Distributed File Management and Job Management of Network-Oriented Operating System

HIDEHIKO TANAKA* and TOHORU MOTO-OKA*

The concept of a network-oriented operating system (NOS) is based on the message-oriented, interprocess communication facility, which can be used independently of the process locations. A distributed file management system and a distributed job management system are implemented on the experimental computer network, TECNET, as a set of system processes on the system nucleus. In this paper, we discuss the structure of these management systems and show the implementation results. We also discuss the enhancement of system reliability against system failure.

1. Introduction

Since the success of the ARPA network project, many computer networks have been developed. In the past 10 years, the basic concepts of networking were developed, such as interprocess communication beyond machine boundary, communication protocol, layered structure of network architecture, and virtual devices such as network virtual terminals. Presently, almost all computer manufacture support network applications through such network architecture products as SNA. The internetworking of existing computer networks is going to be an indispensable technology.

Our research of computer network began in 1972 [1, 3]. The goals of this project were: to define the basic operating system structure that is appropriate for many kinds of network applications; to develop a model for a network utility manager regarding file and job management; and to test the feasibility of a network oriented operating system (NOS) through real implementation.

We began our project by making an experimental computer network testbed (TECNET) in our laboratory, using a few minicomputers and medium-scale/large-scale computers. This operating system consists of a system nucleus and many processes. The system nucleus provides such facilities as process creation, multiprogramming, interprocess communication, and interrupt handling. Next, we designed and implemented a distributed file access mechanism and a distributed job management system on the system nucleus. In 1979, we finished the implementation of an extended version of a file management system that supports the multiplicated file handling feature to enforce system reliability.

Because the location of each process is transparent to the user of the NOS interprocess communication facility, some management or application processes can be implemented easily. Although all current network architectures are based on the connection-oriented interprocess communication, our interprocess communication facility is message-oriented, and each message transfer is controlled directly by SEND/RECEIVE primitives. This characteristic is especially effective for realizing more tightly coupled control systems for network applications or some transaction-oriented applications. The file management system of NOS enables users to access remote files as easily as local files. When allocating file resources, this manager guarantees deadlock-free allocation of network wide by following a prefixed file request cycle. NOS jobs consist of distributed process sets, which are controlled by the distributed job management system. We developed a high-level system description language to implement NOS. With this language, a programmer can directly use a system uncleus service, such as interprocess communication.

2. System Architecture of NOS

2.1 TECNET

Figure 1 shows the hardware structure of the experimental computer network, TECNET, as it existed in

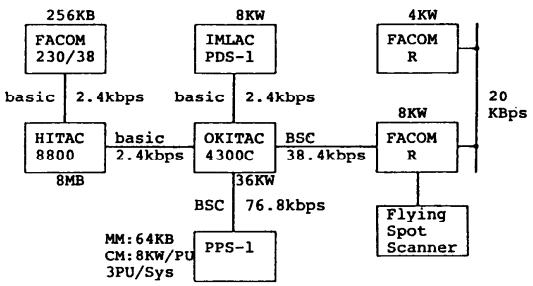


Fig. 1 Structure of the computer network testbed, TECNET.

*Department of Electrical Engineering, University of Tokyo 113, Japan.

March 1980. The data link control procedure is binary synchronous communication procedure of code transparent mode by DLE-doubling. Communication control units consist of TTL ICs and handle framing, CRC code generation/checking, and DLE handling by hardware.

2.2 Structure of NOS [3]

The structure of NOS is shown in Fig. 2. We assume that all element computers in the network have the four-layer operating system consisting of: system nucleus, distributed file management system, distributed job management system, and job proccesses. The functions of the system nucleus are process scheduling, multiprogramming, and primitive operations support, such as interprocess communication facility, interrupt handling, and virtual memory control. All other functions are implemented by many kinds of processes that communicate with each other through primitive operations for interprocess communications, such as SEND msg/RECEIVE msg. These operations activate explicit message transfer. Memory sharing among processes is not permitted in NOS, even if the processes are in the same computer. Although the interprocess communication primitives issued by users are checked for their capability, those issued by system processes, such as device processes and distributed file and job management processes, are not checked.

