RULES OF THE EXAMINATION.

1. Do questions 1 and 2; do one of questions 3, 4 and 5.
2. You must work independently. Phoning your partner to find out what is in your kernel is not considered independent.
3. You may use any source of information you want on this examination. Any information from sources you consult MUST be referenced. (Your memory, course notes and lectures are the only exceptions.)
4. I prefer answers in PDF format (whatever.pdf). If PDF is inconvenient then I accept plain text (whatever.txt) but you will have to stretch a little to make diagrams. Three pages (~1000 words, or less if there are diagrams) is as long an answer as you need for any question, but only if you write the appropriate three pages. Regurgitating the question or course notes get few or no marks. (See 7., below.) Put your name, student number and userid on every page.
5. Your answers should be submitted by e-mailing them to me at wmcowan@cgl.uwaterloo.ca.
6. A strategy that worked well for me as a student was to read the exam twice, then do something else for a couple of hours, then plan my answers, rest again, and finish by writing them.
7. Please remember that the questions are open-ended: you get most of your marks from going beyond simple answers; explicit instructions are intended as prompts to get you started in the right direction. Answering only what they request gets you about half the available marks for a question. The other marks come from ideas that go beyond the question as asked.
   You gain marks for the thoughts that you contribute to your answers. Write other people’s thoughts, including mine, only to the extent that I need them to understand your answer.
8. When the examination says ‘your kernel’ it means the kernel you actually created, not an ideal kernel or the kernel you wish you had created. When the examination says ‘your OS’ it means your kernel plus the other tasks (couriers, notifiers, servers) on top of which applications run. When the examination says ‘your train application/project’ it means the application you planned to create, in what you would consider to be its final form.
9. All measurements and estimates must have units. If your unit is ticks translate it into milliseconds using the size of tick in your kernel.
10. There may be places in the examination where you must make assumptions. Do so, being certain that you explain your assumptions and how they are related to what you are saying.
11. Read each question carefully, and more than once. More marks are lost because of misunderstood questions than from any other single cause.
12. In all questions you should give your reasoning. More marks are given for reasoning than for correctness.
13. The cover page exists only to fulfil the registrar’s regulations.
14. Read 7., above, once more.

* To understand why I handle the exam as described please consult the Introduction to the course.
Question 1. Performance Analysis of your Kernel.

This question concerns two aspects of your kernel, Send/Receive/Reply and AwaitEvent. (Feel free to run your kernel to get estimates for the values required in this question.)

1.a. Send/Receive/Reply (SRR). In the second part of the kernel you measured the time taken by SRR for small and large messages with optimization turned off.
   
   (i) What are your times for the final version of your kernel?
   
   (ii) During a system call three things occur, context switch(es), copying memory, and manipulating kernel data structures. Estimate the parameters $A$ and $B$ in the equation
   
   $SRR\text{Time} = Am + B,$
   
   where $m$ is the amount of memory copied in bytes. Answers should have units. Which of the three things that occur contribute to $A$? to $B$? Give your reasoning.

1.b. AwaitEvent. Assume that an interrupt occurs while execution is in the kernel processing an AwaitEvent system call on the interrupt’s event. Between a call to AwaitEvent and its return the three things mentioned above may occur, context switch(es), copying memory, and manipulating kernel data structures. If you measured the minimum call/return time for AwaitEvent, give the result, otherwise estimate it. Estimate the amount of time spent in switching context, copying memory and manipulating kernel data structures.

1.c. Context Switches. Counting the number of context switches in the two parts above, you can estimate the time for a context switch.

   (i) What is the context switch time?
   
   (ii) Look at your kernel code and consider whether the time required to run its instructions is commensurate with the time you gave above. Give numbers and explain your result.
   
   (iii) Not counting turning on the caches and compiler optimization, what did you do after the third kernel assignment to speed up your context switch? Explain your answer. Estimate the speed up.

1.d. Caches. Assume that both instruction and data caches of the ARM are lockable. Explain how you could use cache locking to improve the time taken by your context switch. Estimate the execution speed-ups when you turn on the caches, both with and without locking. Give as many concrete details as you can, such as how many cache lines to lock, what to put in the locked cache lines, the effect of a smaller cache on performance of user code, and so on. Remember that all context switches are either from the kernel or to the kernel.
Question 2. Deadlock, etc.

