<table>
<thead>
<tr>
<th>Course Abbreviation:</th>
<th>cs452</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Title:</td>
<td>Real-time Programming</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time and Date of Examination:</th>
<th>10.30, Friday, 9 December, 2011 to 13.30, Saturday, 10 December, 2011.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration of Examination:</td>
<td>27 hours.</td>
</tr>
<tr>
<td>Number of Pages:</td>
<td>10.</td>
</tr>
</tbody>
</table>
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. All numbers 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 you 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 ‘Blueland’ 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. Task descriptor and data cache

The task descriptor (TD) is the representation of a task in the kernel. How big it is has a large effect on kernel performance. How do you know?

After measuring the performance of your kernel in Kernel2, we found that turning on the caches reduced the average send-receive-reply (SRR) time by a factor of 7, equally for copying the message and for the remainder of your kernel (context switch, scheduling, etc.). A big part of the reduction was the result of putting kernel data structures into the data cache. However, a group that tuned its TD to the cache obtained a much bigger time reduction. This question is about designing a TD with the data cache in mind.

For the purposes of this question, the data cache in the EP9302 has the following characteristics.

1. Its size is 16 Kbytes.
2. Its line length is 8 words.
3. It has 512 lines.
4. It is 64-way associative.
5. There are 8 sets.

Thus, the 32-bit virtual address that appears in your program is broken up as follows

1. [0:4] addresses within a line.
3. [8:31] is the tag.

1.a The TD in your kernel.
   (i) What did you include in the TD of your kernel?
   (ii) Where is the remainder of the task state?
   (iii) How big is your TD?
   (iv) How many cache lines would be required to hold all your TDs?
   (v) How many cache misses does it take to load a TD into the cache? The word 'alignment' should appear in your answer.
   (vi) Discuss the effect of the above characteristics of your TD on the performance of your kernel.

1.b The smallest possible TD. Answer (i)–(vi) above for the smallest possible TD.

1.c The biggest possible TD. Many embedded systems use what they call a ‘control block’, which has all the information we save and restore when entering and exiting the kernel plus other elements of the TD. We might consider this to be the biggest possible TD. answer (i)–(vi) above for this TD.

1.d The time-optimized TD. Answer (i)–(vi) above for the TD you think would minimize the time taken by TD manipulation in your kernel.

1.e Performance. Showing your work give estimates for the SRR time of each of the four TDs above. (Assume a reasonable value for memory references that hit the cache and ones that miss it and ignore instruction fetch times.)
Question 2. Relativistic Reservations.

I suppose AM radio stations still have programs to which you can phone in and dedicate songs. If songs, why not examine questions? Therefore, this question is dedicated to Simon, who is viscerally unhappy with the reservation system I teach in class.

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 on curves where you can’t get going again.
- The rest of the track 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.
- Another 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, we call them relativistic reservations (RR).

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

2.b RR rules. In class I described a reservation system, basing it on a set of rules 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.

2.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.

2.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.
Question 3. Effect of Caches on Kernel Performance

Part of the speed-up mentioned in Question 1 is the effect of the instruction cache on the time required to fetch instructions. This question is about the amount of speed-up that should be attributed to the instruction cache. On this question I am looking for answers that are as precise as possible.

When doing this question assume that the instruction cache is identical to the data cache described in Question 1.

Hint. When answering this question it will be very helpful to have the assembly code of your kernel and a load map of your kernel available, especially for 3.d.

3.a  Hits and misses. The instruction clock of the CPU runs at 200MHz; the high-speed system bus (HSB), used to access memory, runs at 100 MHz. Loading a register from the cache takes one cycle of the CPU clock; loading a register from memory takes two cycles of the HSB.
(i) How long does it take to load a word from the cache?
(ii) How long does it take to load a word from memory?
(iii) Does this mean that turning on the cache is limited to a factor of four speed-up? Explain.

3.b  Small message size. Suppose your kernel gets a Receive request when there is already a Send request on the Receiving task’s sendQ, the message size is four bytes, and no kernel instructions are in the cache when the context switch occurs.
(i) How many instruction fetches occur?
(ii) How many instruction fetches are cache misses? How many are hits?
(iii) How many loads and stores to the data cache occur?

3.c  Large message size. Answer the questions in 3.b for a 256 bytes message.