3. File Management

3.1 File Management Structure

The objectives of the NOS file management system are:

1. Easy access to distributed files
2. Fully distributed control
3. Built-in countermeasures against deadlock
4. Extension to distributed DBMS
5. Multiple copy support

The structure of the NOS file system is shown in Fig. 3.

The user requests a file through an access interface (FACI). When the file access command is 'OPEN', the local File Main Manager (FMM) communicates with a remote FMM and locates the Access Process that controls the physical access to the required file. FACI keeps the process number of the access process and transmits

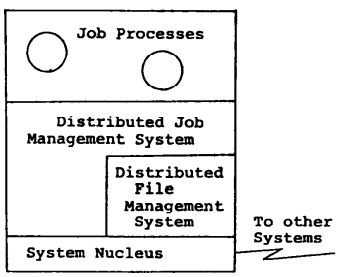


Fig. 2 NOS system structure.

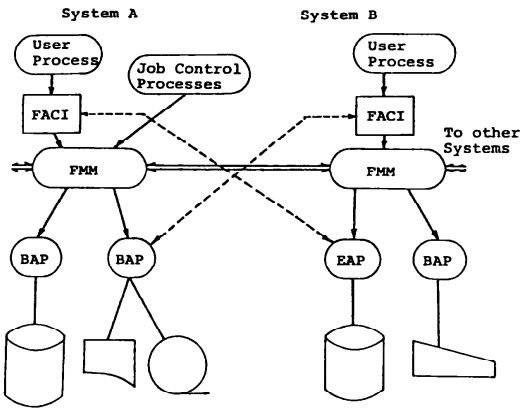


Fig. 3 Structure of NOS distributed file management system.

file access commands, such as 'READ' or 'WRITE', when they are issued by the user process. The processing of 'CLOSE' is via the FMMs, just as 'OPEN'. There are two kinds of access processes—Basic Access Process (BAP) and Extended Access Process (EAP). BAPs support conventional file access, such as disk read/write, line-printer output, and card-reader input. EAPs support higher level file access, such as a search operation for data that satisfies some condition. EAPs process data where it exists, thus minimizing the volume of transmitted data and making remote file access efficient.

3.2 Distributed File System

There are several kinds of directory organization schemes for distributed files. One involves placing a central directory that contains directories of all files in the network. Another scheme is to query every computer about the existence of the files. A third scheme is to make every user designate the system name in which the file is stored.

NOS directory organization is different from all these scheme. We assume that every user has his home site where the main directory of his files is placed. When a user issues a file access command, he specifies a file name that he assigned. Using this name as a key, the system searches a User File Name Table (UFNT) until the owner name, the file name assigned by the owner, and the home site of the owner are found. The physical location and disk address of the file can be obtained by searching an Owner File Name Table (OFNT) at the system pointed to by the home address of the owner. The access control field in OFNT is used to check the file-access authorization. When a user wants to enter a shared file into his UFNT, he must get the owner's name and the file name assigned by the owner. We assume the user does this by referencing data dictionaries or printed manuals.

An example of file access is shown in Fig. 4. In this example, user 'a' accesses a file 'ufn' that is owned by 'b'. The owner's file name is 'nnn', and the owner's home system is 'Y'.

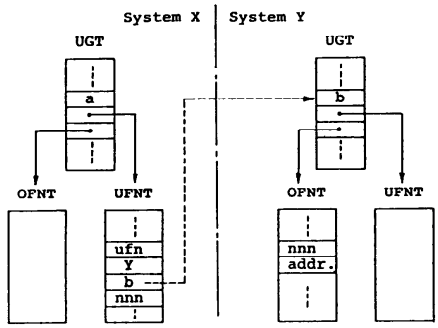


Fig. 4 Example of remote file access.

