The University of Melbourne
School of Computing and Information Systems
SWEN90004 Modelling Complex Software Systems

Assignment 1a, 2024

Released: Friday 1 March, 2024. *Deadline:* 23:59, Friday 22 March, 2024

Objectives

The objectives of this assignment are: to convert a description of a system into a simulation model of that system; to implement that simulation in a shared memory concurrent programming language; to use the implemented simulation to explore the behaviour of the system; to gain a better understanding of safety and liveness issues in concurrent systems.

Background and context

There are two parts to Assignment 1, which is worth 25% of your final mark. This first part of the assignment (ie, this part) deals with programming threads in Java, and is worth 12.5% of your final mark. The second part of the assignment (to be released in a few weeks) deals with modelling in FSP, and is also worth 12.5% of your final mark. Your task is to implement a concurrent simulation of a hospital emergency department.

The system to simulate

An emergency department (ED) is a part of a hospital provides emergency care to patients who need urgent medical attention. A typical (simplified) view of an emergency department is as follows (italicised numbers below correspond to labelled transitions in Figure 1):

Patients arrive at the **Foyer** of the ED, where they are admitted (*1*). A **Nurse** is allocated to a Patient and takes them for **Triage** (*2*) where they are examined and the severity of their injury or illness is assessed to determine their treatment pathway. Patients who are assessed as *severe* will be taken (by their allocated Nurse) for **Treatment** (*3*) by a **Specialist**. All other Patients will be returned to the Foyer (*4*), where they will be released from their allocated Nurse and discharged from the ED (*6*). Once Patients who are assessed as *severe* have been treated by a Specialist, they too will be taken to the Foyer (*5*), released, and discharged (*6*).

Whenever a Nurse takes a Patient from one location to another (ie, between Foyer, Triage, or Treatment; transitions *2–5* in Figure 1) they must be assisted by a number of support staff known as **Orderlies**. Note that a Nurse is allocated to a Patient for the entire duration of their stay in the ED (ie, from the time they are admitted to the time they are discharged). However, Orderlies are only required when a Patient is being transferred from one location to another. Once the Patient enters a location, the Orderlies are released to assist other Nurses/Patients. Orderlies are *not* required for admission to the ED or discharge from the ED.

There is only one Specialist, who also has duties elsewhere in the hospital. Therefore, they leave the ED to attend to these other duties after treating each Patient, returning after period of time.

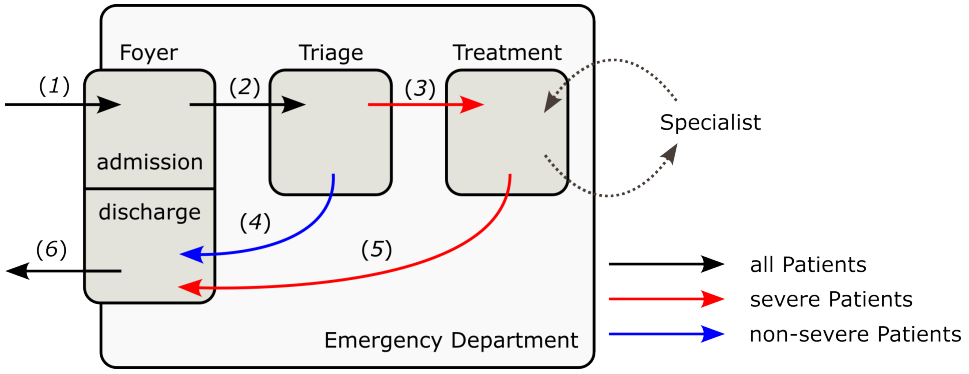

Figure 1: A schematic of the system, showing the flow of admitted Patients to the Foyer (*1*), where they are allocated to a Nurse, from the Foyer to Triage (*2*), where they are assessed and either returned to the Foyer (*4*) or taken for Treatment (*3*) by a Specialist before being returned to the Foyer (*5*). In the Foyer they are released from their Nurse and discharged (*6*). The Specialist enters and leaves the Treatment room in between completing duties in other parts of the hospital. For transitions (*2*)–(*5*), Patients must be allocated to a Nurse, who is assisted by Orderlies.

