Systems Programming (GA) — Assessed Exercise #2

version 20241206

Lauritz Thamsen (building on material from Tom Wallis)

*due **19 — 12 — 2024** at 22:ooh, worth 15%*

Overview

This exercise aims to assess your understanding of concurrency and synchronisation means as well as memory management techniques: For this, you will summarise the concurrency and memory management features of three programming languages and compare the suitability of the languages for a given hypothetical project.

You are asked to:

- Summarise the concurrency and synchronisation means of three programming languages,
- summarise the languages' memory management techniques,
- choose a language that would be best for the project scenario,
- and justify your choice of language in the context of the scenario.

Your report should be around 1,000 to 1,200 words. You should submit it as a PDF or Word document. You are encouraged to work and submit in teams of two students (but you are also welcome to work and submit on your own). The report must include your name(s) and matriculation number(s) in its file name *and* first page.

At a high level, your report should be two things:

1. An accurate and thorough *summary* of the concurrency and synchronisation means and memory management techniques of three programming languages.
2. A convincing *argument* in favour of picking a particular language for the team and application of the given project scenario.

If, after explaining the various properties of these languages, you *do not* think there is any reason to pick a particular language (or more than one), feel free to explain this instead of picking one (or more) particular language(s), so long as you can still convincingly draw your conclusion.

You will get marks for showing an *understanding* of the topic, so details and facts are a good start, but full marks will be given only for explaining why those facts *matter*, in general and in the context of the given project scenario.

Project Scenario

You are an employee of *SpaceY*, a company which specialises in various space-adjacent technologies. Your CEO has just unexpectedly shared on the social media service Y that your team is to embark on an unusual energy technology, "*space-based solar power*"[1], causing several things to happen:

1. Your team has dropped their existing projects and began work on space-based solar power
2. Hardware designs have been made by another team outlining some high-level requirements of the satellites
3. Your team lead handed in their notice, to work for the new social media service BrightSky, making you the new team lead
4. As the new team lead, you currently have one thing on your to-do list: "Work out how to write the software for the sensors running on *SpaceY*'s new solar power satellites."

[1] `https://en.wikipedia.org/wiki/Space-based_solar_power`, basically solar panels in space, and the energy collected is beamed back to Earth as microwaves.

The hardware team's report told you that the satellite has *many* sensors. These are needed to course-correct, measure energy collected, safely beam energy through Earth's atmosphere, detect and protect against solar winds, collect measurements for experiments, and so on. There are a total of 1,326 sensors on the satellite. It is vitally important that the satellites work efficiently, and that information from *all sensors* can reliably be processed by running code at the same time. This means there are many concurrent processes which should all be run near real-time.

Having worked at *SpaceY* for some time, you are aware that future functional requirements might be placed on your software after several satellites have already been deployed. You can foresee a need for many different kinds of processing (for different sensors and other pieces of hardware); for example, frequency modulation of the satellite's microwave beam or the re-use of the company's orbital hardware for other purposes (such as satellite communications or low Earth orbit edge computing). In other words, you are already handling thousands of processes, but the more concurrent processes you can manage in the future, the better. Meanwhile, efficient and low-latency sensor data processing will likely remain a priority.

Your task is to write a concise report highlighting the strengths and weaknesses of various programming languages your team already has proficiency in, as well as any others you think would be good choices, so as to convince them of the language you want this project to be implemented in.

You know your team has previously written code in:

- C & C++
- Java
- Python
- Golang

So you can expect to have to summarise, compare, and relate most of these to the requirements of the project in order to convince your team of your choice.

Marking Breakdown

15% Summary of concurrency and synchronisation means of the *first language*

15% Summary of concurrency and synchronisation means of the *second language*

15% Summary of concurrency and synchronisation means of the *third language*

15% Summary of memory management in these three languages

20% Comparison of the three languages, relating them to project requirements and team capabilities, & deriving a recommendation for the project

20% Writing, structure, & presentation quality

Appendix: Recommendations for Writing the Report

There are a few things to consider in writing the report.

First, which programming languages are interesting to compare? The languages listed above all have some concurrency details to discuss, but you can write about anything. Maybe you write C#, Ruby, or Rust at work. Would they be interesting to discuss?

Second, what makes a programming language a "good" choice for the hypothetical team? When you are justifying the language you are choosing for the project, what would make a convincing argumentation? The language will have to be able to support many concurrent processes without major slowdowns, it might need easy access to synchronisation primitives, and it might — or might not, depending on the solution *you are proposing* — create and destroy

processes quite regularly. With many inputs and outputs, and a need to respond to new data quickly, what do you imagine would be a barrier for your program's performance?

Third, how should you write the text? There are many styles you could pick writing this. One suggestion is to write the report from the perspective of the team lead, as if you were writing a document explaining to your team why you have chosen a certain language to get everyone on-board with your choice.[2] Another idea is to write it as a technical report [3], explaining and summarising details in a neutral, academic style. Alternatively, you could write it explaining the potential thinking of a hypothetical team lead[4] — It is up to you how you write the report, but remember that there are marks for writing quality, structure, and the quality of the presentation.

Finally, how do you earn high marks? Regardless of how you are writing this, remember the marking breakdown. You will get a few marks for each programming language's description, yet most of the marks are for clearly showing an understanding of how the language might apply in the context of the project scenario. There are marks for each language's memory model too. Almost half of the marks, however, are for writing quality and the justification of the language you pick, suggesting to allocate a good amount of effort to those parts of the report, e.g. proportional to the marking breakdown... Remember to actually compare the languages, and that you do not need a thorough, in-depth understanding of each language! Rather, picking a couple of high-level features and focusing on those will let you show understanding and convincingly compare the languages you pick.

[2] "I know we have written a lot of C lately, which uses threads for concurrency and parallelism... At the same time, you will remember from the *Tunneling* project that we had less race conditions to deal with when we relied on message passing between actors in Scala..."

[3] "C's concurrency model has strengths and weaknesses. Threads are powerful but starting them requires operating system resources, as opposed to Go's "goroutines". Goroutines, however, leave programs open to process leaking..."

[4] "The team lead might be curious about Common Lisp, having heard about it from their team, but this would not be a good choice of language for the new project: while continuations are an interesting concept, the developers on the team have heard of Lisp, but have not actually used it much..."