COS40003 Concurrent Programming

Learning Summary Report

Reuben Wilson (9988289)

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment (please tick)			x	

Self-assessment Statement

	Included (please tick)
Learning Summary Report	x
Concurrency Utilities (Semaphore, Rendezvous, Channel, Bounded Channel, Mutex, and Active Objects)	x
Parallel Program	x
Concurrent program (Connection Manager)	x
Language features report	x
Solutions to classic problems (Philosophers, Other)	x

Minimum Pass Checklist

	Included (please tick)
Additional utilities (Latch, Barrier, Reader Writer Lock, FIFO Semaphore)	x
Timeouts, interrupt handling, and optimisations for select utilities	x
Additional synchronisation problem solutions (Gamers)	x

Minimum Credit Checklist, in addition to Pass Checklist

	Included (please tick)
Data Management System – File, Query and Transaction support	x

Minimum Distinction Checklist, in addition to Credit Checklist

	Included (please tick)
Data Management System with Extensions	
Research Report	

Minimum High Distinction Checklist, in addition to Distinction Checklist

Coverage of the Intended Learning Outcomes

This section outlines how the pieces I have included demonstrate the depth of my understanding in relation to each of the unit's intended learning outcomes.

ILO 1: Concurrency Control Utilities

Design and develop complex concurrency control utilities using threading and primitive synchronisation mechanisms of the language used.

During the course of COS40003 Concurrent Programming, a number of concurrent utilities were developed in order to handle concurrency in programs. Programs were developed that required thread synchronisation as well as safety when it comes to values that may be subject to race conditions, I.e. variables that are accessed by more than one thread. The utilities that were developed in order to control the activity and synchronisation of threads include but are not limited to:

- Semaphore: A token issuing mechanism, which will only allow a thread to proceed with activity if the thread successfully acquires a token from the Semaphore. If the Semaphore has no tokens to issue, the thread will wait until such time as it can acquire a token.
- FiFoSemaphore: A Semaphore which works in a first in first out manner, meaning that a thread that has attempted to acquire but is blocked will able to acquire before a thread that has arrived after it and was also blocked.
- Mutex: A single token Semaphoe.
- Barrier: A synchronisation utility that has a limit, where the limit represents the number of threads the barrier will hold. All threads waiting at the Barrier will be blocked from activity until the number of threads waiting at the Barrier represents the Barrier's limit.
- Rendezvous: A Barrier with a limit of two, I.e. will synchronise the activity of two threads.
- Latch: A utility that provides the functionality for the queueing of a number of threads. All threads queueing on the Latch will not be able to commence with activity until such point as the Latch is released. Once the Latch is released, all threads waiting on the Latch will proceed with activity.
- ReaderWriterLock: Utility that allows for many readers concurrently but no writers or a single writer and no readers.

ILO 2: Develop Concurrent and Parallel Programs

Apply contemporary programming languages and frameworks to design and develop concurrent and parallel programs.

The unit has seen the development of many smaller concurrent and parallel programs, which demonstrate the implementation of the concurrent utilities developed throughout the semester in order to handle thread synchronisation and concurrent activity. The piece of work included in this portfolio, which demonstrates this ILO in most depth is the distinction program; DMS.

The DMS sees the implementation of many of the concurrent utilities that were developed during the semester in order to handle the concurrency aspects of the program. In order to demonstrate how the DMS implements the utilities, the threading based aspects of the program will be briefly defined.

- FileManager: A ChannelBasedActiveObject that de-queues Request objects from its Channel. While there is nothing to de-queue on the Channel, the FileManager waits until such time as there is data on the Channel to process.
- TableManager: The TableManager is not a concurrent utility but interacts directly with the FileManager in response to input processed by the QueryManager (defined below). The TableManager places data on the FileManager's Channel and the FileManager processes the data appropriately, based on the kind of Request it receives. A Request object may contain a ResultSet, where a ResultSet contains a Latch. After the TableManager has sent a Request to the FileManager, it immediately acquires the Request's ResultSet's Latch, which will result in the main thread being

blocked until such time as the FileManager has processed the Request and released the Request's ResultSet's Latch.