Your tasks

Your first task is to implement a simulator for the system described above. It should be suitably parameterised such that the timing assumptions can be varied, and the number of Nurses and Orderlies can be varied, as can the number of Orderlies required to assist a Nurse in transferring a Patient. You should use your simulator to explore the behaviour of the system to identify any potential problems with its operation.

You should make the following assumptions:

1. Only one Patient at a time can be in Triage.
2. Only one Patient at a time can be in Treatment.
3. A Patient *must* be allocated to a Nurse before they can leave the Foyer.
4. A Patient is only ever allocated to one Nurse.
5. A Nurse can only have one Patient allocated to them at a time.
6. The Foyer may contain one newly admitted Patient and one Patient waiting to be discharged at the same time (ie, it is still possible for a new Patient to be admitted to the ED even if there is still a Patient waiting to be discharged).
7. The Specialist leaves the Treatment location in between treating each Patient.

Note: you **may** *observe that the system does not behave perfectly if you implement it as described above—you are* **not** *required to fix any observed problems as part of this assignment, though you are encouraged to think about how you might do so.*

The simulator should produce a trace of events matching that below (here shown in two columns). As the behaviour of the simulator is non-deterministic, the output from your code won't match this exactly, but the format of the trace messages should match. In this run, there are 4 Nurses and 10 Orderlies, and 3 Orderlies are required to assist a Nurse for each Patient transfer. These Orderlies are recruited before each transfer between locations, and

released once the transfer is complete. In the trace below, we can see that, following admission to the ED, Patient 1 is allocated to Nurse 1, taken to Triage, and assessed as severe (indicated by "`(S)`"). They hence continue to the Treatment room, where the Specialist is already present. Patient 2 is allocated to Nurse 3, but in Triage is assessed as not-severe, and returns to the Foyer, where they are discharged. Once Patient 1's treatment is complete, they too return to the Foyer and are discharged. At the end of the trace, a third Patient is admitted to the ED and allocated to Nurse 4, and the simulation continues.

```
Patient 1 (S) admitted to ED.
Patient 1 (S) allocated to Nurse 1.
Nurse 1 recruits 3 orderlies (7 free).
Patient 1 (S) leaves Foyer.
Patient 2 admitted to ED.
Patient 2 allocated to Nurse 3.
Nurse 3 recruits 3 orderlies (4 free).
Specialist enters treatment room.
Patient 2 leaves Foyer.
Patient 1 (S) enters triage.
Nurse 1 releases 3 orderlies (7 free).
Nurse 1 recruits 3 orderlies (4 free).
Patient 1 (S) leaves triage.
Patient 2 enters triage.
Nurse 3 releases 3 orderlies (7 free).
Patient 1 (S) enters treatment room.
Nurse 1 releases 3 orderlies (10 free).
Nurse 3 recruits 3 orderlies (7 free).
Patient 1 (S) treatment started.
Patient 2 leaves triage.
Patient 2 enters Foyer.
Nurse 3 releases 3 orderlies (10 free).
Patient 1 (S) treatment complete.
Specialist leaves treatment room.
Nurse 3 releases Patient 2.
Patient 2 discharged from ED.
Nurse 1 recruits 3 orderlies (7 free).
Patient 1 (S) leaves treatment room.
Patient 1 (S) enters Foyer.
Nurse 1 releases 3 orderlies (10 free).
Nurse 1 releases Patient 1 (S).
Patient 1 (S) discharged from ED.
Patient 3 (S) admitted to ED.
Patient 3 (S) allocated to Nurse 4.
Nurse 4 recruits 3 orderlies (7 free).
        :
```

A possible design and suggested components

The various locations in the emergency department (Foyer, Triage, Treatment) could be treated as **monitors**, and the Nurses and Specialist as **active processes** (i.e., Java Threads). Orderlies could be considered as a shared resource used by the Nurses.

