UNIVERSITY *of York*

SCHOOL OF PHYSICS,
ENGINEERING AND TECHNOLOGY

Systems Programming for ARM Assessment

Dr. Andy Pomfret

Autumn 2023

1 Task

You must modify DocetOS to increase its functionality. You may choose to implement any combination of features you like, but you must include:

1. Mutual exclusion, via a reentrant mutex
   - This can be the same as the implementation you completed in the labs, if it is compatible with other aspects of your design
2. A fixed-priority scheduler
   - You may choose a fixed number of priority levels if you wish, though this should ideally be configurable through a `#define`
   - Priority levels should not change at runtime, unless priority inheritance is implemented (see below)
   - The highest-priority runnable task should always be run; if two or more tasks have the same priority, they should operate round-robin
3. An *efficient* task sleeping mechanism
   - You will need to modify the implementation from the labs, to remove sleeping tasks from the scheduler

You must also include **at least three** of the following six items:

- Binary and/or counting semaphores
  - The mutex code can be used as a template for these, although the semaphores are simpler than the mutex
- A queue-based task communication system
  - Copy queues and pointer queues are both acceptable. Careful thought is needed to protect the queue against concurrent modification, and semaphores are useful – please ask for guidance
- Floating-point support for the context switch
  - This requires very little code but is easy to get wrong! Again please ask for guidance
- A more efficient wait and notify system
  - The implementation in the labs used a single queue for waiting tasks, which then requires *all* taks to be notified on any notification event. One queue per blocking item (mutex, semaphore, etc.) would improve this a lot
- Priority inheritance for mutexes
  - Priority inheritance means that a task holding a mutex is temporarily granted the priority of the highest-priority task waiting for it, which ensures that the mutex is released promptly

- A memory pool implementation, properly protected against concurrent modification
  - The implementation from the labs will need minor modification to protect it against concurrent modification

Optionally, to make your design more useful, you may implement more than three of the features from the list above, or change or extend any other features of the operating system as you see fit. Please remember that quality is far more important than quantity, and there are not necessarily any more marks available for going beyond the brief but marks *will* be lost for poor design and implementation.

The implementation details of the features are entirely up to you, but you should ensure that you pay attention to the avoidance of race conditions, to efficient use of CPU time and RAM, and to a sensible, readable code structure. Remember to demonstrate that you understand the things you've learned in this module!

Finally, you must include a demonstration of your modified OS, that shows off its features.

2 Submission

You must submit a technical report on your modifications. It should contain a brief executive summary, a contents page, an introduction and a conclusion. Aim to make it no longer than 4500 words. Include details of your modifications to DocetOS, and of your demonstration code. The report should focus on your **design**, rather than your implementation; if you are unsure about the difference, please ask.

You must also submit your code, being careful to upload all source and header files that you have created or modified. The easiest way to do this is to submit a zip file of the project folder. Before you create the archive please delete the Flash folder from the project (these contain large build artefacts and are not required) and anonymise the filenames in your project by renaming any whose names reflect your identity.

3 Guidance

Tips and resources for implementing each of the requested features will be available on the module web page, to give you a starting point.

4 Marking

Marks will be awarded for:

- Design features implemented
- Effective, readable, commented code

- Good use of assembly language or CMSIS intrinsics where required
- Correct use of `const` and `volatile`
- Efficient use of CPU time – for example, avoiding polling where possible
- Careful treatment of any potential deadlocks or race conditions
- Appropriate use of SVC delegates
- Appropriate choice of data structures, for task storage and other purposes
- Effective demonstration code
- Report structure, presentation and readability

Please note that the mark scheme is based on the quality of the submitted work, on the assumption that it is complete and contains at least the features specified in Section 1. If you do not implement all of the essential features and at least the minimum number of optional features, **marks penalties will be applied**.

Indicative mark breakdown:

Item	Mark
Report structure and presentation	20%
Detailed and effective design	30%
Effective and efficient code	30%
Clear, standardised code practices	20%

Please remember that the report and code will be anonymously marked. Do not include (in filenames, folder names, comments etc.) anything that could be used to identify you personally.

5 Academic Misconduct

A reminder that plagiarism and collusion constitute academic misconduct and will not be tolerated. Do not work with anyone else on this assessment, and do not submit someone else's work as your own.