Rendezvous Facilities in a Distributed Computer System

Liao Xianzhi (廖先治)
Department of Computer Science, Tsinghua University, Beijing 100084

Jin Lan (金 兰)
Department of Computer Science, California State University-Fresno, Fresno, CA 93740-0109

Received July 1, 1993; revised October 30, 1993.

Abstract
The distributed computer system described in this paper is a set of computer nodes interconnected in an interconnection network via packet-switching interfaces. The nodes communicate with each other by means of message-passing protocols. This paper presents the implementation of rendezvous facilities as high-level primitives provided by a parallel programming language to support interprocess communication and synchronization.

Keywords: Rendezvous, packet-switching interface, message-passing protocols, interprocess communication and synchronization, high-level primitive, parallel programming language, interconnection network.

1 Introduction
The development of distributed systems has been stimulated by decreasing hardware cost and performance improvement over single-processor architectures. A distributed computer system named "THUDS" has been constructed in Tsinghua University, Beijing, China. It provides a parallel computing environment that serves a stream of jobs, each consisting of a set of interactive processes distributed among multiple computer nodes for solving a single problem. This application of distributed computers has been referred to as multistream operation and used to speed up problem solving through parallel processing. In order to take the advantage of computer nodes for concurrently executing processes a programming language supporting parallelism is required. The important components in a parallel language are mechanisms for synchronization and communication. The interprocess rendezvous is a language control that provides these mechanisms.

In Section 2 the rendezvous semantics and their implementation in the rendezvous facilities are discussed, and a typical example of parallel program are presented in Section 3. Conclusions are drawn in Section 4.

2 The Rendezvous Facilities
In this section, we describe first the rendezvous semantics, and then the implementation of rendezvous facilities.

The rendezvous is the basic synchronous communication primitive for general concurrent languages, such as Ada, Concurrent C, and Occam. It is also one of several forms of
communication and synchronization in the distributed programming language NIL. The concept of rendezvous construct can be traced back to the process communication primitives described in Hoare's CSP and Brinch Hansen's DP. The term rendezvous was first used by the designers of Ada to describe the construct for communications between tasks, which are referred to as processes in other languages. In our system, rendezvous is a high-level primitive which can be used to send a message through communication network, while avoiding the problems associated with low-level primitives such as semaphores. The two processes are involved in a rendezvous communication by means of a remote procedures call style interface and the suspended execution of processes while data are transferred. If one process requests a rendezvous while the other is busy, the rendezvous request will be enqueued to wait for the other process to become available. The calling process in a rendezvous is capable of handling several different call interfaces, i.e., may multicast a message to several different node-computers. This feature is used to compute multiple tasks in parallel in our system.

Two basic cases of rendezvous semantics are presented as follows.

(a) Caller blocks first

Fig. 1 shows that the caller issues a request before the acceptor is ready to execute, i.e., the receiving queue is not empty. The calling process is blocked by the rendezvous and the request is enqueued in the receiving queue. When the accepting process is ready for service, the request is dequeued and completed. Any return parameters are made available to the caller and are sent through the network. While the caller is ready to receive the return parameters, it is resumed.

(b) Acceptor blocks first

Fig. 2 shows the case when the acceptor is the first process to reach the rendezvous point, i.e., the receiving queue is empty. If a calling message is received, the rendezvous begins immediately. The rendezvous completes in the same manner as described for Fig. 1.

In the following we will discuss the case of simple call and accept, which is straightforward and contains the basic operations of all forms of rendezvous.

In order to provide mutual exclusion for sending the calling messages of any node-computer, the compiler generates a lock of Boolean variable "finish" for testing. It is initialized to true. Before a calling process is ready to send a request message, the Boolean variable is tested. If it is false, then the calling process waits, otherwise the "finish" is set to false, the calling process is blocked, and the request message is sent byte by byte by the output interrupt routine. When sending is completed, the Boolean variable is set to true.