Chapter 1

Real-Time Programs

This chapter surveys the categories of real-time programs, and indicates which types can best be supported by REACT and REACT/Pro. As an experienced programmer of real-time applications, you might want to read the chapter to verify that this book uses terminology that you know; or you might want to proceed directly to Chapter 2, "Basic Features of the CHALLENGE and IRIX™ Architectures".

Defining Real-Time Programs

A real-time program is any program that must maintain a fixed, absolute timing relationship with an external hardware device.

Normal-time programs do not require a fixed timing relationship to external devices. A normal-time program is a correct program when it produces the correct output, no matter how long that takes. You can specify performance goals for a normal-time program, such as "respond in at most 2 seconds to 90% of all transactions," but if the program does not meet the goals, it is merely slow, not incorrect.

A real-time program is one that is incorrect and unusable if it fails to meet its performance requirements, and so falls out of step with the external device.

Major Types of Real-Time Programs

There are three major types of real-time programs: simulators, data collection systems, and process control systems. This section describes each type briefly. Simulators and data collection systems are described in more detail in following sections.

- A simulator maintains an internal model of the world. It receives control inputs, updates the model to reflect them, and displays the changed model. It must process inputs in real time in order to maintain an accurate simulation, and it must generate output in real time to keep up with the display hardware.

Silicon Graphics systems are well suited to programming many kinds of simulators.

- A data collection system receives input from reporting devices, for example telemetry receivers, and stores the data. It may be required to process, reduce, analyze or compress the data before storing it. It must react in real time in order to avoid losing data.

  Silicon Graphics systems are suited to many data collection tasks.

- A *process control* system monitors the state of an industrial process and constantly adjusts it for efficient, safe operation. It must react in real time to avoid waste, damage, or hazardous operating conditions.

  Although Silicon Graphics systems can be used for process control, dedicated process-control computers are sometimes a more economical choice for these uses.

Simulators

All simulators have the same four components,

- An internal model of the world or part of it; for example a model of a vehicle travelling through a model geography, or a model of the physical state of a nuclear power plant.
- External devices to display the state of the model; for example, one or more video displays, audio speakers, or a simulated instrument panel.
- External devices to supply control inputs; for example a steering wheel, a joystick, or simulated knobs and dials.
- An operator (or hardware under test) that "closes the loop" by moving the controls in response to what is shown on the display.

Requirements on Simulators

The real-time requirements on a simulator vary depending on the nature of these four components. Two key performance requirements on a simulator are *frame rate* and *transport delay*.

Frame Rate

A crucial measure of simulator performance is the rate at which it updates the display. This rate is called the *frame rate*, whether or not the simulator displays its model on a video screen.

Frame rate is given in cycles per second (abbreviated Hz). Typical frame rates run from 15 Hz to 60 Hz, although rates higher and lower than these are used in special situations.

The inverse of frame rate is *frame interval*. For example, a frame rate of 60 Hz implies a frame interval of 1/60 second, or 16.67 milliseconds. To maintain a frame rate of 60 Hz, a simulator must update its model and prepare a new display in less than 16.67 ms.

The REACT/Pro Frame Scheduler helps you organize a multi-process application to achieve a specified frame rate. (See Chapter 7, "Using the Frame Scheduler.")

Transport Delay

Transport delay is the term for the number of frames that elapses before a control motion is reflected in the display. When the transport delay is too long, the operator will perceive the simulation as sluggish or unrealistic. If a visual display lags behind control inputs, a human operator can become physically ill.

Aircraft Simulators

Simulators for real or hypothetical aircraft or spacecraft typically require frame rates of 30 Hz to 120 Hz and transport delays of 1 or 2 frames. There will be several analogue control inputs or and possibly many digital control inputs (simulated switches and circuit breakers, for example). There are often multiple video display outputs (one each for the left, forward and right "windows"), and possibly special hardware to shake or tilt the "cockpit." The display in the "windows" must have a convincing level of detail.

Silicon Graphics systems with REACT/Pro are well suited to building aircraft simulators.

Ground Vehicle Simulators

Simulators for automobiles, tanks, and heavy equipment have been built with Silicon Graphics systems. Frame rates and transport delays are similar to those for aircraft simulators. However, there is a smaller world of simulated "geography" to maintain in the model. Also, the viewpoint of the display changes more slowly, and through smaller angles, than the viewpoint from an aircraft simulator. These factors can make it somewhat simpler for a ground vehicle simulator to update its display.

Plant Control Simulators

A simulator can be used to train the operators of an industrial plant such as a nuclear or conventional power generation plant. Power-plant simulators have been built using Silicon Graphics systems.

The frame rate of a plant control simulator can be as low as 1 or 2 Hz. However, the number of control inputs (knobs, dials, valves, and so on) can be very large. Special hardware may be required to attach the control inputs and multiplex them onto the VME bus. Also, the number of display outputs (simulated gauges, charts, warning lights, and so on) can be very large and may also require custom hardware to interface them to the computer.

Virtual Reality Simulators

A virtual reality simulator aims to give its operator a sense of presence in a computer-generated world. (So also does a vehicle simulator. One difference is that a vehicle simulator strives for an exact model of the laws of physics, which a virtual reality simulator typically does not need to do.)

