

Ren Li Assignment

by Ren Li

FILE	12098_REN_LI_REN_LI_ASSIGNMENT_178149_283332113.DOCX (2.16M)		
TIME SUBMITTED	04-APR-2019 09:06PM (UTC+0100)	WORD COUNT	2232
SUBMISSION ID	104113285	CHARACTER COUNT	11890

Critical and Real-Time Application Programming Assignment

Ren Li 1805212

1. Knowledge and Understanding Skills

1.1 Explain the term "critical system"

Critical system is the system whose failure will cause severe outcomes, so it must be highly available and has corresponding treatment measures for potential exceptions [1][2].

1.2 Categorise the heart device software in one or more of the above types, providing appropriate arguments for your choice(s)

Before determine which type of critical system the heart device software is, we firstly discuss the basic meanings of these three types.

As for safety critical systems, they are systems whose working scenario is related to human life, personal health or natural environment. The failure of safety critical systems will cause significant loss like severe personal injury or environmental damage, so the safety issues must be considered critically when manufacturing hardware and software aspects of the system [3]. Mission critical systems are designed to make sure that some objects or goals of system can always be completed to guarantee performance of the system. For a business critical system, it is similar to a mission critical system (some object must be achieved), but it is more relevant to the commercial field and focuses more on profit and reputation of the company. If some operation of system is interrupted or failed, it will cause significant economic loss. Examples of the business critical system are bank accounting system, stock-trading system, etc [2].

Heart devices are used to help people with heart rhythm disorders to have a regular heartbeat. A heart device is placed under the skin and support the heart and circulation, which is closely related to human health, and the failure or interruption of device will cause serious personal injury and even danger to life. Therefore, it should belong to the safety critical system.

1.3 In your answer, consider the implications of building such critical systems on the design and development of this device with appropriate discussion

To design and develop a safety-critical system, there are many challenges and implications we need to consider carefully. One is how to avoid defective software specifications, which are implicated in many serious failures [4]. Safety-critical systems are usually very complex and composed of many subsystems. These subsystems are often developed and maintained by different teams or even companies. Therefore, it is important to make communication and collaboration between development teams smooth and transparent. The overall specification should be divided appropriately for each subsystem and any requirement dependencies should also be discussed fully by each development teams [5]. 

In addition, we also should think about how to ensure the heart device is safe to use. On the one hand, we need to sufficiently learn from antecedent experience to avoid particular

problem occurring again. However, there are also many general problems of incorrect assumptions [6] which may have not occurred in system before. Thus, on the other hand, we should conform to the International standards like ISO14971 [7] to get guidance about medical device risk management, and when developing engineers should communicate well to recognise and manage these problems. We also should design mitigations actions in system to deal with potential problems in prior, and the mitigation actions must be tested fully to make sure that they will not introduce new risks [6]. In regard to testing and verification of the system, because of the complicity of safety-critical application, we should establish a documented formal verification and model checking process. Test cases should be approved prior to testing, and the testing results should also be formally recorded and reviewed [5].



2. Cognitive and Intellectual Skills

Based on the timing constraints and scenario models designed in last assignment, we design a whole sequence diagram UML model of one wash and dry cycle of washing machine. This model remains the same timing constraints and can be regarded as a sequential composition of sub-models in last assignment.

The whole working process of one wash and dry cycle is:

1. Pump water into the drum.
2. Heater heats water to a specified temperature. In addition, if temperature of the water is too low or too high during washing process, the heater will adjust it to the reset value.
3. Motor starts to spin and wash the clothes, and if motor's speed is beyond the safety range, the motor controller will send a warn information to panel and lower it down timely.
4. When washing process finishes, the motor will stop and the pump will pump the water out.
5. Motor starts to spin again to dry the clothes.
6. The wash and dry cycle completes.

