Technical University of Munich, Chair of Communication Networks
Prof. Dr.-Ing. W. Kellerer, Alberto Martínez Alba, Mu He

Analysis, Modeling and Simulation of Communication Networks
SS 2018

Programming Assignment - Part 1

Submission Deadline: 1. May 2018

1 Implementing the Discrete Event Simulator (130 Points)

In the first part of the exercise, you will create a Discrete Event Simulator step by step. This section consists of four parts:

- Modeling Server and Queue
- Implementing the Event Chain
- Processing the Events
- Creating the Simulator

After that, you will verify your system and answer questions on your implementation.

1.1 Modeling Server and Queue

In this part, you will implement the server and the queue. The queue (buffer) stores customers (network packets or shortly packets) that can not be served directly, thus, have to wait for other customers (packets) to be served first. This means the service unit has to finish the operation on the customer (packet) first. To keep your first implementation simple, you will not have to work on a detailed model for network packets in this assignment.

In general, for your implementation, you need an indication (flag) showing whether the server is busy and a counter showing how many customers are currently waiting in the queue. The queue size can be set, thus, be limited by the simulation parameter S. Now, implement the following tasks in the *SystemState* class in the file (Python module) named *systemstate.py*.

Task 1.1.1: Server modeling

5 Points

Create a new boolean variable *server_busy* that indicates whether the server is busy or not. Create a second variable *buffer_content* that indicates the status of the queue (buffer), i.e., how many customers (packets) it contains. Initialize both variables using plausible values in the function *init()*.

Task 1.1.2: Adding Packets to the System

5 Points

Modify the functions *add_packet_to_server()* and *add_packet_to_queue()*. The first function should return *true* if an arriving customer (packet) can be served directly without waiting. The second function should return *true* if the arriving customer (packet) cannot be served directly and has to be added to the queue. Modify the values of the variables that you have generated in the previous task accordingly. Note that you can access (read the value of) the parameter *S* through the member variable *sim*.

Task 1.1.3: Service Completion

5 Points

Modify the remaining two methods *complete_service()* and *start_service()*. The function *complete_service()* is called when the packet is processed completely by the server, i.e., the service is completed. The function *start_service()* returns *true* if the queue is not empty and another packet is taken from the queue and served directly by the server. Modify the variables from task 1.1.1, if needed, accordingly.

1.2 Event Chain

Your DES is based on an event chain that stores all different *SimEvents*. Each *SimEvent* has a time stamp showing the time when it is scheduled, and a priority indication to make sure that when multiple events occur at the same time, they will be processed in the correct order.

In this part, you will implement the functionality of the event chain, that is, inserting and removing events in the right and correct manner. For this, you have to modify two classes that you both find in the file *event.py*: *SimEvent* and *EventChain*.

Task 1.2.1: Comparing two events

10 Points

In order to correctly insert a *SimEvent* into the event chain, the events have to be compared with each other. We use the structure *heapq* for the event chain (`https://docs.python.org/2/library/heapq.html`). Whenever an event is inserted in our event chain, they are sorted by their time and priority. The earliest events are always processed first. If two events happen at the same time, the one with higher priority, i.e., lower priority value, is processed firstly. Implement the function *__lt__()* according to the above specification. Have a look at the python documentation, how it is used. Hint: this is called operator overloading.

Task 1.2.2: Inserting and removing events

10 Points

Modify the functions *insert()* and *remove_oldest_event()* in the module *EventChain* according to the above functionality descriptions of the event chain. The variable *event_list* should be of type *heapq* and would be modified, whenever one of these two functions is called.

1.3 Processing of Events

In the given DES there are three types of events: Customer Arrival, Service Completion and Simulation Termination. In this part you should implement the processing of each event. The following tasks should be implemented in the module *SimEvent* in the file named *event.py*. In each class inheriting from the the abstract class SimEvent, you should implement the process method.

Task 1.3.1: Simulation Termination

5 Points

This event is only processed at the end of a simulation run. Adapt the *process()* function to make sure that the simulation stops when this event occurs. The function should modify the parameters in class *SimState* for this.

