Chapter 8

Conclusion

This dissertation has presented a comprehensive model describing resource reserves, the operations they support, the scheduling algorithm and enforcement mechanism required, and how reservations for various resource types can be encapsulated in a single framework. An implementation and experimental evaluation demonstrate that real applications with non-trivial client/server interactions can achieve predictable real-time performance using resource reserves. The reserves ensure this predictability even when there are multiple real-time and non-real-time applications competing for the same resources.

This work shows that system mechanisms that address entire activities are important for real-time resource management. Furthermore, it shows that enforcement is essential, otherwise a reservation abstraction has no meaning. Programming techniques such as software pipeline architectures with synchronized periods and deadlines are useful for achieving predictable behavior. The following sections detail the contributions of this work and directions that this work opens for future research.

8.1 Contributions

The contributions of this work include the abstraction for operating system resource reservation, its implementation, real applications which use it, and an experimental evaluation of those applications. The following sections discuss these in more detail.

8.1.1 Resource reservation abstraction

The resource reserve abstraction provides a model for how real-time scheduling algorithms and analyses can be incorporated in an operating system design in an integrated manner. This is in contrast to the specification of scheduling algorithms and analyses in the context of a simplified task model where many practical systems issues and programming issues are ignored. Two key features of the abstraction are flexible binding of reserves to threads and enforcement of reservation parameters.

Since reserves are first class objects in the system rather than being tightly and permanently bound to threads (or processes), the management of resources is much easier. For example, reservation parameters can be allocated for a reserve by a thread and then a reference to the reserve can be passed to system service providers invoked by the thread. By allowing the binding of reserves to threads to be flexible, reserves can be passed around in this way, and the resource usage for the abstract activity is tracked and guaranteed throughout all of the server calls.

The enforcement mechanism eases program development and debugging for programmers of hard and soft real-time applications. Programmers of hard real-time applications can exploit the usage accumulation mechanism to measure the requirements of their code during development. During runtime, the enforcement mechanism and usage measurements can be used to isolate timing bugs. Programmers of soft real-time applications can use the enforcement mechanism to ensure isolation between applications and to provide information on resource usage requirements for adaptive applications. This relieves the programmer of doing the exhaustive measurements and analysis usually required to achieve real-time predictability.

8.1.2 Implementation

The implementation of processor reserves in Real-Time Mach and the implementation of several real applications that use reserves demonstrate the feasibility of the approach described in this document. The implementation shows how to design an enforcement mechanism and integrate it with the scheduling policy. It shows how a reserve propagation mechanism can be built to ensure consistent resource reservation and account for abstract activities that span multiple threads (or processes). It also demonstrates how QOS managers can be incorporated into the system to negotiate reservation parameters between reserved applications and the operating system, and how resource usage monitor and control can be used to promote awareness of resource requirements and dynamic adjustment of resource allocation.

8.1.3 Experimental evaluation

The experimental evaluation demonstrates that the applications using the reservation system can achieve predictable behavior with acceptable overhead costs. Experiments with synthetic benchmark applications were able to achieve very consistent real-time performance even in the face of competition from other real-time and non-real-time applications. In the experiments, periodic reserved applications ran over a relatively long duration of time, and measurements of the processor utilization during each period were recorded. The 5-percentile and 95-percentile numbers for these measurements were typically within 5-7% of the average utilization across all the periods, indicating a very tight distribution of processor utilization measurements across periods and a quite consistent pattern of real-time behavior. In other experiments with real applications such as a video player and X Server, processor utilization measurements yielded 5- and 95-percentiles which differed from the average utilization by up to 65%. This is not nearly as tight as the synthetic client/server benchmark applications due to the input/output organization of the X Server which is not ideally suited to reserve propagation techniques. In any case, the behavior of reserved X clients was much better than that of unreserved X clients, which had 5- and 95-percentiles that were as much as 166% of their average utilizations. Experiments with network transmit/receive applications showed processor utilization measurements with 5- and 95-percentiles of 7-16% of the average utilizations. When unreserved, these applications had measurements with 5- and 95-percentiles of up to 360% of their average utilizations. In each case, the reserved application showed much more consistent real-time behavior than its unreserved counterpart.

8.2 Future directions

The work described in this document opens up many avenues for future research. By providing a general framework and testbed platform for operating system resource reservation, this work provides a concrete context for many research topics such as QOS provision and negotiation, resource allocation algorithms, and adaptive application programming techniques.

This work on resource reservation also provides a design point for comparison with other system design approaches [65] and for other scheduling paradigms [119]. The idea of enforced resource reservation can also be used as a building block for exploring higher-level concerns such as the role of user in resource management [57].