The sequence diagram of a wash and dry cycle is following:

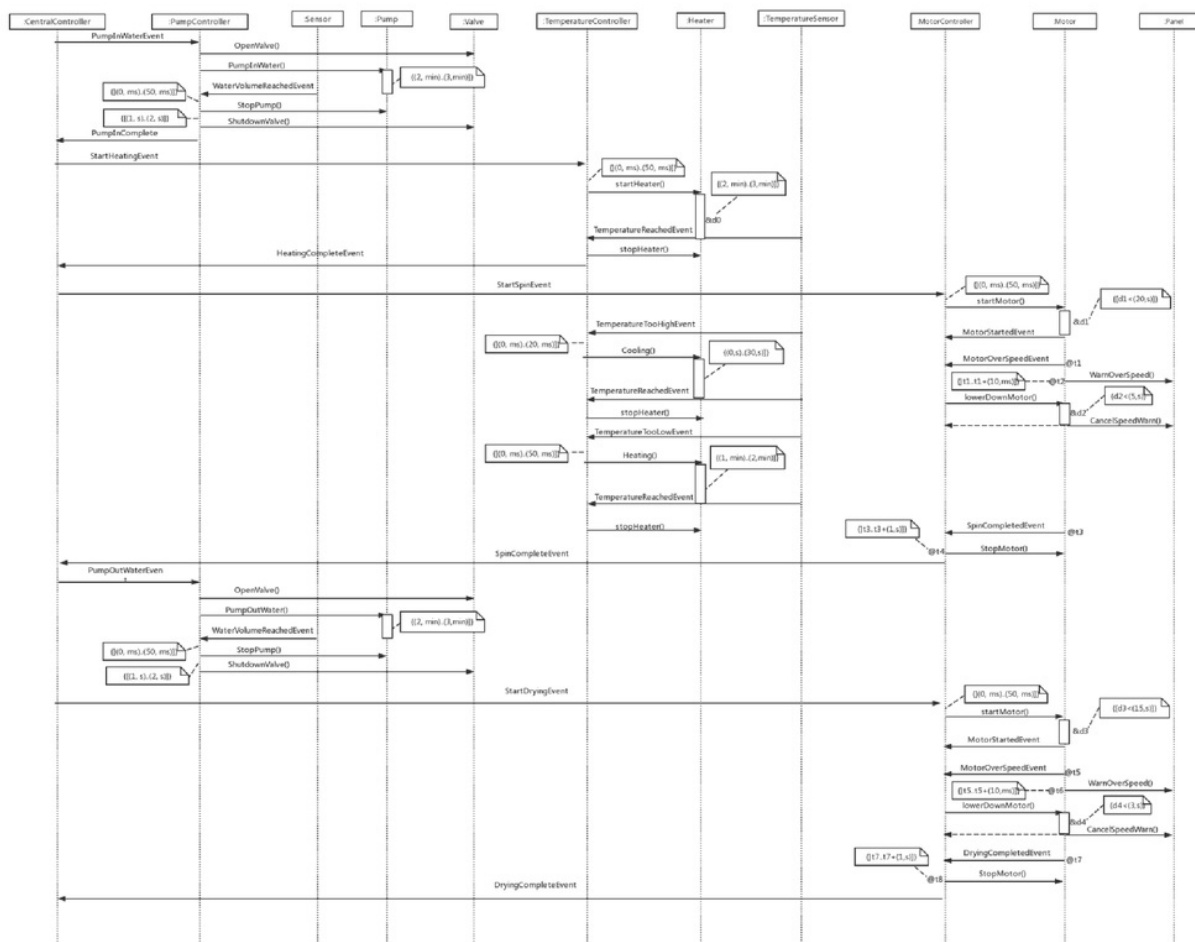


Figure 1 Sequence diagram of one wash and dry cycle

3. Practical Skills

Real-time systems are the systems judged not only by correctness of the output result but also the time spent [8]. There are three types of real-time systems which are background/foreground system, non-preemptive system and preemptive system. In this section, firstly we will discuss the basic characteristics of these three types and compare each system's advantages and disadvantages, then some other important concepts will be introduced including tasks, semaphores, interrupts and device drivers.