We have made some scaffold code available on LMS, which provides:

`Producer.java` Generates new Patients for admission to the ED.

`Consumer.java:` Removes Patients who have been discharged from the ED.

`Patient.java:` Patients can be generated as instances of this class; each patient will have a probability of having a Severe condition.

`Params.java:` A class which, for convenience, gathers together various system-wide parameters, including time intervals.

`Main.java:` The overall driver of the simulation. Note that this won't compile until you have defined some additional classes.

System parameters

The class `Params.java` contains the system parameters, including the number of Nurses and Orderlies, the number of Orderlies required to assist a Nurse, as well as several timing parameters. Varying these parameters will give you different system behaviours. Once you have a working simulator, you should experiment with the parameter settings to observe how the system behaves under different scenarios.

Tips

Some advice for approaching this assignment: Do not attempt to implement the whole system at once—testing it will be too difficult. Rather, I suggest creating a simple version and then incrementally adding complexity. First, create a version of the scaffold code that can compile (ie, by commenting out unimplemented components from `Main.java`). Then implement enough functionality (eg, the Foyer) to enable Patients to move through the system. Then add other locations, perhaps allowing Patients to move through the ED on their own (which in the final system they cannot do). Once this version is working, then start thinking about how to incorporate the constraints related to the Nurses, Orderlies and Specialist.

Reflection

Your second task is to write a brief reflection of **approximately 500 words** that addresses the following questions:

1. Briefly, what behaviours did you observe in your simulation?
2. What, if any, potential problems did you observe in the behaviour of your simulation?
3. If you did identify potential problems, what was the source of these?
4. What were key design decisions or problems that you confronted in designing and implementing your simulation?
5. What other insights did you gain from building and experimenting with the simulation?

Procedure and assessment

- The project must be completed **individually** and all submitted work must be your own work.
- You should submit a single `zip` file via LMS. The `zip` file should include (1) a directory containing all Java source files needed to create a file called `Main.class`, such that "`javac *.java`" will generate `Main.class`, and "`java Main`" will start the simulator; and (2) a plain text file `reflection.txt` containing your reflection. Please ensure that this is a *plain text* file. **All source files and your text file should contain, near the top of the file, your name and student number.**
- 1 mark per *calendar* day will be deducted for late submissions. If you have a valid reason for submitting late, email Nic *well before the due date* to discuss.
- Marks will be awarded according to the following guidelines:

Criterion	Description	Marks
Understanding	The submitted code is evidence of a deep understanding of concurrent programming concepts and principles.	3 marks
Correctness	The code runs and generates output that is consistent with the specification.	3 marks
Design	The code is well designed, potentially extensible, and shows understanding of concurrent programming concepts and principles.	3 marks
Structure & Style	The code is well structured, readable, adheres to the code format rules (Appendix A), and in particular is well commented and explained.	2 marks
Reflection	The reflection document demonstrates engagement with the subject content and project.	1.5 marks
Total		12.5 marks

Nic Geard
29 February 2024

A Code format rules

Your implementation must adhere to the following simple code format rules:

- Every Java class must contain a comment indicating its purpose.
- Every method must contain a comment at the beginning explaining its behaviour. In particular, any assumptions should be clearly stated.
- Constants, class, and instance variables must be documented.
- Variable names must be meaningful.
- Significant blocks of code must be commented. However, not every statement in a program needs to be commented. Just as you can write too few comments, it is possible to write too many comments.
- Program blocks appearing in if-statements, while-statements, etc., must be indented consistently. They can be indented using tabs or spaces, and can be indented 2, 4, or 8 spaces, as long as it is done consistently.
- Each line must contain no more than 100 characters (Google Java Style Guide).