Defence Science Journal, Vol 43, No 2, April 1993, pp 139-144

Object-oriented Real-time Simulation Environment for Analysis of Real-time Software Architectures

K.R.S Iyyengar and R. Srinivasan

Naval Science and Technology Laboratory, Visakhapatnam-530 027

ABSTRACT

The problems associated with real-time software development are discussed and a design of real-time simulation environment (RTSE) to model real-time software architectures is presented. RTSE can be used to model software structure and can dynamically simulate the behaviour of multi-tasking, pre-emptive priority-based real-time software systems. RTSE can, be used to identify RT software anomalies like deadlock, starvation, lockout, signal queueing, race conditions, etc with the help of RTSE Report Analyzer. The modeller can fine tune his design by re-orienting the timing and system specification to remove anomalies and improve performance and reliability.

1. INTRODUCTION

Real-time software systems[1]respond to external asynchronous/synchronous events immediately. The response time depends on the RT O/S and the application architecture. Real-time systems normally perform high speed data acquisition and process information under severe time and reliability constraints. RT systems are normally used for a dedicated applications, viz., military, process control, industrial automation, medical and scientific research, computer graphics, communications, aerospace systems, computer aided-testing, and in industrial instrumentation.

Single task execution systems (like DOS) just load a program into its memory and then pass control to execute the program. A multi-tasking system[2] has multiple tasks to be executed simultaneously. It involves selection of a task from a set of tasks for execution. RT O/S has to manage communication and synchronization among the different processes. It has a kernel for accomplishing the above job. The designer has to ensure that tasks of applications are created, and inter-task communication and synchronization are planned properly to avoid any deadlock, starvation, lockout, etc.

A number of schemes and methodologies have been suggested for real-time software modelling. The most accepted and popular design methodology was proposed by Ward and Mellor[3]. Real-time software design involves of (a) identification of real-time tasks, (b) coordination between real-time tasks, (c) processing of system interrupts, (d) I/O handling to ensure no data loss, (e) specifying internal and external constraints of the system, and (f) ensuring accuracy of data.

The software designer identifies the real-time tasks, and ensures coordination between tasks, external device servicing, etc., to cater the problem requirements. The software architecture thus evolved is made to work in a required scheduling policy, which can be a round-robin, pre-emptive priority or non-pre-emptive. Task synchronization and interrupt handling are time critical and often RT software structure optimization is bogged down with problems of mixed signal problems, deadlock, starvation, race conditions, etc.

The requirements of RT system modelling and analysis are discussed below and an Environment to model the system based on Ward and Mellor[3] methodology is presented. After modelling the specifications, the dynamic simulator visually shows the

Received 26 October 1992

behaviour of the system. Then the software system can be fine tuned to show the optimal performance.

2. RT SYSTEMS DESIGN REQUIREMENTS

To accomplish the job to respond to the external inputs, the software design[4] should have the characteristics: (a) task detinition and resource handling, and (b) task communication and synchronization. Typical RTOS constructs are presented here with examples from popular pSOS[5] and iRMX[6] RTOS kernels.

A real-time software consists of a set of tasks — separate processes/tasks that compete for access to the CPU and a master process that schedules CPU access. Each task can have a priority attached to it, depending on the importance of the task or the timing of the task. Similar tasks can also be grouped together to have more protection.

Usage	pSOS	iRMX
Create task	spawn-p	CREATE-TASK
Activate task	activate-p	
Change priority dynamically	priority-p	SET-PRIORITY
Make pre-emptive task	mode-p	

A task can be suspended for some time, or it can be suspended and resumed later. If the task is suspended, it does not get a CPU share until it is resumed. Suspension and resumption of process can be done among the same group. This provides some level of protection.

Usage	pSOS	iRMX
Suspend process for some time	pause-p	SLEEP
Suspend process until resumed	suspend-p	SUSPEND-TASK
Resume process	resume-p	RESUME-TASK

Inter-task synchronization is achieved by means of shared variables, signals, waits and mailbox messages.

