# Systems Programming Coursework 2

Robert De Venny (2892269d)

## Introduction

The purpose of this report is to outline the concurrency, synchronisation, and memory management features of 3 programming languages. This is with the aim to provide a recommendation for an approach for writing the software for the new *SpaceY* satellites. These satellites have large numbers of sensors gathering a diverse array of information which is required to be processed by the software. This primary requirement is why the language chosen for the implementation of the satellite software must have effective, robust concurrency, synchronisation, and memory management features to ensure the accuracy of the collected data. This report will cover the evaluation of C, Python and Java.

## C Overview

~~C is a lower-level language more associated with operating systems and system software rather than higher-level languages primarily used in modern application software. This leads to the concurrency and synchronisation features linking closely to the concepts themselves without being hidden behind a higher level of abstraction. The concurrency aspects of C utilise the `pthreads` library which is included in most C distributions and is a common piece of learning for developers working with C.~~

### Concurrency

Concurrency is achieved through multithreading and multiprocessing which have some key differences. Threads can be thought of as smaller units of execution within a process, sharing the same memory space and hardware resources. They are much lighter weight compared to processes as they have their own isolated memory.

Multithreading allows multiple threads to run concurrently within a single process. In C, threads are typically implemented using the POSIX Threads (*pthreads*) library. Threads can be created using *pthread\_create* and later joined with *pthread\_join*. Joining ensures that the main thread waits for child threads to complete their tasks before continuing execution. Additionally, joining allows the return values of threads to be collected back into the main thread, which is useful for managing results from concurrent operations. We can still run into issues when programming concurrently in C when we don’t also ensure synchronisation.

### Synchronisation

One of the main challenges faced when concurrently programming is race conditions, which occur when threads access shared resources unsafely. This can be prevented by means of synchronisation. C has supports two common mechanisms: mutexes and semaphores accessed using the *pthread* and *semaphore* libraries. These libraries provide an easy-to-use API for managing the locking and control of shared resources within threads. However, these APIs are very manual and rely on the programmer to avoid issues like **deadlocks**, where threads wait indefinitely for resources held by each other.

## Python Overview

### Concurrency

Python also uses multithreading as a means for concurrency and uses its *threading* module, similarly to C it also requires synchronisation to avoid issues with access to the shared memory resources. Python also, uniquely, has its own Global Interpreter Lock (GIL) which imposes some limitations on achieving true parallelism with its threading mechanisms. Primarily the GIL means that only a single thread executes python bytecode at a time which means that threading in python is more suitable for I/O-bound tasks which traditionally have more natural pauses in execution. Due to it being a higher-level language there is no need for manual thread creation and joining in python as this is all handled by the *threading* module, this reduces the risks of human error.

### Synchronisation

Locking is also available in python but in many flavours and at a higher level of abstraction. Simple *locks* act as mutexes to prevent multiple threads accessing a shared resource in parallel. Semaphores are also available but with a straightforward, clean API making them easy to implement and maintain. Although the GIL does limit functionality in some areas it does drastically simplify thread safety by taking this control away from the developer and can be bypassed using multiprocessing.