3.1 Three Types of RTOS

The background/foreground system is the simplest type of real-time systems. It is composed of two parts which are background and foreground. The background part is essentially an infinite loop where all the functions are called sequentially. The foreground part is composed of Interrupt service routines to mark the occurred asynchronous events which will be responded by corresponding functions in background when their turns come. That is to say,

in background/foreground systems, all the events are considered equally and have the same priority, so the operation with high real-time requirements cannot be dealt with until the background loop gets its turn. The response time of background/foreground system is also nondeterministic and will be whole loop execution time in the worst case. In addition, because of infinite loop structure of the background, if some part of the system gets breakdown, the whole system will crash and other missions will not be processed to end up appropriately. However, because this type of system is easy to implement and does not need much RAM or ROM resource, it is applied widely in the simple embedded applications.

The non-preemptive system is the system where higher priority tasks cannot preempt control of CPU from lower priority tasks which are in operation. The more urgent task must wait current job explicitly give up the control of CPU. In contrast, for preemptive systems, the responsiveness is the most important factor. An executing task can be interrupted by tasks with higher priority and the control of CPU will be preempted. Therefore, in preemptive systems, the critical operations can always be dealt with immediately, which makes task-level response optimum and deterministic [9].

3.2 Tasks

In RTOSs, a task is the smallest unit that can control CPU exclusively. The work is usually split into different parts and be processed in several tasks and each task will be assigned a priority. For a RTOS kernel, its main responsibility is to schedule the tasks to be dealt with according to their priorities. Note that the priority of a task is not always constant, RTOSs can implement a dynamic priorities mechanism to make priorities can be changed during execution, which is helpful to prevent the priority inversion problem.

There are five states for each task: dormant, ready, running, waiting and interrupted, which form the life loop of a task. A task is dormant when it is not available for kernel to assign computing resource. A ready task is ready to execute, but because its priority is lower than current task, it has to wait in the queue to be called. A running is the task that is executing currently. The waiting stage means that the task is waiting for some required resources to be available like I/O report, printer, etc. A task will be interrupted if an interrupt occurs when it is running and CPU goes to serve the interrupt.

3.3 Semaphores

Semaphores are variables mainly used to control access to a common resource in multitasking operating system. A semaphore records how many units of a particular resource are available. In general, a semaphore can be operated by three instructions which are INITIALIZE, WAIT and SIGNAL. A semaphore must be initialized a positive value by INITIALIZE instruction before using. When a task requires the semaphore, it will perform the WAIT operation and there will be two possible cases. One is that the semaphore value is greater than 0, which means it is available, so it will decrease by 1 and task will acquire the required resource. Another case is that the semaphore's value equals 0, so the task has to be suspended until the semaphore is released by current task. When the task possessing the resource finishes, it will perform a SIGNAL operation to release the semaphore and value of the semaphore will increase by 1. In $\mu\text{C}/\text{OS-II}$, the operations are implemented by `OSSemPend()` and `OSSemPost()`, the example code is showed below:

```

OS_EVENT *SharedDataSem;

void Function (void)
{
    INT8U err;

    OSSemPend(SharedDataSem, 0, &err);
    .
    .    /* You can access shared data in here (interrupts are recognized) */
    .
    OSSemPost(SharedDataSem);
}

```

Figure 2 Accessing shared data in μ C/OS-II using semaphores [9]

The usage of semaphores is helpful to solve race condition problem in multitask systems. Semaphore mechanism can specialize the execution order in some critical area and promises the definiteness of system. In addition, semaphores can also be used to signal the occurrence of some event to help communication between tasks.

3.4 Interrupts

An interrupt is a signal to inform the processor that an event has occurred and needs immediate attention. Interrupt mechanism liberates CPU from continuously looking at if some event has occurred and CPU can deal with events when they occur. When an interrupt is recognized, its current activity and state will be suspended and an interrupt service routine (ISR) will be called to deal with the event. After the routine finishing, the program will return. However, note that the returning place differs in different RTOS. In foreground/background systems, the program will return to background to continue the loop of background modules. In the non-preemptive system, the program will return to interrupted task. For preemptive kernels, after interrupting the program will execute the highest priority task ready to run to make sure that critical operations can be responded immediately [9].

