CM0605 - Embedded Systems Engineering

Liam Brand

Chapter 1

Real Time Scheduling

1.0.1 Basic Scheduling Analysis

(a)

The utilisation of a task set is given by

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i}$$

Where N is the number of tasks, C is the task's execution time and T is the task's period. The given task set's utilization is approximately 0.99.

(b)

A set of given tasks can be proved to be or not be schedulable using rate-monotonic priority assignment by using the utilisation-bound theorem. If the test is positive and the CPU utilisation is below the given bound, the set is schedulable.

$$U = U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

As previously discussed, the set's CPU utilisation is around 0.99. The given task set has 6 tasks, so the utilisation bound is around 0.73. This test is therefore false, meaning the task set is not schedulable using rate-monotonic priority assignment.

(c)

First the WCRT (worst case response time) for the task set must be calculated. This can be achieved via the given formula.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where R is response time, C is CPU utilisation, and hp(i) denotes the set of tasks of priority higher than i.

This calculation is repeated for each task until the calculate task response time converges twice. The following shows each iteration of this formula being applied to the task set.

Iteration 1
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 250
B(C=72, T=720, D=660): r = 342
F(C=18, T=4800, D=120): r = 360

Iteration 2
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 412
F(C=18, T=4800, D=120): r = 430

Iteration 3
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 558
F(C=18, T=4800, D=120): r = 576

Iteration 4
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 606
F(C=18, T=4800, D=120): r = 624

Iteration 5
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 620
F(C=18, T=4800, D=120): r = 638

Iteration 6
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 620
F(C=18, T=4800, D=120): r = 652

Iteration 7
E(C=14, T=48, D=48): r = 14
C(C=6, T=240, D=240): r = 20
A(C=42, T=330, D=60): r = 76
D(C=132, T=360, D=300): r = 270
B(C=72, T=720, D=660): r = 620
F(C=18, T=4800, D=120): r = 652

The final iteration shows that the task set is schedulable, as each of the task periods are less than or equal to the task response times. However, tasks A and F have a response time that exceeds their deadline, so these tasks will not meet their deadline.

(d)

The same calculations were performed as the ones in (c), but this time the tasks were organised by deadline (lowest to highest) instead of by their period. As the technique and method for calculation has already been explained and demonstrated, only the final iteration will be shown here.

Iteration 7
E(C=14, T=48, D=48): r = 14
A(C=42, T=330, D=60): r = 70
F(C=18, T=4800, D=120): r = 88
C(C=6, T=240, D=240): r = 94
D(C=132, T=360, D=300): r = 288
B(C=72, T=720, D=660): r = 652

The final iteration shows that each task's response time satisfies the condition of being less than or equal to the task period, so the set is schedulable

using deadline-monotonic fixed-priority assignment. Task A's response time is higher than its deadline however, so it will not meet its deadline.

(e)

Files are an example of a mutual exclusive resource, as only one task can be accessing a file at any given time to perform read or write operations on it. Another example is components like hardware timers, as if multiple tasks want to use the timer there will need to be controlled access to it.

Priority Inversion

An example which exhibits priority inversion can be seen with the Mars Pathfinder. The Pathfinder spacecraft has an information bus which was used to exchange information between the robot's various components. A high priority task ran frequently to manage the data on this bus, and access to the bus was synchronised with a mutex. The Pathfinder had another task for gathering meteorological data, which ran infrequently and had a low priority. This task would acquire the bus' mutex before writing data to the bus. If an interrupt caused the information bus to be scheduled whilst the meteorologic task held it, the bus management task would pend on the mutex and wait for the meteorologic task to be done. There was also a task with medium priority that dealt with communications.

The problem with priority inversion manifested like so. An interrupt would sometimes occur that scheduled the medium priority communications thread to be scheduled during the small moments where the high priority bus thread was blocked waiting on the meteorological data thread. When this happened, the communications task would prevent the meteorological task from running which would in turn prevent the communication task from running, meaning the medium priority task was essentially blocking the high priority task[2].

Priority Inheritance

Priority inheritance can help prevent this. With priority inheritance, tasks that block higher priority tasks will inherit the higher task's priority for the duration of the blocking. Once the task has finished executing and has released the resource it will drop down to its original priority asignment, and allow the high-priority task to use the resource that it has just acquired. For the Pathfinder example, this would mean the meteorologic thread would escalate to the bus management task's priority meaning the communication's task could not acquire the mutex and block the bus management task once the meteorological task had finished executing.

Priority Inheritance Limitations

Transitive blocking can occur. This is when tasks all begin blocking eachother and acquire high importance. For example, there are three tasks with high, medium and low priority, and two resources called R1 and R2. The low task takes R1 and the medium task takes R2. If the medium task wants to take R1, it is blocked and the low tas's priority is set to medium. If the high task then wants to take R2, it is blocked and sets the medium task to high priority. The

4

task holding R1 has an unchanged priority here even though it is the reason the original high priority task is being blocked.

Priority inheritance also does not prevent deadlocks, where all tasks aer stuck waiting for other tasks to release resources preventing them from releasing resources that other tasks need and so on.

As well, priority inheritance might prevent priority inversion but it doesn't minimize the time that tasks spend being blocked.

An Alternative

1.0.2   Scheduling with Shared Resources

Question a

Question b

Chapter 2

A Distributed Real-time System

Chapter 3

Reliability

3.1 Section 1

3.1.1 (a)

Fail-Operational

Fail-operational is when a system is still capable of full performance in the presence of faults, with no external signs of the fault manifesting. For the car production line, this would mean a system fault would still result in cars being produced at a good efficiency, meaning the system's objectives to produce cars won't be impeded and as such neither will car production. The encountered fault might cause an issue where damage is done to cars however, so whilst the production might not fail the actual quality of the cars produced would. As well, there would be no guarantee for the system to be entirely safe during its operation here, so there would be a risk to personnel safety.

Fail-Active

Fail-active systems can continue their operation when a fault is encountered, but will do so at a reduced performance. This is 'graceful degradation' where as more parts of the system experience faults the system can still continue its operation (up until a certain point). With regards to the production line, this will likely involve car production continuing at a reduced rate until the fault can be corrected. This won't always be the case however, as the fault may lie in a specific part of the car assembly. If this was the case, reduced system performance isn't tolerable as a key part of the manufacturing process won't be completed, so the cars being produced here won't be of any use.

Fail-Safe

Fail-safe systems will cease operation upon encountering a fault, and enter a safe mode. For example, elevators are often equipped with breaks that will apply

should the cable holding the elevator snap. For the car production line, this will mean the entire system ceases operation in the event of a fault. This will completely minimize risk to human life but production will suffer the most as each encountered fault will result in the system entering its safe mode. Restarting the system and bringing it back to full operational capacity could take a while, and this will result in a potential failure of the system's mission objectives.

High Availability

High availability systems cease operation upon encountering a fault, but must be returned to operation as quick as possible, this can involve hot swapping system units whilst the system is operating. These systems allow for failures, and aim to achieve a high mean time of operation rather than a long continuous time of operation. The goal here is not to avoid faults, but to minimize time spent rectifying them so that the overall system operation time is as high as possible[1]. The availability of these system can be measured by the following, where MTBF is mean time between failures and MTTR is mean time to repair.

$$Availability = MTBF/(MTBF + MTTR)$$

Out of the different types of fault-tolerance, high-availability would be the most appropriate for the car assembly line. Whilst this will result in a lower amount of continuous operation time, as long as the mean operation is as high as possible (which high-availability aims to achieve) production goals should still be satisfied and the production line's mission shouldn't be impeded. High-availability also allows for shutdowns when faults are encountered, which is an imperative feature of the assembly line's fault tolerance given the potential for harm to personnel.

3.1.2 (b)

Potential Risks

The nature of the car assembly line means any single failure will cause the entire system to fail, so in terms of the network only a single connected node in the distributed system needs to experience a problem for the entire system to stop working.

Timing will also be essential within the assembly line to ensure the robotic tools perform their jobs properly, so there is a risk of performance overhead negatively impacting the system too. Networked processors that are in close proximity will communicate incredibly quickly, but processors that are further away from the network will take longer to communicate and this will need to be accounted for otherwise tools may not behave as expected and, should any issues occur, the entire system will fail as previously discussed. In a similar vain, timing will need to be considered to ensure that all operations across the network are in synchronisation with each other. Messages arriving on time is a

good start, but this doesn't guarantee that all of the production line robotics are cooperating as expected.

There is also the possibility for the system's data to encounter issues, such as addressing errors where nodes send or receive messages that are intended for a different node. The data itself could also be corrupted during its transmission.

Architecture Recommendation

3.1.3 Question c

One hardware risk with the production line components is the contents of the ROM becoming corrupted by electronic pulses or radiation.

Bibliography

[1] Jim Gray and Daniel P. Siewiorek. High-availability computer systems. *Computer*, 24(9):39–48, 1991.

[2] Mike Jones. What really happened on mars rover pathfinder. *The Risks Digest*, 19(49):1–2, 1997.