Reliable Resource Allocation Between Unreliable Processes

S. K. SHRIVASTAVA and J.-P. BANÅTRE

Abstract. Basic error recovery problems between interacting processes are first discussed and the desirability of having separate recovery mechanisms for cooperation and competition is demonstrated. The paper then concentrates on recovery mechanisms for processes competing for the use of the shared resources of a computer system. Appropriate programming language features are developed based on the class and inner features of SIMULA, and on the structuring concepts of recovery blocks and monitors.

Index Terms. Concurrent processes, error recovery, monitors, recovery blocks, reliable programs, resource allocation, software redundancy.

I. Introduction

The realization that even a well-designed and tested system is likely to contain residual faults has increasingly led designers to consider the application of redundancy techniques to software construction. Recently a program structure called a recovery block has been developed that allows redundancy, in the form of standby spares, to be added systematically and efficiently to computer programs to make them more reliable [1], [2]. The essence of this scheme is that it provides a facility for a computation to be backtracked to an earlier state, if an error is detected, and proceed again using a possibly different algorithm.

In this paper we extend this idea to apply to the error recovery problems of concurrent processes of an operating system sharing the limited resources of a computer system. A very general overview of the problem we wish to tackle here can be obtained by considering the progress of a process in an operating system. Suppose that during its “forward motion,” the process is generating results entirely by assignments to variables in its private space. If an error is detected, the “reverse motion” of this process to a prior state is easily performed — the undoing of assignments is equivalent to the restoration of prior values. However, the actions of a process can be quite diverse — for instance, control of a peripheral. In general then, during its forward motion this process will generate results by recording them in various resources, such as storage locations, input-output equipment. Since many of the resources involved will usually be shared between processes, the process under consideration will occasionally be interacting with other processes during its progress. If an error is detected, the reverse motion of this process is no longer as easy as before and it may become necessary to provide algorithms for undoing the effects of previously done operations. Just as the process interacts with other processes during its forward motion, it may also interact during its reverse motion. It is thus seen that when programming for processes that are capable of backtracking, apart from programming for their normal forward progress, we must also be prepared to pro-
gram for their reverse progress. The important point to note is that appropriate programming language tools must be provided to cope with this additional complexity in a systematic manner, otherwise resulting programs are likely to be even less reliable than versions with no redundancy. We believe that the programming language features developed in this paper meet the above criterion; however, the reader must be the ultimate judge.

The paper is structured into seven sections. Since recovery blocks play a prominent role in this paper, they are discussed in Section II where the role of exception handlers is also described. It is necessary to understand the main error recovery problems between interacting processes before appropriate programming language features can be developed. For this reason, we discuss the recovery problems in Section III and then in Section IV, we discuss the recovery requirements for a class of problems for which language features are to be developed. In Section V we describe how recovery blocks can be introduced into resource allocation algorithms implemented using monitors. A program structure called a port is developed in Section VI. A port provides facilities for specifying how a resource should be used and what recovery actions a reversing process should undertake. Finally, in Section VII, we briefly discuss some implementation details and summarize the work presented.

II. Error Recovery

A. Recovery Blocks and the Recovery Cache

A system can contain residual faults both in the hardware and software. Therefore, when an error is detected, quite often it is not possible to determine the sources of the fault; backward error recovery then is the only sensible solution. The recovery action consists of restoring a prior state of the computation and proceeding again in the hope of avoiding the fault. By using a different algorithm, after the restoration, a measure of fault tolerance against software faults can be obtained. A recovery block is a program structure embodying backward error recovery.

A recovery block consists of a conventional block which is provided with a means of error detection (an acceptance test) and zero or more standby spares (alternatives). Its structure is shown in Fig. 1.

```
ensure (acceptance test) by
   (primary block)
else by
   (alternative 1)
   ...
else by
   (alternative n)
else error;
```

Fig. 1