Dralle, Magee, O’Reilly, Dargan, Hawkins

OS-Group Project

Proposal Paper

H2014 – Operating Systems

Contents

[Introduction 1](#_Toc95564027)

[Team-members 1](#_Toc95564028)

[Problem statement – The Dining Philosophers Problem 2](#_Toc95564029)

[Proposed Solution – Mutual Exclusive Locks 2](#_Toc95564030)

[Mutex vs. Semaphores 3](#_Toc95564031)

[Analyzing the problem solution 3](#_Toc95564032)

[GANTT – Chart 4](#_Toc95564033)

[Academic Honesty 5](#_Toc95564034)

# Introduction

In this semesters group project, the research team will analyze one solution to the ‘Dining Philosophers Problem’. This classic multithreading problem invented by E.W Dijkstra, will be looked at under consideration of solving it by use of ‘mutual exclusion locks’ (mutex locks). After explanation of the problem and comparison of three of the ways to solving this problem, the analysis conducted will outline and explain the use of mutex locks in solving the problem step by step through use of the Thread Mentor software, developed by Dr. C.-K. Shene, at the Michigan Technological University in 2001.

# Team-members

Student name: Evan Hawkins  
Student number:

Student name: Jack Dargan  
Student number: B00138090

Student name: Rene Dralle  
Student number: B00138163

Student name: Sam Magee  
Student number: B00132512

Student name: Saoirse O’Reilly O’Riordan  
Student number: B00132517

# Problem statement – The Dining Philosophers Problem

This classic multithreading problem invented by E.W Dijkstra, states that there are five philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each philosopher and five chopsticks. A philosopher needs both a right and left chopstick to eat. If both not available, the philosopher puts down their cutlery and begins to think again.

It is obvious, that the philosophers do not have the resources at the table to all be eating at once, but rather need to wait until there are two chopsticks free to use. This is a demonstration of a classic synchronization problem as it shows a large class of concurrency issues.

# Proposed Solution – Mutual Exclusive Locks

The concept of Mututal Exclusive Locks (mutex-locks) is a renown solution for solving multithreading problems with resource limitation like the above. The thread checks the ressource it needs to acquire its resources and takes ownership of the control instance of said ressource the lock. For as long as the thread is working with the ressources, other threads needing the resource can not access the locked resource (mutual exclusion) and enter a queue waiting idling until the thread in possesion of the lock signals its unlock for the resource, making it available to the other threads again. It is nessecary to say, that one can not assume the queue handling implemented in others’ programs, so caution is advised.

For the above problem every chopstick will be given a lock attribute that is observable by the threads representing the five philosophers, making the program capable of solving the resource issue.

# Mutex vs. Semaphores

The solution of the dining philosopher’s problem by usage of semaphores, is another concept applied. This makes use of a central management unit keeping track of the available resources on the stack allocating them to threads in need of them, and telling threads off, when there are none available at the moment signaling them to go into a waiting state. The solution applied in this analysis will apply the “mutex” solution to better understand inter-resource communication in respect to the threads, than involving a handler required for the semaphore-solution.