Task 1.3.2: Customer Arrival

20 Points

This event should occur once a new customer (packet) arrives in your system. Think about what should happen when a customer arrives and implement the corresponding function. In detail, your function should either serve, enqueue or drop the customer (packet), depending on the current system state. For statistics, do not forget to call the functions *packet_accepted()* and *packet_dropped()* in module *SimState* whenever it is necessary. These functions count how many packets are dropped and how many packets have come into our system. Every packet should be counted in the correct way. Make sure that each CustomerArrival event inserts a new CustomerArrival into the event chain and a Service Completion event, if necessary. The inter-arrival-time between two CustomerArrival events is fixed and the value is stored in *SimParam*. The serving time should be a uniformly distributed random integer between 1 and 1000. Use the function *random.randint()* for that and set your random seed in *SimParam* to your matriculation number (`https://docs.python.org/2/library/random.html`).

Task 1.3.3: ServiceCompletion

20 Points

Whenever a packet has been served completely and leaves your system, this event will be called. Think about how its process function should behave when there are still packets in the queue (buffer) of your system and when the queue (buffer) is empty. Note that you may have to insert new events into the event chain in the processing as well. Note again that the serving time should be random as described in the previous task.

1.4 Creating the Simulator

Now you are ready to setup the whole simulator and make a simulation run. For this, the function *do_simulation()* in the python file *simulation.py* has to be modified. You see that the first and the last event of your simulation have already been inserted. All other events are

inserted on the fly according to the implementation. The basic structure contains a while loop that runs until the *simstate.stop* flag is set. In each iteration of the while loop, only one event is processed and the simulation time (*simstate.now*) is updated accordingly.

Task 1.4.1: Creating the Simulator

10 Points

Modify the above-mentioned function *do_simulation()* such that the simulator processes the events in the right and correct manner. Pay attention here and think carefully about how and when to update the simulation time, since this can strongly affect the way your simulation works.

1.5 Verification

Since you have setup the simulator, it's time to verify your simulation results. We provide you a python unittest file that checks the basic functionality of your code. The unittests in file *part1_tests* check the functionalities of different SimEvents, EventChain and SystemState. Furthermore, it can run whole simulations with the seed ranging from 0 to 5, helping you to check the correct implementation of your DES.

When you run a simulation with a maximum queue size of 4 ($S = 4$) for 100 seconds, the system should count roughly between 5 and 20 dropped packets. Try different seeds to check whether this is the case in your simulation. Run all unittests and make sure they do not return any errors.

1.6 Analysis and General Questions

The last section of the first part of the programming assignment contains a few questions that should be answered separately. Explain your answers and write full sentences.

Task 1.6.1: Confidence

5 Points

In the last section you have run your simulation and received a result. How could you establish statistical confidence in this case?

Task 1.6.2: Event Chain Structure

5 Points

Explain why it makes sense to use a heap as your data structure for the event chain.

Task 1.6.3: Update Event Chain

5 Points

Explain why it makes sense to update the event chain during simulation instead of inserting all events at the beginning of simulation. Consider both types of events, Customer Arrivals and Service Completions.

1.7 Simulation Study I

In the previous tasks, you have created a basic version of a DES. Now, perform a little simulation study. Use the given functions in the files *part1_simstudy.py* and write your code in these files. Explain your answers and write full sentences.

Task 1.7.1: Queue Length Determination I

10 Points

Find out which queue length is required such that after 100 s less than 10 packets are lost in at least 80% of 1000 simulation runs. Describe your idea how to design the simulation runs to achieve this goal. Think about cases, when you are close to fulfilling or not fulfilling the requirements and how you could assure a correct result in these cases.

Task 1.7.2: Queue Length Determination II

5 Points

Does the queue size depend on the simulation time? What happens if you simulate for 1000 s and want to have less than 100 dropped packets in 80% of the cases? Describe your observations!

Task 1.7.3: Comparison of Results

10 Points

Compare the results of task 1.7.1 and task 1.7.2. Does the system behave differently or not? Explain your observations! Hint: Think about the distribution of the blocking probability per run. Possible solution: Plotting cumulative distribution functions of the blocking probabilities might be helpful.

Total: 130 Points