MICROCOPY RESOLUTION TEST CHART

1.0
1.1
1.25
The progress achieved during the second year of the Saguaro distributed operating system project is presented. The major accomplishments were the refinement of the system design and the prototype implementation of several system components. Work was also performed on related research, including the redesign of the distributed programming language SR and the implementation of a system for distributed mixed language programming.
Summary

The progress achieved during the second year of the Saguaro distributed operating system project is presented. The major accomplishments were the refinement of the system design and the prototype implementation of several system components. Work was also performed on related research, including the redesign of the distributed programming language SR and the implementation of a system for distributed mixed language programming.
Status of the Research

In the past year we have implemented prototypes of components of the Saguaro distributed operating system and refined the design of the entire system based on this experience. The philosophy behind Saguaro is to support the illusion of a single virtual machine while taking advantage of the concurrency and robustness that are possible in a network architecture. Within the system, these advantages are realized by the use of pools of server processes and decentralized allocation protocols. Potential concurrency and robustness are also made available to the user through low-cost mechanisms to control placement of executing commands and files, and to support semi-transparent file replication and access. Another unique aspect of Saguaro is its extensive use of a type system to describe user data such as files and to specify the types of arguments to commands and procedures. This enables the system to assist in type checking and leads to a user interface in which command-specific templates are available to facilitate command invocation. A mechanism, channels, is also provided to enable users to construct applications containing general graphs of communication processes.

Several reports have been issued describing various aspects of Saguaro. The preliminary design of the system and its components was presented in [1]. That report has now been extensively revised to reflect the current design of the system [2]. The file system mechanisms that support file availability are described in [3]; a prototype implementation of these mechanisms in the UNIX environment has confirmed their utility and is near the point of being a very useful tool in its own right.

Two related research avenues were also pursued. One involved constructing a formal axiomatic derivation of an algorithm for rectangle partitioning—a problem motivated by the window manipulation necessary for the Saguaro user interface. A report on this topic appeared last May [4]. The second involved applying the Saguaro type system to the problem of writing distributed, mixed-language programs. A system for constructing such programs has been described in [5] and is currently being implemented.

A large amount of effort was also expended revising our implementation language Synchronizing Resources (SR) and implementing a compiler for the new version. The main language constructs are still resources and operations: resources encapsulate processes and variables they share, operations provide the primary mechanism for process interaction. One way in which SR has changed is that both resources and processes are now created dynamically. Another change is that the mechanisms for operation invocation—call and send—and operation implementation—proc and in—have been extended and integrated. Consequently, all of local and remote procedure call, rendezvous, dynamic process creation, asynchronous message passing, multicast, and semaphores are supported. We have found this flexibility to be very useful for distributed programming. The language has also been refined in numerous additional ways to provide additional flexibility. Moreover, by basing SR on a small number of well-integrated concepts, the language is also relatively simple and has a reasonably efficient implementation.

The new version of the SR language is defined in [6]. An overview of the language and its implementation are described in [7]. The compiler and runtime support currently execute on top of UNIX; they have been in use since November 1985. A detailed discussion of how SR has
evolved—what has changed and why as well as what has not changed and why not—is given in [8].

An important aspect being considered in the design of Saguaro is providing some degree of tolerance to hardware failures. One reason such failures are difficult to handle is that they occur asynchronously with respect to program execution. We have recently begun investigating how this asynchronous nature of hardware failures might be handled within distributed programming languages. Since we do not yet know what the best approach will be, we have begun two lines of investigation. First, we have incorporated relatively low-level mechanisms—a failed primitive and status returns from uses of operations—into the new version of SR. Second, we have defined high-level mechanisms that treat a failure as an asynchronous event in a concurrent system of processes and thus regard failure-handling as the problem of dealing with these events. In particular, the user specifies operations that are to be implicitly invoked when failures occur; these operations are serviced in much the same way as normal, programmer invoked operations. This latter approach is described in [9].

Finally, the Sun workstations intended to support this research were installed during the summer. The purchase of this equipment was financed by a grant from the DoD University Research Instrumentation Program (URIP) [10].

References


END
DATE
FILMED
DEC.
1987