• Deadlock Handling

When a user is going to use several shared files at a time, he faces a deadlock problem. There are 2 ways to cope with this problem. One way is to assign at one time, all the resources required with deadlock free. Another way is to assign required resources dynamically, and to do backout when deadlock is detected. We used the former scheme. In our network, TECNET, all files stored are numbered sequentially. Prior to the start of processing, the user requests all files needed and File Main Manager creates a file request command with a list of required file names. This command is circulated along a fixed loop of systems. Each FMM checks the availability of requested files stored in that system, and forwards the command to the next computer system. As the result of these transfers, the file request is granted only if all checks of files passed successfully. Otherwise, the request fails and a failure message is returned to the user. Fig. 5 shows an example of the request circulation. Generally, any system can be skipped if it has no files named in the list.

• File Management Protocol

We provided several commands that are standard in the network, as shown in Table 1. These commands are issued by FACI when the user process uses this interface. GETFN is used to get a physical file location, owner name, and owner file name, and is sent to the user's home FMM. OBTAIN is a command to get the file access right of all files to be used. FREE is used to disable the file access right. CATALOG is used to catalog a specified file into his UFNT with a name that he assigned. PERMIT is used to permit sharing of his file with other users. All the commands are accompanied by their corresponding reply messages.

The basic access methods are sequential-access and

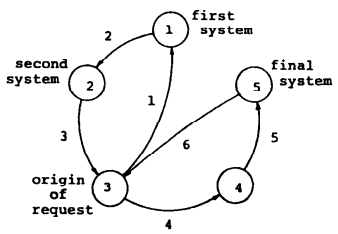


Fig. 5 Example of file request traveling.

Table 1 Distributed file commands.

command name	destination
GETFN	Home FMM
OBTAIN	FMMs
FREE	FMMs
OPEN	FMM(s)
CLOSE	FMM(s)
CATALOG	Home FMM
UNCATALOG	Home FMM
PERMIT	FMM
INHIBIT	FMM
READ	BAP/EAP
WRITE	BAP/EAP

direct-access. At present, the Virtual Sequential Access Method is supported on PPS-1, but this can be only used locally.

The open modes are Input, Output, and Update.

3.3 Multiple Copy Feature

The reliability of the file system can be improved by having copies of files. We can then access one of the copies of a file that cannot be accessed because of system failure. In the network environment, another benefit of having copies is the communication costs decrease, because the effective distance to the file gets shorter when copies are placed at several sites. However, the existence of copies should be transparent from the user's point of view. That is, the contents of all copies should be consistent, even if many users try to update the file concurrently.

In the following discussion, we assumed two kinds of files: logical file and physical file. Logical file is the file from the user's point of view and is independent of the existence of copies. The name of the physical file is used to indicate each copy of the file.

• Distributed File Directory with Copy Feature

The OFNT used priviously is broken into a new OFNT and several Physical File Node Tables. The OFNT has all the location information for the copies. When a user wants to use a logical file, he selects one of the physical files, depending on the location of the user process and the availability of each copy. This selection algorithm plays an important rule in economizing the distributed access when some query decomposition of high-level DML is concerned. Accordingly, we provided a way of getting all the location information for copies at the time of the access-right request (OBTAIN). This method will be useful for implementing distributed DBMS. PFNT maps an owner file name to the location information for a physical file (disk name, address, etc.). This table is placed at the site where the physical copy is stored.

• Distributed File Protocol with Copy Feature

When OBTAIN is used, the location information for all copies is returned with the reply message while the file access-right check is performed. A new command, CREATE, creates a physical file and its PFNT. To associate the PFNT with an OFNT, the OBTAIN command

is used with a OFNT registration flag set after the CREATE command. To add a copy of a logical file, the command, ADDCOPY, is used to make up a physical copy and registrate it into OFNT. The command, DLTCOPY, deletes a copy. To get the contents of UFNT, OFNT, and PFNT at some FMM, the DISP command is sent to the FMM. CREATE, DELETE, ADDCOPY, DLTCOPY, and DISP cammands are added, and OBTAIN/FREE commands are modified to facilitate the copy feature.