# Analyzing the problem solution

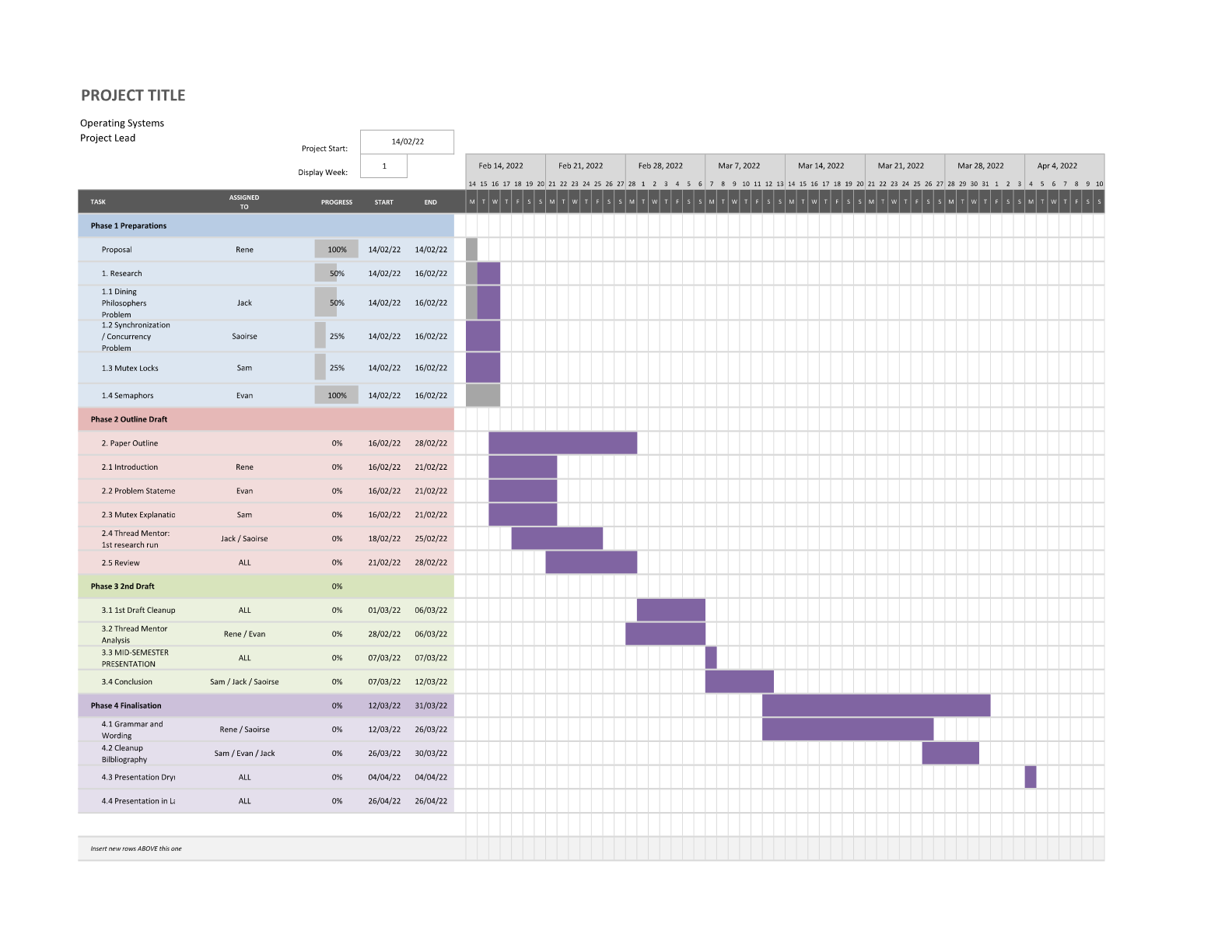
Over the course of this project the problem will be observed using a pre-build C program for Thread Mentor. This visualization tool depicts every interaction between the threads and resources during runtime. With it, it is possible to understand every lock and unlock operation during the execution of the program.

While pointing out the individual steps done by the system during one run of the program. Data will be collected through output analysis and pictures captured from the visualization.

After the run, this collected data will be interpreted for its individual actions and summarized in an analysis and conclusion.

Things to be kept in mind and look out for are, how deadlocks and concurrency issues are avoided and what mechanisms ensure a smooth operation of the program.

# GANTT – Chart



# Academic Honesty

We are aware of the University policy on plagiarism in assignments and examinations (3AS08). We understand that plagiarism, collusion, and copying are grave and serious offences in the University and we will accept the penalties that could be imposed if we engage in any such activity. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study. We declare that this material, which we now submit for assessment, is entirely of our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work.”