University of Glasgow

SAMPLE EXAMINATION
(Duration: 2 hours)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

ADVANCED OPERATING SYSTEMS (M)

(Answer 3 out of 4 questions)

This examination paper is worth a total of 60 marks

You must not leave the examination room within the first hour or the last half-hour of the examination.

1. You are given a system of independent periodic tasks to be scheduled on a single processor in a pre-emptive manner: $T = \{T_i\}$ for $i = 1 \ldots n$ where $T_i = (\phi_i, p_i, e_i, D_i)$ for each i.

(a) Assume $p_i = D_i$ for each i. What is the maximum schedulable utilization of the Earliest Deadline First algorithm for this system? Is this a necessary and sufficient condition?

If the inequality $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ is satisfied, then the system is schedulable using EDF. This is a necessary and sufficient condition.

[3]

(b) Assume that $p_i \geq D_i$ for each i. What is the maximum schedulable utilization of the Earliest Deadline First algorithm for this system? Is this a necessary and sufficient condition?

If the inequality $\sum_{i=1}^{n} \frac{e_i}{D_i} \leq 1$ is satisfied, then the system is schedulable using EDF. This is a sufficient condition unless $p_i = D_i$ for all i, in which case it is also necessary.

[3]

(c) Assume that $p_i = D_i$ for each i. What is the maximum schedulable utilization of the Rate Monotonic algorithm for this system? Is this a necessary and sufficient condition?

If the inequality $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq n\left(2^{1/n} - 1\right)$ is satisfied, then the system is schedulable using RM. This is a sufficient condition; there may be systems which do not satisfy this inequality that still yield feasible schedules.

[3]

(d) Assume that $p_i \leq D_i$ for each i. Under what conditions will the maximum schedulable utilization of the Rate Monotonic algorithm for this system be identical to that stated in response to part (a) above?

If the system is simply periodic – i.e. if for every pair of tasks T_i and T_k in $\mathbb{T}$ such that $p_i < p_k$, p_k is an integer multiple of p_i. This is a necessary and sufficient condition.

[3]

(e) You are provided with the following system definition (all tasks are independent, and are scheduled pre-emptively on a single processor system): $T_1 = (0, 2, 0.4, 2)$, $T_2 = (1, 4, 1, 4)$, and $T_3 = (0, 5, 1.5, 5)\}$

(i) Can these tasks be scheduled using the Earliest Deadline First algorithm? Explain your answer.

Total utilization U = 0.4/2 +1/4 + 1.5/5 = 0.20 + 0.25 + 0.35 = 0.75. Since this is less than 1.0, and since the relative deadlines are identical to the periods, then this system is schedulable using EDF.

(ii) Can these tasks be scheduled using the Rate Monotonic algorithm? Explain your answer.

Total utilization U = 0.75. $n\left(2^{1/n}-1\right)$ for n = 3 is 0.779. Since U ≤ 0.779, this system is schedulable using RM.

The parameters of the system are changed, such that $T_3 = (0, 8, 4, 8)$.

(iii) Can this new system be scheduled using the Earliest Deadline First algorithm? Explain your answer.

Total utilization U = 0.4/2 +1/4 + 4/8 = 0.20 + 0.25 + 0.5 = 0.95. Since this is less than 1.0, and since the relative deadlines are identical to the periods, then this system is schedulable using EDF.

(iv) Can this new system be scheduled using the Rate Monotonic Algorithm? Explain your answer.

Total utilization U = 0.95. The system is simply periodic, so the system is schedulable if U ≤ 1.0. Therefore, this system is schedulable using RM.

[2+2+2+2]

2. You have been hired into a software engineering firm to replace a design engineer that has recently left. His legacy is the design of a real time embedded system. Your first task upon arrival is to critically review the design, since the implementation phase is to start within two weeks.

(a) The system consists primarily of N independent periodic tasks that must meet their deadlines, along with random aperiodic jobs involved in the user interface of the system. It is desirable to minimize the average response times of the aperiodic jobs. All tasks execute on a single processor, and are pre-emptively scheduled. Your predecessor had decided to use a simple deferrable server. The total utilization of the periodic jobs is U_p, and the maximum utilization permitted while still being able to meet all deadlines is $U_{max} > U_p$. Your predecessor has specified that the deferrable server size should be $U_{max} - U_p$.

Do you agree with his choice? If so, indicate why; if not, how would you change the design?

A deferrable server is the simplest bandwidth-preserving server algorithm. It retains its budget whenever it is not executing, and loses any unused budget at replenishment time. The deferrable server is usually the highest priority task in the system. The schedulability condition for such a system is dependent upon which scheduling algorithm is chosen.

Regardless of the scheduling algorithm chosen, the deferrable server algorithm retains its budget at times when it should not – i.e. it is too aggressive. As a result, the utilization constraints are of the form $U_p + u_S + e_S/p_N \leq$ Umax.

The system should use a simple sporadic server with size $U_{max} - U_p$. This is guaranteed to restrict the utilization by the server to u_S, thus guaranteeing that the periodic tasks will all meet their deadlines.

[5]

(b) Various reasons led to using an earliest deadline first scheduling algorithm for the periodic tasks and the server. Several of the periodic tasks compete for exclusive access to a shared resource, and it is essential that the system not deadlock. Your predecessor designed the system to use the priority inheritance protocol to minimize blocking due to resource contention. Do you agree with his choice? If so, indicate why; if not, how would you change the design?

