√

MASTER

PREPRINT UCRL- 82958

CONF- 791102 - - 35

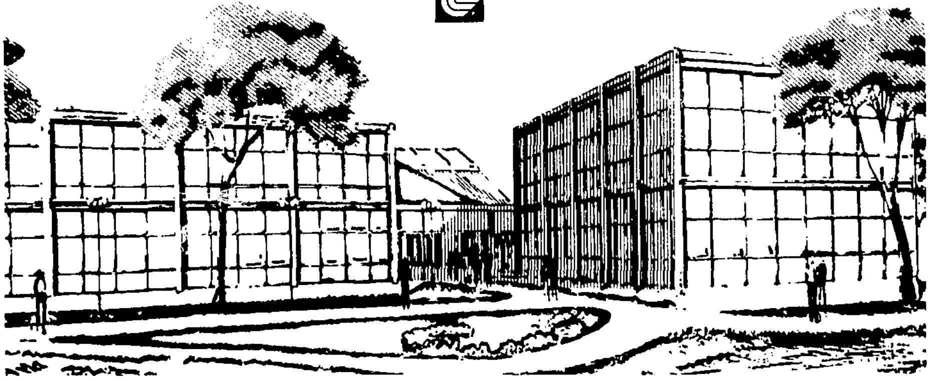
Lawrence Livermore Laboratory

SUPERVISORY CONTROL AND DIAGNOSTICS SYSTEM DISTRIBUTED OPERATING SYSTEM

P. R. McGoldrick

November 8, 1979

This paper was prepared for submittal to the 8th Symposium on Engineering Problems of Fusion Research, San Francisco, CA, November 13-16, 1979



SUPERVISORY CONTROL AND DIAGNOSTICS SYSTEM DISTRIBUTED OPERATING SYSTEM*

MASTER

P. R. McGoldrick
Lawrence Livermore Laboratory, University of California
Livermore, CA 94550

SUMMARY

This paper contains a description of the Supervisory Control and Diagnostics System (SCDS) Distributed Operating System. The SCDS consists of nine 32-bit minicomputers with shared memory (Fig. 1). The system's main purpose is to control a large Mirror Fusion Test Facility (MFTF). The facility is so large, containing over 3000 devices to control and 7000 sensors to monitor, that it is not cost effective (or possible) to have only manual control of the facility. This type control system places certain requirements on the system design.

- Availability: The control system must be available. Our design criteria are such that downtime for a single-point failure should be no more than 5 min.
- Flexibility: The devices that the MFTF experiment controls or monitors may change regularly. Also, new devices may be added (possibly requiring additional computers, memory, data storage devices, or interfaces for control.)

Data Throughput: SCDS may have to collect, process, and display in meaningful form as much as four million bytes of information every 5 min.

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

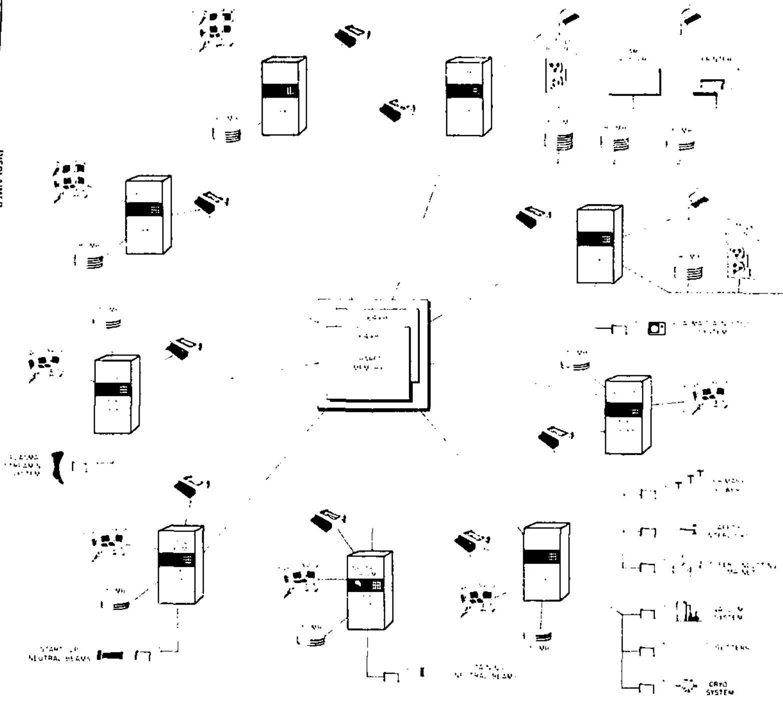


Fig. 1. MFTF control and diagnostics system.

