University of Waterloo

cs452 – Final Examination

Spring 2009

Student Name:	_____
Student ID Number:	_____
Unix Userid:	_____

Course Abbreviation:	**cs452**
Course Title:	**Real-time Programming**

Time and Date of Examination:	**15.00, August 10, 2009 to 15.00, August 11, 2009.**
Duration of Examination:	**roughly 150 minutes.**
Number of Pages:	**7.**

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. One to two pages (~400 to ~800 words or less if there are diagrams) is about right for each question. 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 two to three hours.
6. Do question 1 and any two of questions 2–4.
7. You will gain marks for the thoughts that you contribute to your answers. Write other people's thoughts 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' it means the application you tried to create in what you would consider to be its final form.
9. 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 'Giant Hogweed' at the top of your first page. The advice to read at least twice is self referential.

Question 1. Short Answers.

1.a Real-time. Most real-time processing courses distinguish between two qualitatively different types of real-time programming: hard real-time, in which failure to meet real-time constraints results in disaster, and soft real-time, in which failure to meet real-time constraints is merely inconvenient or annoying.

(**i**) This distinction was treated as unimportant or even non-existent in the course. Why? Instead the course emphasized acceptable response time as the feature differentiating one sort of real-time system from another.

(**ii**) Give an example of a real-time system requiring ten picosecond* response times. How must systems like this be constructed?

(**iii**) Give an example of a real-time systems that requires response times measured in days. What is the easiest way of constructing such a system?

(**iv**) Give the range of response times over which your kernel would be useful. Explain the thinking behind the lower and upper limits of the range.

1.b Link Register.

(**i**) In the ARM architecture the link register, `r14`, plays a special role. What is it?

(**ii**) When a hardware interrupt occurs two different link register values exist. What are they and how is each one used?

(**iii**) In your kernel where did you save each of these values?

(**iv**) You don't have to worry about the existence of two values when a software interrupt occurs. Why? (Hint. The compiler helps you.)

1.c Processor Modes. Most likely, the context switch in your kernel made heavy use of processor modes. System mode is a recent addition to the ARM architecture, added in v4.

(**i**) Describe two properties that distinguish system mode from other modes.

(**ii**) Give two examples from your kernel illustrating how and why each of these properties was important.

(**iii**) Suppose system mode was not available Describe a serious problem you would have had in your kernel and explain how you could have worked around it.

1.d Calibration. In your train application it was necessary to obtain actual measurements in centimetres and seconds. You got centimetres with a tape measure.

(**i**) Describe how you obtained measurements of time intervals, how accurate the measurements were, and how you knew the accuracy.

(**ii**) Did the accuracy limit the performance of your application? If so, how? If not why not? Explain your answer.

---

* 1 picosecond = 10^{-12} seconds.

Question 2. Bottlenecks

2.a Cyclic Execution. When we talked in class about bottlenecks we did so in the context of cyclic execution, and I argued that worst cases in your system looked a lot like cyclic execution. A simple abstraction of the lowest level of execution of your kernel might include three external triggers (interrupts) that trigger changes in execution.

1. An interrupt from the timer. Suppose these happen at most once every 50 milliseconds.
2. An interrupt that occurs when a byte arrives on the UART. Suppose these happen at most once every 100 milliseconds.
3. An interrupt that occurs when the UART is ready to transmit a byte. Suppose these happen at most once every 100 milliseconds.

(This abstraction considers only one UART, and ignores the flow-control interrupts.)

(**i**) For each trigger draw a time line showing the sequence of task executions that occurs in your kernel when each type of trigger occurs. Use typical execution times for each task taken from your kernel to label the time lines.

2.b The Critical Instant. The critical instant occurs when all triggers appear at the same time. Answer the following questions assuming that all other tasks have lower priority than the lowest priority task in part **2.a**.

(**i**) For the critical instant draw a single time line indicating everything that occurs following it. Use the priorities you used in your OS.

(**ii**) What is the length of time from the critical instant until processing is complete for each trigger?

2.c Maximum frequencies. On the basis of the numbers calculated in part **2.b**(ii) you surely see that there is lots of spare time for your kernel to get through this bottleneck. Now, assume that the triggers speed up by a factor of k, trigger 1 occurring every $50/k$ milliseconds, triggers 2 & 3 occurring every $100/k$ milliseconds.

(**i**) Define the criterion you would use for not getting through the bottleneck.

(**ii**) What is the maximum value of k your OS can handle, assuming it is doing nothing else. (Remember that triggers might occur several times before all processing is complete.)

(**iii**) Draw a time line for your maximal k.

Question 3. Distributed Implementations of Your OS

You could build your OS as a distributed OS. Messages might, for example, be distributed over a dedicated ethernet, thereby allowing tasks on one machine to interact with tasks on another.

(i) Here is a simple way to do it. Every machine contains a separate kernel. 'Something' examines each Send or Reply, and determines which machine has the task ID to which it is addressed, after which the message is sent to the appropriate kernel for processing. Assuming that kernel data structures are magically synchronized across machines describe the sequence of events that occurs when Send/Receive/Reply occurs. Include a time line as part of your description, showing when processing shifts among components of the system.

