KF5010: Operating systems and concurrency Course Work 2017-18

Module tutor: David Kendall / Alun Moon

1 Dates and mechanisms for assessment submission and feedback

Date of hand out to students: 30 October 2017

Mechanism to be used to disseminate to students: eLP

Date and Time of Submission by Student:

- Report and C program: 23.59 on 11 December 2017

Mechanism for Submission of Work by Student:

- The report should be submitted via eLP using the Turnitin link `Assessment->Report`.
- The C program `tunnel_controller.c` should be submittted via eLP using the assignment link `Asessement->Program`

Date by which Work, Feedback and Marks will be returned to Students: Within 20 working days of submission date.

Mechanism(s) for return of assignment work, feedback and marks to students: email and appointment on request

2 Systems and Concurrent Programming (50%)

2.1 The Programming Problem

You are asked to imagine that a company is looking at the possibility of setting up an alternative to the Tyne Tunnel. The alternative tunnel will be to the west of the existing tunnel, connecting the south to the north side of the river Tyne. The company so far has struggled to raise investment and is able to build only a single-lane vehicle tunnel. This means that when traffic is travelling from north to south, vehicles on the south side must wait before entering the tunnel; similarly, when traffic is travelling from south to north, vehicles on the north side must wait. It is essential that a situation never arises where there are vehicles in the tunnel travelling in opposite directions at the same time. For safety reasons, it has been decided also that there must never be more than 3 vehicles travelling in the tunnel at the same time. Finally, in order to ensure some kind of fair access to the tunnel, a further requirement on the operation of the tunnel is that no more than 6 vehicles should be allowed to proceed through the tunnel in the same direction, if there are vehicles waiting to enter the tunnel in the opposite direction. Notice that this does not necessarily mean that traffic flow in both directions should be equal; just that a vehicle waiting at the front of the queue for entry to the tunnel will never see more than 6 vehicles pass it in the opposite direction before it is allowed to proceed.

It is possible that the business plan for this company requires some further scrutiny. However, for the moment, the company requires software to simulate the flow of vehicles through the tunnel to ensure that it is possible to develop control software for the operation of the tunnel that will satisfy the requirements stated above.

2.2 What to do

You should begin your solution to this problem by cloning the `git` repository `http://github.com/DavidKendall/kf5010_tunnel`. This provides you with the files: `main.c`, `tunnel_controller.h`, `tunnel_controller.c`, `console.h`, `console.c`, and `console_safe.c`.

It is possible to build and run this software as follows:

```
$ gcc -Wall -c console_safe.c
$ gcc -Wall -c tunnel_controller.c
$ gcc -Wall -o main -pthread main.c console_safe.o \
              tunnel_controller.o -lncursesw -lm
$ ./main
```

You should do this and observe the behaviour of the system. You should see a console window with a display that looks something like the one in Figure 1.

```
THE OTHER TYNE TUNNEL
---------------------


In Tunnel (NORTH) :  4
In Tunnel (SOUTH) :  3
Total     (NORTH) : 235
Total     (SOUTH) : 239
```

Figure 1: Display of simulated vehicle flow through tunnel

The display shows the number of vehicles currently in the tunnel, with their direction of travel. It also shows the total flow of vehicles through the tunnel since the software was started. You can see from the example display above that the control software, in its current form, fails even to meet the basic requirement that there are never vehicles in the tunnel at the same time travelling in opposite directions.

You are required to construct a solution to the problem by modifying only the file `tunnel_controller.c`. This file should be modified only by adding to it. You must not remove or comment out any lines of code that it contains now. The other files must not be modified at all. On completion, `tunnel_controller.c` should implement a *monitor*, providing the operations `enter_tunnel` and `exit_tunnel`, maintaining their types as given in `tunnel_controller.h`.

The use of these operations should ensure the satisfaction of the main requirements for the system:

1. Vehicles never in the tunnnel travelling in opposite directions at the same time

2. No more than 3 vehicles in the tunnel at the same time

3. No more than 6 vehicles pass in one direction if there are vehicles waiting to enter the tunnel in the opposite direction.

You should include an assertion in your program for each requirement that checks that the requirement is satisfied. You may need to introduce new variables into your program in order to be able to state your assertions. Refer to line 17 of the initial implementation of `tunnel_controller.c` for an example of the use of an assertion. Uncomment this line and rebuild and run the program. You should see that the program fails almost immediately with an assertion violation, confirming that this initial implementation does not yet satisfy requirement 1.