The availability requirement has been the major factor in our design of the SCDS. Fortunately, our design requirements are such that we can allow the entire SCDS to crash on a single-point failure as long as we recover in 5 min. In our design we attempt to prevent crashes of this type, but preventing certain hardware failures from crashing the SCDS is virtually impossible.

The SCDS Architecture

Nine computers comprise the SCDS (as seen in Fig. 1), and each has its own local memory while each shares some common memory. To keep in line with our availability requirement, the shared memory consists of two separate units. If one shared memory fails, the other is usable. Notice that the nine computers divide up the MFTF workload. The division was made so that each computer could do its function with minimal interaction with the others.

With shared memory it is important for efficiency to minimize access to shared memory. The memory cycles in shared memory are divided among competing machines. It is entirely possible for a machine to be locked out of shared memory for eight memory cycles (about 11 μs). We therefore have each computer perform as much as possible locally.

To assure availability after single-point failures, each major component in the system has a backup component. One approach would have been to have a spare control system that switched in whenever there was a failure, but this is cost-prohibitive. Instead we designate a backup machine for each machine in the system, which does its primary function and the backup function when necessary. Devices necessary for the backup function can be switched to the backup machine via bus switches. This architecture allows multiple failures provided there is no more than one failure per machine and its backup. When we are in the backup mode, performance of SCDS is degraded.

The bus switches are potential single points of failure that are not backed up. In our investigation these units are very reliable. When a bus switch fails, we will replace it. This operation will take from a few minutes to an hour. There is other common hardware that is critical. We have attempted to minimize these; but without redesign of the manufacturer's hardware, they cannot be entirely eliminated.

Distributed Operating System

We designed a Distributed Operating System to fulfill the requirements of SCDS. Managing mulitple processors is not different from managing a single processor.[1] The technology for designing and implementing multiprocessor operating systems exists,[2] so that the SCDS system need not be developed in an ad hoc manner. Brinch Hansen defines an operating system to be a set of manual and automatic procedures that enables a group of people to share a computer installation effectively.[2-4] Here the key work is sharing. We must manage access to shared memory and other shared resources. Rather than concentrating our design on shared memory and interprocessor communication, we designed a virtual machine that processes could use regardless of the need for shared memory. In fact, a process should not have to know whether it is running in a one machine environment or nine. This allows us greater flexibility in deciding where processes run.

The main issues in developing the SCDS Distributed Operating System are:

- Maintaining availability.
- Store management.
- Process control.
- Mutual exclusion.
- Error handling.

Logical Machines and Availability

To simplify our thinking about the processes that could be operating anywhere in the SCDS, we developed the concept of logical machines, which is a logical grouping of tasks that work together to perform a desired result. For example, the tasks that control the vacuum system in the MFTF were grouped to form a logical machine. A key element of a logical machine is that it has a higher degree of interactivity or utilization of a certain set of resources between its members than its members have with other logical machine members.

The rules governing logical machines follow:

- All members of a logical machine run on a single physical machine (i.e., logical machines are mapped onto physical machines).
- A physical machine may run as many logical machines as it is capable.
- Logical machines may be moved from one physical machine to another.

We accomplish the logical-to-physical machine mapping by a logical-to-physical machine table in shared memory. Moving a logical machine from one physical machine to another is our method of maintaining machine availability. We can remove or add physical machines to the SCDS by moving logical machines from them or to them.

Tasks that form a logical machine use a checkpoint system. Whenever a task passes a checkpoint, it records the fact in shared memory. The checkpoints can be thought of as milestones. If a physical machine fails, the network operator can reassign the logical machines that had been assigned to the failed physical machine to one that is operational--the backup machine. When the logical machine is then restarted, the tasks that form the logical machine can determine where they should restart by examining the checkpoint status in shared memory, thus allowing a smooth switchover.

Store Management

Each machine has its own local memory while sharing a common memory. We could set aside a portion of local memory for members of a logical machine to share. When logical machines need to share data, shared memory will be used. In this manner we cut down access to shared memory.

In these common memories we are only sharing data, not programs. If we placed programs in shared memory, we believe it would waste memory cycles on instruction fetches. Programs could be placed more efficiently in local memory and only access common memory when using data. The terms that we will use for these common memories are:

Local common: Memory that can only be accessed by a single processor but can be accessed by mulitple tasks on that processor.

Global common or Shared memory: memory that is shared among processors.

One should realize that any information stored in local common is lost when a logical machine is moved from one processor to another.

