CAB403 Assessment 2: Major Project

40 Possible Points

27/10/2023

Add comment

11/09/2023

Details

Changes to spec / Errata

- 17/09 - The requirement for the callpoint to check for confirmation has been removed - it now just sends packets in a loop. This change was made because the callpoint is not provided with a port to bind, so the original specification was impossible. The requirement for the fire alarm to send a confirmation to the callpoint has also been removed. This should simplify both programs.
- 13/09 - An erroneous 'delay' argument was removed from firealarm - it has been replaced with a 'reserved' argument that serves no purpose (to avoid breaking anything). Feel free to just pass any value in there when invoking it, and ignoring the argument in firealarm.

Overview

Task description:

Your task is to develop and submit multiple software components relating to a building's access, fire alarm and security systems. These components will each serve a unique role and will communicate with each other (with a combination of TCP and UDP over IPv4) to achieve larger outcomes.

These components include:

- An **overseer**, which talks to the other components and controls most of the building. A system consists of exactly one overseer.
- The **card reader controller**, which communicates to the overseer and sends successful card scans. A system consists of 0 or more card reader controllers.
- The **door controller**, which controls whether the security door it is attached to is open or shut. A system consists of 0 or more door controllers.
- The **fire alarm unit**, which connects to the overseer, the security doors and the temperature sensor mesh network, and has the job of activating fire alarms and opening certain doors in the

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

- The **temperature sensor controller**, which receives local temperature readings from the building and forms a mesh network with other sensors (as well as the overseer and fire alarm system). A system consists of 0 or more temperature sensor controllers.
- The **fire alarm call-point controller**, which allows anyone to activate the fire alarm manually. A system consists of 0 or more fire alarm call-point controllers.
- The **elevator controller**, which can be used to control an elevator's doors and motion, to transport authorised users between floors, depending on what floors that user is authorised to access. A system consists of 0 or more elevator controllers.
- The **destination select controller**, which will scan a user's card and allow a floor to be selected, and connect to the overseer (which in turn connects to the elevator controller to instruct it of the new route.) A system consists of 0 or more destination select controllers.
- The **security camera controller**, which reads pixel data from a camera and can also control whether the camera is pointing in a fixed direction or turning. If the camera controller detects motion, it will send a report to the overseer, which may raise a security alarm and close certain secure doors. A system consists of 0 or more camera controllers.
- A **simulator**, which will read in a scenario file, spawn the controllers and fire alarm unit and make simulated changes to shared-memory regions at particular times to simulate people entering the building, manipulating the swipe card readers, showing up on cameras, as well as simulating events like fires. A system consists of a single simulator.

Some of these components (the door controllers, fire alarm unit and fire alarm call-point controllers) are considered to be **safety-critical systems** and must be written to appropriate standards, with all deviations/exceptions documented.

Unit Learning Outcomes assessed:

ULO 2: Articulate industry standards and critically apply best practice for developing safety-critical systems.

ULO 4: Construct low-level systems programs to carry out authentic systems programming tasks.

Estimated time for completion	Weighting	Group or Individual	How I will be assessed
20-40 hours (per student)	40% of final grade	Group (1-5 students)	Using a 7-Point grading scale

Group composition

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

students in different practical sessions.

You may have up to 5 students in your group. The components you need to develop depends on your group size:

1 student: You must implement the **overseer**, the **simulator**, the **fire alarm unit** and the **card reader, door** and **fire-alarm call point controllers**. The overseer and simulator need to have the functionality to support the aforementioned controllers as well.

2 students: In addition to the above, you must also implement the **temperature sensor** controller, and add appropriate support for it to the overseer and simulator.

3 students: In addition to the above, you must also implement the **elevator** and **destination select controllers**, add support for elevator control to the simulator and add support for elevator dispatching and authorisation to the overseer.

4 students: In addition to the above, you must also implement the **security camera controller** and add support for it to the simulator and overseer, but the security camera controller does **not** need to do motion detection and the overseer does **not** need to automatically detect trespassers and raise the alarm.

5 students: In addition to the above, you must implement **motion detection** in the security camera controller and add logic to the overseer for raising a security alarm if there are no authorised users recorded entering a particular section where motion was detected.

See Assessment 2: Requirements for different group sizes (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-requirements-for-different-group-sizes?wrap=1) for more detail.