• Procedure of File Creation

When a user creates a logical file with a few copies placed at several sites, he does it as follows. (In the following discussion, "user" means some system processes, such as job manager in the real sense.)

1. The user sends a CREATE command to the FMM of a site where he is going to place a physical file.
2. The FMM makes BAP allocate space to store the file, registrates the physical address in the PFNT, assigns a physical file identification number, and sends the user a responce with the identification number.
3. The user makes as many file spaces as he wants, at several sites, by following the proceding procedure.
4. The user sends an OBTAIN command with its OFNT registration flag set to the FMM of the system where he is going to place the OFNT. This command is followed by parameters, such as site addresses and physical file identification numbers, for all copies.
5. The FMM that received the OBTAIN command registrates the logical file into the OFNT, gets the access-right, and returns a responce to the user.
6. The user prepares data and writes it into the logical file, using WRITE commands. For each WRITE command, data is transferred toward each physical file space separately.
7. The user sends a FREE command and releases the access-right.
8. After receiving the responce for the FREE command, the user sends a CATALOG command to the FMM at his home site. The FMM registrates the file in the UFNT with its user's file name set to the same name as the owner's file name.

4. Job Management

4.1 Network Job Management Structure

In the network environment, a job is defined as a set of processes that are located at several computer systems and that communicate with each other to perform a function for the user. Accordingly, the job management system in the network environment should manage jobs across the computer boundary. The architecture of the network job management system of NOS is shown in Fig. 6. Each job is managed by a Master Job Management Process (MJMP) and several Local Job Management Processes (LJMPs), which are located at each computer system in which some job processes of the job are running. The MJMP manages all LJMPs and is the center of the job management. The hierarchy structure of MJMP and LJMPs are made up at job creation time for each fob with the help of a logger process. We designed a network job control language to define these network job structures.

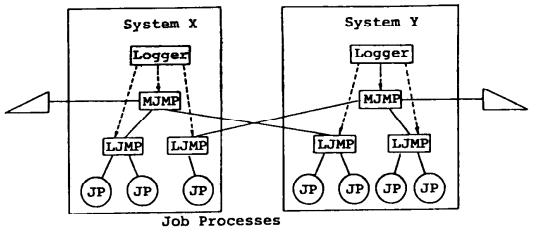


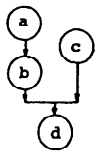
Fig. 6 Structure of distributed job management system.

4.2 Network Job Control Language

We provided a set of job control statements that are enough to realize the arbitrary structure of a network job on the computer network. The JCL is shown in Fig. 7. The JOB statement defines a job for which the home site for file management is specified by ⟨sitename⟩. When the user pushes an attention key, a logger is connected to the user keyboard. The MJMP for the user is created at the log-in site when the user keys in a JOB statement. The FD statement defines files that are needed for the job. ⟨filedefname⟩ defines a name used in the user program. In ⟨unitstatmnt⟩, the description 'UNT1 = ⟨deviceinf⟩, UNT2 = ⟨deviceinf⟩' is used when the user wants to create a logical file with two physical files, UNT+ is used when the user wants to add one more copy, and UNT− is used to delete a physical file. The EXEC statement defines a program to be executed as a process specified by ⟨processname⟩. This statement

