CHAPTER 19

Real-Time Systems

Chapter 19 introduces real-time computing systems. The requirements of real-time systems differ from those of many of the systems we have described in the previous chapters, largely because real-time systems must produce results within certain deadlines. In this chapter we provide an overview of real-time computer systems and describe how real-time operating systems must be constructed to meet the stringent timing requirements of these systems.

19.1 Interrupt latency is the period of time required to perform the following tasks: save the currently executing instruction, determine the type of interupt, save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency needs to be minimized in order to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so the interrupt does not get serviced immediately. For hard real-time systems, the time-period for which interrupts are disabled must be bounded in order to guarantee the desired quality of service.

19.2 Hard real-time scheduling constraints are required for the nuclear power plant and for the jet airliner. In both settings, a delayed reaction could have disastrous consequences and therefore the scheduler would need to satisfy real-time scheduling requirements. On the other hand, the thermostat and the fuel economy system can be operated within the context of a scheduler that provides only soft real-time constraints. Delayed triggering of these devices would result only in suboptimal use of resources or a slight increase in discomfort.

19.3 Consider when P_1 is assigned a higher priority than P_2 with the rate monotonic scheduler. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. When P_1 is assigned a lower priority than P_2, then P_1 does not meet its deadline since it will not be scheduled in time.

19.4 The priority inversion problem could be addressed by temporarily changing the priorities of the processes involved. Processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question. When they are finished, their priority reverts to its original value. This solution can be easily implemented within a proportional share scheduler; the shares of the high-priority processes are simply transferred to the lower-priority process for the duration when it is accessing the resources.

19.5 Consider two processes P_1 and P_2 where $p_1 = 50$, $t_1 = 25$ and $p_2 = 75$, $t_2 = 30$. If P_1 were assigned a higher priority than P_2, then the following scheduling events happen under rate-monotonic scheduling. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. The earliest deadline schedule performs the following scheduling events: P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 55$, and so on. This schedule actually meets the deadlines and therefore earliest-deadline-first scheduling is more effective than the rate-monotonic scheduler.