HCMC University Of Technology
Faculty of Computer Science & Engineering

Assignment 1
The Queuing Systems-based Performance Evaluation Project
Course: System Performance Evaluation

Phuong-Duy Nguyen

September 20, 2023

Contents

Chapter 1

Assignment 1

1.1 Objectives

Build a queuing system application according to a predefined parameter set. The developed queuing system can be built on top of popular libraries. One common library is SimPY, a discrete-event simulator. Build the mapping or translation between your real-life problem parameters and the associated queuing system parameters.

1.2 Project descriptions

1.2.1 Single queuing system

Notations A queuing system is described using Kendall's notation: A / B / m / K / n/ D.

We can verify that the combination of different sets of values represents a different model. The work involves implementing the queuing system with these pre-defined parameters.

- M/M/1 queue
- M/M/1/K queue, system with finite capacity
- M/M/∞ queue
- M/M/n/n queue, Erlang-Lost system
- M/M/n queue
- M/M/c/K queue, multiserver, finite capacity system

An illustration of parameter setting in sample code as listed in the listing 1.1. The translated values from Kendall notation system

- maxServers=m
- maxQueueSize=K
- arrivalRate $\sim$A distribution
- serviceRate $\sim$B distribution

Listing 1.1: Application setting parameters

```
###Server###
# If server capacity is available
self.maxSize = maxQueueSize
# By initialization
self.maxServers = maxServers
self.mean_interarrival_timeTime = 1.0 / arrivalRate
self.mean_service_time = 1.0 / serviceRate
```

1.2.2 Multiple-queuing system

A multiple-queuing system includes many single-queuing systems connected together serially, parallelly, or to form a network. An illustration of a multiple-queuing system is presented in Figure 1.1.

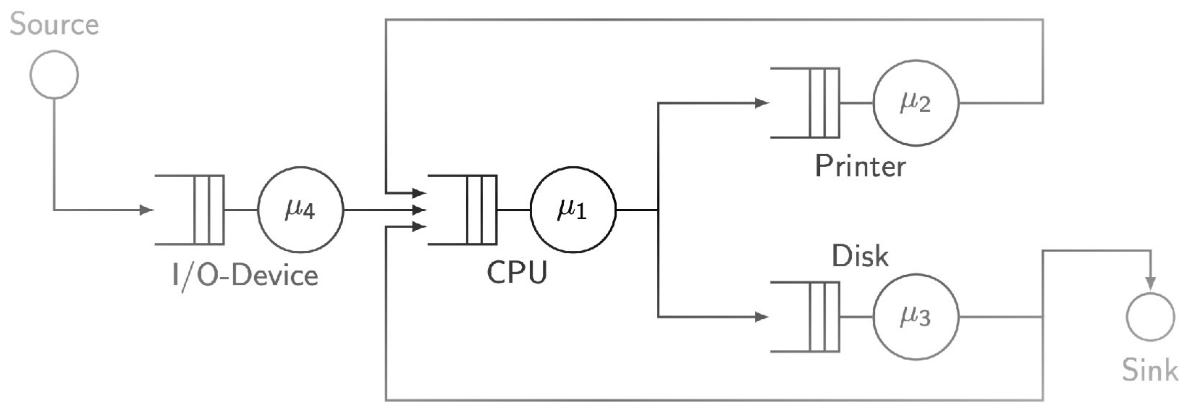

Figure 1.1: An example of multiple-queuing system.

In order to invoke a further event generator inside the current event handling, it is recommended to refer to the following API:

```
class simpy.events.Process(env: Environment, generator: Generator[Event, Any
    , Any])
```

1.2.3 Queuing system reference design

The project referenced design is based on 3 components as in Figure 1.2.

- Arrival Event: generate the arriving data with the given distribution and parameter setting (i.e. mean_inter_arrivaltime).
- Server: the number and capacity of the server are associated with the 3 or 4-tuple param.
- Processing Event: create event for notification or logging in event processing step.

