RULES OF THE EXAMINATION.

1. Do one of questions 1–3, one of questions 4&5, plus one other question.
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 require answers in PDF format (whatever.pdf). 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 two thirds of the available marks for a question. The other marks come from ideas that go beyond the question as asked.*
8. 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. Provide references to thoughts that are not your own or the instructor’s.
9. 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.
10. All measurements and estimates must have standard units. If your unit is ticks, for example, translate it into milliseconds using the size of tick in your kernel.
11. There may be places in the examination where you will want to make assumptions. Do so, being certain that you explain your assumptions and how they are related to the question you are answering.
12. Read each question carefully, and more than once. More marks are lost because of misunderstood questions than from any other single cause.
13. In all questions you should give your reasoning. More marks are given for reasoning than for correctness.
14. The cover page exists only to fulfil the registrar’s regulations.
15. Read 7., above, once more.

* To understand why I handle the exam as described please consult the Introduction to the course.
DO ONE OF QUESTIONS 1–3 AND
ONE OF QUESTIONS 4 & 5 AND
ONE OTHER QUESTION.

In the first part of the kernel you had to design the memory layout of your kernel, which probably stayed
pretty much the same during kernel and program development. Many students ask how much memory a task
needs, and I give them the standard answer, ‘It depends.’ You have had a lot more experience with
programming tasks by now, and maybe it’s time to think about how one might estimate the memory needed
for a stack.

This question asks about bounds on the size of the stack. (When the question says ‘bounded’ it means
bounded by a finite number. You may assume that the number of tasks in the system is bounded by a finite
number, \( N \), which is the size of the array of task descriptors.)

1.a. A Courier. A courier has a very simple skeleton*.
FOREVER {
    Send(srcTid, REQ, message);
    Send(dstTid, message, ACK);
}

(i) Draw a sketch of the data elements on this courier’s stack when it is as big as it ever can be.
(ii) Give the size of the stack, making realistic assumptions about the size of the data elements it
uses.

1.b. The Receive Notifier for a Warehouse. The skeleton of a receive Notifier is something like this.
FOREVER {
    AwaitEvent(RCV_EVENT);
    ch = *UART_DATAREGISTER & 0x7f;
    Send(warehouseTid, &ch, ACK);
}

(i) List the elements that are on the Notifier’s stack when the stack is at its biggest, giving the
maximum size of each and the total size of the stack.

* ‘Skeleton’ means that I have omitted error-detection, initialization and corner cases, leaving only the data
structures used in the FOREVER loop. You are expected to estimate or assume how big are the bounds on
those data structures.
1.c. **The Warehouse Server.** Packaging in of a warehouse is more complex, with a skeleton like the following.

```c
FOREVER {
    Receive(reqTid, req);
    switch(req.type){
    case CLIENT:
        enqueue(clientQ, reqTid);
        if(!empty(pkgQ)) Reply(dequeue(clientQ), dequeue(pkgQ));
        break;
    case NOTIFIER:
        insert(pkg, req.data);
        if(full(pkg)){
            enqueue(pkgQ, pkg);
            if(!empty(clientQ)) Reply(dequeue(clientQ), dequeue(pkgQ));
        }
        break;
    }
}
```

(i) List the elements that are on this server's stack when the stack is at its biggest, giving the maximum size of each and the total size of the stack.

(ii) What must be true for this stack to be bounded?

(iii) This bound requires a bound on the environment of the Client/Warehouse/Notifer/Interrupt System. Explain in detail.

(iv) As a programmer you do not have the option of providing a stack bigger than the amount of memory available for this part of your program. What can you do in practice if there is a theoretical, but very unlikely, possibility of unbounded input? Give two examples from everyday life: one in which the solution is good, the other in which the solution is bad.
Question 2. Idle Time.

In the course I stressed that real-time performance indicators are useful for debugging and tuning. The percentage of run time taken up by the idle task as part of kernel 3, and maintained it during your project. Most of you have some estimates of running times from kernel 2. There may be others you can easily obtain by running your kernel, but if that’s not easy there are other ways to get them.

2.a. Kernel 3. In the third part of the kernel you implemented an idle task, which ran when no other task was ready to run. You were also required to display on your terminal screen, the fraction of time that the idle task was running. Here you calculate how much time you would expect the idle task to run when the test program in the assignment is run. You should give numbers with units in answering this question.

(i) Once initialization is complete, all computation is initiated by an interrupt from the timer, which occurs once every ten milliseconds. Two sequences of execution can occur. The simpler of the two is the following
   1. Idle task interrupted by timer.
   2. Kernel processes hardware interrupt.
   3. Notifier returns from AwaitEvent, sends to Clock server.
   5. and so on.
   Complete this sequence until the Idle task recommences running.
   