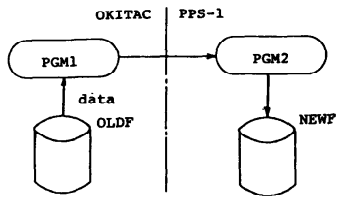
```
$JOB  username  [@<sitename>]
$FD [<filedefname>] [,FN=<flname>] [,<unitstatmnt>]
    [,SP=<space>] [,DISP=<disposition>] [,FCB=<fcbparam>]
$EXEC [<deviceinf>] [,<flname>] [<processname>)
      [@<sitename>] [<execorder>]
$STEP [<filewaitmode>]
$CAT <userfilename>,<ownerfilename>.<ownername>@<sitename>
$UNCAT <userfilename>
$PERMIT <filename>, [<username>], [<accessmode>]
$INHIBIT <filename>, [<username>]
$QUIT <pnamelist>
$ABORT <pnamelist>
$ALLOCATE <memorysize> [@<sitename>]
$END
where, <flname>::=[%<ownername>.]<filename>|+
      <unitstatmnt>::=UNT1=<diviceinf>[,UNT2=<deviceinf>
          [,UNT3=···]]|UNIT=<deviceinf>|UNT+=<deviceinf>
          |UNT-=<deviceinf>
      <disposition>::=(<createmode>,<keepmode>)
      <keepmode>::=KEEP|DEL
      <fcbparam>::=([<fileorganization>], [<recordform>],
          [<blocklength>], [<recordlength>]
      <fileorganization>::=PS|IS|DA
      <deviceinf>::=[@<sitename>.]<devicename>[#<device-
          number>] [.<volumenumber>[#<filesequencenumber>]]
      <devicename>::=DK|MT|CR|LP|TW|TR|DP
      <execorder>::= |/[<pnamelist>]
      <pnamelist>::=<processname>|<processname>,<pnamelist>
      <filewaitmode>::=WAIT|NOWAIT
```

Fig. 7 Network job control language syntax.




```
$EXEC  fa, a, @PPS
$EXEC  fb, b, @PPS/a
$EXEC  fc, c, @OKI/
$EXEC  fd, d, @PPS/b,c
```

Fig. 8 Example of process execution order control.




```
$JOB  USR1@OKI
$ALLOCATE  CORE=10@OKI
$ALLOCATE  PRCS=01@PPS
$FD   OLDF, FN=MSF1, DISP=(OLD, DEL)
$EXEC , PLIB(PGM1)@OKI
$FD   NEWF, FN=MSF2, DISP=(NEW, CAT),
      UNIT=@PPS.DK.SYS, SP=10
$EXEC , ELIB(PGM2)@PPS/
$STEP
$END
```

Fig. 9 Example of network job.

also can define the order in which job processes are to be executed, so that this process is activated after processes specified by ⟨pnamelist⟩ are executed. The STEP statement defines the end of a job step. When the MJMP interprets this statement, it gets all the files needed and activates the job step. CAT and UNCAT are used to catalog or uncatalog a file. PERMIT and INHIBIT are used to permit or inhibit shared access by users defined by ⟨username⟩. QUIT interrupts the processes defined by ⟨pnamelist⟩. ABORT stops the execution of processes specified by ⟨pnamelist⟩. The ALLOCATE statement allocates an address space for the job and creates an LJMP at site ⟨sitename⟩. END defines the end of job. Fig. 8 shows an example of controlling the execution order of processes. Fig. 9 shows a very simple example of a job that reads file data from an OLDF at site OKITAC and writes the processing results into NEWF at site PPS.

4.3 Distributed Job Management Protocol

The communications among MJMP, LJMP, and the logger define the distributed job management protocol. We provided 12 basic commands, as shown in Table 2. These commands are followed by their appropriate responses. When a job is started, MJMP sends a 'request to start job-process' command to LJMP and indirectly controls the process. LJMPs monitor job processes that are local to the site, provide management information reports, such as accounting and termination status of each process, and send them to MJMP by a 'report of job-process end' command. The division of job management work between MJMP and LJMPs is made so that local management tasks are dedicated to LJMPs, and others that are common to all systems are left to MJMP.

Table 2 Distributed job control commands.

command name	source	destination
set MJMP initial data	Logger	MJMP
request to create LJMP	MJMP	Logger
request to remove LJMP	MJMP	Logger
request to remove MJMP	MJMP	Logger
set LJMP initial data	MJMP	LJMP
request to create jobprocess	MJMP	LJMP
request to start jobprocess	MJMP	LJMP
request to stop jobprocess	MJMP	LJMP
request to remove jobprocess	MJMP	LJMP
request to abort jobprocess	MJMP	LJMP
request to quit jobprocess	MJMP	LJMP
report of jobprocess end	LJMP	MJMP

Accordingly, this protocol is independent of specific implementation.

