

Computer Modelling Final Project

Mihaila Victor Matei

November 29, 2023

Contents

1	Introduction	2
2	Part I: Making the car as fast as possible	2
3	Part II: Logging important data	5
4	Results and difficulties	5
5	Conclusion	7

1 Introduction

The final project for the Computer Modelling class consisted of simulating an automotive system using software based on the strategy described in the article "Functionally and Temporally Correct Simulation of Cyber-Systems for Automotive Systems", studied previously in class. The main advantages the approach proposed by the paper has over other similar tools designed for the same purpose is that it accurately models function and timing behavior at points of interaction with the physical interface. This is achieved through using ranges defined through best- and worst-case execution times rather than assuming this times constant, and solving non-determinism as it arises.

The main idea of the algorithm is to process the scheduling through a two step process. The first one is an offline guider phase, where a precedence graph that models both deterministic and non-deterministic precedence relations between all jobs taking place in a hyperperiod (least common multiple of the periods of all tasks). The second phase, called online progressive scheduling, uses the guidance of the graph and information about individual jobs such as read or write constraints and the resulting deadlines, the schedule is constructed by solving non-determinism along the way.

In the version of the tool used for this project, a certain set of simplifying assumptions is employed. First, tasks have well defined, constant execution times, meaning non-determinism is much less of an issue and many of the tool's features remain unused. Furthermore, clear constraints are defined for all of the tasks' parameters, such as periods, execution times, deadlines and utilization. Under these assumptions, we are asked to improve the performance of the system compared to a given baseline, and to provide logs of useful information such as reading and writing operations, as well as scheduling of all relevant events. The project is divided in two parts, which I will present separately.

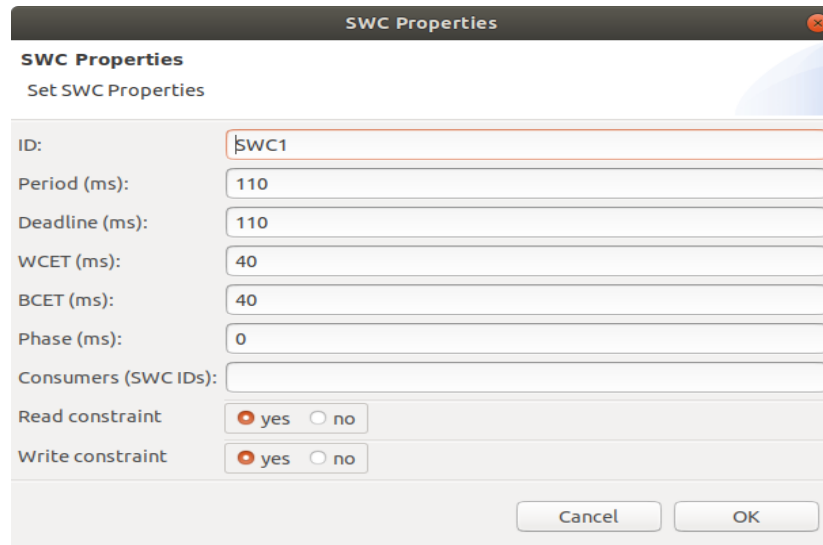
2 Part I: Making the car as fast as possible

In the first part of the exercise, we are asked to improve the lap time of the car compared to the provided baseline. To get ideas about how this may be achieved, I decided to implement the system described in the baseline and observe the behavior of the car. What I observed was a combination of low speed and very sudden steering, that occurs fairly rarely, and that causes a sort of jittery, zig-zag like trajectory along the track.

The tasks provided by the systems that actually interact with the physical interface are "Lane Keeping" (LK), and "Cruise Control" (CC). Four other dummy tasks without actual attached code are provided, and they do not directly affect the car's behavior. With this in mind, I theorized that the sudden movements are caused by the car not properly sticking to the center of its lane. LK aims to bring the car to the centre of the lane, meaning that doing it more frequently would lead to a trajectory that more closely resemble the shape of the track. Furthermore, cruise control keeps the car at a speed parameter controlled in the task's code, which can be increased up to 80 km/h.

With this in mind, I re-assigned the parameters of LK and CC to be performed more often, while trying to avoid scheduling conflicts with the dummy tasks. This is easily achieved by moving the dummy tasks to a separate ECU, without a need for modifying their parameters from the ones provided in the baseline.

In the following images, we can observe the configuration of these different tasks. In addition, in the code of CC the target speed is set to the maximum of 80 km/h. With this configuration, the time is improved from 2 min 40s to 1 min 16s.