   (ii) Estimate the computation time for each item in the sequence, describing how you arrived at it. Sum up the numbers and estimate the likely error.

   (iii) Describe the second more complex sequence that can occur and make estimates as you did above.

2.b. Serial Input. Assume you are getting full bandwidth input from the terminal. Further assume that the input is going no further than the server where it is buffered in an effectively infinite buffer and that there are no timer interrupts.

(i) How often do receive interrupts occur? Give your calculation.

(ii) List the sequence of execution that is initiated by each receive interrupt, from the interruption of the Idle task until the Idle task is running again. Estimate the execution time for each item in the list, and give the expected fraction of time that the Idle task is running.

2.c. Combining the Estimates. You now have two estimates of CPU usage for two independent pieces of execution.

(i) The simplest estimate would add together the estimates, weighted by how often they are executed. Calculate the idle time using this method.

(ii) The two executions interact when an interrupt occurs before processing of a proceeding interrupt is complete. Write down the sequence of execution that occurs when a timer interrupt and a receive interrupt are raised simultaneously. You should be able to create the sequence largely by cut and past of your previous sequences.

(iii) Is there any difference other than order of execution? Is there a significant difference in idle time? There may be significant differences in other important metrics of real-time performance. Suggest two and explain why they matter more than idle time.

(iv) Think of your everyday life. Give an example of poor service you have received because a human server was multi-plexing two requests at once. How could you solve that problem without hiring more servers?

*The legendary E. Hunter Harrison, asked for the secret of his success, replied, ‘I never try to solve an operational problem by capital investment.’*
Question 3. Hardware Protection of Tasks.

In a shared computer system user programs must be prevented from damaging the operating system. Equally, user programs should be prevented from damaging one another. Commonly, this is accomplished using hardware protection of memory. You certainly saw the hardware component that implements protection, the memory management unit (MMU), in your second year hardware course, and a variety of ways of using the MMU in your third year operating systems course.

Your kernel probably did not program the MMU; RedBoot set the MMU to give you a convenient view of memory, which you used in your kernel. You may want to refer to the MMU section of the ARM processor manual when answering this question.

3.a Handling an Illegal Access. When the MMU in our ARM CPU detects an attempt to access protected memory it causes a data abort exception, which should trigger code that cleans up the problem. For example, most versions of Unix kill a program that tries to access protected memory, providing only a cryptic message like ‘Segmentation violation’.

(i) Suppose you wanted to do what Unix does when a data abort exception occurs. You would need to enter the kernel, print ‘Segmentation violation’ on the terminal and exit. (I am assuming that you would immediately debug the problem and start a new bug-free execution.) List the sequence of steps that occur as this happens, including special values that are needed to support the steps.

(ii) To help the person debugging the program the kernel might do a little more than just printing a generic error message. List three pieces of useful information that the kernel might provide, explaining how you obtain them and why they would be useful for debugging.

3.b. Protecting the Kernel. A simple static protection mechanism might protect all kernel memory from access by user tasks.

(i) List three distinct blocks of memory that should be protected from access by user tasks, explaining what is special about each one.

(ii) Describe how you would set up the MMU to provide this protection.

(iii) This protection scheme catches many stray pointer errors. What is the most common value of an uninitialized pointer? Why?

3.c. Protecting Individual Tasks. Protecting the kernel helps, but it’s also common for tasks to interfere with one another. You may have experienced, for example, user task stacks running into one another. So here’s something you might want to do. When a user task is active arrange the MMU so that it can access only its own stack. An attempt to access other memory would cause a data abort exception.

(i) Describe in words how you would set up the MMU to get this protection.

(ii) Describe how the MMU tables would change when a context switch occurred from a user task to the kernel, and from the kernel to a user task.

Usually table entries needed by the MMU are found in a small fast associative memory called the table lookaside buffer (TLB). Otherwise, they must be loaded into the TLB from RAM, which is very slow.

(iii) If you had fifty tasks, how often would the new table be found in the TLB?

(iv) Compare the TLB to the L1 cache. How much speed-up did you get in the message copying part of message passing when you turned on the L1 cache? What does this tell you about having to go to memory for a new MMU table? Remember that every memory access, data or instruction, goes through the TLB.
**Question 4. Sensor Attribution**

When a train is moving on the track your program only gets reports of triggered sensors, and has to turn the location of a triggered sensor and the time of its triggering into the location of a train. Most groups do this as a two stage process. In the initialization stage each train is registered, which gives it an initial position. Subsequently, when a sensor is triggered, the train that triggered it is inferred on the basis of earlier positions and velocities of trains, and on the track state. Some groups do the inference prospectively, predicting in advance which sensor a train is likely to trigger next; others do it retrospectively, working backward from the trigger to earlier train states.