5. Reliability Consideration

5.1 Principles

Up to this time, reliability technologies of computer systems have been applied only to a single computer system. But, in a network environment, each operating system should take into account the failure of other systems, and should be more secure against many kinds of disturbances. Our principles for these problems is to preserve the autonomy of each system and to manage failure with distributed facilities.

To realize these principles, the basic structure of NOS is based on the interprocess communication of message oriented. Processes cannot send or receive data without the permission of others. Accordingly, a failure of one process is well shielded from the others. This interprocess communication facility is the only way of intersystem interaction, so the autonomy of each system is preserved. The process name registration scheme is to make the interprocess communications secure statically, and efficient in terms of communication overhead. This scheme is also substantial for keeping independence between programs and processes.

5.2 Reliability Enhancement of Distributed File Management System

We introduced a multiple copy feature to make the distributed file usage invulnerable. But, other enhancements should be implemented to make the distributed file management system itself more reliable. These enhancements include:

1. Distributed file directory enhancement
2. Consistent update mechanism

In case a system failure occurs when the user is reading a file, the access history of the file should be kept in the file-using system in order to change the physical file dynamically, though the location information of alternative files can be gotten easily, because it was set in the local area of FACI when the OBTAIN command was used.

In a multiplicated-files environment, a network or computer system failure can cause inconsistency of file content. One inconsistency is internal to the file; that is, incorrectness of each file's contents. Another inconsistency is external, which means there are discrepancies in the contents of copies. Because these inconsistencies occur when the file is updated, some corporation rule among copies is needed to preserve consistency. For example, we can decide whether the update transaction can be forwarded or not depending on the condition:

1. Whether the user can communicate with some specified site.
2. Whether user can communicate with more than half sites which store the physical files.

In any case, if the conditions are not met, the transaction is aborted. Each update transaction is assigned, at file-request stage (OBTAIN), a sequence number that identifies uniquely the processing order of transactions for the file. The update operation should be done for all copies, though some systems that store the copy may have failed, and the update operation cannot always be performed at exactly the same time. Accordingly, the transaction and the sequence number are kept at some sites, and can be forwarded to the failed systems as soon as the systems recover. To ensure that files are updated consistently, even when failures occur, we should provide mechanisms, such as the "Two Phase Commit" procedure developed by Lindsay of IBM [5].

5.3 Failure Handling of Distribured Job Management System

Failures can be classified as follows:

1. Internal Failure of a Job Process

The job process can't proceed because of a programming error, resource shortage, etc. In this case, the termination notice is sent to the LJMP. When the job process enters an infinite loop, the timeout event is activated, and the LJMP forces the job process to stop. The LJMP sends the termination status to the MJMP. Depending on the status, the MJMP sends the 'STOP' command to all LJMPs, receives responses with status information, and terminates the job.

2. Computer Systems Failure

When a computer system that is processing a job fails because of hardware damage or the operating system being down, the MJMP or LJMPs detect the condition by the timeout because the MJMP and LJMPs exchange 'Are you ready?' and 'Yes I am' messages periodically. An LJMP failure, is detected by MJMP, which sends an 'ABORT' or a 'WAIT TILL NOTIFIED' command to all LJMPs. The WAIT TILL NOTIFIED command makes each LJMP save all the job processes, and the LJMP returns to MJMP a key in the response. The MJMP checks the recovery of the failed system and restores the job by the 'RESTORE' command when appropriate. RESORE is accompanied by the key and is sent to the loggers of all systems concerned.