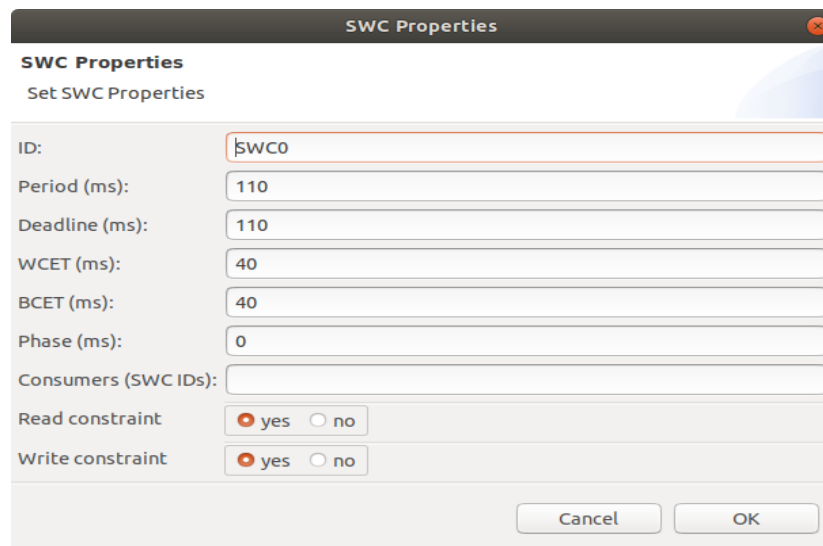


The image shows a dialog box titled "SWC Properties" with a subtitle "Set SWC Properties". It contains the following fields and controls:

- ID:
- Period (ms):
- Deadline (ms):
- WCET (ms):
- BCET (ms):
- Phase (ms):
- Consumers (SWC IDs):
- Read constraint: ☒ yes ☐ no
- Write constraint: ☒ yes ☐ no

At the bottom right are "Cancel" and "OK" buttons.

Figure 1: LK parameters



The image shows a dialog box titled "SWC Properties" with a subtitle "Set SWC Properties". It contains the following fields and controls:

- ID:
- Period (ms):
- Deadline (ms):
- WCET (ms):
- BCET (ms):
- Phase (ms):
- Consumers (SWC IDs):
- Read constraint: ☒ yes ☐ no
- Write constraint: ☒ yes ☐ no

At the bottom right are "Cancel" and "OK" buttons.

Figure 2: CC parameters

SWC Properties

Set SWC Properties

ID:

SWC2

Period (ms):

500

Deadline (ms):

500

WCET (ms):

100

BCET (ms):

100

Phase (ms):

0

Consumers (SWC IDs):

Read constraint

☐ yes
☒ no

Write constraint

☐ yes
☒ no

Cancel

OK

Figure 3: Dummy parameters

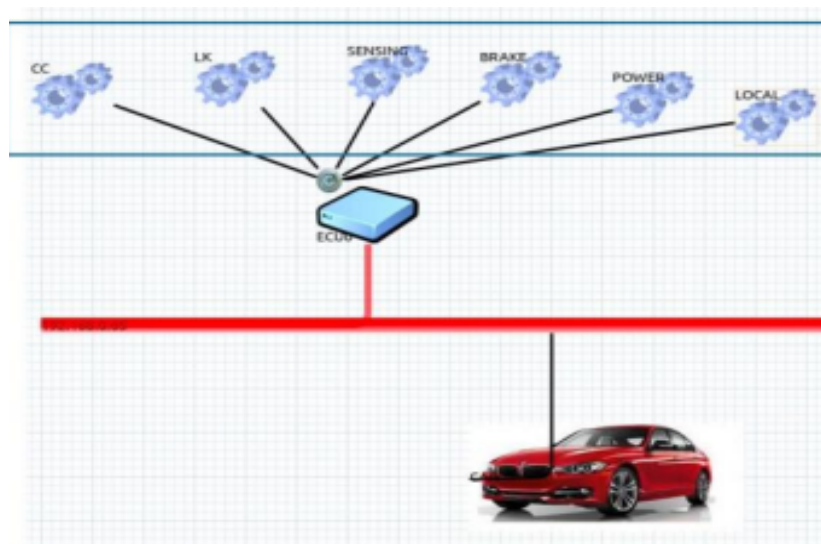


Figure 4: ECU system architecture

3 Part II: Logging important data

The second part of the project consisted of logging the read and write timings of a given task in one file, and the timings of all job releases, starts, finished, and possible deadline misses in another file.

For obtaining information about read and write operations, we can use the actual start time and actual finish time attributes of elements of the Job class, easily accessible in the `run_function()` function in the job class.

For obtaining all events and their associated timings and job IDs, we can use data accessible in the `run_simulation()` function in the Executor class. These events can be stored in an array that can be sorted by time, and whose elements can later be printed. Job IDs are reset after each hyperperiod, which makes it easy to observe patterns of issues such as deadline misses.

4 Results and difficulties

The following images present the lap time of my configurations, as well as extracts from the log files. The format for the first log is [TIME] [READ/WRITE] [TASK NAME] [DATA NAME]. The format for the second log file is [TIME] [JOB ID] [EVENT TYPE].