Usage	pSOS	iRMX
Signal a process about occurrence of some event	signal-v	SEND-SIGNAL
Wait for some event	wait-v	RECEIVE-SIGNAL
Create message	create-v	CREATE-MESSAGE
Send message	send-x	SEND-MESSAGE
Request for message	req-x	RECEIVE-MESSAGE

The RT O/S kernel has got some memory management routines and semaphores for task synchronization.

With the above basic constructs a complete RT software system can be designed.

3. DESIGN OF RT SOFTWARE SYSTEM

Let us consider a problem in which the following activities are to be met: (a) data will be received from a sensor at constant intervals, (b) system has to process the data, (c) monitor keyboard input, and (d) display the earlier processed data as per the key board input.

The designer has to know the quantum and frequency of the data the system is going to receive and the timing of receiving data. The data flow diagram according to Ward-Mellor approach for the above real-time problem is given in Fig. 1.

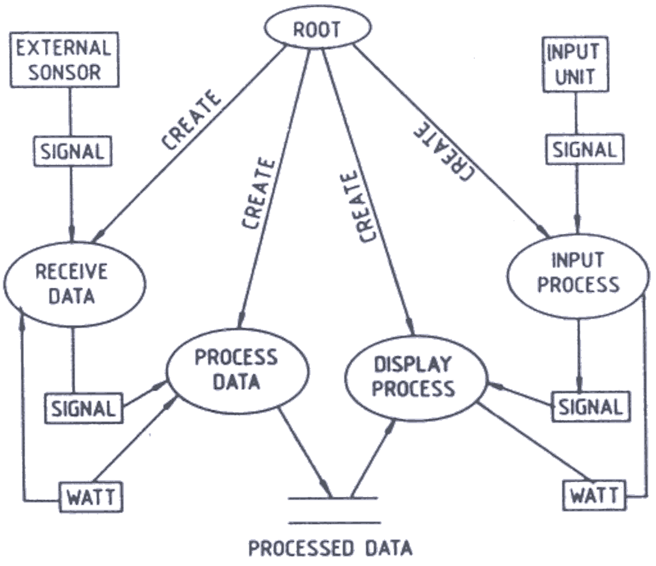


Figure 1. Data flow diagram.

The tasks and the thread of events are conceived by the designer. The data receive task is given a higher priority, as the task should not miss any data. As soon as the receive task receives data, it gives a signal to process data task.

The display task has a lower priority than other tasks. It waits for signal from input unit. The code corresponding to it in pSOS will be as shown in Appendix A.

The anomalies can be manually analyzed for the entire activity. The first look at the system seems to be perfectly correct. Consider the case in which the external data is received by the Input-Process continuously. In such cases, the signal queue increases considerably, finally leading to missing of signals or system hangup.

If the designer can foresee the problem, he can tune his design by allowing external input to proceed only if the display process is waiting for external input, or the external input buffer exceeds two inputs in the queue.

Analysis and testing of the real-time software system are very difficult if we follow manual debugging method. This is particularly true in the case of large projects where it has to cater for more inputs, and a number of tasks and interrupts are working on the system asynchronously. An analysis tool will help the designer to tune his model while designing a real-time software system.

4. REAL-TIME SIMULATION ENVIRONMENT

The real-time simulation environment (RTSE) developed by the Computer Centre, NSTL, resolves the problems of real-time software modelling[7,8] and verifies the design by dynamically simulating the design; it also reports the anomalies. The RTSE has four components (a) modelling real-time system and GUI, (b) dynamic simulation, (c) report analyser, and (d) Fine tuning the system.

4.1. Modelling Real-time System and GUI

The environment provides an interactive user-friendly interface for defining the specification of the RT system. The specifications are entered into the system through the graphical user interface (GUI) in an interactive manner. The GUI environment provides selection of task, signal, wait, create-task, suspend-task, resume-task and other constructs needed for building a real-time software system.