When the system that has the MJMP fails, the corresponding LJMPs detect the failure. Each LJMP saves the final status information of job processes with the identifier of the MJMP. The resynchronization of processes is accomplished by message sequence number. When the failed system recovers, the latest message number that was used by the restored program to communicate with some other system (A, for example) can be defined from a journal tape. This message number is passed from the failed system to system (A), which resets the processing status to just before the beginning of the next communication statement. Both systems can then start processing arbitorary after that.

3. Network Failure

Handshaking between the MJMP and LJMP are interfered with in this failure. This condition detected by the MJMP and LJMP through the timeout event, or by a failure report from the communication control processor. The failure-handling procedure is the same as in failure 2.

6. Implementation

6.1 Languages

To implement NOS for the OKITAC 4300C, we developed an assembler language that is enhanced with macro instructions and modified to provide appropriate code for our program loader. This assembler is a cross assembler written in PASCAL on the HITAC 8800/8700 connected by a 2400-bps communication line. For PPS-1, we developed a micro assembler and used it to implement the OS nucleus and device control programs that are closely related to hardware. Other programs are implemented by a language called NPLAN, which is an ALGOL-like high-level language and is based on a language called PLAN that was developed by Takeichi [4]. The NPLAN compiler, written in PASCAL, produces V-codes, which are interpreted by a microprogram of PPS-1. We added a few primitive operations to PLAN so that NPLAN can be used as the basis to write programs in the network environment. Statements related to the primitive operations are compiled into V-code 'KCALL' (kernel call), the parameters of which include operation identifier, process name of opposite side, message identifier, buffer identifier, etc. The V-code interpreter, written in microcode, expands the KCALL to corresponding primitive operations written in microprogram. For example, a SEND statement can be written as:

```
Cstatus:=Send (Hostname, Processname, Waitmode,
              Message)
```

Hostname and Processname define the destination process, and Waitmode specifies whether this process (source) waits for the end of the send communication event or processes the next statements. 'Send' is an internally-defined function procedure. Cstatus, therefore is the resulting status of the Send execution. The V-code interpreter is a virtual stack machine that is provided for system portability. The size of this interpreter is about 1500 steps in the microprogram.

6.2 Implementation Results

Table 3 shows an implementation result of NOS. Implementation is performed for three machines: OKITAC 4300C, PPS-1, and FACOM R. A simplified version of the NOS nucleus is implemented on the FACOM R. The distributed file manager and job manager are implemented on the OKITAC and PPS-1. The HITAC 8800 is handled as a virtual file system through a process implemented on the OKITAC. Table 3 shows the result of two versions of the distributed file manager. The 'Single' version does not have the Copy feature; the 'Multi' version has the Copy feature at the PFNT level. Multiple UFNT and OFNT support, consistency maintenance, and job recovery feature are not yet supported. By implementing the Copy feature, the size of FMM increased 30%~40%, and that of FACI increased 70%, for the OKITAC, and 400% for the PPS-1. This 400% size increase includes the increase of code as a result of improving the user interface. The amount of code for the PPS-1 program is in terms of microcode or V-code (16 bit/step). The processing time of user file commands can be divided into two parts. One part is the time used in the FACI routine, and the other part is the time used in the FMM process. These times are shown in Tables 4 and 5, respectively. The time shown in Table 5 doesn't include the time needed for interprocess communication (about 2.0, 8.5, and 41 ms for PPS-local, OKITAC-local, and PPS-OKITAC communications). The difference in time values between the systems shown in Table 4 is due mainly to cycle-time difference (1:6 for PPS vs OKITAC), because the FACI routine of PPS is written as a microprogram. Because the FMM of PPS is written in NPLAN, the measured value of the PPS processing time shown in Table 5 is very high compared to that of the OKITAC, due to V-code interpretation. If the values are expressed in terms of processing steps, the difference between systems is fairly small, as shown in the tables.