In general, the application is called by 2 basic steps:

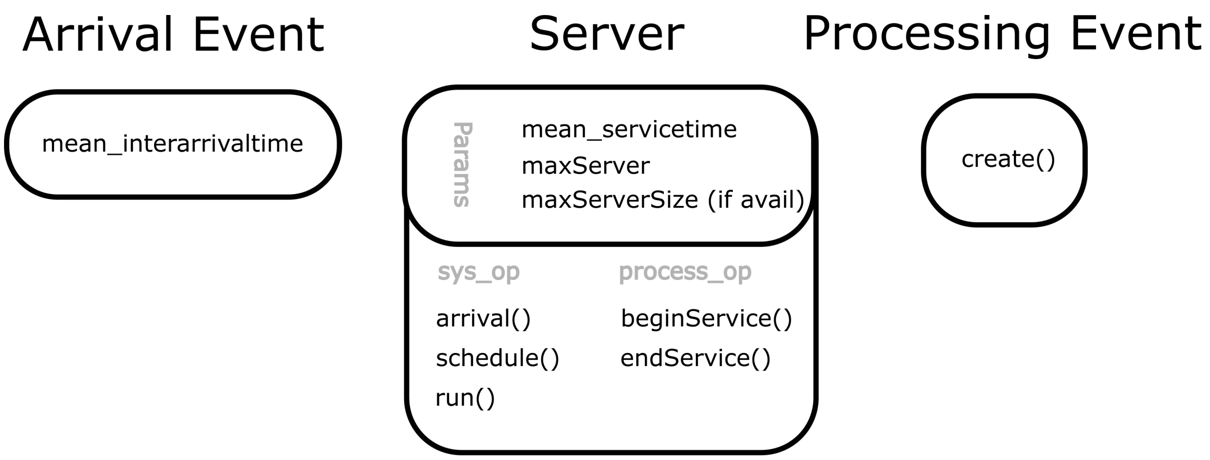

Figure 1.2: Referenced Queuing System.

```
sys = MMcKn(...)
sys.run()
```

The (input) data generation will be performed in the first step of the system declaration, in which the step init() will be triggered. The execution of run(), in turn, will activate the system.run() to start the event processing or scheduling to be performed.

Event list the sample code

```
event_list = PriorityQueue()
```

Event list/queue may be unlimited size (with NULL or 0 size) or be limited by the input number of queue length.

Operations

init() the sample code

```
def init():
    while event < n event s :
        inter_arrivale_time = np.random.exponential(self.
            mean_interarrival_timeTime)
        next_arrival_time = next_arrival_time + inter_arrivale_time
        self.schedule(self.arrival ,...)
```

arrival() the sample code

```
def arrival()
    self.customer_number += 1
    if self.numAvailableServers > 0:
        serverid = getServer()
        self.schedule(event=self.beginService,serverid, delay=0)
```

This is a prototype-like suggestion, but it is not limited to this design.

schedule() this handles the occurred event.

```
def schedule(event,serverid):
    if validate(serverid) = OK:
        self.Event(event,...)
```

If there is no Kendall K=maxQueueSize argument then validate(serverid) is always OK

beginService() this starts the processing of the event that occurred. It can be performed immediately or postponed to another schedule.

```
def beginService(self):
    self.numAvailableServers -= 1
    self.schedule(self.endService,serverid, delay=np.random.exponential(self.
        mean_service_time))
```

endService() finalize the processing step.

```
def endService(self):
    self.numAvailableServers += 1
```

1.2.4 Build the mapping/translating between the problem and the associated model

Some sample projects We provide some suggestions, but do not limit your problem to these samples. At this stage, we are currently limited by one type of model, the queuing system. So you need to figure out whether the tackling problem is compatible with the associated queuing system. The associated system is mandatory to be a multiple-queuing system; simple M/M/1 is not approved. Each single-queuing system is encouraged to be a multiple-server M/M/c at least.

Here is the suggestion list:

1. Write a simulation of hierarchical paging memory allocation; it might include many cache levels, external storage, etc.

2. Write a simulation of contiguous memory allocation with variable partitions; it might include many cache levels, external storage, etc.

3. Write a simulation of multiple feedback queues in CPU scheduling; it can employ the internal application scheduler among threads in the whole application or sub-module in each application.

4. Write a simulation of indexed allocation (based on the i-node of UNIX) of a file system. A distributed file system with multiple levels of storage space is a good case study.

5. Write a simulation of packet arrival processing at a gateway router in a multi-hop network.

6. Write a simulation of buffet self-service at multiple entrances to different food sections' tables.

7. Write a simulation of a multiple dentist chair/minor surgery system. It is not limited to empowering your topic by increasing server capacity by more than one, such as in a healthcare check-up system where many slots can be processed at each department.

8. Write a simulation of a phone teller system employing multiple departments, i.e., customer service, product guarantee team, and technical support. For more variants of system settings, you can employ the up-to-date robotics/AI-enabled teller to increase the system capacity.

Mapping the input data and system parameters to the associated queuing system To make a systematic map, you need to ensure that your work satisfy some basic requirements, which include:

1. Following the steps fora performance evaluation project.

2. Technique to be used is simulation.

3. Selecting at least 2 kinds of workload to test the simulation models

4. Analyzing experimental data and interpreting it.

5. All code and report are written in a Jupiter Notebook.

Mapping the output simulation measurement and mathematical formula calculation Since we are learning, we need to verify and match all our estimations in both simulation and mathematical ways. This step helps you make a co-relation between the two different kinds of estimations.

In this project, we design for a large group, so the workload must be proportional to the number of group members. We aim to get the outcomes of superior design, whereas:

- Support the stress test with a high number of data points to make the probability approach the expected numbering. Many formulas are based on the law of Little then they should be performed well when the time goes to infinity.

- Separable testing of each part of the system since our base is multiple-queuing should be able to do partial tests on each elemental queue.

- Support automation integration as we work in a large team; each member's work can be performed individually, and the integration needs to be well designed.

- Support distribution management among components of the system is a plus.

- Events are traversed between or among queues, just as signals pass. Support for signal management, i.e., redundant, intermediate canceling, is a plus.

- Testing members in the group should show one superior and at least 2-3 limitations of your design.

1.3 Working plan

Working in a team

- Each group has up to 6 members

- Phase 1 (First 2 weeks):

  - Define specifiction of the queuing system.
  - Define the interfaces and params/argument passing used for each function.

- Phase 2 (Next 2 weeks):
  - Implement and refine the application according to the functions and protocols defined in Phase 1.

1.4 Results

- Phase 1 (softcopy):
  - Submit the report file of the first phase to shared folder
  - File format is ASS1-P1<<**Group_name**>>.ipynb (Jupiter Notebook) or pdf
  - This report includes:
    * Define and describe the functions of the application.
    * Define the programming interface used for each function.
- Phase 2 (hard and soft copy)
- Submit the report file of the assignment to shared folder
- File format (.rar) ASS1-<<**Group_name**>>.rar
- This report includes:
  - Phase 1 content
  - Describe each specific function of the application
  - Detailed application design (architecture, class diagrams, main classes,dots)
  - Validation (sanity test) and evaluation of actual result (performance)
  - Extension functions of the system in addition to the requirements specified in section 2
  - Participants' roles and responsibilities
  - Manual document
  - The report must be all designed illustration or (abstractful) prototype of the code. All copied code to the report document will has penalty
  - Souce code is separated in a different folder.

1.5 Evaluation

Time The submission is only accepted the **announced deadline** on **BKeL**.

Scoring Assignment 1 is worth 50% of the assignment grade.