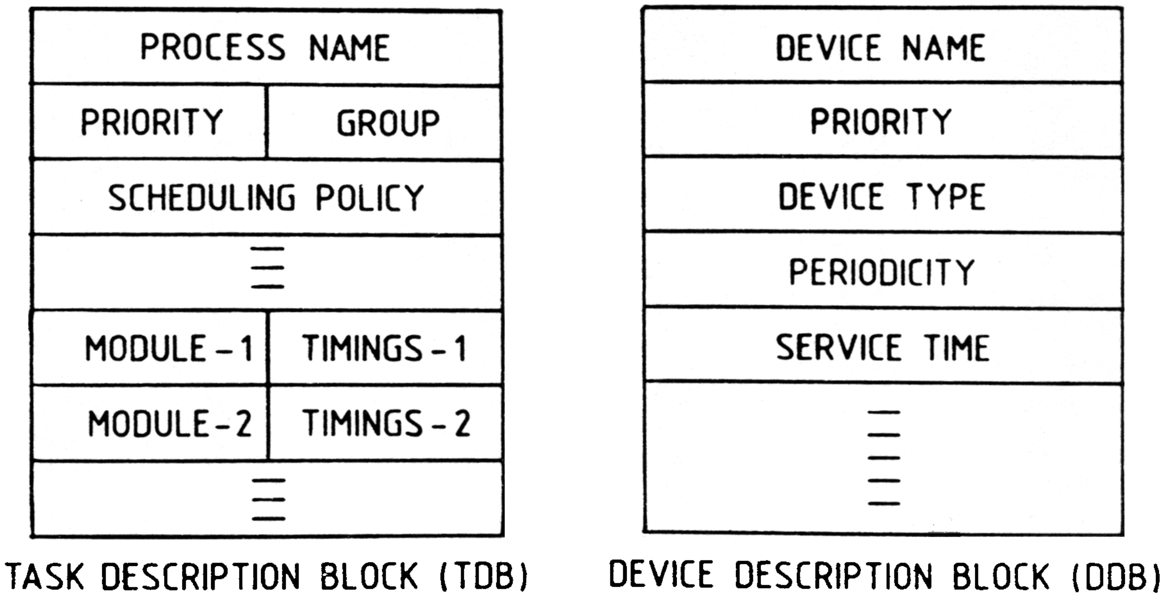
Tasks are represented by circles. Creation of the task and the relationship among tasks are represented by thick lines. Rectangles represent different entities like signal, wait, suspend, resume, pause, etc. The specifications of the tasks and external devices are described by using task description windows and device description windows.

4.1.1 The Task Description Window

The circles represent different tasks of a problem. The task characteristics are described in a task description window. An interactive dialog window appears in which the designer has to enter the specifications of the particular task in the form of (a) timing (approximate execution time), (b) interrupt or normal task, (c) priority of the task, (d) group to which the task belongs, (e) sequence of sub-tasks and their approximate execution time, and (f) scheduling policy.

The model creates a task description block (TDB) structure associated with each task described (Fig. 2).

The priority of the task can be selected between 0 and 255, 255 being the highest priority. Group specifies the Task group to which the current task belongs. Only



TASK DESCRIPTION BLOCK (TDB)

DEVICE DESCRIPTION BLOCK (DDB)

Figure 2. Task description and device description blocks.

the tasks that are in the same group can be deleted, suspended or resumed by the task.

The scheduling policy also can be specified. The policy can be priority-based scheduling, non-pre-emptive multi-tasking or round-robin scheduling. If the priority of all processes is same, then the task switching takes place in a non pre-emptive first-in first-out scheduling.

4.1.2 The External Device Description Window

The real-time system always interacts with some external units. The external device is described in the external device describtion window. An interactive dialogue window appears in which the designer can enter the specification of the external devices in the form of (a) input or output device, (b) asynchronous or synchronous device, (c) interval of occurrence, (d) service time, (e) priority/no priority, and (f) coupled with interrupt or normal.