3.d  Speeding up your kernel. There are several ways it is possible to manage the instruction cache so as to speed up your kernel. Describe a few of them in as much detail as you can. (They may be things that you actually implemented, or ones that you would be able to implement. Please, no science fiction.)
Question 4. Busses *

Several times during the term we talked about busses. The clock of the EP9302 SoC used in CS452 is usually 200MHz. The EP9302 implements two on-chip busses:

1. Advanced High-Speed Bus (AHB),
   (i) defined by ARM,
   (ii) clocked at 100MHz†,
   (iii) fully arbitrated,
   (iv) up to sixteen wait states can be inserted by slave,
   (v) four, eight and sixteen word overlapping bursts, and
   (vi) split cycles,
   and
2. Advanced Peripheral Bus (APB),
   (i) defined by ARM,
   (ii) clocked at 50 MHz†,
   (iii) only one master (no arbitration),
   (iv) no wait states possible, and
   (v) no other exotic features.

Each UART has a bus-facing zone clocked by the APB and a serial-line facing zone clocked by sixteen times its bit-clock. That is, if the UART is transmitting/receiving at 2400 bits/sec, the serial-line facing zone is clocked at about 38KHz, and if at 115200 bits/sec, at about 1.8 MHz. The UART clock is not synchronized to the APB clock.

Beside the link to the exam on the Assignments page is a link to more detail about the AHB, the APB and the bridge that joins them.

Note. Addresses on the APB are all uncachable, which means that any load or store on control/status or data registers of a UART require cycles on the AHB and APB.

4.a Read/Write. How long does it take to read a register in the UART? How long does it take to write a register in the UART?

4.b Inside the UART. When an APB read/write requires a transfer from the UART zone to the APB zone what are the minimum and maximum transfer times for each of the bit rates used in your project?

4.c Status responses. With FIFOs, a write to the transmit hold buffer triggers, within the UART clocked region, negation of the transmit ready bit. How long is it from the completion of bus arbitration on the HSB until the transmit ready bit reads correctly on the APB? Give answers for both bit rates. Is it possible to read an incorrect status bit at either of the bit rates used in your project?

4.d At the other end of the serial line. At the other end of the serial line is another UART in another computer system (U2). Suppose the two UARTs (U1 is the one we control) use flow control to avoid over-running internal buffers.

Let’s focus on the trigger that changes the flow control state in U2 when it is receiving. It must be the arrival of a byte, which occurs only when the stop bit is successfully detected. What precedes the trigger?

1. In U1 a byte was transferred from the hold register to the shift register.
2. U1 asserts XMIT_IRQ in the ICU, followed by the usual interrupt processing.

* The spelling busses looks wrong to me – as if it is the plural of ‘buss’ – even though it’s the plural given in my dictionary (Concise Oxford). However, the alternate form ‘buses’, which I see from time to time, looks as though it should be pronounced ‘boozes’, which is even worse.
† The AHB and the AHP can be configured to operate at lower clock rates: a power-minimizing OS would vary the speed as its load varies.
3. The shift register put the byte on the serial line, ended by the stop bit.

After the stop bit arrives in U2 software, or firmware
1. detects that another byte cannot be handled without overflow,
2. negates CTS by writing the U2 modem control register, and
3. U2 negates the CTS signal on the serial line on the next edge of its UART clock.

In U1 the change in CTS immediately asserts the MODEM_IRQ in the ICU, followed by interrupt processing.
   (i) Draw three time lines, one for events within U1, the second for events in U2, the third for
       interrupt processing as it would be done in your kernel. Estimate the length of temporal
       intervals on the time lines.
   (ii) For each of the two bit rates used in your project, is it possible that CTS would be too late to
       prevent overflow in U2? Explain your answer.
   (iii) If your answer to (ii) is ‘yes’ explain how you would modify your OS to prevent overflow. If it is
       ‘no’ explain how your software prevents overflow?

4.e Wait states on the APB. Suppose it were possible for the slave (UART) to inject wait states into the APB
   just as the bridge can inject wait states into the AHB. Then when a zone transfer is pending the UART can hold
   off responding to a read request until the zone transfer is complete. It does so by conditioning the APB
   transaction on an edge in the bit clock combined with asserted Enable.
   (i) What is the maximum number of wait states injected into the HSB at 2400 bits/sec? at 115200
       bits/sec?
   (ii) How would this change in the APB affect your implementation of the serial server, the serial
       notifier and/or the kernel interrupt handler?
Question 5. QNX

QNX (www.qnx.com), recently acquired by RIM, is a company based on a real-time operating system kernel created by Dan Dodge and his partner in an early offering of cs452. But you may be puzzled if you look at the API of QNX: it seems to have every communication/synchronization primitive you have ever encountered.

