IL2206 EMBEDDED SYSTEMS

Reflection Assignment 2

Qiran Qian

3/10/2021

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?

The core of classic real-time theory is scheduling. Since the seminal work of Liu and Layland in 1973 and other research regarding the theory, we can decide which scheduling algorithm(e.g. Fixed Priority Scheduling or Dynamic Priority Scheduling, RM -based Scheduling or EDF-based Scheduling) to take to build a real-time system via well-rounded theory for feasibility analysis, task interaction, resource sharing and overload management. With the theory, the engineers can easily design real-time systems under certain standards and search for the optimal policy.

The term ”classic” indicates there are limitations. We used to design small and simple real-time systems or so-called federated systems which have isolated hardware resources and distribute characteristics. The existing real-time computing theory is just well enough to build such systems. However, engineers are trying to build more complex systems with integrated architecture, which means the components, e.g. sensors, data buses and processors are extensively shared. As for the highly integrated systems, it is hard for the classic scheduling analysis tools offer solutions efficiently as there are too many configurations of the system. We need a methodology to automatically help us search the design space and offer feasible solutions to each system configuration.

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)?

The period for each task is specified by the designer. We can simply set the period equal to the desired deadline. The execution time should be measured. When determining the execution time in practice, we often determine the Worst Case Execution Time(WCET), as it can be proved that if the task has feasible schedule solution with WCET, it can also be scheduled without overload with execution time that lower than WCET. That is to say we consider WCET as the upper bound of the execution time so we can guarantee the existence of solution. Then we consider the measurement of execution time. In practice, the task instances run on single/multi-processor, and the execution time actually depend on many factors e.g. frequency of the processor, advanced feature of the processor(caches, branch prediction, etc.), memory access time and so on. Thus, it's quite challenging to measure the execution time of a task instance accurately, though there are many research papers and tools regarding the problem.

The accuracy of WCET measurement depends on many factors. The predictability is of utmost consideration when we evaluate the accuracy. Maybe we can develop(or already exist) methods to evaluate how much the different features influence the measurement accuracy, and analyze in different levels to decide.

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?

Semaphore is a mechanism used to control access to a common resource by multiple processes and avoid critical section problems in a concurrent system such as a multitasking operating system. Semaphore can be used for process synchronisation or to ensure mutual exclusion by dispatching permissions to tasks to control the tasks suspend or continue so that synchronize the tasks.

Protected objects is a low overhead, data-oriented synchronization mechanism. From the client perspective, operating on a protected object is similar to operating on a task object. The operations on a protected object allow two or more tasks to synchronize their manipulations of shared data structures. Protected objects are specifically designed for synchronizing access from concurrent tasks.

Rendezvous is a mechanism that used for communication between tasks. We have a task as rendezvous server and several tasks as clients. The clients call the server and the server will decide whether to accept the requests.

In my view, to design a safety-critical real-time system, we would flexibly use the combinations of the above communication mechanisms. For instance, we shall use semaphores to ensure mutual exclusion when accessing to critical shared-resources. When implementing watchdog timers, we prefer rendezvous mechanism. When we try to model the system in a higher level or want to write OOP-style codes, we prefer protected objects. Actually, I think we can implement semaphores or rendezvous mechanism by protected objects. Protected objects is more flexible.

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?

With the real-time theory, we can develop useful and automatically tools for the design process in industry. We can also build a set of standards to standardize the design process of real-time system. Several methods already exist to aid the design of real-time systems e.g. MASCOT, HOOD, Real-Time UML, AADL, the Ravenscar profile, and Real-Time Java. And of course Ada is one of the most widely used tool to design real-time systems in industry, with its powerful real-time annex.

Base on the existing theory, we can design and model the real-time system in a higher abstraction level and search design space more efficiently. In addition, we can also easily emulate and validate our design in high level with the models.

In industrial design process, we program in Ada to support concurrency and real-time features of software, or we can also use RTOSs e.g. uC/OSII, to directly support real-time features which provided by the kernel of operating system. These build-in real-time features are powerful so to be utilized in avionic, defense and automotive. The real-time annex of Ada and the kernel of uC/OSII offer mechanisms like semaphore, mailbox and so on, meanwhile they provide support for scheduling algorithms, which make it much easier for engineers to design and program real-time systems.

Ada was originally designed to provide a single flexible yet portable language for real-time embedded systems to meet the needs of the US Department of Defence, but its domain of application has expanded to include many other areas, such as large-scale information systems, distributed systems, scientific computation, and systems programming. Furthermore, its user base has expanded to include all major defense agencies of the Western world, the whole of the aerospace community and increasingly many areas in civil and private sectors such as telecommunications, process control and monitoring systems.

However, these seems not sufficient because the complexity of the systems is increasing, which means we need more and more new features to design our systems in higher level. Further, to ensure the correctness regarding both functionality and timing, we need verification tools all through different levels. The engineers are still trying to provide new standards with new features(Ada83, Ada87, Ada95, etc.), from every iteration and revision, we obtain a larger annex which make the language redundant, while it’s still hard to meet all the requirements.