The external units can provide input to the system or to output data to another system. The asynchronous events happen at random time intervals and the synchronous events happen at fixed time intervals. The interval of occurrence of an event can be set to random, or the time interval can be specified. Service time specifies the amount of time the device takes to perform input/output. Priority specifies the priority associated with the device. The specification includes whether the interaction happens through interrupt mode or through normal mode. The description of devices is loaded into a device description block (DDB).

The first level of the task diagram can be drawn as described above, and if the designer is interested in the next level, he can further 'explode' the task into sub-tasks, which can inherit the parent task characteristics.

4.1.3 The Build

Once the design is complete, the system provides a build check on the completeness of the design. Any clash in the signals and waits of tasks can be detected. A check is made on the signals and the corresponding waits of the tasks. Any task which is trying to violate group rights can be checked. Any unconnected tasks or devices are intimated to the modeller by build. Any static anomalies detected are reported. The designer can correct his design at this stage itself.

4.2. Dynamic System Simulation (DSS)

Once the preliminary design is syntactically and semantically (to a little extent) verified, the designer can view the performance of the system.

The DSS is a scheduling manager and checks the validity of the design. The DSS does (a) CPU scheduling, (b) asynchronous/synchronous communication with external devices, (c) manages inter-task communication and synchronization, and (d) handles interrupts.

The tasks created can be in any of the three states, namely, ready state, blocked state or running state. The ready state indicates that the task can be allocated to CPU; and blocked state indicates that the task is expecting some other task or external device to signal. Running state indicates that the task is currently allocated to CPU.

DSS maintains a ready queue, blocked queue and running queue, The queue containing the entities is shown in Fig. 3.

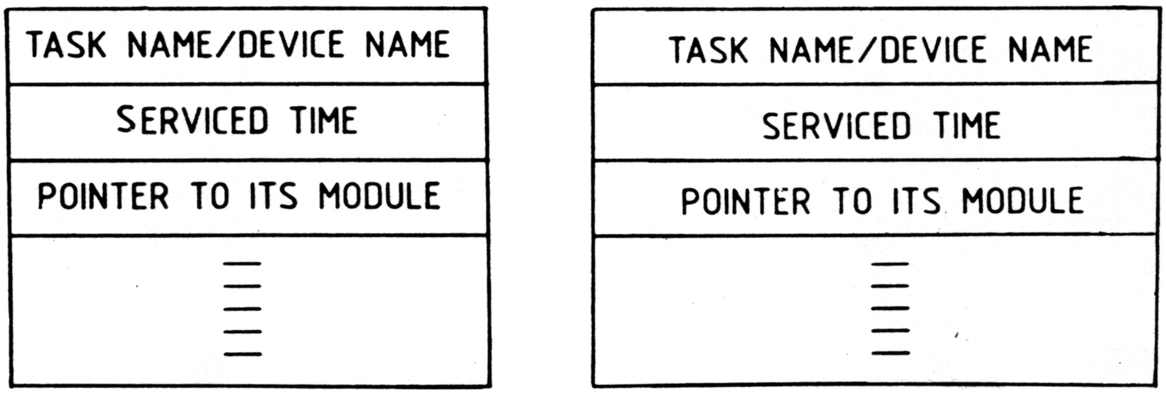


Figure 3. Typical queue structure.

DSS initializes all the queues and the root task (main task) is selected and put in the ready queue, with initial execution time set to zero, and the pointer points to the first module of the task description Table. As and when a task is initialized, it is added to the ready queue.

DSS selects the appropriate task from the ready queue, depending on the scheduling policy opted for. DSS allocates CPU for a time unit (execution of an instruction) for the task and then checks the status and updates the status of different tasks and schedules CPU again. Periodic/random inputs are simulated by DSS and fed to the RT software. The complete control flow sequence is recorded by the DSS.

4.3. Report Analyzer

The report analyzer goes through the execution sequence of the tasks. The task execution sequence is graphically shown on the display. Conditions like deadlock, starvation, multiple events queueing are detected and reported by the report analyzer.