Table 3 NOS element program size.

program element	OKITAC 4300C		PPS-1			
	single	multi	microprogram		V-code	
			single	multi	single	multi
nucleus	3700		3000	4000		
(comm.)	3400		1600	(system data)		
(others)	1600		400			
FMM	1900	3100			4500	6100
BAP	1550	4000	717	719		
FACI	900	1550	141	805		
MJMP	4000				5000	
LJMP	500				1300	
Logger	750		500			
Loader	1500		250			

Table 4 Processing steps and time for FACI.

routine name	no. of P files	PPS-1		OKITAC 4300C	
		steps	time ms	steps	time ms
OPEN	1	330	0.17	350	1.05
	2	480	0.24	560	1.68
CLOSE	1	110	0.06	90	0.27
	2	180	0.09	160	0.48
READ	1	70	0.04	70	0.21
	2	130	0.07	140	0.42
WRITE	1	70	0.04	90	0.27
	2	140	0.07	160	0.48
CHECK	1	70	0.04	90	0.27
	2	160	0.08	210	0.63

Table 5 FMM processing steps and time.

command	PPS-1		OKITAC 4300C	
	steps	time ms	steps	time ms
GETFN	520	8.8	480	1.4
OBTAIN (cat. in OFNT)	1350	23.0	1250	3.8
OBTAIN (without cat.)	940	16.0	960	2.9
FREE (normal)	520	8.8	510	1.5
FREE (uncat OFNT)	740	12.6	650	1.9
CATALOG	510	8.7	500	1.5
UNCATALOG	460	7.8	320	1.0
PERMIT	580	9.9	460	1.4
INHIBIT	550	9.4	400	1.2
ADDCOPY	620	10.5	370	1.1
DLTCOPY	630	10.7	360	1.1
CREATE	790	13.4	570	1.7
DELETE	620	10.5	380	1.1
DISP	650	11.1	610	1.8
OPEN	1350	23.0	370	1.1
CLOSE	450	7.7	190	0.6
initialize	7640	129.9	10	0.0

7. Conclusion

In this paper, we presented the structure of the network oriented operating system and the feasibility shown through implementation results obtained with the experimental computer network TECNET. The elements of NOS are the system nucleus, which enables interprocess communication beyond machine boundary, the distributed file management system, which permits multiplicated files, and the distriburd job management system, which expands the concept of local processing to a virtual network environment yet preserves the autonomy of each local system.

Following are projects that should be completed in the future:

1. Implementation of reliability feature other than the multiple physical file feature
2. Development of network oriented programming

language
3. Development of a high-level, job-control language for the network environment
4. Extension of the distributed file management system toward a distributed data base management system.

Acknowledgement

We acknowledge the following people for their work in developing the elements of NOS on TECNET. F. Wagai, N. Yamanouchi, and A. Nakagawa (now with Fujitsu) implemented the system nucleus for each TECNET computer. The distributed file management system was developed by K. Horita of Fuji Electric Co., E. Aoki of Fujitsu, M. Horiguchi (now with Hitachi) and K. Honda (now with Fujitsu). The distributed job management system was developed by N. Kurobane (now with Fujitsu), M. Oda of Fujitsu, M. Horiguchi, K. Honda, and Y. Yamanouchi. We also thank the many people of our laboratory for their useful discussions.

References

1. TANAKA, H. and MOTO-OKA, T. An Experimental Computer Network TECNET, Papers of Technical Group on Electronic Computers, IECE Japan, EC-73-57, (Dec. 1973) (In Japanese).
2. MOTO-OKA, T. and YAMAMURO, Y. Polyprocessor System PPS-1, *JIP* 15, 7, (July 1974) 557-564 (In Japanese).
3. TANAKA, H. and MOTO-OKA, T. Network Oriented Operating System-Process Control and Distributed File in TECNET, *Pacific Area Computer Network Symposium*, (Aug. 1975) 171-179.
4. TAKEICHI, M. A Mini Compiler of a Mini Language, *bit*, 6, No. 8-13, (1974) (In Japanese).
5. LINDSAY, B. G., et al. Notes on Distributed Databases, Research Report, IBM Research Lab. San Jose, RJ2571, (June 1979).