University of Glasgow

DEGREES OF M.Eng., B.Eng., B.Sc., M.A. and M.A. (Social Sciences)

COMPUTING SCIENCE 4H:
REAL TIME AND EMBEDDED SYSTEMS
Sample Examination

(Answer 3 out of 4 questions.)

1. Assume that you have a system of periodic, independent, preemptable tasks to be scheduled on a single processor,

$\mathcal{T} = \{T_i\}$ for i = 1..n, where $T_i = (\varphi_i, p_i, e_i, \text{and } D_i)$ for each i.

(a) Assume $p_i = D_i$ for each i. What is the schedulability condition 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 schedulability condition 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 schedulability condition 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 schedulability 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 $\mathcal{T}$ 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 have been provided with the following system definition (all tasks are independent and preemptable, and to be scheduled on a single processor):

$\mathcal{T} = \{T_1 = (0, 2, 0.4, 2), T_2 = (1, 4, 1, 4), T_3 = (0, 5, 1.5, 5)\}$

(i) Is $\mathcal{T}$ schedulable using EDF? 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) Is $\mathcal{T}$ schedulable using RM? 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 have changed, such that T_3 becomes (0, 8, 4, 8).

(iii) Is $\mathcal{T}$ schedulable using EDF? 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) Is $\mathcal{T}$ schedulable using RM? 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, preemptable, 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. 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) Several environmental considerations lead 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 the schedulability of a system. 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 $2 * T_{CS}$.

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 $N * T_{CS}$.

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 $2 * N * T_{CS}$.

[5]

3. **(a)** Many real-time systems comprise tasks which have to be executed periodically. An operating system can support such tasks through the abstraction of a "periodic thread", or they can be implemented using a standard thread that loops with an appropriate period. Compare the two approaches, commenting on their relative advantages and disadvantages.

A periodic thread is simpler for the programmer since the operating system ensures the thread is executed at the appropriate times, cleanly abstracting the timing properties of the task. A looping thread that block between cycles is simpler for the operating system but requires the programmer to handle timing. A looping thread may be less reliable, since it requires the programmer to calculate the correct delay to ensure the next execution of the thread happens at the appropriate time: this gives a potential source of error in each program, rather than having a single – presumably well tested – implementation as part of the operating system.

[3]

(b) The POSIX 1003.1b API provides the ability to schedule processes at a range of priority levels, according to either FIFO or round-robin scheduling disciplines. Using a diagram of the scheduler's priority queues, explain how both real-time and non-real time tasks may be supported by a single operating system. Your diagram should show tasks in the ready, executing, sleeping and blocked states, and you should explain when transitions between states occur.

The diagram will look like that shown on slide 27 of lecture 12. Key points are separate ready queues for real time and non-real time tasks, but a single set of queues for tasks that are blocked.

When tasks are created, they are initially in the sleeping state. When a task becomes eligible for execution, it moves into the ready state and is inserted into either a real time or non-real time ready queue according to its requested scheduling discipline and priority.

The scheduler picks the highest priority task to run, moving it from the ready queue to the executing state. Real time tasks run in preference to non-real time tasks. Real time tasks using SCHED_FIFO and SCHED_RR share ready queues, and are scheduled together. Real time tasks scheduled using SCHED_RR have a time quantum enforced, which limits the amount of time they can spend in the executing state, before being de-scheduled and moving to the back of the appropriate ready queue. Real time tasks using SCHED_FIFO execute until they block or yield the processor.

Non-real time tasks execute using some form of background server. In the simplest case this is a slack stealing server, where non-real time tasks only execute if all real time tasks are idle. More complex systems may use a bandwidth preserving server to guarantee that non-real time tasks make progress.

[8]

(c) Explain how the presence of non-real time tasks might disrupt the operation of real-time tasks running on the same system.

A non-real time task may lock a resource desired by a real time task, causing the real time task to block, and leading to an uncontrolled priority inversion. Alternatively, a non-real time task may execute a system call or other non-preemptive operation, delaying the execution of a real time task until that operation completes.

[4]

(d) You are debugging a soft real time networked multimedia application running on Linux. The application uses the POSIX real time extensions to select high priority FIFO scheduling, so that it has priority over non-real time tasks in the system and can achieve smooth video playback. Unfortunately, the application is faulty and goes into an infinite loop when trying to decode certain video sequences, never blocking or yielding the processor. What would be the effects of this on other tasks running on the system? How would you debug the program?

A high-priority real time program that gets stuck in an infinite loop using SCHED_FIFO will starve all lower priority tasks of processor time. In Linux, non-real time tasks execute using a slack stealer, so they will never run in the scenario described, making it difficult to debug the program since the debugger will not get any processor time. The solution is to run a debugger with the highest real-time priority, to allow it to preempt the application, and attach to it for debugging. This will not disrupt the normal operation of the application, since the debugger is blocked waiting for user input when not in use.

[5]

4. **(a)** You have been given the task of implementing a streaming audio application, as part of a media server to be used by an Internet radio station. You can implement this application using TCP/IP or UDP/IP. Which would you choose, and why?

UDP/IP provides an unreliable datagram service, exposing the timing behaviour and loss inherent in the underlying IP network to the applications. TCP/IP layers both reliability, through retransmission, and rate adaptive congestion control onto the IP service. These features make an application reliable and allow it to adapt to changes in available network capacity. However, they also disrupt an application's timing. Accordingly, you would pick UDP/IP for the streaming audio application.

[4]

(b) When testing the streaming audio application, which sends packets with constant spacing, you notice that the inter-packet timing is disrupted on reception. This is causing problems for your application, which tries to playout the audio as soon as each packet is received. Assuming you cannot change the network, how would you adapt the application to make it robust to this timing variation?

You would add a buffer which delays processing of packets for some small time. The playout delay is chosen to absorb the timing variation, allowing packets to be processed at a constant rate, even when their arrival times are non-uniform.

[2]

(c) Explain why the inter-packet spacing at the receiver may have been disrupted, describing the effects of each type of problem. Is disruption of inter-packet timing a problem with the sender, the network, or both?

The problem could be with either the sender or the network.

The network will disrupt the packet timing as a result of variable queuing delays in intermediate routers, or if the route changes. Variable queuing delays will cause jitter, a route change may cause a step change in the end-to-end delay, and hence a sudden discontinuity in the timing of the received packets.

The sender could be the cause of problems if its clock rate is skewed relative to the receiver clock, or if there are delays in transmission of the packets. The latter is indistinguishable from queuing delays in the network. Clock skew manifests itself as a continual increase or decrease in relative inter-packet spacing.

[8]

(d) Flush with success from developing the streaming audio application, you take another job with a bank that wishes to stream real-time stock quotes to their customers. A key requirement of this new application is that all customers receive the quotes at approximately the same time, to prevent one customer getting an unfair advantage over others. Your employer is concerned that it may not be possible to build such a system using an IP network. Is he right to be concerned? Justify your answer.

Your employer may be right to be concerned, but it depends on the accuracy of the synchronization required. The network path from the bank to each customer will be slightly different, so each customer will see different amount of network jitter. This would mean that the data gets to each customer at a slightly different time. If the accuracy with which the quotes need to be presented is of the same scale as the network jitter, there may be a problem, otherwise your employer doesn't need to worry.

[6]