Chapter 1

Introduction

This dissertation presents the design, implementation, and experimental analysis of a model for operating system resource reservation. The reservation system supports predictable performance in real-time and multimedia applications, enabling them to meet their timing requirements, and facilitating adaptive resource management. This approach is suitable for real-time programming problems that arise in personal computers and workstations where users may want to run real-time multimedia applications or other real-time programs. The approach is also applicable to embedded system design where better resource reservation abstractions at the system level aid in the design, debugging, and maintenance of such systems.

1.1 Motivation

Recent increases in processor speed and network bandwidth combined with the wide availability of digital audio and video devices have enabled a plethora of multimedia applications and services. Examples of these include audio/video presentation and playback, audio/video phone and conferencing, persistent multimedia data storage services, telephone answering and call management, speech processing, and virtual reality applications.

Stringent timing constraints and large volumes of data characterize these applications. Existing operating systems are not designed to support such applications, especially when real-time multimedia applications execute alongside a non-real-time workload. A key requirement of systems for multimedia applications is predictability, and this means that possible contention for resources must be identified and managed to ensure the timeliness of multimedia data processing and delivery. Although it is possible to manage contention for system resources in an *ad hoc* manner based on the specific requirements of a particular class of applications, this dissertation describes a more structured approach to managing contention based on:

- A reservation model that provides an abstraction for resource capacity reservation and a system-level mechanism for scheduling resources.

- A higher-level layer for implementing resource management policy using the system-level mechanism.
- Programming techniques for structuring applications in a way that can take full advantage of the resource reservation model.

The reservation mechanism and allocation policy provide abstractions that relieve the system designer from relying on complicated high-level application information to make low-level scheduling decisions. And the programming techniques facilitate the programming of real-time applications that meet their timing constraints.

1.2 Background

Real-time system designers must take timing constraints into account when developing real-time applications and the systems to support them. The programming techniques and resource management policies that have been developed for real-time systems typically apply *ad hoc* solutions for each application area. Several issues and problems arise which can be addressed with appropriate system abstractions.

1.2.1 Programming real-time applications

In applications with timing requirements, the software must be designed to satisfy the timing constraints. The user-level servers and system services used by such applications must be designed with timing constraints in mind. The resource management policies underlying those system services must also be designed to support applications with timing constraints.

Traditionally in real-time system design, application programmers use small, simple operating systems that provide fixed priority processor scheduling, priority queueing for various system resources such as semaphores and mailboxes, and perhaps priority inheritance protocols. The application programmers must then build many of their own real-time system services such as database management systems, file systems, and network communication. These programmers must carefully schedule the various applications running on the system and manage contention for the processor and other system resources. In doing the design and scheduling, the computational requirements of applications must be carefully measured and characterized, and resource sharing must be carefully planned. Applications are therefore very sensitive to the misbehavior of other applications. For example, if a high-priority real-time application has a bug that sends it into an infinite loop, the effect on other applications and the system as a whole would be devastating. The errant application would take over the processor and would not relinquish it, forcing a system crash.

This extreme sensitivity among applications is the result of a lack of suitable system abstractions for effectively managing shared resources in real-time systems. Many abstractions exist in real-time scheduling theories, but typically the assumptions of the theoretical results are implicitly embedded in real systems. An example of an assumption that many theories require is that the worst-case execution times (WCET) for computations in real-time applications are known at system design time. Most systems designed using analysis techniques that have this assumption do not explicitly check or enforce worst-case execution times for computations. Appropriate system abstractions would explicitly bring the assumptions into the actual system where they could be checked and, in the best case, even enforced.

1.2.2 Resource management problems

Suitable system abstractions could effectively address several problems that arise in real-time system. These problems fall into three broad areas: resource management policies that can satisfy timing constraints, mechanisms to support the policies, and analysis techniques based on the available mechanisms.

Many systems do not provide scheduling policies that directly support real-time resource management. For processor scheduling, most systems provide either fixed priority or deadline scheduling policies. Both of these policies lack complete information about real-time requirements and therefore do not address important problems such as how priority should be assigned or how to prevent overload. The following issues arise in the design of resource management policies:

- **Priority assignment problem:** Simple priority schedulers are hard to use, especially if there are many activities with timing constraints. If there is no global repository of knowledge about the timing constraints of different activities in different applications, there is no basis for deciding what the priority ordering of the activities should be.
- **Overload problem:** To prevent overload, the scheduling policy requires information about real-time constraints such as the computational requirements and frequency of execution. Even if the designer can express the resource and timing requirements for real-time applications, a system with no admission control policy cannot protect itself from overload.
- **Flexibility requirement:** The timing constraints and resource requirements for dynamic real-time applications may change dynamically during execution. The scheduling algorithm must support efficient adjustment of requirements.