(ii) '"Something" examines each Send or Reply, and determines which machine has the task ID to which it is addressed, after which the message is sent to the appropriate kernel for processing.' Suggest how the 'something' in the above sentence might work. Describe what needs to be synchronized between machines, and how synchronization might be accomplished.

(iii) Here is another simple way to do it. There is a single kernel on a special machine, and all memory is unified into a single address space.* Memory references outside the address space of the local machine are automatically mapped into requests to the memory subsystems on other machines, which respond after performing the requested operation. Describe the sequence of events that occurs when Send/Receive/Reply occurs. Include a time line as part of your description, showing when processing shifts among components of the system.

(iv) Each of these 'simple solutions' has a serious drawback. Describe each drawback. Which do you think is worse? Consider the nature of typical applications and hardware features that might diminish the cost of the drawback when giving your answer.

(v) Distributed implementations should be robust against machine failure. Choose the system you consider to be the most robust and describe how it might cope with the failure of a machine.

---

* For example, the high four bites of an address could be the machine id.

Question 4. Communicating Sequential Processes (CSP).

As discussed in class CSP communication primitives allow the compiler to find types mismatches between sending and receiving code in multi-task applications. In Occam ii the syntactic construct used to support CSP is the channel, which is declared (instantiated) with a protocol, essentially a type. For example,

```
CHAN OF BYTE: pipe
```

instantiates a channel on which a sender task can write individual bytes

```
pipe ! 'a'
```

and from which a receiver task can read individual bytes

```
pipe ? nextChar
```

(**i**) You will note immediately that sending and receiving code must share a name space. What does this mean, and why is it necessary?

In Occam ii name space sharing is usually accomplished by the task instantiation mechanism, using something like the following,

```
CHAN OF BYTE: pipe
PAR
    producer ( pipe )
    consumer ( pipe )
```

where the common name space is provided by scoping rules.

(**ii**) Your kernel has a component that purportedly creates a common name space that is more flexible than the one in Occam ii. Describe what it is, what the names are, and what values are returned. Give arguments that the name space in your kernel is more flexible than the one in Occam ii?

(**iii**) Write high level pseudo-code for your component in Occam ii for client code equivalent to `WhoIs( "ClockServer" )`.

(**iv**) As mentioned in class your kernel merely finesses the name space problem by using reserved strings. Explain.

(**v**) Draw the task communication diagram for the highest level of your train application, and write Occam ii pseudo-code that initializes the application.

(**vi**) Briefly, i.e. in about 100 words, discuss how static the Occam ii implementation is compared to your actual implementation.

Question 5. Trusted and Untrusted Tasks.

Your kernel implementation was significantly simplified by assuming that all application programmers are trustworthy, a common assumption in traditional embedded systems. In 3G mobile telephones, however, real-time embedded systems confront the untrusted programmer whenever a foreign application is loaded, which happens frequently.*

For this question assume a more simple scenario: a Unix shell task that is created in response to successful use of a login task. To mimic typical multi-user Unix the login task is created automatically on every terminal connected to your OS.

(i) The shell is a user interface, albeit a very simple one. Assume the user is a real-time peripheral with the following time scale:

- 100 milliseconds, fine,
- 1 second, tolerable, and
- 10 seconds, unacceptable.

How do you protect tasks with shorter latencies from the shell? How do you protect the shell from tasks with longer latencies? Consider priority inheritance in your answer.

(ii) Describe four ways in which a foreign user task started by a shell could subvert the normal operation of your OS, and the state into which each one would put your OS.

(iii) For each answer you give above describe in detail a modification to your kernel that would either prevent the subversion or detect and terminate it, explaining how and why it works.

* It is, for example, possible for a malicious application to put up web pages identical to app store pages, downloading key loggers and even root-kits in work-alike applications. It would allow spammers to put their communication costs on your phone bill.

Question 6. Bonus Question.

Although we used a new processor this term I maintained the set of services supported by the kernel, which correspond roughly to the hardware of a first generation PC. Our processor board has a wide variety of unused capabilities, including

- ethernet hardware,
- USB hardware,
- a parallel port,
- two analogue to digital inputs,
- a five-channel digital to analogue converter,
- floating point hardware,
- 8 Mbytes of Flash RAM,
- an IDE compact flash connector, and
- a PC/104 bus connector.

(i) Suggest ways that any of the above capabilities could be used to improve the course. Bear in mind that it is unrealistic to expect significant alterations in the train sets. In your suggestions, explicitly take into account

- the amount of support software required by students,
- the difficulty and the amount of programming done by students,
- the bottlenecks in work flow you experienced this term, and
- anything else you think would improve student learning in the course.

I would like to thank all of you for your understanding when I was pre-occupied by difficult events in my private life, and for the sympathy expressed in your card.

Your feelings support my optimism that all is unfolding for the best.