep until waken by new jobs
  - Serve the queued jobs on waken until there is no job in queue

```
class Job:
    def __init__(self, name, duration):
        self.name = name
        self.duration = duration
```

More attributes could be added in order to collect statistical data, such as

- arrival time
- start time of service

FIFO queue

A queue with single server (service process)

```
class Server:
    def __init__(self, env):
        self.Jobs = list(())
        env.process( self.serve(env) )

    def serve(self, env):
        while True:
            if len( self.Jobs ) == 0 :
                self.serversleeping = env.process( self.waiting( env ))
                yield self.serversleeping
            else:
                j = self.Jobs.pop( 0 )
                yield env.timeout( j.duration )

    def waiting(self, env):
        try:
            print( 'Server is idle at %d' % env.now )
            yield env.timeout(1000)
        except simpy.Interrupt as i:
            print('A new job comes. Server waken up and works now at %d'
                  % env.now )
```

```
class JobGenerator:
  jcnt = 0

  def __init__(self, env, server):
    self.server = server
    env.process( self.jobgen(env) )

  def jobgen(self, env):
    while True:
      job_interarrival = random.randint(1,5)
      yield env.timeout( job_interarrival )

      job_duration = random.randint(2,5)
      self.jcnt += 1
      self.server.Jobs.append( Job('Job %s' %self.jcnt, job_duration) )
      print( 'job %d: t = %d, l = %d' %( self.jcnt, env.now, job_duration ))

      if not self.server.serversleeping.triggered:
        self.server.serversleeping.interrupt( 'Wake up, please.' )
```

```
env = simpy.Environment()
MyServer = Server( env )
MyJobGenerator = JobGenerator( env, MyServer )
env.run( until = 20 )
```

And results

```
Server is idle at 0
job 1: t = 4, l = 2
A new job comes. Server back to work at 4 by 'Wake up, please.'
Server is idle at 6
job 2: t = 9, l = 5
A new job comes. Server back to work at 9 by 'Wake up, please.'
job 3: t = 13, l = 5
job 4: t = 14, l = 3
job 5: t = 19, l = 4
```

M/M/1 queue

Simulation parameters

- Inter-arrival rate: $\lambda = 5$ (jobs/time unit)
- Service rate: $\mu = 8$ (jobs/time unit)
- Simulation time: 5000 (time unit)

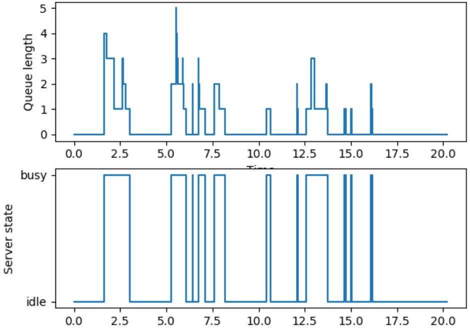
M/M/1 queue

Simulation parameters

- Inter-arrival rate: $\lambda = 5$ (jobs/time unit)
- Service rate: $\mu = 8$ (jobs/time unit)
- Simulation time: 5000 (time unit)

Performance

- Analytical modeling: $\overline{W} = 0.21$ (for validation)
- Simulation modeling: $\overline{W} = 0.19$



- Simulation functions

- Model enhancement

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)

- Model enhancement

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model ($M/M/1/\infty/\infty/FIFO$)
  - Statistical report (mean, variance)

- Model enhancement

M/M/1 queue

What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model ($M/M/1/\infty/\infty/FIFO$)
  - Statistical report (mean, variance)
  - Simulation logging

- Model enhancement

M/M/1 queue

What else should be doned ?

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model ($M/M/1/\infty/\infty/FIFO$)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement
  - Different types of single queue in Kendall's notation M/M/c/K/N/D
    - Queueing strategies (FIFO, SJF, priority, round-robin,...)
    - Different types of workload

- Simulation functions
  - Simulation model verification/validation
    - verification: develop a synthetic workload to verify the model)
    - validation: validate single queue model (M/M/1/∞/∞/FIFO)
  - Statistical report (mean, variance)
  - Simulation logging
  - Simulation plotting
- Model enhancement
  - Different types of single queue in Kendall's notation M/M/c/K/N/D
    - Queueing strategies (FIFO, SJF, priority, round-robin,...)
    - Different types of workload
  - Queueing networks