To effectively schedule real-time applications such that applications cannot monopolize system resources requires some usage measurement and enforcement mechanisms. Information gleaned through these mechanisms can be used in the scheduling policy to make decisions about how priorities or deadlines should be dynamically adjusted to reflect the requirements and usage patterns of applications. The problems that motivate the use of these mechanisms are described briefly below:

- **Enforcement problem:** An application that specifies resource and timing requirements may accidentally or deliberately attempt to exceed its stated requirements. This may interfere with the satisfaction of timing constraints for other reserved activities, and it may cause starvation among unreserved activities.

- **Measurement problem:** Enforcement of resource requirements means that resource usage of each application must be accurately measured and compared to the allocation that has been made on its behalf. If the activity uses external modules, servers, and system services, the measurement must include that usage as well.
- **Coordination problem:** Many time-constrained activities are composed of multiple sub-activities implemented by other modules, user-level servers, or system services. Allocating resources for a single activity that spans multiple modules, possibly in different memory protection domains, is complicated. It is necessary, however, to be able to place timing requirements on the overall activity and to track the resource usage for the overall activity.

Other problems deal with higher level issues of how to analyze system behavior given more sophisticated abstractions and mechanisms for scheduling and resource management. The following issues arise in this context:

- **Distributed real-time scheduling problem:** An activity that uses external modules and services may require the use of multiple resources within certain time constraints. Coordinating usage across multiple resources to meet timing constraints is a hard problem.
- **QOS management problem:** Real-time applications may dynamically change their quality of service (QOS) requirements. In a system with many such applications, a high-level resource manager (sometimes called a QOS manager) is needed to resolve conflicts and negotiate between applications.

In traditional real-time systems design, many of these problems are avoided in the design phase by careful measurement and analysis of simple computations and their resource requirements and by ad hoc scheduling techniques. This approach results in inflexible systems that are difficult to maintain [43,68]. In particular, supporting dynamic real-time activities with timing constraints and resource requirements that may change freely at run-time stretches the traditional approach beyond its limits.

Recent work in real-time systems addresses some of these problems. Several systems (e.g. [3,53,113,124,125]) allow specification of timing requirements instead of forcing the programmer to determine an appropriate priority assignment. A few systems have on-line admission control policies [3,78,113], but many others use off-line analysis [53,124,125]. Still others have no admission control at all [19,21,90]. Some limited flexibility requirements for hard real-time have been addressed recently [122]. This work focuses on mode changes, which are radical but infrequent changes in the task sets of a real-time system. For example, the real-time system in an airplane might experience a mode change after takeoff as it switches from the ground-based task set to the air-based task set.

Very few software systems address the measurement and enforcement problems with respect to resource usage although most systems can detect and react to missed deadlines [43,125]. In networks, the notion of enforcement or policing is much more common [33,98,128]. The work on priority inheritance protocols [8,16,51,86,97,108] addresses some aspects of the coordination problem.

The distributed real-time scheduling problem is an active area of research [38]. The QOS problem is another active area of research. Some operating system research in this area focuses on best effort approaches [21] although other research emphasizes guarantees [46,53,78,81,126].

1.3 Resource reserves

This dissertation defines a resource reservation model that provides an operating system abstraction called a reserve. Reserves explicitly represent reservations on resources such as processors, memory pages, disks, and network devices. In particular, reserves support

- specification of reservation parameters,
- admission control,
- scheduling based on timing constraints and usage requirements,
- reservation enforcement,
- reserve propagation in the RPC mechanism,
- flexible binding of threads to reserves.

Reserves prevent applications from over-running their allowed resource usage and interfering with other reserved activities or starving unreserved activities. Applications reserve capacity on the resources they need to carry out their computations. For example, an application can reserve 10 ms of computation time on a processor for every 100 ms of real-time. The application then binds to the reserve, and the processor scheduler uses the information associated with the reserve to control the scheduling of the application. The system also performs an admission control test before granting the reservation to make sure that the resource can support the reservation being requested. The enforcement mechanism ensures that an application does not use more than its reserved time if doing so would interfere with other reserved activities.