4.a. Basic Sensor Attribution. Almost every group accomplished some level of sensor attribution. Base your answer in this part of the question on what your group did.

(i) When your train project receives information that a sensor has just been triggered, how does it determine which train triggered the sensor. Give your answer in terms of significant system components*, such as track server, train driver, etc. Describe the sequence of information exchanged between the components and its content. Explain the sub-problems being solved and the nature of the solutions.

(ii) We expected sensor attribution to take possible failures into account. How did your project handle† failures?

4.b. Sensor Attribution for Following Trains. When one train follows another closely the nature of sensor attribution changes. A set of closely following trains is normally treated as a single train for route finding and/or track reservation: the single train then triggers each sensor several times.

(i) My description suggests that several sensor reports are merged into a single sensor attribution. Explain how you would accomplish this, using the same granularity of description asked for in 4.a. above.

(ii) There are axle counting sensors on many rail lines. As a train passes they count the number of axles in the train and send an audio report to the engineer, ‘218 axles.’ It’s a fail-safe against the end of a train falling off. Describe how you might use such a sensor to make sensor attribution easier for closely following trains. In your answer consider the practical problems you would have to solve.

4.c. Sensor Attribution in Grand River Transit. Inside each bus Grand River Transit provides audio and video information announcing the next stop which requires the bus to know its position. Doing so seems simple: the bus uses GPS to know its position and the current schedule to know the sequence of stops. But the failure modes I observe suggest that the story is more complex.

(i) When a bus makes a detour its sign announces the next few stops along its route when it would reach them on an undetoured trip. I saw this behaviour in final demos when sensor attribution failed because a train went the wrong way through a switch. Use this analogy to rule out possible simple models of sensor attribution for the bus sign. Give your reasoning.

(ii) Here’s a new failure mode that has appeared in the last few months, an off-by-one error: the bus traverses its route announcing as the next stop either the stop it just left or the next next stop. (I have not yet seen an off by two fault, but it might just be rare.) What models can you rule out on the basis of this failure. Again, give your reasoning.

---

* A component is virtual, not physical. It may be a task, a collection of tasks, or even part of a task.
† Or, ‘How would your project have handled ...’
Question 5.  Timed Reservations.

Every term several groups consider implementing timed reservations. Two level train control systems in the real world usually have time controlled reservations at the upper level where trains are receiving routes, backed up by a lower level system that accommodates states when trains fail to maintain their schedules.

5.a. Time-distance Diagrams. We visualize train location using time-distance diagrams like the one to the right. The black pattern shows the space-time occupied by a train that is stopped, then moves to the right at a constant speed. When the track runs in a circle, we cut the circle and unroll it. Thus, the grey pattern shows a train doing a trip that goes twice around a circular track. Taken as a whole the diagram shows two collisions, in which the grey train and the black occupy the same location at the same time.

(i) Which train has the greater velocity? Explain.

(ii) The diagram has no units. Label points on the diagram that show the length of the grey train, the amount of time it takes to cross a point on the track and its velocity.

5.b. Trains Passing without Colliding. We can make a slightly more complex distance-time diagram. (Distance-time diagrams need not be drawn to scale.)

(i) Draw a time-distance diagram showing two roughly circular tracks sharing, like the track shown in the diagram to the right. Hint. When one track branches from or joins to another track you need to make the paper branch. The diagram in 5.a. has a hack that joins a flat sheet of paper into a virtual circle. Use a similar hack to make the a single sheet of paper split and rejoin.

(ii) On your diagram show the paths of two trains travelling anti-clockwise, one on each loop, passing one another without colliding. If you want them to travel indefinitely without changing velocity and without colliding they must have correlated velocities. Give an equation that describes the correlation.

(iii) On another copy of your diagram show the paths of four trains, two running clockwise on the upper circle, two running anti-clockwise on the lower circle, all running at constant velocities, and none ever colliding.

(iv) The last part has an obvious solution when all velocities are the same. Are different velocities possible? If so, give a non-trivial example; if not explain why not.

5.c. Scaling Time-distance Diagrams. Time-distance diagrams are neat, especially so Einstein used them to describe trains passing one another in his 1905 theory of special relativity.* But in the real world train dispatchers use two dimensional diagrams like the ones many groups created for their projects, where the trains move along a two-dimensional diagram in real-time, which is great for seeing what the program is doing right now, but not so great for seeing what will happen to trains in the future.

(i) Build enough of a time-distance diagram for a track layout in the trains lab to convince me that even the best time-distance diagram does not make the future comprehensible.

(ii) Many user interfaces exist to make intelligible phenomena that mix time and space, from editing video to understanding the flow of air in a wind tunnel. Sketch an idea that could augment virtual trains moving in real-time on a virtual track to improve a user’s ability to understand future collisions in a train control interface.

* He called them space-time diagrams.