Note that the CAB403 teaching staff have very little latitude for mediating groups, and even less for intervening in group-related issues (e.g. group members not contributing equally). In practice, the advice I usually end up giving to members of dysfunctional groups is to 'submit the best work you can.' Unfortunately I am simply unable to compel uncooperative group members to get their work done; nor am I able to force overly controlling group members to listen to the other members of their group.

For this reason it is **strongly advised** that you form a group with similarly-minded students who are aiming for a similar grade. You do not need to register your groups in advance or tell us who you are grouped with, but when you submit to Gradescope, whoever submits the assignment will need to add the other members to the submission.

Software components

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507)

(https://canvas.qut.edu.au/courses/1

use OpenMP (-fopenmp), but your code will be running on a very minimal Linux distribution.

While you can name your source files whatever you want, the resulting programs have **required names** that you must use (and typing `make <program name>`, for example `make simulator`, must build that program.)

The following table lists all of the components, with links to pages with more detailed information about implementing that particular component.

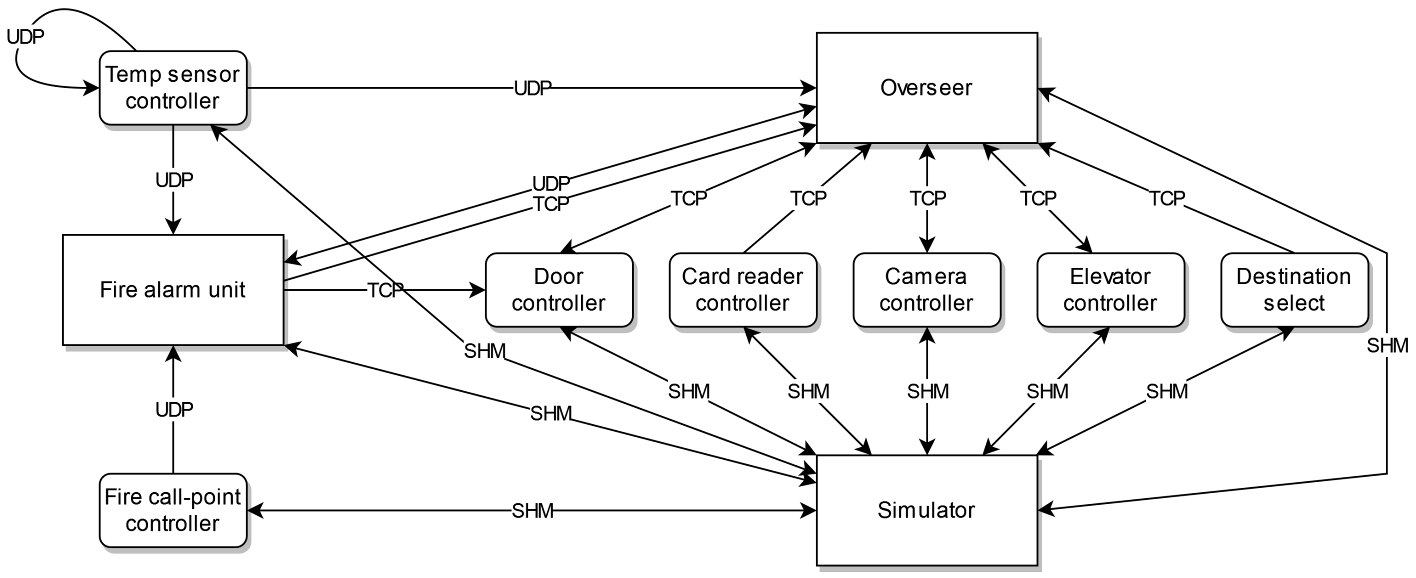
Component	Program name	Safety-critical?	For group sizes
Overseer (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-overseer)	`overseer`	No	1-5
Simulator (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-simulator)	`simulator`	No	1-5
Fire alarm unit (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-fire-alarm-unit)	`firealarm`	Yes	1-5
Card reader controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-card-reader-controller)	`cardreader`	No	1-5
Door controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-door-controller)	`door`	Yes	1-5
Fire alarm call-point controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-fire-alarm-call-point-controller)	`callpoint`	Yes	1-5
Temperature sensor controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-temperature-sensor-controller)	`tempsensor`	No	2-5
Elevator controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-elevator-controller-2)	`elevator`	No	3-5
Destination select controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-destination-select-controller)	`destselect`	No	3-5
Security camera controller			

