School of Computing
Assessment Brief

Outline	Basic Information
Module code and title:	COMP6018 Theory and Practice of Concurrency
Who to Contact Name & Email Address:	Marco Paviotti (m.paviotti@kent.ac.uk)
Assessment name:	Assessment 1
Assessment type and component:	Mini practical exercises – Component 1
Assessment weighting:	20%
Module learning outcomes assessed in this assignment:	• Have a critical understanding of the principles of concurrent programming, as well as its advantages and challenges. • Critically evaluate and appraise the properties of a concurrent process and compare the behaviour of different processes. • Apply the principles of concurrency theory to real scenarios. • Apply simple abstract concepts of concurrency theory to master complex concrete scenarios.
Submission deadline:	Tue, 21 Oct 2025
Where to submit:	Moodle
How to request an extension:	Unexpected events such as illness, an accident, or bereavement sometimes happens. If so, you may need to request an extension, late submission or to be absent on the day of an assessed event. For instructions, see "in-course extenuating circumstance" in My Kent.
When marks and feedback on your submission will be provided on Kent Vision:	Tue, 11 Nov 2025
How to find your feedback in Moodle	Your tutor's feedback on your assessment is one of the most valuable learning resources you have on your course. Please review it, compare it with feedback you've received on other assignments, and use it to identify skills where you can improve on future assignments. Your tutor, academic advisor, or a member of the Skills for Academic Success (SAS) team can help you apply the lessons in this feedback to future work.

1 The Task

The goal of this assessment is to gain experience with shared-memory concurrency primitives in Java. The aim is to build a simple file server which can be accessed concurrently by multiple clients.
The accompanying source files on the Moodle provide interfaces and key classes which should not be modified:

- `FileServer.java` – the main interface;
- `File.java` – a class representing *files* which clients can see;
- `FileFrame.java` – a class representing file information internally on the server;
- `Mode.java` – public enumeration of *file modes* (describes the status of a file and is also used to request access to a file with a certain mode e.g., readable or read-writeable).
- `TestSuite.java` – a standalone application for testing your program.

1.1 What do I need to do?

You need to develop a shared file server by defining a class which implements the interface provided by `FileServer.java`:

```
public interface FileServer {
  // Create a new file on the server
  public void create(String filename, String content);

  // Attempt to open a file -- may block if the file is not available at that mode
  // Returns an Optional.empty() if no such file exists
  public Optional<File> open(String filename, Mode mode);

  // Close a file
  public void close(File file);

  // Check on the status of a file (the mode it is currently in)
  public Mode fileStatus(String filename);

  // What files are available on the server?
  public Set<String> availableFiles();
}
```

Clients can access files via `open(filename, mode)` with the following behaviour:

- A client can open a file for reading (`Mode.READABLE`) if it is currently closed or currently already open for reading (by other clients perhaps).

- A client can open a file for writing (`Mode.READWRITEABLE`) if it is currently closed only (`Mode.CLOSED`). If the file is in a different mode, the call to `open` should block until the file is closed (and therefore available).

For example, if a client opens a file `stuff.txt` for reading, then other clients can also open `stuff.txt` for reading. If a client opens `stuff.txt` for writing, then `stuff.txt` must already be closed (no other clients can have it open for reading or writing), otherwise `open` should block until it available. Furthermore, whilst a client has a file open for writing, any other use of `open` for that particular file (whether reading or writing) should block until the writing client has closed it.

You do not need to actually open and save actual files (you shouldn't be importing a file handling library); instead, just represent files inside your file server implementation (i.e., via a list or hash map), for which you can also use the `FileFrame` class which captures the mode and the content (string).
You should aim for some kind of fairness but there are different choices about who gets priority (e.g. readers, writers, etc.).

1.2 Where do I start? Tips for success.

Read the assessment brief – see **Section 1.1 What do I need to do?**
Start early: give yourself enough time to complete the assessment.
Look at the lecture slides and the class materials. We have seen programming techniques is in the lectures. It will help revise them before attempting these tasks.

1.3 May I use Generative AI?

See the latest guidance at the My Kent page on Generative AI use in assessments.
The use of generative AI is **not permitted**, as it would invalidate the learning outcomes (problem solving). Furthermore, AI will likely produce wrong or too generic solutions for the purpose.

1.4 What will I submit? Are there any special formatting or referencing styles expected?

Your submission should be at least one Java file for your file server, called `MyFileServer.java`. Include a short description (as a comment) at the start of the file (clearly marked) that explains your approach to concurrency in solving this problem including how you avoid race conditions and starvation, and how you provide fairness and mutual exclusion with respect to writing.
You can create extra files (e.g., for extra classes) but you should have at least the above files.
Do not modify and do not submit the provided classes.

1.5 I have an Inclusive Learning Plan (ILP), how will this plan be accommodated?

In general, assessments have been designed to be inclusive. For timed assessments, extra time will automatically be given if your ILP recommends it. If you need more time for a coursework assessment due to the impact of your disability or mental health condition, please contact the student engagement team for

your School to request an in-course extenuating circumstance. If you have an ILP that you think requires further accommodation, please contact the module convenor to discuss how it would work.

2 Support Available

2.1 Module activities or tasks that specifically help you prepare you for this assignment.

Lectures and classes in the 1st and 2nd week of term will provide the foundational tools and skills needed for this assignment. The class exercises in week 14th are designed to help you prepare for this assignment.

2.2 Further assistance

For questions related to this assignment, you can contact Marco Paviotti at m.paviotti@kent.ac.uk

See My Kent for links to support services and resources to help you with your studies. These include the Course Administration Team, Library Search and Subject Librarians, Skills for Academic Success, and Mental Health and Wellbeing Support.

3 Marking Criteria- How will my assignment be marked?

You can run your implementation of the file server against the test suite by compiling `TestSuite.java` and running:

```
$ java TestSuite MyFileServer
```

where `MyFileServer` is the name of your file server class. There are 52 tests that have to pass in order to get full marks (100/100). The final mark is calculate by rescaling the number of passed tests to 100 using the formula:

$$\text{YourMark} = \frac{100 * \text{PassedTests}}{52}$$

You may use things from Java's standard library like collections and Java's `ReentrantLock` and `Semaphore` which we covered in lectures.

On ReadWriteLocks: You may use Java's `ReentrantReadWriteLock` library to solve this, but if you do the maximum possible mark will be capped at 40/100. This still means you could (just) get a first-class mark overall, but the implication is that if you can solve it using your own techniques (involving some locks/semaphores) then you can score more highly for task 1. The idea is that it would be great to see you think for yourself about how to solve this using a smaller set of basic components, though if you are having real trouble solving the problem you can still default to the `ReadWriteLock` implementation and get a good mark.