IL2206 Embedded Systems

Reflection Assignment 2

Weikai Zhou

October 3, 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?

I think the largest benefit may be that each task has its own priority and generally the priorities vary from task to task so that the concurrent feature can be enabled in the way that the CPU will switch between different tasks according to their priorities, And the tasks can be finished within their deadlines. Therefore, the concurrent feature is achieved with a single CPU.

There are some limitations. The first and the most important one is that how to decide the priority of each task. In class, we have talked about Rate-Monotonic Scheduling(RMS) Algorithm, where the priority of each task is decided by their period. The shorter the period is, the higher the priority is. Therefore, the priorities of tasks are determined when the periods of tasks are determined. And they will not change during the execution. However, this algorithm has the problem that it will not figure out a feasible schedule for some task sets. For example, from the lecture, we know that the utilization of task set $\tau_1(9, 3)$, $\tau_2(18, 5)$ and $\tau_3(12, 4)$ is $\frac{17}{18} < 100\%$. And this task set actually has a feasible schedule for all tasks to finish within a hyperperiod. However, the algorithm cannot find such schedule.

We also talked about another algorithm, Earliest Deadline First(EDF) Algorithm, where the priorities of tasks are given by their deadlines. The earlier the deadline is, the higher the priority the task has. Therefore, the priorities are dynamic. The same job may have different priorities at different time. However, it requires the additional knowledge of the deadlines of each tasks. And when implementing such an algorithm, we will definitely put more effort into it. Through these two case, we can find there is a trade off between the complexity of designing the priority and the degree to finishing within the deadlines.

Another limitation may be that the switch between different tasks according to their priorities may need extra computational time and decrease the efficiency of the system.

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)?

First, we can run the program for multiple times. For example, we can set a constant n which is large enough. In a for loop, we can decrease n every time until it equals zero and record the time needed for execution in each loop. After finishing this process, we can add up all the time and divide it by n to get the average time for execution.

To get the period, we should be aware that the period should be longer than the worst case execution time. Besides, the priority of task can also affect the period of it. For example, the higher the priority is, the shorter the period is. And generally, the period should be the longer than the relative deadline so that there will be some more time to buffer the system. Considering all these situations, we can find that it is really difficult to determine the period and the accuracy of the period is really low.

The average time for execution can be regarded as the execution time of the task. However, the answer we get is not accurate since the worst case execution time can also have a large gap between the average time. Therefore, the worst case execution time will influence the result greatly. Besides, we can also not simply use the worst case as the execution time since worst case is a rare case, meaning that the probability of the happening of worst case is really low and most cases can be executed shortly. Therefore, it is also difficult to determine the execution time and the accuracy of it is also low.

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?

Semaphores: A semaphore can be viewed as a record with two fields, which are a non-negative integer part and a set of process part respectively. It is used to control access of a shared resource and avoid critical section problems. Generally, only one task can access the resource at a time. The tasks will first wait for the permission from the semaphore. When the semaphore gives a signal to a certain task, it will be then allowed to access the shared resource. And the integer is used to help judge whether a task is waiting or has received a signal. The integer can also be used to limit the number of waiting tasks.

Protected objects: It is a powerful and flexible communication mechanism that is based on the concept of the monitor. It is specified with access protocols and has implementation details in the body. In the specification, there are generally some private variables and some function declarations. The private variables can only be accessed by these functions and other functions are blocked to access the private variables. The functions are then be implemented in the body part. Although the declared functions can access the private variables, they can also be blocked if they do not meet certain requirements. And the block is achieved by a boolean expression guard (when-statement).

Rendezvous: It is a communication mechanism that it servers as a server and the tasks are clients requesting services. The clients will send requests to the server and wait for the acceptance of the server. After the server finishes processing this client's request, the server will start to process the next client's request. Similarly, the rendezvous also has the when-statement as the guard. It also has a select-statement to switch between different tasks. It also generally obeys the FIFO fashion.

Therefore, the main difference is that for semaphores, all the tasks can have access to the shared resource but only one task can access it at a time under the control of the semaphores; for protected objects, only the declared functions which satisfy certain requirements at the same time can have access to the private variables; for rendezvous, tasks will wait for the acceptance of the rendezvous, and the rendezvous will process the requirements from the tasks directly and gives back the answer to the tasks.

From my point of view, rendezvous should be the one most suitable for a safety-critical real-time system. The function of the semaphores can be replaced by a rendezvous since the rendezvous also only process one task at a time. Therefore, there will not be critical section problems for the shared resources. And rendezvous provides other features that a semaphore cannot provide. For example, there are some guards (when-statement) in the rendezvous to block unsatisfied tasks. While for the protected objects, several functions can access the private variables at the same

time, which can cause some severe problems. Therefore, rendezvous may be the one most suitable for a safety-critical real-time system.

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?

People need to arrange the tasks with proper priorities, periods, relative deadlines and so on. Then the tasks should be finished within a hyperperiod. And it should be executed enough times to check its feasibility. People also should use scheduling algorithms like Earliest Deadline First(EDF) Algorithm to schedule the tasks. People can also design a overload detection and a watchdog to ensure the program is functioning properly and the tasks are finished within deadlines. When it detects some errors, it can reset the system or put the system in a sate state.

The support from programming languages like Ada or real-time operating systems like MicroC/OS-II is not sufficient. To achieve a safety-critical real-time system, both hardware and software should participate. For example, a safety-critical real-time system generally needs to get the information from the environment timely, then the sensors will play an important role. Besides, the hardware should also be as reliable as possible to handle the harsh environments. Therefore, the hardware is also an important part for a safety-critical real-time system.