In this project, object-sharing schemes for both real-time and non-real-time concurrent systems have been investigated. A particular focus has been uniprocessor and shared-memory multiprocessor systems with processors that are multiprogrammed (many processes executing on the same processor). Much of the work in this project has been concerned with lock-free and wait-free shared object implementations. Such implementations are not lock-based, and therefore are immune to performance problems associated with process preemptions in multiprogrammed systems. A variety of new algorithmic techniques for efficiently implementing concurrent objects have been developed and tested in this project. In addition, research has been conducted on schedulability tests for use in real-time systems in which the proposed object-sharing techniques are used.
Mechanisms for Scalable Object Sharing in MIMD Multiprocessing Systems

Final Progress Report

James H. Anderson

July 1998

U.S. Army Research Office

Grant Number DAAH04-95-1-0323

Department of Computer Science
The University of North Carolina at Chapel Hill
Campus Box 3175
356 Sitterson Hall
Chapel Hill, North Carolina 27599-3175

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

THE VIEWS, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.
Problem Studied

In this project, object-sharing schemes for both real-time and non-real-time concurrent systems have been investigated. A particular focus has been uniprocessor and shared-memory multiprocessor systems with processors that are multiprogrammed. In multiprogrammed systems, several processes may execute on the same processor. Processes on the same processor are scheduled for execution either by priority or by allocating a scheduling quantum. One of the main reasons for adopting a multiprogrammed execution model is that it enables problems to be solved without static constraints on the number of processes that may be employed. One price to be paid for this flexibility is having to deal with frequent process delays due to preemptions.

Much of the work in this project has been concerned with lock-free and wait-free shared object implementations. As explained in the original proposal and in previous interim reports, such implementations are not lock-based, and therefore are immune to performance problems associated with process preemptions in multiprogrammed systems. Most of the research conducted in this project has focused on two main goals: (i) to develop efficient algorithmic techniques for implementing concurrent objects, and (ii) to determine how to account for object-sharing overheads that arise when using such techniques in real-time schedulability tests.

Summary of Results

In most conventional wait-free and lock-free shared object implementations, performance does not scale well with an increase in either the number of processes sharing the object, or the size of the object. To deal with the former problem, we developed object implementations that incorporate lock-based synchronization techniques to limit the number of processes that can concurrently access an object [1]. Performance studies conducted by us have shown that, in multiprogrammed systems, the use of such techniques results in performance that is better than that of pure wait-free objects. To deal with the problem of implementing large objects, we developed wait-free and lock-free object constructions in which the state of the object is fragmented into smaller pieces that can be updated and managed more efficiently [15]. Performance studies conducted by us on a KSR-1 multiprocessor have shown that, for many common objects, these implementations perform significantly better than previous ones.

Our work on real-time object sharing has focused both on scheduling conditions for real-time tasks that share objects, and on algorithms for implementing such objects. For real-time applications on uniprocessors, we have shown through work on task scheduling that lock-free objects often outperform conventional lock-based schemes by a substantial margin [2,5,11,16]. The good performance of lock-free objects in this context is primarily due to the fact that they avoid priority inversions with no kernel support for object sharing (in contrast to lock-based schemes) and with low algorithmic overhead (for most common objects). We have validated these claims both formally, based on scheduling models, and experimentally, based on research involving a desktop videoconferencing system.

We have also shown that it is possible to optimize wait-free algorithms in real-time multiprocessor systems by exploiting the inherent synchrony that exists in such systems [5,7,8,10,13]. We have evaluated the performance of these optimized wait-free implementations by conducting a variety of simulation experiments and by performing stress tests on a four-processor SGI Origin. These experiments indicate that wait-free objects implemented using the proposed techniques outperform other object-sharing schemes by a wide margin and lead to better schedulability in real-time systems.

We have also initiated new work on memory-resident real-time databases (RTDBs) [3,4,9,14]. RTDBs differ from conventional database systems in that transactions can have soft, firm, or hard deadlines. In most existing RTDBs, soft or firm deadlines are supported by modifying conventional database protocols to provide preferential treatment to high-priority transactions. In such systems, best-effort scheduling protocols are employed to minimize the number of transactions that miss their deadlines. In many military applications, however, certain critical transactions must be guaranteed to
satisfy hard real-time constraints. For example, in an air defense system designed to neutralize enemy ballistic missiles, database updates may be triggered to track and identify potential threats and to issue appropriate counterstrikes; such updates must complete by their deadlines. Our work on RTDBs has focused on hard-deadline uniprocessor and multiprocessor systems. Our approach is to implement such transactions using highly-optimized lock-free and wait-free algorithms that execute at the user level. Since no underlying system support is needed for transactions, this allows RTDB functionality to be achieved in embedded applications without complicated protocols for avoiding priority inversion and deadlock, for supporting mode changes, and for handling transaction abort/recovery.

Publications

Journal Papers.


Book Chapters.


Conference Papers.


September 1997.


**Participating Scientific Personnel**

**Faculty.**

James H. Anderson, PI, Associate Professor.

**Graduate Students.**

Rohit Jain (partially funded by other sources).
Mark Moir (partially funded by other sources).
Srikanth Ramamurthy.

**Honors/Awards/Degrees.**

Alfred P. Sloan Research Fellowship, February 1996, Prof. Anderson.


M.S. awarded, May 1998, Rohit Jain.

**Report of Inventions (by Title Only)**

Wait-free algorithms for implementing large shared objects.

Scalable shared object algorithms for large multiprogrammed systems.

Algorithms for implementing lock-free transactions on priority-scheduled real-time uniprocessor systems.

Algorithms for implementing wait-free objects on priority- and quantum-scheduled uniprocessor and multiprocessor systems.

Algorithms for implementing wait-free transactions on priority-scheduled real-time multiprocessor systems.

Scheduling conditions for lock-free, real-time uniprocessor systems.

Scheduling conditions for wait-free, real-time uniprocessor and multiprocessor systems.