SYNCHRONIZATION AS A FRAMEWORK FOR DISTRIBUTED SYSTEM
FAULT-TOLERANCE DESIGN
Alexander B. Romanovsky, St. Petersburg Technical University

1. Introduction
We shall regard a computer system as a whole which comprises software, hardware and mixed components each of which can, in its turn, present a system. Then the entire system is a multilevel hierarchy.

The purpose of this paper is to single out and generalize about the essential features and properties of synchronization and to argue in favour of the idea of designing and developing fault-tolerance (FT) for distributed systems on the basis of a multilevel synchronization system. Getting aware of and constructing a synchronization system like this can be, and often is, a basis for providing system FT (with masking, recovery, voting, reconfiguration, migration, warm-starting, etc. subsequently built over). The synchronization system controls and coordinates the operation of all main and redundant components and it is by controlling redundancy that FT is achieved. Thus, error recovery can be done precisely at that system level where there is some redundancy; to do this, the operation of the main component and of the redundant ones has to be synchronized. Besides, for that purpose all those components have to be in a certain known state (e.g. active, backup, fault, identical, alive, etc.) and providing this is just what synchronization is meant to do.

We propose to consider designing and developing a hierarchical system of synchronizing (HSS) redundant components in distributed systems the first and foremost stage in providing FT in application systems. In this way the entire FT system can be structuralized naturally. HSS can be used as a basis underlying the implementation of any of the known FT schemes in which redundant components are controlled by different CPUs or can be active on their own (disk-drivers, plants, processes, etc.).

Our idea and approach is a special case of the approach developed in [1]. We hope to add to the practical applicability of the latter (for FT design, particularly in Distributed OSs - DOSs) and at the same time to single out and analyse the most essential, underlying feature (i.e. component synchronization) in providing FT.

2. Redundancy Synchronization
The operation of many FT schemes results in providing the user with a virtual fault-tolerant (ft) resource by means of employing several redundant ones. In this way a fault in one or several components can be parried transparently. This is true for the well-known schemes employing hardware redundancy: workby, standby, TMR [2]. In a similar way a ft servicing is created not just for resources but for components of any type as well.

Let us call synchronizing the operation of redundant software, hardware or mixed components for FT purposes a redundancy synchronization (RS). With the help of RS a higher level component (or a DOS user) has a virtual ft-resource at its (his) disposal. For instance, a user gets one ft-disk, one ft-process (process replication), one ft-file (computer, bus, etc.) instead of several ones.

Since there naturally arises a time shift in the operation of identical components (primarily due to differences in computing speed), the main purpose of RS is to make them synchronous. For any
recovery scheme to be employed, this is a condition sine qua non.

What we require is not for the components to be identical but for their states to be coordinated. For instance, in case of software redundancy each component can be developed independently, but every one of them provides the same interface (service) for higher level components. It is not necessary either that the redundant component should be used directly to replace the faulty one, since the recovery scheme employed may involve not just replacing a component, but voting and averaging the results in this or that way. Besides, for software or mixed components one can use a standby recovery scheme; in it such information is saved (backup, logging, recovery point) that ensures that the redundant component can be made identical with the main one at the synchronization moment.

3. Redundant Resource Synchronization on DOS Level
Since DOS controls such resources as CPUs (which implies controlling processes), devices, files, etc. as well as nodes as a whole, it seems natural to construct a synchronization system of available redundant resources of this kind as a DOS subsystem. It is in this case that FT becomes transparent for DOS users. Let us show how RS is used for controlling the main DOS resources:

- Controlling redundant processes and redundant CPUs: this is known as process replication [3]. Several identical copies of the same program are executed simultaneously (in different nodes). They are synchronized most often at message transmission or plant input/output levels or at the result returning stage (for procedures).
- Controlling redundant nodes (computers). This is necessary when there is identical software running. For this final or intermediate results have to be synchronized. Common clock, common bus cycle or, at software level, result synchronization can be used [2].
- Controlling redundant devices: device replication based on RS at the moments of performing operations over devices. This is the method underlying, among other things, shadow disk construction.
- Controlling redundant plant inputs/outputs: this can be done by synchronizing input from and output to plants (e.g. with voting). For continuous data it may prove necessary to average them immediately after synchronization.
- Controlling redundant files: those are always synchronized at the moments of performing operations over them. This is the basis for constructing shadow files.
- Controlling redundant timers: the problem is similar to the abovementioned problem of matching input data, but is complicated by timers working in different nodes and by it being necessary to have global time for DOS and application software purposes [4].

Thus, RS is a basis for controlling DOS resources of a variety of types. Note the range of synchronization for the components we already mentioned: from step-by-step synchronization (micro instructions, cycles of retrieving data from memory, timer ticks, bus access [2]) to software (at the topmost level).

4. Redundant Software Synchronization
Controlling redundant software is, actually, a service facility for introducing software FT. We shall consider those software FT schemes only in which several software variants are executed concurrently. This is our reason for not taking into consideration recovery block - RB[5] or exception handling[6] in which several
software variants are executed sequentially and no need arises for synchronization. Unlike RB and exception handling, N-version programming (NVP) [7] implies synchronous execution of software variants and matching the results of their operation.

In the same way RS is achieved in a RB modification - distributed repairable RB [8] in which different variants run in different nodes and their operation is synchronized at the moments of executing the acceptance test.

