Rules of the Examination.

1. You must work independently.
2. 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.)
3. 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. Two pages (~800 words, or less if there are diagrams) is as long an answer as you need for any question; some questions require less. Put your name, student number and userid on every page.
4. Your answers should be submitted by e-mailing them to me at wmcowan@cgl.uwaterloo.ca.
5. A strategy that works well for me is 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. The total writing time should be about three hours.
6. 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. 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.
7. 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’ it means the application you tried to create in what you would consider to be its final form.
8. When asked to estimate something you should respond using an almost logarithmic, which contains only the values ..., 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, .... All measurements and estimates must have units.
9. There may be places in the examination where you can say something clever by making assumptions. I encourage you to do so. Be certain that your explain your assumptions and how they are related to what you are saying.
10. Read each question carefully, and more than once. More marks are lost because of misunderstood questions than from any other single cause. To show that you read this far, for one mark put the phrase ‘nolens volens’ at the top of your first page. The advice to read at least twice is self referential.
11. The cover page exists only to fulfil the registrar’s regulations.
Do Questions 1 & 2.

Question 1. Performance Analysis of your Kernel.

This question concerns two aspects of your kernel, Send/Receive/Reply and AwaitEvent.

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 were your times? During a system call three things occur, context switch(es), copying memory, and manipulating kernel data structures. Using the logarithmic scale introduced in the introduction, estimate 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. Give your reasoning.

   (ii) Assume that an interrupt occurs while execution is in the kernel while it is processing the AwaitEvent call. Between a call to AwaitEvent and its return the same three things 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 each of the three things.

   (iii) Counting the number of context switches in (i) & (ii), you can estimate the time for a context switch. 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.

   (iv) Assume that both instruction and data caches of the ARM are lockable. Explain how you could use cache locking to improve your context switch time. Estimate the execution speed-ups when you turn on the caches, both with and without locking. Give as many concrete details as you can.
Question 2. Bottlenecks.

2.a Performance. 'Bottlenecks play a disproportionate role in system performance in a real-time system.' Explain what this sentence means, taking into account the difference between response time to input events to an application, and throughput from the average rate of event occurrence. In explaining, feel free to agree or disagree.

2.b Your Train Application. Many times in class I stated that the main bottlenecks in your train application would occur in communication between your application and the train control interface. Describe the worst case bottleneck that can occur for
   (i) communicating with the train control interface,
   (ii) communicating with the console/monitor/display, and
   (iii) CPU execution.
Estimate response times for worst cases in each of these bottlenecks, giving the concrete details underlying your estimates. (Remember the role of priorities.)

2.c Clever Algorithms. When a bottleneck provides less performance that your application needs it is sometimes possible to compensate by using clever algorithms: a simple example might be to poll only the sensors that will provide data you can use, and only at times when the data is likely to appear, doing extra work in the CPU to get good predictions to guide polling. Explain, with concrete detail, what you did/would have done, to substitute one resource for another
   (i) in your train application as demoed, and
   (ii) in the train application you hoped to demo.
Question 3. Calibration.

3.a Control. ‘The details of calibration always depend on the controlled system, including the application task it performs.’ Explain this statement, giving concrete examples from your train application, and how it differs from other possible applications.

3.b Static versus Dynamic Calibrations. Explain the benefits are drawbacks of static and dynamic calibrations, giving concrete examples from your train application. Which of the following did you actually implement:

(i) static calibration,
(ii) dynamic calibration, or
(iii) a mixture of static and dynamic calibration?

Giving concrete examples explain which you implemented and why. Would you do things differently if you were starting your project again? Why?

3.c Calibration Performance. You are expected to have enough familiarity with the train set that you can provide estimates for

(i) train speeds,
(ii) time lags in giving commands to trains and switches,
(iii) times and distances for stopping trains,
(iv) and so on.

The train control and reservation systems in your train application had some difficult cases. Describe two such cases, giving all of the

(i) concrete details and estimated numbers with units,
(ii) performance requirements,
(iii) technical details, and
(iv) so on

that you consider relevant.
Question 4. Task Structure.

Warning. This question is a little more fuzzy* than most of the others. Do it if you are in the mood to be imaginative and creative.

4.a The Server. No doubt, you got tired of hearing the word ‘server’ over and over again during the course. But, as far as the task structure described in the course is concerned, the server is an essential concept: in fact, the SRR semantics we use was defined so as to make servers easy to write. Linking server roles to message passing semantics, give three features of a server that are essential for it to play the role of a server, and explain why each is essential.

Possible key words. Request, priority, early.

4.b Weaknesses. Servers can be considered to be

(i) a high-level program structuring concept,
(ii) support dynamic communication structures, or
(iii) single-threaded persistent objects.

Servers have weaknesses, such as

(i) living in global name spaces,
(ii) awkward simulation of Unix-pipe semantics, and
(iii) requiring globally-defined, but statically uncheckable message types.

Explain how these weaknesses are inherent, and how they are related to the different views we can have of the server as a program structuring concept.

Hint. There is not intended to be a one-to-one mapping between the lists of views and weaknesses.