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507)

(https://canvas.qut.edu.au/courses/1

In testing, the simulator will be started first, and it will start all the other components depending on the scenario file (for instance, if this scenario consists of 3 doors, 6 card readers and 4 temperature sensors it will start a controller for each of those.) Each component, except for the fire alarm call-point controller and the temperature sensor controller will connect to the overseer via TCP-IP to register itself (so the overseer knows which components are online).

Component architecture



Each controller is designed to interface with a specific piece of hardware. The hardware is simulated by the Simulator program and interactions with this hardware are performed via accesses through shared memory. The page for each controller shows the protocol for using shared memory, but in most cases the protocol is provided as a C struct containing some data, a mutex and a condition variable. The standard approach for working with the shared memory is that changes to it are made by locking the mutex, changing the shared variable, unlocking the mutex and then signalling the condition variable. If the component needs to wait for changes to be made to shared memory (for example, the card reader waits for someone to swipe a card) this is done by locking the mutex and waiting on the condition variable. More specific instructions are given on the pages for each component.

The controllers receive a path to the shared memory segment and an offset into that segment via command-line arguments. As the simulator spawns all other programs, it will create that shared memory segment and launch the other programs.

Safety-critical components

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

The fire alarm unit, door controller and fire alarm call-point controller are all considered **safety-critical components** and must be programmed to a higher standard accordingly. See the Week 6 lecture (https://canvas.qut.edu.au/courses/14485/pages/6-dot-1-lecture-safety-critical-systems) and Week 7 practical (https://canvas.qut.edu.au/courses/14485/pages/7-dot-2-practical) for more information about safety-critical programming.

The main source file for each of these components needs to feature a block comment near the top detailing any exceptions or deviations to safety-critical standards in the code and justifying your decision to deviate. This will be taken into account when marking your code for safety-critical adherence.

Marking criteria

Criterion A: Implementation of systems software following provided specification

Under this criterion we will evaluate how well your software runs, both through testing it in isolation and through manual inspection of your program source code. When you submit your code to Gradescope we will use your Makefile to build each of the components separately and put it in a scenario with our own versions of these components, so it is important that your code adheres to the shared memory, TCP stream and UDP datagram formats precisely. You can therefore receive partial marks if you implement some components but are unable to get others to work correctly.

Gradescope submissions will become available in Week 9, so we encourage you to submit to Gradescope early and often to ensure that your code is building and running correctly in our environment.

Marking criterion A

7: High Distinction (20 marks)	The implementation meets all requirements for a grade of 6 and there are absolutely no issues or limitations that compromise adherence to the specification
6: Distinction (16 marks)	The implementation meets all requirements for a grade of 5 and there are at most only very minor issues or limitations that compromise adherence to the specification
5: Credit (14 marks)	The implementation meets all requirements for a grade of 4, except that all functionality described in the specification has been implemented
4: Pass (10 marks)	The implementation contains most of the functionality contained in the specification, with the software performing reliably in our testing environment and with a respectable level of code quality

›
(https://canvas.qut.edu.au/courses/14485/modules/items/1431507)

›
(https://canvas.qut.edu.au/courses/1

2: Fail (5 marks)	The implementation greatly deviates from the specification and is unable to run properly in our testing environment, but a considerable amount of work has been done and that work appears to be on the right track
1: Low Fail (0 marks)	The implementation is either missing or non-functional

Criterion B: Use of systems programming constructs to meet project requirements

As Criterion A already tests how well your code functions, in this criterion we are looking at your use of system calls from a correctness perspective. Poor thread cancellation practices, race conditions and risky synchronisation practices will be penalised here even if the functionality is unaffected during testing.

