

Operating Systems Coursework 1 (30%)

Dr James Stovold

Released: Mon Week 5

Due: Mon Week 7, 3pm German Time

1 General Rules

This assessment is covered under the the relevant sections of Lancaster University's Manual of Academic Regulations and Procedures^[1] of particular relevance are the '*General Regulations for Assessment and Award*' and '*Academic Malpractice Regulations and Procedures*'.

Late submissions will be downgraded as per these regulations. This is an **individual** assessment, so your work must be your own, and you are not permitted to discuss the problem or request help from other students. Remember: if you are unsure about whether something constitutes academic misconduct, it is better to ask than to guess.

Note that the use of generative AI tools (such as ChatGPT or similar) is not permitted for this assessment and will constitute plagiarism under the current Lancaster regulations. This includes its use for either the code or the report.

Questions about the assessment should be submitted to James via email. For fairness, the (anonymised) question and answer will then be sent via email to the entire cohort.

2 Task

Your task is to design part of a robot controller. The robot has various sensors which provide input to the controller, and various actuators to which the controller can provide output. The controller needs to be able to handle the task workflow (sensing, analysing, actuating) using a concurrent program.

2.1 Task workflow

The task workflow is the part of the robot controller which completes tasks. For example, a packing robot would have tasks related to putting things in boxes and then taping them up, whereas a cleaning robot would have tasks related to wiping or hoovering the floor.

In our case, the robot controller has an 'analyse' step which accepts a 'task' from the sensors, analyses it internally, and then passes the 'result' to the actuators (see fig. [1](#)). If there are no tasks arriving from the sensors, the analyser won't have any results to pass to the actuators. If the analyser has too many tasks incoming, the sensors have to wait until the analyser has worked through

¹<https://www.lancaster.ac.uk/academic-standards-and-quality/marp/>

some of its backlog. While the analyser is analysing (or the actuator actuating) the incoming tasks/results need to be stored until it is time to consume them.



Figure 1: Diagram depicting the task workflow.

You are required to implement the task workflow in Java using appropriate concurrent programming techniques. The three stages of the workflow should operate concurrently, and no tasks or results should be lost or corrupted. Tasks contain two variables: c , a single real-valued number corresponding to the complexity of the task ($0.1 \leq c \leq 0.5$), and id which is a unique identifier of the task (incremented each time a new task is created).

Sensor Tasks should be generated by your sensor at a rate which corresponds to a Poisson distribution with λ provided as an input to the program (see section 2.4). As a starting point, you can assume that $\lambda = 2$. In other words, you will have to produce a random number of tasks every second. In any given second, the probability of producing k tasks will be:

$$P(k \text{ tasks in 1sec interval}) = \frac{\lambda^k e^{-\lambda}}{k!}$$

The complexity of the task should be randomly generated following a uniform random distribution across the range described above.

Analysis Your analysis step should ‘analyse’ the incoming tasks, producing results to pass to the actuators. This analysis takes the same amount of time as the complexity of the task c (in other words, have the analysis step sleep for c seconds while it performs its ‘analysis’).

Finally, your analysis should produce a ‘result’ which is sent to the ‘actuate’ step. The result should contain the id and c from the original task, along with the outcome of the analysis \mathbb{Y} , which can be calculated using the following formula:

$$\mathbb{Y} = \sqrt{\frac{1}{c}}$$

Actuator The actuator step accepts the result of the analysis, and uses the result to move the robot by updating a floating-point variable called ‘position’ ($0 \leq position \leq 1$). Once the robot’s position reaches 1.0 or 0.0, it will turn round and continue back in the other direction (as if there is a wall at each end). When it reaches the wall, if it still has some distance to travel (as in the bottom example in fig. 2) then it will continue in the other direction to travel the correct total distance (you will need to work out how to decide if the robot is facing left or right).

Each time the actuate step processes a result, it should print to stdout a single line with the following details (completed as appropriate):

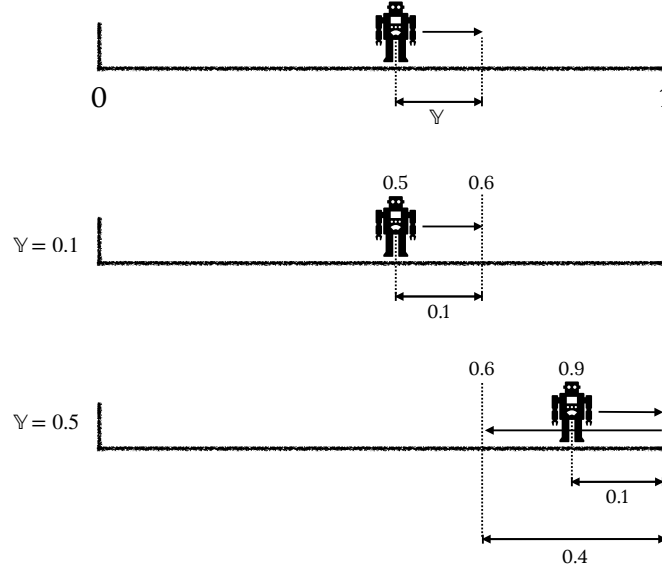


Figure 2: Three diagrams depicting the robot moving around its one-dimensional arena. The left and right bound of the arena space are 0.0 and 1.0 respectively. If Y is more than the remaining space in the arena (as in the third diagram), then the remaining distance is covered in the other direction. (Diagrams not to scale)

Robot moving. Task id {id}, task complexity {c}, result {Y}, old position: {pos}, new position: {pos}.

Every time one of your three steps is unable to do something (for example, if there are no results for the actuator, or there are too many results waiting to be processed so new results can't be added), then you should output an error message similar to the following (depending on the problem):

Actuate error: no results to process. Last task processed {id}.

2.2 Multiple Sensors

For the second part of this task, you need to extend your task workflow so that it can handle multiple sensors (i.e. fig. 3). Each task will now include a sensor ID, which specifies which sensor generated it. The logs produced by the actuator should be updated to reflect the extra information.

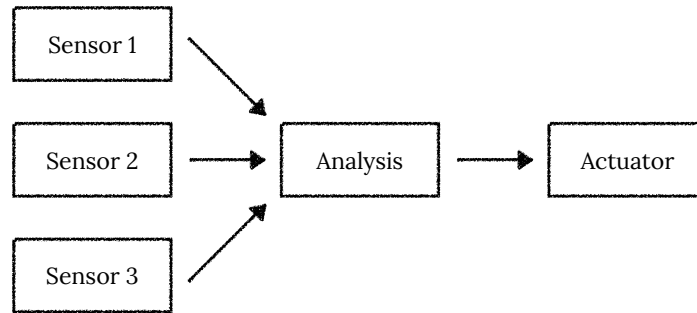


Figure 3: Multiple sensors

This will have an impact on the throughput of your entire workflow, which should be represented in your documentation.

2.3 Documentation/Analysis

You are required to write a short (max. 4-page) report detailing your approach to the problem, why you picked that approach, and how well it worked. While describing how well it worked, you need to measure the **throughput** of your implementation as you systematically vary the input variables. This should be presented in your report using an appropriate representation (e.g. a graph, rather than just a list of numbers).

Ensure that you use appropriate academic citations and referencing throughout your report (N.B. a link to StackOverflow or similar is not an appropriate reference).

2.4 Testing

Your program should include bash scripts that build and run the code on the SCC VMs. I will test your code exclusively on the SCC Lab VM (accessible via mylab.lancaster.ac.uk). Specifically, I will unzip/untar your code into a directory, move into that directory and then run your two scripts in order: `build.sh` then `run.sh`.

Your program should accept three values as input parameters (for example, included in `run.sh` or from the terminal):

- i. λ , to adjust how often the tasks are produced
- ii. pos_0 , the initial position of the robot

In your report you will be expected to vary these and measure the change in behaviour based on an appropriate metric.

3 Submission

You should submit *two files* for coursework 1. The first is a **pdf** file which should contain all of your report and analysis, along with any references you have used (appropriately cited within your work). The second file is a **tar.gz** or **zip** file containing your Java code and bash scripts.

Please note: if you include your **pdf** file inside the **zip/tar.gz** file, this will **not** count as a submission of your report. You must upload *both* files separately.

The files should be uploaded to Moodle where the report will be analysed by Turnitin for plagiarism and collusion. **Do not include your name anywhere on your submission, your work will be marked anonymously.** You should, however, include your Lancaster student number on the front page of your report.

The coursework will be marked, and feedback provided, within 4 working weeks of submission. Be sure to familiarise yourself with the marking rubric available at the end of this document. The allocation of marks to component is as follows:

Robot controller	50%
Analysis of behaviour	20%
Report	30%
<hr/>	
Total	100%

Component (weight)	Fail [0, 40)	3 [40, 50)	2.ii [50, 60)	2.i [60, 70)	1 [70, 100]
Robot Controller (50%)	Code fails to provide even basic functionality. Code may have syntax errors, or may not compile properly on the SCC VMs as required. If code compiles correctly, it may not solve the basic workflow problem or may not be a concurrent program.	Code compiles correctly, basic workflow code appears to work, but may be very inefficient or fail to use core techniques to manage concurrency. There might be some missing functionality (such as handling multiple sensors), or might fail when parameters are changed from their default values. Code might be difficult to interpret due to poor coding practices.	Code provided appears to solve the problems sufficiently. Some management of concurrency has been attempted, but approach could be simpler, or fails to provide fair access to all agents. Code might be a little difficult to follow, and some focus on good coding practice would help.	Controller has clearly been designed well, and the code reflects this. The code for the task workflow and multi-sensor problems are provided and work well. The approach to handling concurrency could be a little better, but the controller can handle a variety of different parameter variables, not just the default values.	A well-designed and well-implemented robot controller, making effective use of appropriate concurrency management techniques. Code practices are at the level that I could be looking at professional code.
Analysis of Behaviour (20%)	No analysis attempted, too poorly presented to be interpretable, or analysis which does not match the controller provided for part 1.	Limited surface-level analysis presented. Some core errors in the approach, analysis might not be follow a logical progression, or might include qualitative aspects such as opinions.	A logical approach to analysis provided, but the rationale or justification for the approach is lacking. A quantitative measure has been defined, but might not be the best measure for the task at hand.	An appropriate, quantitative measure has been defined and presented in a clear manner. The approach to analysis has been justified and the analysis undertaken and reported well.	An appropriate, quantitative measure has been defined and presented in a clear manner. The choice of measure and approach to analysis have been justified and the analysis undertaken and reported well. There is a clear consideration of edge and boundary cases, with further analysis undertaken as required.
Report (30%)	No report provided, or so poorly presented as to preclude reading.	Report is provided, but might not follow a coherent structure, or has many spelling and grammar issues that make understanding difficult.	Report follows a logical, coherent structure. There might be a few spelling/grammar mistakes but these do not impede the reader too much. Analysis is presented but the rationale for each step might be unclear.	Documentation follows a logical, coherent structure. The thought process and design decisions are clear to the reader, with appropriate analytical techniques employed when needed. A few minor spelling/grammar issues may persist, but this does not distract from the overall message.	Documentation follows a logical, coherent structure that helps to guide the reader through the thought process of the designer. Analysis techniques are used when appropriate, with results presented to justify the decision-making process. Minimal, if any, spelling/grammar issues.