2.3 Additional requirements

You must include your code for `tunnel_controller.c` as appendix 1 in your report. It must be possible also to download the raw code file (`tunnel_controller.c`) from the eLP at `Assessment -> Program` and to include it with copies of the other files to build and run the system exactly as described earlier. Failure to satisfy one or more of these requirements will lead automatically to the award of a mark of 0 for the program as described in § 2.4

2.4 Marking of the program

Your program will be assessed on:

1. the extent to which it satisfies the specified functional requirements, including your use of assertions to demonstrate this;
   (10 marks)

2. the quality of the code: correct use of mutexes and condition variables to implement a monitor, layout, naming, comments, functional decomposition etc. Note, when built with `gcc -Wall`, your program should compile without errors or warnings. *(10 marks)*

2.5 What to include about the program in the report

The first section of your report should be entitled 'The tunnel program' and should include the following:

1. *The tunnel program*

   (a) A description of what you understand by a *monitor* in this context and a defence of why you believe your program correctly implements a monitor in `tunnel_controller.c`. *(5 marks)*

   (b) A description of how you have implemented the code to ensure that it is not possible for vehicles to be travelling in the tunnel in opposite directions at the same time. Explain any features of the `pthreads` library that you have used in this part of your implementation. *(5 marks)*

   (c) A description of how you have implemented the code to ensure that there are never more than 3 vehicles in the tunnel at the same time. Identify the assertion that you have included in your program to check that this requirement is satisifed and explain why the use of the assertion allows you to be confident that the requirement is satisfied. *(5 marks)*

   (d) A description of how you have implemented the code to ensure that it is never possible for more than 6 vehicles to pass in one direction while there are vehicles waiting to enter the tunnel in the other direction. Your answer should explain what you understand by the term *deadlock* and how you have avoided it in your program. *(5 marks)*

3 OS theory and concepts (50%)

Sections 2 and 3 of your report should include the following:

2. *OS abstraction and process management*

   (a) It is commonly observed that an operating system acts both as a *hardware abstraction layer* and as a *resource manager*. Explain what is meant be each of these terms and give *two* examples of an OS acting in each of these ways.

   (10 marks)

(b) Describe in detail the actions taken by an operating system to achieve a context switch between processes. Illustrate your answer with diagrams.

(10 marks)

3. *Security*
   An important requirement of many operating systems is to provide a secure communication function between processes. One approach to the provision of such a function is the *Secure Socket Layer* (SSL).

   (a) Identify what potential security violations are addressed by SSL. Explain briefly how SSL offers protection against each potential violation that you identify.

   (6 marks)

   (b) SSL makes use of both *symmetric* and *public-key* cryptography. Explain what you understand by these concepts, distinguishing clearly between them. Give examples of each. Identify how each of these cryptographic techniques is used in SSL, explaining the reasons for the choice of technique in each case.

   (7 marks)

   (c) SSL is susceptible to a *man-in-the-middle* attack. Explain the nature of this attack. Identify the precise vulnerability of SSL to this attack and discuss how users of SSL can protect themselves against it.

   (7 marks)

4 Further information

Learning Outcomes assessed in this assessment:

1. Describe the architecture of an operating system (OS) and its services, and evaluate its use in a variety of scenarios.
2. Discuss the process model and the scheduling, IPC and synchronisation services provided by an OS and reason informally about the behaviour of a multitasking system under a variety of scheduling algorithms.

3. Review the principal concepts and methods of memory management and file system implementation.
4. Identify a variety of security threats and examine appropriate OS mechanisms to protect against them.
5. Design, implement and evaluate solutions to problems of I/O device handling, synchronisation, communication and timing for multitasking systems, using appropriate OS services and concurrent programming techniques.

Assessment Criteria/Mark Scheme: The coursework consists of

1. a software development project assessing systems programming and concurrency (50%)
2. questions on OS theory and concepts (50%)

More detailed marks allocation is provided in § 2 and § 3.

Referencing Style: Harvard

Expected size of the submission: Your report should be about 5 to 7 A4 pages in length (assuming 10pt, normal margins, and excluding appendices). There is no fixed penalty for exceeding this limit but unnecessary verbosity, irrelevance and 'padding' make it difficult for the marker to identify relevant material and may lead to some loss of marks.

Assignment weighting: 100%

Academic Integrity Statement: You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of plagiarism or any other form of misconduct in your work. Refer to the University's Assessment Regulations for Northumbria Awards if you are unclear as to the meaning of these terms. The latest copy is available on the University website.