4.3.1 Deadlock Detection

Deadlock occurs due to a task expecting some signal/resource that is not released by another task which may be waiting for the earlier task to signal/release resource. On analyzing the blocked queue at regular intervals, the deadlock situation is reported. Report analyzer displays in a window as to why the deadlock has occurred and the tasks involved when the deadlock occurred in real-time.

4.3.2. Starvation

Starvation is defined as a task not getting CPU share, because some task of higher priority job always takes CPU. The indefinite wait for CPU is called starvation. The report analyzer checks CPU allocation, the task description blocks and the ready queue at regular intervals and reports the tasks that are starving for CPU.

4.3.3. Race Conditions

The report of execution sequence provides important information about the order of execution of simultaneous tasks. The over-run of time by some process may lead to clash with the continued execution of the system, resulting in hangup in several iterations.

4.3.4. Multiple Signal Queueing

Even if the process is given a signal, due to some other reasons if the process is not able to service it before the arrival of the next signal, it leads to a signal queueing which might lead to signal or data loss. The report analyzer checks signal queueing at regular intervals and reports such conditions.

4.4. Fine Tuning of RT Software

If anomalies are found in the design, the design can be re-tuned by changing the specifications of the required task or module. Fine refinement is possible by adjusting the specifications on getting the feedback of system performance by repeatedly running DSS.

5. CONCLUSION

The RTSE has been implemented in C++ under MS-WINDOWS. The choice of C++ made the design simpler. The usage of virtual contructs in C++ has reduced the burden of programming repeatedly. MS-WINDOWS provided an excellent GUI and constructs for building the modelling part of RTSE. The environment simplifies the job of the software designer and eliminates the necessity of manual RT software probing.

ACKNOWLEDGEMENTS

The authors thank Rear Admiral RS Chaudhry, AVSM, VSM, IN, Director, NSTL, Visakhapatnam for the support provided for the work and Dr DK Chattopadhyaya, Deputy Director for giving constructive guidance at every stage of development of this work.

REFERENCES

Pressman. Software engineering, Ed. 2. McGraw-Hill, New York, 1988. pp. 368-99.

2 Milkinov, Milan. Operating systems—concepts and design. McGraw-Hill, New York, 1987. pp. 70-168.

3 Ward, P.T. & Mellor, S.J. Structured development for real-time systems, 3 V. Prentice-Hall, Englewood Cliffs, 1987. 468 p.

4 Ripps, David L. An implementation guide to real-time programming. Yourdon Press, Reading, 1989.

5. pSOS 68 K user manual. Software Components Group Inc., California, 1988.

6. iRMX user manual. Intel Corporation, California, 1985.

7 Lewis, Ted. & Reisman, Sorel. Tools fair. *IEEE Software*, 1990, 67-76.

8. Falk, Howard. CASE tools emerge to handle real-time systems. *Computer Design*, 1988, **27**, 53-57.

APPENDIX A

pSOS PSEUDO-CODE FOR THE SAMPLE PROBLEM

```
root()

/* Receive-Data is assigned the highest priority */
spawn-p(Receive - Data,Priority 255,Group 0);
activate(Receive Data);
spawn-p(Process - Data,Priority 254, Group 0);
activate(Process - Data);
spawn-p(Display - Process, Priority 254, Group 1);
activatep(Display - Process);
spawn-p(Input - process, Priority 254, Group 1);
activate-p(Input - Process);

Receive-Data() /* Receives data every 3 s */

Wait-for signal from external device.
Receive Data. /* computation 0.5 s */
Signal to Process Data.

Process-Data()

Wait-for signal from ReceiveData.
Process data. /* computation 1.0 sec */

Input-process () /* Random Data Input */

Wait for signal from external device.
Collect Data. /* computation 0.25 s */
Send signal to Display process.

Display-process()

Wait-for signal from Input Process.
Display Requested /* computation 2.0 s */
```