Figure 5: Lap time

8950	READ	LK	TARGET_SPEED
8990	WRITE	LK	ACCEL_VALUE
9060	READ	LK	TARGET_SPEED
9180	WRITE	LK	ACCEL_VALUE
9170	READ	LK	TARGET_SPEED
9210	WRITE	LK	ACCEL_VALUE
9280	READ	LK	TARGET_SPEED
9320	WRITE	LK	ACCEL_VALUE
9390	READ	LK	TARGET_SPEED
9430	WRITE	LK	ACCEL_VALUE
9500	READ	LK	TARGET_SPEED
9540	WRITE	LK	ACCEL_VALUE
9610	READ	LK	TARGET_SPEED
9650	WRITE	LK	ACCEL_VALUE
9720	READ	LK	TARGET_SPEED
9760	WRITE	LK	ACCEL_VALUE
9830	READ	LK	TARGET_SPEED
9870	WRITE	LK	ACCEL_VALUE
9940	READ	LK	TARGET_SPEED
9980	WRITE	LK	ACCEL_VALUE
10850	READ	LK	TARGET_SPEED
10890	WRITE	LK	ACCEL_VALUE
10100	READ	LK	TARGET_SPEED
10200	WRITE	LK	ACCEL_VALUE
10270	READ	LK	TARGET_SPEED
10310	WRITE	LK	ACCEL_VALUE
10380	READ	LK	TARGET_SPEED
10420	WRITE	LK	ACCEL_VALUE
10490	READ	LK	TARGET_SPEED
10530	WRITE	LK	ACCEL_VALUE
10600	READ	LK	TARGET_SPEED

Figure 6: Read/Write log

1180	J11	FINISHED
1200	J3	STARTED
1200	J3	FINISHED
1210	J12	RELEASED
1210	J12	STARTED
1250	J12	STARTED
1250	J12	FINISHED
1290	J12	FINISHED
1300	J3	STARTED
1300	J3	FINISHED
1320	J13	RELEASED
1320	J13	STARTED
1360	J13	STARTED
1360	J13	FINISHED
1400	J3	FINISHED
1400	J13	FINISHED
1430	J14	RELEASED
1430	J14	STARTED
1470	J14	STARTED
1470	J14	FINISHED
1500	J4	RELEASED
1500	J4	STARTED
1510	J14	FINISHED
1540	J15	RELEASED
1540	J15	STARTED
1580	J15	STARTED
1580	J15	FINISHED

Figure 7: Schedule log

When implementing this logging, I had difficulty finding the "DATA NAME" parameter requested for the log, so I decided to use the ones seen in the example, which are "TARGET SPEED", and "ACCEL VALUE", which is consistent with what we would expect a lane keeping task to read and write respectively.

Another (much more interesting) issue I was faced with arose after changing my configuration following the progress report on Monday. After my presentation, it was pointed out to me that all parameters should be modified by increments of 10, meaning that I had to change the execution time for my Lane Keeping task from 25 to 30, but I did not change anything else. Normally, such a change would be expected to have little to no impact on the final lap time. However, after running this configuration, I noticed an increase in lap time of almost 2 seconds, as well as a return (to a much lesser extent than the baseline) of the sudden, sharp, stirring adjustments. The problem was very confusing until I implemented my schedule logging function. In this log, I could see all events occurring during the simulation with associated job IDs, as well as time stamps. Looking at it, I noticed numerous missed deadlines for my LK task. This came to be as a result of a very particular configuration in which the 5 ms increase in execution times caused every second LK to not be executed as planned and have its deadline missed. I adjusted the configuration to avoid this situation and got to a performance basically equivalent to the one observed before the small adjustment. This is a very obvious examples of why storing logs can be very useful in simulating cyber-systems.

In reality, the fundamental purpose of a simulation is to accurately asses the behavior of a system without actually implementing it, avoiding the risk of wasting valuable resources such as time and money on an idea that does not perform as desired. Therefore, in the simulation phase of a project we want to be able to detect and fix all sorts of issues, even those that might escape the human eye. For this, storing variables of interest in log files is an excellent starting point. As seen in my example, many issues can be detected through this method and later be easily fixed.

In realty, a deadline miss on a crucial task means the task has to wait at least another period to be executed, which can lead to serious problems such as loss of control over the car and eventually serious accidents. To emphasize this importance, the constraints for the project say that the presence of deadline misses incurs a zero grade.

5 Conclusion

This final project was a good opportunity to actually experience working with an algorithm that we theoretically analyzed in class, although not using all of its capabilities. Part I of the project was useful to visualize how simulation code transitions to actually simulating the phenomenon. Meanwhile, the much more technical Part II was a good demonstration of why logging important data is a good practice when modelling computer systems.

Overall, I was able to achieve the desired results for both parts, even though for Part II I used an approach different from the suggested one as I found it much more convenient. My implementation provides all of the required functionalities for "debugging" the simulation, meaning for fine-tuning the parameters until a desirable performance is achieved. I believe this project help me solidify my knowledge acquired along the course of the semester as well as better understand certain aspects that were previously

not totally clear. Additionally, it helped me gain experience in working with large quantities of code developped by other programmers, as initially it was quite difficult to understand what all of the different classes and functions were designed to achieve.