We could preallocate all buffers and data areas in the above commons, but the commons are not big enough to hold everything and certainly would not easily allow change. We decided to use dynamic memory allocation/deallocation. Each data area is allocated based on need. Our algorithm allows only one process to be allocating or releasing memory in an area at a time, so the dynamic buffer area has its access controlled by a semaphore. If we only have one dynamic buffer area in shared memory that all machines use, only one machine can use it (for allocation or release) at a time.

A good solution to the above inefficiency is to have multiple dynamic buffer areas, preferably one per logical machine. Each logical machine is assigned a dynamic buffer area where it can obtain buffers. If each logical machine has its own area, it will never have to wait for access. This accomplishes another very important step toward availability. By assigning dynamic buffer areas, we limit the amount of shared memory that each logical machine controls. An aberrant logical machine, generally, can at most fill up its area (hardware failures notwithstanding).

Shared memory is initialized, and can be modified, by a program called NETCTRL. Local Common is automatically initialized upon boot-up of a computer by a program called LMINIT.

Another important issue is data integrity in common memories. How does one prevent programs with anomalies from contaminating shared memory? Our solution is to only give a user program direct Read-Only access to a common memory. If a program wishes to Write, it must use procedures that are part of the Distributed Operating System. (These procedures are described in SCDS Software System Manual, Section 6.1.) These system procedures perform certain consistency checks to verify correct operation.

MAIL

Buffers that are allocated in a dynamic buffer area in common memory can be passed around between processes. Therefore, we call any buffer in common memory MAIL, which is used to implement mutual exclusion, process control, and communication of data via three mail types: semaphore, command, and data.

When each piece of mail is made, the system encases it in an envelope called a header. The header contains valuable information that is used by the distributed system such as:

- Postal zone: Local or global. Local mail is in local common and can only be sent between processes on the same logical machine. Global mail is in shared memory and can be sent to anyone at any location.
- DBA: Pointer to the dynamic buffer area the mail is in.
- Parent, younger-sibling, elder-sibling: These fields allow mail to be created in a hierarchy. This aids in debugging because we can obtain a snapshot of the system at anytime. Other benefits of placing mail in a hierachy will be discussed later.

Process Control

Every process in our distributed system has one or more activation records in common memory. These activation records are command pieces of mail. Each command piece of mail contains the logical machine, taskname, and load file of the process the command activates. A process is said to own a command if the logical machine and taskname in the command are the same as the process'. Processes are created by mailing a command piece of mail to the process to be created.

The postal system requires the process that is mailing the command to specify the logical machine, taskname, and load file of the process that is to receive the command. If the intended recipient of the command is not already running (for another command), the postal system loads and starts the process from the load file. Mailing a command results in the command being owned by the recipient. When a process mails all its commands to others, it is no longer thought to be part of the distributed system and is not allowed to make or send mail. Note that in order to make mail, a process must own a command piece of mail. This prevents unauthorized users from tampering with the system.

Mutual Exclusion

There are times when it is safe for only one process to manipulate a shared resource at a time. An example is allocating or deallocating buffers in shared memory. A resource may also be able to be shared simultaneously by members of one set of processes (readers), but must be used by only one process at a time by members of another set of processes (writers).

All shared resources that require exclusive access hav piece of mail of type semaphore. We have written three routines to control access via a semaphore: PS (P sequential), PC (P concurrent), and V.

Executing a PS operation on a semaphore suspends a process until it is the process' turn to have exclusive access to the resource. Executing a PC operation on a semaphore suspends the process until the resource is not being used by a process that executed a PS operation. Executing a V operation will release the process' access to the shared resource.

Error Handling

Our very strict rules on making and posting mail tends to show up errors before they impede the system.

A command piece of mail has two fields that point to mail a process owning the command made in local or global common. This allows a hierarchy of commands to be made that correspond to processes running in the distributed system.

If a process that owns a command blows up, pauses or goes End-of-Task in a suspicious manner, the parent of the process is notified by returning the command to the parent with the appropriate error status.

Each processor sets an "I am alive" flag in shared memory once a second. When a processor failure is detected, the logical machines that were running on the failed processor are restarted on another processor.

References

1. Price, R. J., Multiprocessing Made Easy, National Computer Conference, 1978.

2. Brinch Hansen P., Operating Systems Principles, Prentice Hall, 1973.

3. Brinch Hansen P., "The Programming Language Concurrent Pascal," IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975, pp. 199-207.

4. Brinch Hansen P., Concurrent Pascal Introduction, Information Science, California Institute of Technology, July 1975.

5. Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557.

6. Wallentine, V. and McBride, R., Concurrent Pascal--A Tutorial, Department of Computer Science, Kansas State University, November 1976.