Usually the operator can see only the simulated display, and has no other visual referents. Because of this, the frame rate must be high enough to give smooth, nonflickering animation, and any perceptible transport delay can cause nausea and disorientation. However, the virtual world is not required (or expected) to look like the real world, so the simulator may be able to do less work to prepare the display.

Silicon Graphics systems, with their excellent graphic and audio capabilities, are well suited to building virtual reality applications.

Hardware-in-the-loop (HITL) Simulators

The operator of a simulator need not be a person. In a hardware-in-the-loop simulator, the role of operator is played by another computer, such as an aircraft autopilot or the control and guidance computer of a missile. The inputs to the computer under test are the simulator's display output. The output signals of the computer under test are the simulator's control inputs.

Depending on the hardware being exercised, the simulator may have to maintain a very high frame rate, up to 1000 Hz. Silicon Graphics systems can be used for some hardware simulators. Special-purpose systems may be more practical or more economical for very demanding frame rates.

Data Collection Systems

A data collection system has the following major parts:

1. Sources of data, for example telemetry. Typically the source or sources are interfaced to the VME bus.
2. A repository for the data. This can be a raw device such as a tape, or it can be a disk file or even a database system.
3. Rules for processing. The data collection system might be asked only to buffer the data and copy it to disk. Or it might be expected to compress the data, smooth it, sample it, or filter it for noise.
4. Optionally, a display. The data collection system may be required to display the status of the system or to display a summary or sample of the data. The display is typically not required to maintain a particular frame rate, however.

Requirements on Data Collection Systems

The first requirement on a data collection system is imposed by the *peak data rate* of the combined data sources. The system must be able to receive data at this peak rate without an *overrun*; that is, without losing data because it could not read the data as fast as it arrived.

The second requirement is that the system must be able to process and write the data to the repository at the *average data rate* of the combined sources. Writing can proceed at the average rate as long as there is enough memory to buffer short bursts at the peak rate.

You might specify a desired frame rate for updating the display of the data. However, there is usually no real-time requirement on display rate for a data collection system. That is, the system is correct as long as it receives and stores all data, even if the display is updated slowly.

Achieving High Transfer Rates to Devices

The Challenge/Onyx systems support a variety of I/O types with different bandwidth and latency characteristics:

- VME device registers can be mapped directly into the program's address space, where they can be read and written as memory variables. This is implemented as *programmed I/O (PIO)*.

  Memory-mapping makes I/O programming simple, especially when large numbers of devices or complex device protocols are involved. Memory-mapped, programmed I/O can transfer data from 250 KB/second to 1 MB/second. (See "PIO Access" on page 168.)

- When transferring 32 or more consecutive bytes, transfer rate can be increased using *direct memory access (DMA)* to devices on the VME bus. The Challenge/Onyx architecture allows a user-level process DMA access to VME master and slave devices through a unique DMA engine (see "Program Access to the VME Bus" on page 168.)

- Maximum transfer rates on the VME bus are achieved with a VME device that supports block mode transfer as a bus master. Challenge/Onyx systems can achieve VME transfer rates greater than 50 MB/second to such devices.

- Multiple SCSI controllers can be attached to all Silicon Graphics systems. SCSI transfer rates can reach 14 MB/second on each channel for 16-bit SCSI-II controllers (see "SCSI Hardware on CHALLENGE and Onyx Systems" on page 161).

Achieving High Transfer Rates to Disk

A data collection system can exploit two features to achieve a high rate of data transfer to disk,

- asynchronous disk I/O
- Guaranteed-rate I/O (GRIO), a feature of the XFS filesystem

Asynchronous I/O that conforms to POSIX 1003.1b-1993 is a standard feature of IRIX. You use asynchronous I/O library calls to initiate disk I/O in a separate process, while your real-time process continues to work with the input data. (In fact you can start asynchronous I/O to any device, not only to disk files.) You can ensure that the asynchronous process performing the I/O executes on a different CPU than the one used by the real-time process.

Using GRIO, your real-time program can claim a specified portion of the bandwidth of a device. I/O requested by other processes is deferred, if necessary, to ensure that your process achieves the promised data rate.

For details of both these features, see Chapter 8, “Optimizing Disk I/O for a Real-Time Program.”

Real-Time Programming Languages

The majority of real-time programs are written in C, which is the most common language for system programming on UNIX. All of the examples in this book are in C syntax.

The second most common real-time language is Ada, which is used for many defense-related projects. SGI sells Ada 95, a new implementation of the language. Ada 95 programs can call any function that is available to a C program, so all the facilities described in this book are available, although the calling syntax may vary slightly. Ada offers additional features that are useful in real-time programming; for example, it includes a partial implementation of POSIX threads which is used to implement Ada tasking.

Some real-time programs are written in FORTRAN. A program in FORTRAN can access any IRIX system function, that is, any facility that is specified in volume 2 of the reference pages. For example, all the facilities of the REACT/Pro Frame Scheduler are accessible

through the IRIX system function **schedctl()**, and hence can be accessed from a FORTRAN program (see "The Frame Scheduler API" on page 98).

A FORTRAN program cannot directly call C library functions, so any facility that is documented in volume 3 of the reference pages is not directly available in FORTRAN. Thus the **mmap()** function, a system function, is available, but the **usinit()** library function, which is basic to SGI semaphores and locks, is not available. However, it is possible to link subroutines in C to FORTRAN programs, so you can write interface subroutines to encapsulate C library functions and make them available to a FORTRAN program.