Marking criterion B

7: High Distinction (5 marks)	The use of low-level systems programming constructs meets all requirements for a grade of 6 and all use of systems programming constructs is appropriate and correct, considering all potential exceptional circumstances
6: Distinction (4 marks)	The use of low-level systems programming constructs meets all requirements for a grade of 5 and systems programming constructs are used correctly and appropriately for the task at hand. All race conditions and undocumented/unspecified behaviour are avoided
5: Credit (3.5 marks)	The use of low-level systems programming constructs meets all requirements for a grade of 4 and all systems programming constructs required to complete the task are used
4: Pass (2.5 marks)	The implementation uses low-level systems programming constructs with only occasional omissions or incorrect use thereof
3: Marginal fail (2 marks)	The implementation uses low-level systems programming constructs to meet the requirements of the specification, but with omissions or errors that make the code less reliable
2: Fail (1.5 marks)	The implementation uses low-level systems programming constructs to meet the requirements of the specification, but with significant omissions or errors that cause poor reliability
1: Low Fail (0 marks)	The implementation does not use or barely uses systems programming constructs

Criterion C: Applying best practices for developing safety-critical software

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

scenarios, and in this assignment we want you to program these controllers as though they were part of a real life safety-critical system. Although these components are relatively simple programming exercises in terms of pure functionality, the additional care that needs to be taken makes each of them a significant task.

Marking criterion C

7: High Distinction (10 marks)	The submitted safety-critical components meet all requirements for a grade of 6 and follow best practices for safety-critical software development to the greatest extent possible, with all exceptions fully documented in a comment at the top of the source code for each component
6: Distinction (8 marks)	The submitted safety-critical components meet all requirements for a grade of 5, and any departures from best practices are documented
5: Credit (7 marks)	The submitted safety-critical components meet all requirements for a grade of 4, except there are only very minor departures from best practices, and these do not realistically compromise the safety of the software
4: Pass (5 marks)	All of the required safety-critical components have been submitted and have only minor flaws from a safety-critical perspective. Deviations/exceptions are documented and justified in a comment at the top of the source code for each component
3: Marginal fail (4 marks)	All of the required safety-critical components have been submitted and they have been clearly written to a higher standard, with care to avoid undefined and unspecified behaviour, unchecked input and memory leaks, but there are flaws or exceptions that either have not been documented or are insufficiently justified
2: Fail (2.5 marks)	Only 1 or 2 safety-critical components have been submitted, or the components that were submitted have serious flaws from a safety-critical perspective
1: Low Fail (0 marks)	No safety-critical components were submitted, or the submitted components are non-functional

Criterion D: Applying best practices for software engineering

While software engineering best practices are not a focus of this unit, it is important that you keep up your application of the fundamentals, as software that does not meet a certain baseline standard is not considered fit-for-purpose, irrespective of how well it runs.

Marking criterion D

7: High Distinction (5 marks)	The submitted software meets all requirements for a grade of 6 and shows that best practices for software engineering have been applied to a professional standard consistently across all submitted software

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

	professional standard
5: Credit (3.5 marks)	The submitted software meets all requirements for a grade of 4 and shows that best practices for software engineering have been applied consistently across all submitted software
4: Pass (2.5 marks)	The submitted software shows the application of best practices for software engineering in terms of code readability, consistent formatting, identifier choice, appropriate scoping of variables, no unnecessary casting, appropriate commenting and a lack of unnecessarily repeated/duplicated code
3: Marginal fail (2 marks)	The submitted software shows some application of best practices for software engineering
2: Fail (1.5 marks)	The submitted software is flawed in serious ways that have implications for maintaining the software, analysis of the software for correctness, performance of the software etc. Many best practices for software engineering are ignored
1: Low Fail (0 marks)	Submitted software is non-functional or highly incomplete, or there are very serious flaws in the engineering and quality of the software

What to submit

Upload your program code to **Gradescope** (https://www.gradescope.com.au/) (Gradescope submissions will be available from Week 9). Only one person per group needs to submit.
Make sure you include the following in your submission to Gradescope:

- The C source code (may consist of .c and .h files) of the components you've implemented.
- A Makefile with targets to build each of the components.

After submitting your project, click 'View or edit group' and add the other group members to your submission. See **Adding Group Members** (https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members) for more information.

Feedback:

Under normal circumstances, you will receive marks for each criterion via Gradescope within 15 working days of the deadline- check Gradescope for feedback from your grader.

Moderation:

The marking team will meet and discuss your submissions to ensure consistent grading.

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1

(https://canvas.qut.edu.au/courses/14485/modules/items/1431507) (https://canvas.qut.edu.au/courses/1