Networks & Operating Systems Essentials 2 (NOSE 2)

Tutorial for Assessed Exercise 2

The aim of the 2nd exercise is to have students use the knowledge they've acquired so far in the second part of the course, in the context of implementing a set of scheduling algorithms. They will be using and extending a simple discrete event simulator written in Python, by creating functions that simulate the scheduling and dispatching of processes performed by four scheduling algorithms: First-Come-First-Served (FCFS), Shortest-Job-First (SJF), Round-Robin (RR) and Shortest-Remaining-Time-First (SRTF). This tutorial will be a gentle introduction to such concepts as queuing theory and discrete event simulation.

Queueing Theory Primer

Queuing Theory is a discipline of mathematics (specifically, of probability theory) that studies waiting lines, also known as queues. Queues (sometimes also referred to as *queuing nodes*) are usually described using a notation of the form X/Y/Z, where X denotes the distribution between arrivals of successive jobs to the system, Y denotes the distribution of job sizes, and Z indicates the number of servers for the queue. Given a queue, a model is then constructed to allow for prediction of statistics on the queue length and waiting times.

One of the simplest and most frequently used models is the M/M/1 queue. In this case, the inter-arrival time follows a Poisson process with rate parameter λ (i.e., on average λ jobs will arrive per time unit), the job sizes follow an exponential distribution with rate parameter μ (i.e., on average each job will require $1/\mu$ time units), and there is only one server for the queue.

The system is said to be *stable* if $\lambda < \mu$; that is, if jobs arrive less frequently than they are completed. In this case, one can compute the *stationary distribution* – that is, the limiting distribution over an infinite amount of time – and therefore deduce various performance measures. Let $\rho = \lambda/\mu$. Then, for example, the average number of jobs in the queue is given by $\lambda/(1-\lambda)$, the average waiting time is given by $\rho/(\mu-\lambda)$ and the average response time is given by $1/(\mu-\lambda)$.

The simple discrete event simulator you will be provided with, already features an implementation of a M/M/1 queue. That is, new processes are added to the system following a Poisson process, process service times follow an exponential distribution, and there is supposed to be only one CPU (i.e., only one process should be active/running at any time).

Discrete Event Simulation Primer

A discrete event simulation (DES) simulates the operation of a system via a series of discrete events ordered in time. That is, given a set of events ordered by time of occurrence from earliest to latest, the internal clock of the simulator does not take on a continuum of values but rather jumps from one event to the next. The simulated system is considered to be in a constant state in between any pair of successive events, while every event triggers a possible change in the system's state. The DES, in essence, implements the following algorithm:

```
Start:
      # Initialise ending condition to FALSE
      # In this case, populate a queue with all arrival events,
      # ordered by time
      Queue event_queue = ...

      # Initialise system state variables
      # Could include initializing statistics, clock, etc.
      system_time = 0

Simulation loop:
      # While termination condition is FALSE
      while event_queue is not empty:

            # Select next event and advance system clock
            current_event = event_queue.remove_first_event()
            if current_event.time > system_time:
                  system_time = current_event.time

            # Service the event and update statistics
            service(current_event)

End:
      # Generate/print statistics
      # ...
```

where `service(current_event)` can potentially advance the internal clock and/or end up adding new events to the queue.

For the sake of this tutorial, try to play out the example below. Think of a roadside canteen selling hot food. People arrive and queue up for their turn to be served. When their turn comes, they describe what they want to eat and then wait for it to be prepared. Assume it takes a fixed amount of time (1') to place their order, but its preparation depends on what exactly they ordered. As such, after placing their order, they stand to the side waiting to be called to collect their food. When called, they instantly take their food and leave. Assume there is infinite amount of space around the canteen (so that no restriction is imposed on how many people can queue up), and that the food is prepared on its own.

Now consider the following clients:

Client no.	Time of arrival	Time to prepare order
1	13:00	1'
2	13:00	4'
3	13:03	1'

In small teams or on your own, think:

- What types of events would you expect to have in a system like this?
- What would the service(…) function look like?

Then, play out the simulation on paper and compute at what time each client left.