The diagram to the right shows a closed track on which two trains, 1 & 2, are running. Each is driven by a driver task, and the two tasks have the same executable code. Each train has a destination, indicated on the diagram as d1 and d2. At the moment the diagram is drawn both trains own track from their current positions to the striped divider, and have initiated requests for more track. Both requests will fail and the trains will stop nose to nose at the striped divider.

Two servers provide the train drivers with resources they need to carry out their missions: a route server provides the shortest route to a requested destination that is not blocked by another train; a reservation server provides ownership of track when it is requested, but only of track that is not already owned. The train drivers are well-behaved: perfect calibration means that they never overdrive a reservation; track is immediately returned when it is vacated.

Reservations are of arbitrary length. The maximum amount of track that can be owned at a time by a single train is the length of one side of the square track.

The red dots mark the position of sensors.

2.a. Servers that queue. Assume that the reservation server queues requests that cannot be granted, waiting to reply until the requested track has been returned by its current owner, at which point the request is successful. Assume also that the route server queues requests for routes that are blocked or otherwise impossible, replying a route only when the blockage is removed. What happens next in the pictured configuration? Give details of what the drivers and the servers do?

2.b. Servers that refuse. Assume that the reservation server refuses requests that cannot be granted, replying the reservation if it is available and ‘refusal’ otherwise, and assume that the route server does likewise. What happens next? Give details of what the drivers and the servers do?

2.c. Early re-routing. Train driver tasks often respond to refusal by polling the server, which generally recreates queuing with substantial additional overhead. A different strategy I have seen many times in final demos is early re-routing: every time the train driver passes a sensor it requests a new route. Early re-routing might be a substitute for reservations, assuming that a train on a collision course will, sooner or later, block the route of the potentially colliding train, creating a re-route that guides each train away from the other, thereby avoiding a collision. Using the track in the lab give two examples where early re-routing works well, and two where a collision ensues. What aspects of your collision avoidance policy does early re-routing violate?
Question 3.  Tasks and Objects

A task is an entity, consisting of executable instructions plus data (state), provided and supported by an operating system; an object is an entity, consisting of executable instructions plus data (state), provided and supported by the run-time environment of a computer language. These properties of a task are superficial, but the similarities of server tasks to objects runs much deeper. For example, requests to servers run well-defined code sequences with arguments, return values and context, like the methods of objects. In this question you consider properties of objects and how they can be implemented using tasks.

3.a. Basic concepts. Below is a short list of basic object properties. For each of them describe – in words or pseudocode – a task or set of tasks having those properties.

(i)  Data.
(ii) Methods.
(iii) Interface inheritance. (Type extension.)
(iv) Dynamic typing.
(v) Polymorphism.

3.b. Other object concepts. Add to the above list other object concepts you consider important, including descriptions of schematic task implementations for each concept.

3.c. Properties objects usually don’t have. There are a few properties most objects don’t have, but which are natural to tasks.

(i) Persistence is one: tasks, like most daemons, persist for the entire time that the operating system is up. Objects, in contrast, persist only during an execution of the program that creates them. Explain how it can be useful to have the internal state of a task persist across the running of different applications.

(ii) Persistence across executions of an operating system is also potentially useful, and provided, not always exhaustively, for objects in the form of serialization. Explain how you could implement it as a feature of our task system.

* Some languages, such as Self, are integrated with an operating system in order to provide persistent objects.
Question 4. Making One Train Follow Another

In the past few terms several groups have made projects in which one train followed another very closely, without colliding and without the two trains getting too far apart. For example, several terms ago a group did a project in which trains could, at random times and places, be disabled on the track. Then a rescue train went out to bring it back. The rescue train approached the disabled train closely, then both went together to a siding, the disabled train leading, the rescue train following. The rescue train then matches its velocity to that of the disabled train and the two trains travel as one*. An analogous situation occurs when a train is supposed to travel a route with its position matching a predetermined moving location.