- Table: In order to accommodate for the Serialised Read Transaction isolation level, the Table implements a single Mutex, which is used to block requests to create new Row's while a Serialised Read Transaction is active.
- QueryManager: A ChannelBasedActiveObject that de-queues Message objects from its Channel. The QueryManager is responsible for processing input requests placed on its Channel by the ConnectionManager (defined below). Each Message object is the result of data sent from a connected client. The Clients are managed by the ConnectionManager. The DMS implements different Transaction isolation levels, where the access to the Table and or it's Rows is restricted to a manner in which is specified by the Transaction in use. I.e. If a Read Serialised Transaction is active, creating new Rows is explicitly blocked. If a Read Uncommitted Transaction is active, all data read from Rows has to have been committed, meaning that if a read is requested on a Row that is currently being updated, the read will be blocked until the changes for the Row are committed.
- ConnectionManager: An ActiveObject, which manages client connections to the DMS and places any input received by connected clients on the channel for the QueryManager(s).

In summary, the DMS demonstrates the implementation of a complex program in which concurrency is handled appropriately by the concurrent utilities developed during the semester.

ILO 3: Identify issues related to liveness, performance, and reusability

Analyse algorithm and utility designs to identify, explain, and correct issues related to safety, liveness, performance and reusability in concurrent contexts.

Before the optimisation of the utilities developed during the semester, there were issues with liveness, performance and reusability. The issues related to each are defined below:

- Liveness: Before implementing the support for interrupts and timeouts, the issue related to liveness stems from the fact that any given thread could be waiting indefinitely on a concurrency control utility. Take the Semaphore as an example, where a thread could wait on the Semaphore for a token. If the thread never successfully acquires from the Semaphore, that thread has lost liveness and is in limbo. By implementing interrupts and timeouts, we can ensure as much a possible that a thread can have continual liveness even in the event that it may not be able to acquire from the Semaphore by interrupting the idle thread or by specifying a maximum number of time that the thread should wait to acquire from the Semaphore before continuing on with activity.
- Performance: The main area in which performance was an issues before utility optimisation was the manner in which locking was implemented within the utilities. When locking was required, a reference to the object in which the locking was required was used as the key. I.e. This object. The issue with this is the fact that too much was being locked and by locking the reference to the object, other callers on methods of the object being locked would often wait unnecessarily until the lock is released. As an example, if a method has been called on the object that locks itself by one caller and another caller calls a method on the object that requires no locking, the second method call will not be executed until the reference to the object is released from the lock. This particular performance issue was fixed by implementing individual object locks for each concurrency control utility so that when locking was required, the lock object was locked instead of a reference to the object itself, which removes the issues as previously defined and hence forth improves on utility performance. Another area in which performance was improved was by implementing a custom thread safe List that sees the inclusion of lock splitting over the top and bottom nodes. The List was utilised in the Channel classes and replaced the use of a .NET Queue in which locking was always enforced before en-queuing and de-queueing from the Queue in order to implement thread safety. This was a

performance issue because only one thread could ever have access to the Queue at a given time, where by implementing the custom list with lock splitting, the List could be en-queued and de-queued to concurrently. Performance also relates directly to liveness in the respect that, if a thread is waiting indefinitely on a concurrency control utility, the performance of the program is directly inhibited.

- Reusability: By addressing the issues related to liveness and performance, the reusability of the concurrency control utilities is greatly improved due to the fact that they can be used in any concurrent program while guaranteeing minimal detriment to liveness and performance.