The reservation parameters associated with an application's reserves are not necessarily fixed for the lifetime of the application. A dynamic real-time application must be prepared to change its behavior and timing requirements based on changing requirements of users and possibly the changing availability of resources. A user may want to change the frame rate on a video player or change the resolution, and the application must be ready to adjust its resource reservation levels appropriately. Likewise, reserves must support an operation that modifies the reservation parameters, subject to the admission control policy.

Reserves are first class objects in the operating system; a reserve is associated with a particular thread (or process) by explicitly binding the thread to the reserve. This allows an application to reserve all of the resources it will need for its computation, including resources that will be needed by various modules, servers, and system services it intends to invoke. The application can then pass references for its resource reserves to modules or servers along with the operation invocation. The module or server can bind its thread to this reserve when performing the operation, and it can then take advantage of the resources that have been reserved by the client. Having modules and servers charge a caller's reserve for work done on behalf of the caller also maintains a consistent view of the resource usage that is being consumed on behalf of that client.

The reserve model presented in this dissertation addresses the problems identified in the previous section. The scheduling policies embedded in the reserve model address the priority assignment problem and the overload problem while remaining flexible in terms of accommodating dynamic changes in application resource requirements as discussed below.

- **Priority assignment problem:** Reserves avoid the priority assignment problem by accepting reservation specifications that include the timing constraints and usage requirements.
- **Overload problem:** The admission control policy of the reservation mechanism prevents overload. This is possible since the scheduling policy has information about both the computational resource requirements and the timing constraints (such as period of invocation) for each application.
- **Flexibility requirement:** Changes in reservation parameters can be made at any time, subject to the admission control policy.

The reserve model makes use of several system mechanisms that support the abstraction. These mechanisms provide information about resource usage and that coordinate the consumption of resources for an activity that crosses memory protection boundaries as follows:

- **Enforcement problem:** Reservation enforcement isolates reserved activities from undue interference from other reserved activities.
- **Measurement problem:** The flexibility in binding reserves makes it possible to accumulate resource usage charges for an activity even when work is done by external modules, servers, or system services.
- **Coordination problem:** The reserve model supports reserve propagation which includes a "priority" inheritance mechanism and which takes advantage of the flexibility in binding reserves to threads.

The reserve model provides an abstraction that can be used for distributed real-time scheduling and QOS management. The approaches to these problems are described briefly below:

- **Distributed real-time scheduling problem:** The reserve model supports reservations for remote resources, and pipeline-style software architectures are supported. This is not optimal, but future work on this problem could take advantage of the reserve model as a base.

- **QOS management problem:** A high-level QOS management layer can use reserves to carry out resource allocation policy decisions. The QOS managers can carry out their allocation decisions by manipulating reservation parameters for the applications being managed.

1.4 Contributions

This dissertation describes and analyzes a resource reservation solution to the problem of supporting predictable execution of real-time and multimedia applications with specific quality of service parameters. It shows that:

> Resource reserves, an enforced operating system resource reservation abstraction, effectively supports real-time and multimedia applications. Reserves allow the software designer to specify timing requirements on resources required, thus providing a method for guaranteeing deadlines in real-time applications. The reservation abstraction accommodates non-real-time applications as well as real-time applications.

This dissertation defines an enforced resource reservation model called reserves and then describes programming techniques for developing applications using reserves. It presents an implementation of one type of resource reserves, processor reserves, in Real-Time Mach along with several real-time applications that use reserves to satisfy their timing constraints. These applications included a suite of synthetic benchmark programs, a QuickTime video player, an MPEG player, and a version of the X server. Experiments with these applications showed that reserves can indeed provide predictable behavior for real-time applications even with competition from other real-time and non-real-time applications.

Reserves are a tool that real-time system designers can use to raise the level of abstraction for resource scheduling in real systems. This allows applications' timing requirements to be defined and guaranteed in isolation with the system providing high-level guarantees that resources will be available when needed by each real-time application.

1.5 Overview of the dissertation

Chapter 2 describes the background and motivation for the dissertation in more detail. Chapter 3 describes the reserve model, and Chapter 4 discusses techniques for structuring programs to best take advantage of reserves.

Chapter 5 describes the implementation of processor reserves in Real-Time Mach, a quality of service (QOS) manager, and several reserved applications and servers. Chapter 6 presents an experimental evaluation that explores the predictability achieved by using reserves and the overhead involved in providing that predictability.

Chapter 7 discusses related work, and Chapter 8 summarizes the contributions of the dissertation and presents the conclusions and future directions.