4.a Two trains travelling as one. When two trains travel as one, sensor attribution can be tricky. Each sensor is hit twice, once by the leading train, once by the following train.
   (i) What is the time interval between the sensor hits? Give times in milliseconds for several speeds, using measurements from your project.
   (ii) Suppose you detect the time interval by continual sensor polling, and time-stamp each complete sensor poll as you receive it from the train controller. What intervals would you measure between successive time-stamps? Give the range of true time intervals you should associate with each measured value. (Numbers from your train project please.)
   (iii) Suggest an alternative method of measuring that would estimate the intervals more precisely.

4.b Maintaining equal velocities. When the following train gets too close you must slow it down a bit, and when it gets too far behind you must speed it up a bit. This is called feedback control: measured error is fed back into the control unit, affecting the velocities in the system. It is described mathematically as
   \[ v(t + \delta) = v(t) - a\delta(v(t) - v_d(t)) \]
where \( v_d(t) \) is the desired velocity and \( a \delta \) the gain of the signal fed back. (Notice the similarity of this equation to the one used for dynamic calibration.) If you have seen feedback control previously you have probably seen the continuous time form \( v'(t) = -a(v(t) - v_d) \), which you get in the limit \( \delta \to 0 \).
   (i) You actually measure the times of sensor hits, and want to control the distance between two trains using velocity error. Explain in detail how you use the measured times to control velocity. (I would like to see both words and equations.)
   (ii) Moving the speed back and forth between two different values is suggested as a method moving at an intermediate velocity. As a result the train is almost always accelerating, never at a constant velocity. Do you need an acceleration calibration to make this work? Why or why not?

4.c Avoiding collisions. The main cause of collisions is measurement lag. The differential equation is really simultaneously a difference equation and a differential one: \( v'(t) = -a(v(t) - T) \), where \( T \) is the amount by which the velocity used is stale. We solve it by assuming a solution of the form \( \exp(-\lambda t) \), which reduces the differential equation to an algebraic one \( \lambda = a e^{\lambda T} \).
   (i) This equation has positive solutions for \( \lambda \) for some values of \( aT \), and complex solutions for others. What would you observe when the solution is complex?
   (ii) When you design a control system using negative feedback you get to choose \( a \), which you choose to negate the harmful affects of \( T \). But you would prefer not to choose a very small value for \( a \). Why?

* In the presence of reservations it is common, but not necessary, for the two trains to travel as one on a single reservation.
Question 5. Relativistic Reservations.
Here is a different method for preventing the collision of trains. It’s based on the following idea.

- There are dangerous places to stop on the track, such as on top of switches, or with the pick-up raised by a sensor.
- The rest of the track (light grey) is safe.
- A reservation (dark grey) is a region of space time, as shown to the right, where the train starts stationary in one safe region (light grey), and ends stationary in another safe area.
- A second reservation (black) allows the train to remain in a safe place for a finite amount of time.

Because these reservations use space-time pictures like the one on the right, which also occur frequently in Einstein’s theory of special relativity, I call them relativistic reservations (RR).

5.a Shape of an RR.
   (i) An RR has a minimum extension in space. Why? How wide must it be?
   (ii) The two RRs shown above have linear sides. Is this realistic? Why or why not?

5.b RR rules. In class I described a reservation system, basing it on a policy, the enforcement of which is distributed across several tasks. The rules guaranteed that there would never be a collision.
   (i) Give a set of rules for RR systems, and explain why it avoids collisions.
   (ii) Describe how to distribute responsibility so that rules are followed, using the task structure of your train application as a model.

5.c Sharpening up RRs. As the train travels it passes sensors.
   (i) How might you use sensor feedback to improve an RR as the train travels?
   (ii) Sharpening up reservations means that they are temporally dynamic. Explain how the reservations you used changed over time. Draw a changing reservation as drawn above.
   (iii) Draw on a diagram like the one above showing your reservations changing over time.
   (iv) Another way to have changing reservations is to have two time dimensions. Try drawing a changing RR in a 3D diagram with two temporal dimensions.

5.d Failures of train control. As all of you know from the project demos trains that refuse to move as directed are common in the real world*
   (i) Explain how you would modify RRs to make them more resistant to trains that stall.

* ‘Real world of the trains lab’ that is.