3.5 Device Drivers

A device driver is a software program that controls a particular type of computer peripheral device. A driver is composed of many interfaces to hardware device, through which operating system or other computer programs can access hardware functions without knowing how to work directly with hardware [10]. For example, when a USB device being connected to the computer, if the specific device driver has been installed, there will be a device object created on the driver to represent the device and its physical structure. The methods of this object allow operating system to control its functions. When we choose an operation of the device, the device driver will issue corresponding commands to microcontroller within the device, so the device can know how to finish the task.

Following we will discuss how a serial device driver work to control a USB device. Because data bits flow parallelly inside the computer, serial device driver needs to convert the data from parallel to serial firstly, so the computer can communicate with the USB device. When USB port receives a number of bits into its buffer from USB device, the driver will send an interrupt to inform CPU to fetch them. This process is done frequently so from outside view

it seems like data is sent directly to CPU. When the CPU wants to send data out through USB port, it will send a command to USB driver at first and the driver will make serial port to prepare for the data. After preparation process finishing, the driver will send an interrupt to CPU, then CPU starts to transmit data to USB port. During transmitting, there are also many types of interrupts to inform CPU states of the port like buffer is full, buffer is clear, etc. Therefore, interrupts play an important role in USB driver to make sure data can be received and transmitted smoothly.

Reference

- [1] Hinchey, Mike, and Lorcan Coyle. "Evolving critical systems: A research agenda for computer-based systems." 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems. IEEE, 2010.
- [2] "Critical system." Wikipedia: The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 10 July 2018. Web. 1 April 2019, en.wikipedia.org/wiki/Critical_system
- [3] "Safety-critical system." Wikipedia: The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 12 December 2018. Web. 1 April 2019, en.wikipedia.org/wiki/Safety-critical_system
- [4] Neumann, Peter G. "Illustrative risks to the public in the use of computer systems and related technology." ACM SIGSOFT Software Engineering Notes 19.1 (1994): 16-29.
- [5] Bowen, Jonathan, and Victoria Stavridou. "Safety-critical systems, formal methods and standards." Software Engineering Journal 8.4 (1993): 189-209.
- [6] Knight, John C. "Safety critical systems: challenges and directions." Proceedings of the 24th international conference on software engineering. ACM, 2002.
- [7] Lincoln, John E. "Product risk management under ISO 14971: 2007." Journal of Validation Technology 15.4 (2009): 10-19.
- [8] Shin, Kang G., and Parameswaran Ramanathan. "Real-time computing: A new discipline of computer science and engineering." Proceedings of the IEEE 82.1 (1994): 6-24.
- [9] Labrosse, Jean J. MicroC/OS-II: The Real Time Kernel. CRC Press, 2002.
- [10] "Device driver." Wikipedia: The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 24 March 2019. Web. 4 April 2019, en.wikipedia.org/wiki/Device_driver

Ren Li Assignment

GRADEMARK REPORT

FINAL GRADE

70/100

GENERAL COMMENTS

Instructor

Critical System (10) 10
Cat heart device (15) 4
Implications on design (15) 9
UML timing diagrams (20) 15
Research RTOS (20) 14
Device driver Int/DMA (20) 18

You have mixed the answered to 1.1 and 1.2. As a result your analysis of the heart device is far too brief. UML diagram is a bit small but well drawn. Most of the required features are present. Nice general discussion of RTOSs, but I also wanted a comparison of two of them. However, I will give you extra marks because you have gone into some detail in this discussion. Discussion of device driver is very good (you missed a point at which DMA might have been used).

PAGE 1



Comment 1

But how does this improve "safety" within the design?

You need to provide more justification for this point.

PAGE 2



Comment 2

Good, you have highlighted testing.

PAGE 3

PAGE 4

PAGE 5



Comment 3

Good



Comment 4

OR this could be done by DMA