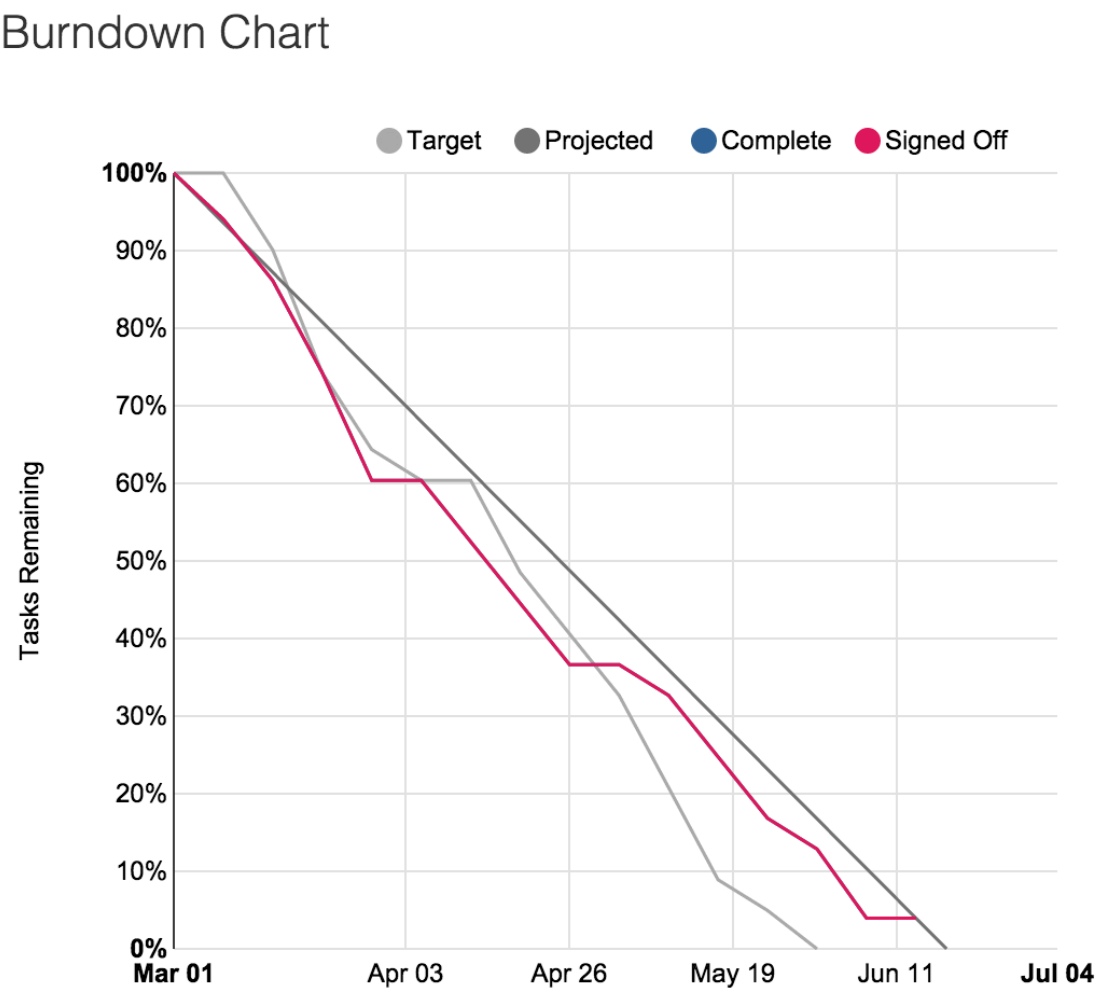
ILO 4: Compare and Contrast Language Features

Compare and contrast concurrency support provided by different programming language frameworks.

The Language Features report compiled and submitted with this portfolio demonstrates an adequate comparison between the concurrent utilities developed through out the semester to a concurrent framework written in another language. Specifically, the Language Features report submitted with this portfolio compares a concurrent library for the language Ruby. The conclusion derived from the Language Features report supports the notion that regardless of the language in which the concurrent utilities are written and the differences that may be evident from language to language, there is little to no difference when it comes to the functionality of those concurrency control utilities.

The Language Features report provided as an opportunity to further understand the utilities written throughout the semester by drawing on developed understandings of how the utilities work in order to support concurrency control while at the same time learning more about the same kind of utilities that are available in another language.

Reflection



Throughout my degree so far, there have been a number of subjects in which I feel I have developed my skillset and experience as a programmer. Such subjects include:

- Introduction to Programming
- Object Oriented Programming
- Data Structures and Patterns

All of the above named subjects have been extremely important for my learning and understanding in relation to the computer science field. That being said though, having sat Concurrent Programming this semester, I feel as if I have just completed my most important subject, my favourite subject and by far the most challenging.

I can recognise that Concurrent Programming is the most important subject I have undertaken due to the fact that it has helped me develop my skillset even further by drawing on knowledge that I have already obtained and applying that to a challenging subject area. More so than any other subject, Concurrent Programming promoted the use of critical thinking as well as polishing my problem solving skills. I genuinely feel that I'm twice the programmer I was before commencing with Concurrent Programming.