4.c Disquisition. The server will never disappear. It reappears in one context after another, which suggests that it is a significant organizing concept in human thought.† For example, CSP, a programming language construct with Unix-pipe semantics appears, appears with channels accepting assignment, so that appropriately dynamic servers are possible; objects use locking mechanisms to achieve thread safety so that tasks can use object services within Smalltalk message passing concepts.

Many network servers accept requests from anonymous, possibly malicious and certainly incompetent, requesters. To enhance reliability and security they enforce strict protocols that do not expose internal data structures. Protocol negotiation is an essential first step in setting up a connection. What would protocol negotiation add to your kernel? What sort of application might use it? Should it be done by the kernel or by user tasks associated with the server? Draw a possible task configuration and describe how it works.

---

* ‘A little more fuzzy’ is probably just a synonym for ‘off the wall’.
† The assumption underlying this statement is the legitimate (?) inference from ‘We can only think about things for which we have concepts,’ to ‘We can think most easily about things for which we have simple concepts.’
Question 5. Distributed Kernels.

Recently, Ron Minnich at Sandia Labs, combined 196 TI OMAP3530 ARM CPUs running at 720MHz into a small supercomputer. It is connected as follows.

- Seven CPUs on boards 17mm by 58mm on a ‘motherboard’* 70 mm by 293 mm. One ethernet connector and seven IP numbers per motherboard.
- Seven motherboards plus a switch per shelf.
- Four shelves.

So we could abstract this as 196 ARM CPUs on an ethernet in a star topology.

You can see pictures of a preliminary box here,
- http://picasaweb.google.com/rminnich/Strongbox?authkey=Gv1sRgCI6PwZXM57iZ8gE#,
and of a box with a cleaner hardware layout here,

5.a Thin-wire Kernels. For the purpose of this question assume that we are going to put a distributed kernel onto the seven CPUs of a single motherboard, each CPU having 32M of memory. These CPUs do not share a single address space: memory bus congestion would kill them immediately. The CPUs have communication hardware with the following properties.

(i) A CPU puts into a buffer at a standard location in its address space a message addressed to another unit†, followed by a message in any format.
(ii) A command flushes this buffer onto a common bus.
(iii) When the addressed unit hears its name on the bus, it picks up the message and puts it in a standard place.
(iv) The addressed unit then interrupts its CPU.

This is called a thin-wire system because the CPUs do not have the thick wire of a system bus connecting them.

5.b Message Passing. Architect a kernel for such a system. It should support message passing between CPUs so that all tasks on all CPUs are a single cooperating application. Here are some criteria.

(i) Tasks do not know about the multiple CPUs at all. There is a global TaskId space in which the tasks live.
(ii) Each CPU handles a distinct set of interrupts and tasks waiting on those interrupts are tied to that CPU. Otherwise tasks can be created on any CPU.
(iii) Make any other assumptions you like.

In your solution explain carefully how kernel data structures are shared, how shared data is minimized, and how shared data is synchronized.

After describing you architecture describe step-by-step two tasks on different CPUs doing SRR.

5.c Bonus. Answer in detail any of the following.

(i) Describe how your design could continue executing given the failure of any single CPU.
(ii) Describe how the kernel parts would negotiate to place a newly created task in the best location.

* In the Gumstix product literature this component is called a ‘stagecoach’.
† By its IP number in the example described above.

You may have experienced frustration with the train set at one or more times during the last month. A very common response to this frustration is to think about using computer vision for monitoring the trains. This question examines some aspects of this alternative.

6.a Levels of Performance. A computer vision system can make available a variety of different inputs to your program. Here is an ordered list of a few.

(i) Exact position on the track and velocity of an identified train: ‘train 52 at 452 mm past sensor A5, moving toward sensor B4 at 67 mm/second’.
(ii) Exact position on the track of unidentified trains: ‘452 mm past A5, 36 mm to switch 16, ...’.
(iii) Real-space position on the train set: ‘452 mm north and 36 mm west of fiducial mark’ X’.
(iv) Bit map of the train set from above.

Each input requires the CPU to do more work than the one above it, with an added time lag. Explain, with concrete examples and estimates, how you would fit each level of performance into a train application.

6.b Bottlenecks. With different levels of performance in the computer vision system the bottlenecks change. Describe what you think is the critical moment for each input configuration described in Part 6.a and estimate the performance you require. Use velocities, response lags and track distances you measured while doing your train application, and remember units.

6.c Control Modes. All the calibration problems that plagued you while you developed your train application would disappear, or so it seems, if the computer vision subsystem were fast enough that you could switch from dead reckoning to immediate feedback for position estimation. Describe, giving concrete details such as task structures, how you train application would change if feedback control was possible. Using the changed application structure and concrete values from your train application, estimate the response time required from computer vision input to switching a turn out or changing the speed of a train in order to use feedback control.

6.d Preserving the Trains Course. Feedback control would change the trains course significantly. Would this be a benefit or drawback in terms of learning in the course? You already can predict my answer, just by knowing that I think calibration is an ubiquitous problem in real-anything computing, and that you can only learn it by doing it. But feel free to disagree. Be concrete in your answer, stating specifically what educational value would be lost and gained.

---

* A fiducial mark is an agreed upon location from which things are measured.’
† Informally, the ‘critical moment’ is the set of circumstances that is most likely to lead to an error, such as a lost train, a collision, or a derailment.