I know that at least one group implemented locks as a way of controlling access to resources. There are two ways of implementing locks, as a kernel primitive and as a lock server. There is a third way of doing the task that a lock performs, putting the resource itself in a server which accepts requests using Send/Receive/Reply and performs the requested operations.

5.a Software Management. In my view, ‘mixing primitives from several synchronization models is a problem looking for a program to inhabit.’

(i) If you agree with the above statement explain what the problem is, and what happens when it occurs. If you disagree with the above statement, please explain why.

(ii) As the manager of a programming team developing using QNX how would you avoid the problem?

5.b Lock Server. Let’s reject locks as a kernel primitive compatible with a micro-kernel.

(i) Explain the incompatibility, or disagree if you choose. (Even if you disagree you must do the remainder of the question.)

(ii) Describe the API you would provide for user code accessing a lock server, explaining why each part of it is necessary. Give a schematic description of how you would implement the API. (Your API should take into account the possibility that the set of lockable resources may not be known at compile time.)

(iii) How would you exclude a rogue task from using the resource without acquiring the lock?

5.c Resource Proprietor. One reason often given for preferring locks to servers is execution efficiency. This part of the question makes an elementary comparison between the efficiency of accessing a data structure controlled by a lock server or a proprietor. Each requires

1. synchronization, acquiring and relinquishing the lock (lock server) or receiving and responding to the request (proprietor),
2. manipulating the data structure by the user (lock server) or the proprietor, and
3. transferring data.

(i) In terms of efficiency of manipulating the data structure, (2.), is a wash, but synchronization and data transfer are not. Describe how the two solutions differ in those two cases.

(ii) Estimate, using the timings you have for your kernel how much time each consumes if $n$ bytes of data are transferred.

(iii) At what value of $n$ do you change from preferring one solution to preferring the other?

5.d Priorities. Comment on the following statement. ‘A resource proprietor uses the resource at a priority determined by the importance of the resource, while a lock server allows a resource to be used at the priority of the user.’
Question 6. Distributed Kernels.

Last night at Morty's a few students had a short discussion with me that is relevant to this question. If you choose to answer this question, and were part of the discussion, please indicate so at the beginning of your answer. P.S. I wondered this morning if I should just remove this question from the exam, but it's one students like.

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,


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.

6.a 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.

6.b 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.
Question 7. Cyclic Execution and Polling Loops

7.a. The polling loop that you wrote at the very beginning of the course, and cyclic execution, which we discussed at the end of the course, are two different ways of doing almost the same thing. Briefly describe how each is structured, specifically indicating code that you could reuse if you were converting a polling loop to cyclic execution.

7.b. When we think about polling loops we think about response time, the worst case amount of time that occurs between an interrupt and the execution of the first line of code responding to it. When we talk about cyclic execution we think about admission control, how to decide whether a new task submitted to the cyclic executive will be able to execute without overflowing the period of the cycle in the worst case.

(i) How do we calculate the worst case response time in a polling loop?
(ii) How do we calculate whether or not a new task can be admitted in cyclic execution?
(iii) I claim that admission control and worst case response time are very similar. Am I right? Give reasons for your answer.

7.c. In your polling loop you learned two programming techniques for improving the response time of critical tasks:
1. putting a high priority task in the polling loop more than once, and
2. splitting a long-running low priority task into more than piece to open up a window for polling by other tasks.
What are the analogous techniques for cyclical execution?

7.d. Cyclic execution is very natural for a device like a mobile telephone.
1. At a level below the operating system, time is divided into basic periods.
2. Time-critical tasks, like animation, playing music, or communicating with the line card in the ground station are scheduled at the beginning of each period. They are subjected to admission control when they start to ensure that adequate time is available for them.
3. The leftover time in the period is provided to an operating system, possibly your kernel, which provides asynchronous, non-time-critical interaction with the user. The OS is told each time it is scheduled the amount of time it has to run, and is expected to yield the processor in time for the next cycle to start.
4. The OS can insert new time-critical tasks into the period, when the user wants to start a new activity, and delete old tasks from the period when an activity is complete.

(i) Obviously, such an architecture is interesting only when the OS can interact with the time-critical tasks. Describe possible who must block on whom, and why, when this interaction occurs.
(ii) The communication is naturally CSP-like, which provided a different way of structuring a server. Describe how you could have time-critical code supply services to asynchronous tasks.