The greedy nature of the priority inheritance protocol means that the system cannot be guaranteed deadlock-free.

The priority ceiling protocol could be used, but it has a higher run-time overhead than the preemption-ceiling protocol. The latter should be used in this system.

[5]

(c) We usually assume that the context switch time is negligible when determining if a system can be scheduled. Your predecessor has instrumented a prototype of the running system, and has determined the maximum context switch time for jobs in

execution to be T_{CS}. He has, therefore, increased the execution time for each of the periodic tasks (including the bandwidth-preserving server) by $2 \times T_{CS}$.

Do you agree with this approach? If so, indicate why; if not, how would you approach this problem differently?

This is the correct way to approach the problem. Each job incurs a context switch when it pre-empts the current job, and incurs another context switch when it completes and releases the processor. Therefore, the execution time of each task in the system should be increased by $2 * T_{CS}$.

[5]

(d) Each job in periodic task T_i queries an array of sensors; each query to each sensor takes ~1ms to complete, and there are N sensors to be queried serially; after issuing the query, the job self-suspends until the I/O completes. Your predecessor has accounted for this situation by increasing the execution time e_i for jobs in the task by $N \times T_{CS}$.

Do you agree with this approach? If so, indicate why; if not, how would you approach this problem differently?

Every time one of the task's jobs self-suspends, it generates two additional context switches: the first to schedule the highest-priority, ready-to-run job when it releases the processor, and the second when it resumes itself, preempting the lower-priority job running on the processor. Therefore, the execution time e_i for jobs in the task must be increased by $2 \times N \times T_{CS}$.

[5]

3. **(a)** A simple form of automatic heap management is *reference counting*. Briefly explain how a reference counting system manages memory allocated to objects, and explain under what situation it can fail to reclaim objects.

[5]

Each object is augmented with a count of the number of references to that object [1 mark]

The reference count is incremented each time a reference to the object is created, and decremented each time a reference is destroyed [2 marks]. When the count reaches zero, the object is reclaimed [1 mark].

Reference counting can fail to reclaim objects if there are cyclic references [1 mark].

(b) Describe one advantage of reference counting over other forms of automatic heap management?

[2]

Reference counting is incremental: collection occurs in many small operations, and does not "stop-the-world" for long periods of time [2 marks]

(c) *Tracing garbage collection algorithms* are often used as alternative to reference counting for automatic heap management. Briefly outline the concepts behind the operation of tracing garbage collection.

[4]

A tracing garbage collector starts from a known *root set* of objects, and explicitly scans for the set of live objects that can be reached from that root set [2 marks]. All unreached objects are garbage, and can be disposed of in a separate garbage collection phrase [2 marks].

(d) A way of implementing tracing garbage collection is to use a *copying collector*. Outline what is a copying garbage collector, and explain how a copying collector implemented using *semispaces* works.

[9]

A copying collector works by tracing through all the live objects and copying them into one region of memory [2 marks]. This is done in a single pass [1 mark]. All the objects not copied to the new region are garbage, and can be reclaimed [2 marks].

The implementation using semispaces splits the heap into two halves, each being a contiguous region of memory [1 mark]. Allocations are made linearly from one half of the heap only [1 mark]. When that half of the heap is full, the collector is invoked and copies live objects into the other half of the heap, compacting as it goes [1 mark]. The first half of the heap then only contains garbage, which is overwritten eventually (when the other half is full and the collector runs again) [1 mark].

4. We discussed the MacOS X I/O Kit as an example of how operating system kernels can be improved through the use of modern object-oriented software engineering languages and practices, to improve the reliability of device drivers. The I/O kit allows the use of a limited subset of C++ in the MacOS X kernel, and organises device drivers into a hierarchy, pushing functionality that is common to several devices up to super-classes from which particular drivers are derived. Outline the advantages and disadvantages of this the I/O kit model compared to the C-based device drivers used in Linux. Discuss the trade-off in using object-oriented languages for kernel development in general – is the use of such languages a net win compared to writing kernels in C?

[20]

The advantages of the I/O kit model are ease of development and robustness. It's easy to build a device driver by creating a sub-class in an appropriate family of devices, and only implementing the minimum set of features needed to describe the unique features of the new device. Robustness comes from well-controlled integration with the rest of the operating system via framework code, implemented at the high-levels of the inheritance hierarchy, which is automatically inherited by specific drivers [5 marks].

The main disadvantages are performance and flexibility. The use of C++ has some (small) overhead compared to C, but more importantly there is less scope of optimise the drivers to their full extent, since they must fit within the generic framework (which is designed to be easy to use, rather than necessarily high performance). Lack of flexibility is an issue when implementing drivers for a new class of devices: it's harder to build something that doesn't fit into the existing frameworks, since there is more to implement to integrate the new family of drivers into the frameworks [5 marks].

With regards to the general question of whether object oriented languages offer benefit compared to writing in C, there is no single correct answer. The trade-off is between ease of programming, complexity of the runtime, and overheads. The C runtime is very simple and high performance, it also offers a great deal of control for the programmer; however it's also difficult to program. The object oriented languages tend to have a more complex runtime, which operates at a higher level of abstraction. This makes them easier to program, but less predictable (since the runtime hides a lot of detail). Up to [5 marks] are available for this discussion; [1 mark] for making an explicit statement on whether the candidate believes these languages are a net win; and [4 marks] for quality of written argument (e.g., logical progression of concepts; coherent argumentation – not English grammar).