IL2206 EMBEDDED SYSTEMS

Reflection Assignment 2

Badi Mirzai, badim@kth.se

September 25, 2020

1 Reflection Assignments for Seminar 2

1.1 What do you view as the largest benefits of the classic real-time theory? Are there some important limitations?

One of the benefits with classical real-time theory are that we can prioritize tasks and run them depending on how we set up their priorities. This makes sure that we can plan the task to be run on a concurrent (single core) program so that the CPU manages to execute the tasks within their deadlines by switching between the tasks. A challenge of a real time system is during a concurrent program, which is when a single core is switching between multiple tasks. The challenge is to plan which tasks to run at what time by prioritizing them so that no deadlines are violated while not having the processor's waiting time by waiting in between. This requires good planning and testing by the designer, ensuring that no system failure might occur or deadline violations but it will also make the system more efficient since the CPU can jump between multiple tasks. This is also more cost efficient than having the system run on multiple cores, which will ensure better performance but a higher price for the hardware.

For some systems it is also possible to use modularity, meaning that the system may be run separated and then recombined later after the tasks are finished, which will save time for the computations if the separated parts are run in parallel.

Limitations are of course the available hardware and the trade of between performance and cost, which will determine how the planning of the real time system should be done in the prioritization.

1.2 The real-time theory assumes that the period and execution times for each tasks are well-known. How can you determine period and execution time in practice? How accurately can you determine these parameters (period and execution time)?

One could determine the execution time by creating a loop around the task that needs to be executed. One measures how long it takes for the task to run in a loop of n times, and then divide the total execution time by the number of times looped through. This can be done for a set of values of n = {10, 100, 1000, 10 000}. This will give some sense of the average execution time but it might not be the perfect way to be secure since the worst case might be what you're looking for to prove in a safety critical system to secure no deadline violation for certain tasks. To get the worst case execution time one could look at the worst case time complexity of the algorithm and and assume that that will be the execution time, which will give a good margin for the execution time in a safety critical system. The same can be done when measuring the execution time in the loop mentioned above, where one just picks the worst case execution time among the set of values that were run. There would still be uncertainty of the execution time since running multiple time might be randomized depending on if the task is running on multiple cores and it also depends on what processor the task is tested at, which will probably need to be the same on as the one that will be run on in the SC system.

The period of the tasks can be determined by the the priorities of the tasks. A task that we want to have a higher priority will need to have a shorter period time, since we will want it to run more frequent to ensure that it is being executed enough times. A higher priority will need to have a shorter period, since the deadline will most likely also be shorter. The deadline can be set the same as the period, but can be a little larger to ensure that there is safety margins due to small random delays in the system.

With all of this said, it is however hard to determine the periods and execution time since one needs to optimize the scheduling based on the tasks and resources to minimize the chances of deadline violation which could lead to damage.

1.3 What is the main difference between semaphores, protected objects and rendezvous? Which of these communication mechanisms seems to be most suitable for a safety-critical real-time system?

A protected object consists of shared variables that can be accessed through its protected procedures/functions. The protected functions can only read the shared variables and the protected procedures can also change the values of them. The shared variables are thus private among these private functions/procedures and the protected object will block other tasks that is trying to access them from the outside. So it is only the well defined access functions of the protected object that can operate within the object, everything else will be blocked. This is good if the object doesn't need to be accessed by other parts of the system and provides a sense of privacy and cyber security. It does however cause some problems if there are other components that need to overwrite/read the values within the protected object. The protected object blocks the tasks that does not meet certain requirements, for example to avoid corruption of data by avoiding multiple tasks to spontaneously access the same data. This is done by the Boolean expressions inside the function's body of the protected object that specify the conditions that needs to be met.

Rendezvous mechanism on the other hand is based on communication via an active task, acting as a server and all of the other tasks can request services from the server tasks, thus acting as clients. This service-client relationship will, unlike the protected objects, have clients send requests to the tasks and instead of being blocked, they will wait for the client to be available by a availability request. This makes the relationship more abstract for the clients, since they are just waiting for access instead of trying to break into the object and being blocked (in protected object). Clients can also withdraw their request for access and they can be cancelled after a certain waiting time. A Server will provide a request that can be accessed by the client tasks and the clients waits until the server is available. During the process, the server receives the parameters from the client, executes the task and send back the results to the client. After finished, the server will provide availability for the next clients. The tasks in the Rendezvous can also communicate with each other directly by sending messages to each other, which is not the case in the protected object. The Client must know the identity of the accepting process and the entry. The server process on the other hand does not need to know the identity of the client and the identity of the rendezvous belong to the server (accepting process).

A semaphore is a communication object that consists of a set of processes it will handle and a non-negative integer it counts its availability for the tasks with. A semaphore is different from the protected object and the Rendezvous in the sense that it only makes sure that one task is running at the time and does not handle which task should be allowed to run. In other words, the semaphore only handles the distribution of tasks in the input/outputs and does not separate which tasks should be given access, which is different than the other two communication objects. Furthermore, a strong Semaphore can only handle the task in a first-in-first-out fashion, meaning that they only handle them in a queue fashion. This ensures that a process in the waiting for being accessed by the semaphore will eventually be so. Semaphores are different in the way that they do not support the concept of ownership, meaning that any process can execute the command signal(S) which means that they can be fed to the semaphore. There is in other words no requirement that the process have previously been successful in the wait(S) execution, which is the availability check for the semaphore. This can cause problems in a safety critical system that does not have the right implementation of the Wait(s) and signal(S) commands.

The semaphores should in my view not be considered for a SC-RT system, since they do not prioritize the tasks and they do not decide any distribution of which tasks should be run but rather in a First in First out fashion. It is from my understanding that both protected object and Rendezvous could be considered in a SC-RT system, depending on the objective of the system. The protected object is the most strict network when it comes to which functions/tasks can access the object. However in a real time system, there are deadlines that needs to be handled and therefor it is much more important that the system can choose tasks based on priority so that no deadlines are violated for the safety-critical parts. Both the protected object as well as the Rendezvous mechanism can be considered, but I would choose the latter since it provides more flexibility and lets the functions directly communicate with each other.

1.4 How can the real-time theory be used in an industrial design process for safety-critical real-time systems? Is the support from programming languages like Ada or real-time operating systems like MicroC/OS-II sufficient?

One could run the most critical parts of the safety-critical system in parallel as a back-up so that if there are some problems in the tasks running, the system could use the other running process as a backup in case of system failure.

One might need to analyse how the tasks meet their deadlines during several periods and if this will causes any delays or missing deadlines during multiple (hyper) periods. It is necessary to simulate these things in programming languages such as Ada, since they provide simulations for the real system beforehand which is necessary for the design process. One needs to be able to simulate the tasks in real time.

One could use Watchdog to handle deadlocks and/or delays for a safety-critical feature which will be handled by the overload detection. This feature means that the system might be able to predict errors based on missing deadline which can otherwise have catastrophic consequences.

Ada was basically developed by the US department of defence, and there is thus a lot of support for safety critical real time systems with the Ada language.