I feel I have learned enough about concurrency in programs and concurrency control utilities to maintain a clear cut perspective of what is involved in concurrent programming and discuss the subject matter with future students and professionals alike with confidence. For me, this is quite a milestone, considering I had little to no idea what concurrency was before taking the subject. As an example that supports my last claim of little to no knowledge of the subject matter, I'd never heard of concurrency control utilities and was not aware of their existence and uses.

For me, there was not one task that was more important than the other. I feel as If the learning was adequately tapered from beginner to advanced throughout the semester. Everything that I learned in the earlier weeks, I could build upon in terms of understanding and application while the work was continually administered and completed. At stages, I was actually unsure of how the utilities work, even if there was an understanding of their appropriate application. Exercises like commenting the utilities and writing tests really helped me to be able to establish an understanding of how the utilities serve their intended purpose in regards to concurrency control in programming.

I was most interested in working on the distinction program; the DMS. For me, I feel as if that if I didn't attempt the DMS, I would be cheating myself by shortcutting the learning process. I feel this way because the DMS was an excellent opportunity to apply all of what I'd learned from the pass and credit tasks to something much more sophisticated. The DMS helped me to build on my understanding of the utilities that I developed and their appropriate use in concurrent programs. More so, the DMS I found interesting because it challenged me, it kept me awake at night. I can honestly say, that at times, it is the only programming task that I've ever committed to that has kept me absolutely obsessed with progress and continual development. I learned a lot from the DMS exercise and not just about programming. I learned about my patience, I learned more about he ability to critically think and apply practical solution derives from learned skills in order to solve complex problems.

This unit will help me in the future because I feel that I'm a better software developer for having taken it. I'm not entirely sure that I'm interested in concurrent programming as a discipline, but that is irrelevant due to the fact that I've learned a new language and greatly improved on my programming skills.

The activities that really helped me were the roll playing activities. Due to the nature and behaviour of multi-threading, in the beginning I found it difficult to understand how to utilities are working due to the fact that any number of threads could be calling on the same utility. The role playing during the lectures very much helped me to visualise the functional aspects and mechanics of the utilities by demonstrating with props, thread activity.

After completing this unit, I do not think differently about software development, but I have learned a lot more about a particular discipline within the software development industry. After completing the unit, I feel as if I have developed on my programming skillset and am looking forward to applying what I have learned to projects in the future.

In the beginning, I was not aware or had not considered the full scope of what I may or may not have learned by undertaking the unit. I feel as if I learned more than I originally would have perceived, although I did not much think about it. I'm very satisfied with the materials that were covered throughout the unit and I feel as if the materials that were covered were more than sufficient in terms of conveying the core concepts of concurrent programming. In terms of the exercises, the DMS is what I learned most from it due to the required application and understanding of the concurrent utilities tat we developed.

I have never had great time management skills, but I feel that for this unit, because of my keen interest in the topic and my motivation to learn and achieve results on the higher end of the academic scale, that I used my time well in relation to working on and completing the required work. This unit is one of the only units that I've been completely obsessed with in terms of progressing with the unit material. I would even dream about the task. It's really quite disturbing actually, but it's an indication to the enthusiasm and commitment that I had for the subject.

In future, based on this unit, I would feel that I could improve on my learning experience by having a more well planned study schedule, in which I've set aside clearly allocated time slots for each unit of study that I participate in. If I could do the unit, I would love to be able to commit to going for a High Distinction. I just burned out and lost interest in the last task, which is the DMS extension. I justify it by telling my self that, I feel as if I've learned all that there is from the unit and by extending on the DMS, I would not learn any more in regards to the ILOs that are relevant to the unit, Concurrent Programming.

I would like to thank Andrew Cain for an awesome experience and all the help and support that he gave to me. The unit was tough, but not without reward. Well done, Andrew. Thank you for a great semester.

Reuben Wilson.

Conclusion

In summary, I believe that I have clearly demonstrate that my portfolio is sufficient to be awarded a distinction, at least. I've worked more in this unit than any other unit I have completed at Swinburne, and as previously stated within this learning summary report, this has been my favourite unit to date due to the amount of learning I achieved and the subject matter.

Although I did not complete the work relevant for a High Distinction, I don't think it is a reflection on my work ethic. I strived continually for development and progress within this unit. There were very few days during the semester where I did not work on the unit. I feel as if there are many aspects of my submitted work that demonstrate this, from my thorough documentation to my extensive DMS.

I've had loads of fun and I look forward to continuing my studies at the Swinburne University of Technology.