The kinds of RS described in this section are essentially identical with synchronization in process replication, but here it is to be performed after obtaining the results only (to match them) and there can be neither intermediate synchronization nor any harder types of lower level one.

5. Synchronization System Analysis
Let us introduce the notion of synchronization point (SP) to denote the moment of performing RS in components. RS is necessary for the following reasons:
- to eliminate the time shift in the operation of components;
- to perform error detection based on RS since we are able to match their states when establishing SPs only;
- for recovery (after an error has been detected).

The point essential for RS is the interaction of the given component with its environment. It is at these moments that SPs are more often established. In practice one has to make decisions as to what is to be considered environment: RS can be performed every time data are retrieved from memory, input or output of the plant, procedures are called, etc. Besides, internal synchronization is sometimes used, e.g. for matching the intermediate results of component operation.

All actions to provide FT are carried out at those moments only when components are synchronized, i.e. are in a known state.

RS has been successful if a sufficient number of elements are in the working state and have signalled this within the acceptable time limit. In those cases when RS (and recovery) can not be done at this level (there can be several reasons for that: the main or redundant element being faulty, there being no signal from them within the acceptable time limit, etc.), one can has to use redundant resources of higher level. For instance, if both disk drivers (in a shadow disk system) of a computer fail, one should use a redundant computer, whose operation is to be synchronous with that of the first computer which will be regarded to be out of working state.

It is for describing this structure of redundancy in a system that HSS is intended for and controlling this redundancy is based on it.

The choice between software or hardware synchronization as more adequate for a particular component depends both on its type and on the nature of application. Thus, it is known, for instance, that purely software synchronization (as in SIFT [9]) is not applicable for hard real time flight control system.

6. Synchronization System Unification
In HSS there is a RS server (RSS) to provide joint operation for every group of components. Each component is to establish a SP for sy-
Synchronization: SetSP (Name, Result), where Name is the unique name of SP, Result (optional) is the information required for further operation, e.g. output data to be averaged or operation results to be voted on.

RSS receives information from all functioning components. In its operation two built-in parameters are employed: Number - the minimum number of components required to continue operation; Time - the timeout depending on the time shift that can appear in the operation of components and on the times of message transmission between them. Besides, RSS employs one of the schemes to parry the fault and may need the Result information to do this. Having taken a decision, RSS either sends out all working components an command to continue or stops (which means stopping all of its components). Thus, RSS and operation SetSP functioning in this way will provide the fail-stop property: if a system does not have the necessary number of working components it stops, and if it does they go on operating synchronously. Thus, within this approach fail-stop is considered the only scheme for spreading the fault through levels. A similar scheme is followed by most known FT synchronization systems.

We have considered just those functions of HSS that are concerned with masking and voting. HSS is used in a similar way to arrange rollback, reconfiguration, warmstarting, migration and degradation.

To determine where SPs are to be established, one should take into account the acceptable time shift, application area requirements, times of components exchanging messages, etc.

To execute the protocol (operation) of establishing SPs at a purely software level, one generally has to execute the Reaching Agreement protocol [10]. In practice, a high degree of reliability can be ensured by executing simpler algorithms; besides, employing special equipment can make them simpler still (shared memory; hardware comparator which runs every time a record is made in a special memory field [11], etc.) The two-phase commit protocol [12] used to modify replicated data and DBS provides software synchronization similar to the one we have described.

Besides, synchronization algorithms differ significantly depending on whether in a group of resources one of them is chosen as the master one or all of them have equal rights; whether RSS actually exists or its functions are distributed; what is the acceptable time shift and response time for the application system on the whole. A number of researchers have been concerned with developing synchronization algorithms for a long time and have proposed numerous algorithms of various characteristics and application areas [1-4, 8-12]. HSS can be part of DOS or network, a structure built over the existing DOS or over several one-computer OS(s).

7. Software Engineering for FT System Design
Let us describe the order of actions for providing FT by creating HSS:
- After making a list of most undesirable and most probable faults one should determine the components that are liable to fail in this way and can be made redundant to parry those faults. It is these elements that generally form the required HSS.
- Then the points and the parameters of these component operation synchronization are to be determined.
A recovery scheme (RSS operation algorithm) should be chosen for every group of components as soon as they have been successfully synchronized. To do this, Result parameter is used and additional service information exchange among components may be required.

After successful external synchronization it may prove necessary to employ special techniques of smoothing, averaging or filtering the application information. Unlike the rest of HSS elements, these procedures are to be developed by application programmers.

8. Summary

In the present paper a general unified approach to create FT systems that are based on using redundant components is proposed. A considerable part of FT schemes is shown to be constructed precisely in this way. We have singled out the basic facilities that all systems like this can be based on, and have proposed a way of arranging them in the form of HSS. Thus, HSS serves as a frame for FT design. This was the basis for our description of engineering rules to unify FT system development.

It seems to us that though a large part of papers on FT is implicitly concerned with synchronization problems, still, synchronization has not yet been adequately discussed as a concept, a framework and a potential general-purpose structurization facility.

The author has contributed to developing several HSSs for FT systems to be considered in the report: shadow disk system (disk operation synchronization); tightly-coupled system with bus switch and dual port disk (disk access and dialog synchronization); workby PC-based navigation system (dialog, task, input/output synchronization); workby multi-I80186 system (specialized hardware [2,11], which allows to change the synchronization level - the HSS structure - dynamically; OS, timer, task, bus access, input/output, interrupt, memory access synchronization).

References: