Synchronization Services

This chapter explains the NetWare services that control file and record sharing. Developers can use these calls to manage file access among users on the network. This chapter includes the following topics:

- An Introduction to File Sharing
- Locking Files and File Sets
- Locking Records
- Semaphores

An Introduction to File Sharing

One of the major aims of a NetWare network is to allow many users to easily access data that is kept on common storage devices. The file server controls access to these storage devices, retrieving and writing data as requested by workstations. Such files can be considered network files, in contrast to local files that are stored on a workstation's local disk drives and are accessed only by that workstation.

Through the file server, workstations not only have access to the same network files, but they have access to the same network files at the same time. Several workstations can open the same file, retrieve its contents into workstation memory, make changes and write the file back to storage.

So long as network files are intended to be read only, no conflict should occur between workstations as they retrieve files and read their contents. However, when network files need to be altered and updated, the potential for conflict becomes very great.

If even two workstations, independent of one another, are changing the contents of a file at the same time, the results can be disastrous. Since files are copied into workstation memory as they are read, both workstations are only dealing with a ªworking copyº of the file and must write any changes they make back to the disk. As both workstations update the network file, neither one is aware of the changes the other is making.

Synchronization Services are designed to help applications avoid the sort of file-sharing conflicts described above. These services are based on the concept of data locking. Before an application allows a user to modify a file, the file is locked so that only that user has access to it. As soon as the modifications are made, the application releases the file, allowing other users to modify it.

In situations that involve many interdependent files, the potential for problems is compounded. Groups of files may have to be locked together until a single modification is carried through to the end. In the meantime, the application may permit other users to read the locked files, so long as they do not attempt to change anything.

Data locking reduces the likelihood of two workstations attempting to update the same file simultaneously, but it is only a partial solution. Data locking does not solve the problem of one station being unaware of the changes another station has introduced into a file.

For example, suppose that workstation #1 locks a file and modifies it while workstation #2 is reading the file. When workstation #1 releases the file, workstation #2 is free to write to it. But workstation #2 is working on an instance of the file that does not include the current changes made by workstation #1. From banking records to travel reservations to inventory control, the scenarios for a catastrophic error resulting from this arrangement are easy to imagine.

To ensure that a workstation always has a current copy of a file when making changes, an application can do one of two things. On the one hand, an application can simply lock the data so that only one user has access to it at a time, whether for reading or for modifying. On the other hand, an application can perform a read-modify-write cycle every time the user makes a modification. This latter approach is usually the more desirable, since it promotes a true multi-user environment.

An application should perform a read-modify-write cycle in conjunction with data locking. The exact arrangement

for synchronizing changes will depend on the nature of the data the application is dealing with. Typically, an application should present the data to the user and wait for the user to make a change. When a change is requested, the application locks the data and rereads it. The application can determine whether the change is feasible according to the current status of the data. Then the application has the choice of aborting the change or carrying through.

While data locking solves many problems, it also introduces others. One serious concern is that data locking can result in a deadlock between two workstations. Typically, if a workstation attempts to lock a file that is already locked, the workstation will wait for the file's release. A deadlock can occur when two workstations attempt to lock the same two files together as a set. If each workstation succeeds in locking one of the files and then enters a waiting state, both workstations can remain there indefinitely waiting for the other to release its file.

These are the basic issues involved in synchronizing file usage on the network. Synchronization Services provide several ways of solving these problems. The following sections will look at each approach in detail.

Locking Files and File Sets

The simplest way to handle data locking is in terms of files. NetWare provides an automatic file-locking mechanism through its system of file attributes. However, Synchronization Services present a more powerful tool for locking files and coordinating their usage.

File Locking Through File Attributes

By default, NetWare places an automatic file lock on any file that is in use through NetWare's system of file attributes. NetWare's FLAG utility allows users to mark a file as shareable or non-shareable, as well as read-write and read-only. Developers can provide the same effect by using the calls found in File Services .

Using file attributes to lock a file allows only one user to access the file at a time. This is an easy way to protect data, and is adequate for many situations. For example, word processing text files are generally designed for a single user to access. By using NetWare's automatic file attribute, a word processing program can save itself the trouble of testing and assigning file locks.

Locking Files Manually

Synchronization Services contain a group of calls that allow an application to lock specific files individually or together as a set. These calls include:

NWClearFile
NWClearFileSet
NWLockFileSet
NWLogFile
NWReleaseFile
NWReleaseFileSet

As might be inferred from the calls listed above, working with file locks consists of four separate tasks: logging, locking, releasing and clearing.

A file server maintains a log table for each client, listing the file or files a workstation wants to lock. Taken together, the files in the log table are a file set. The file server will attempt to lock all of these files together. If any one of the files is already locked by another workstation, the attempt to lock the file set will fail.

An application can use the call NWLogFile to enter a file in the log table. The call includes a parameter, lockDirective, that allows the application to specify if the file should only be added to the log table or if the file server should go ahead and attempt to lock the file. Another parameter, timeoutLimit, indicates how long the file server should attempt to lock the file if the file is currently in use.

When an application has entered all the files it needs to lock in the log table, this file set can then be locked by making the call NWLockFileSet. Like NWLogFile, NWLockFileSet also has a parameter, timeoutLimit, that controls the file server's efforts to lock the file set. Once the file set is locked, no other workstation can access any one of the files until it is released.

NWReleaseFile allows an application to release the lock on a particular file. The file remains in the log table and will be locked with the other files in the table the next time NWLockFileSet is called. NWReleaseFile affects only the file that is specified. All other locked files remain locked. The call NWReleaseFileSet releases all the files a workstation has locked.

Finally, a pair of calls, NWClearFile and NWClearFileSet, can be used to remove files from the log table of the requesting workstation. If a file is currently locked, these calls release the lock. NWClearFile can be used to remove individual files from the table. NWClearFileSet empties the entire log table.

Locking Records

Although file locking is an effective way to protect data, it may create quite an inconvenience for other users. This is especially true if a lot of data must be locked while only a tiny portion is being updated. Record locking allows an application to restrict data locking to individual records, structure and variables. These records can be locked while the remainder of a file is available to other users. NetWare allows an application to use either physical or logical record locks.

Physical Record Locks

Like a file lock, a physical record lock is associated with specific bytes of data on the file server's disk storage. Physical record locks can be used in conjunction with file locks to obtain a wide range of data locking features.

The following Synchronization Services are relevant to physical record locks:

NWClearPhysicalRecord
NWClearPhysicalRecordSet
NWLockPhysicalRecordSet
NWLogPhysicalRecord
NWReleasePhysicalRecord
NWReleasePhysicalRecordSet

The calls for manipulating physical record locks are equivalent to those for manipulating file locks and include logging, locking, releasing and clearing. Like files, physical records are logged into a table that the file server uses to coordinate the record locking. An application can manage this process the same way it would manage file locking.

An application can log a physical record using the call NWLogPhysicalRecord. This call passes a DOS handle for the file that contains the record. The record is indicated by passing its starting offset along with the record's length. Another parameter, lockDirective, indicates whether the record should lock at the time it is logged. If the record is locked, it can be made available to other workstations for reading or it can be made available exclusively to the requesting workstation.

An application can lock all the records in the log table by calling NWLockPhysicalRecordSet. Individual records and record sets can be released and cleared using the calls NWClearPhysicalRecord, NWClearPhysicalRecordSet, NWReleasePhysicalRecord and NWReleasePhysicalRecordSet. These calls work virtually the same as their file lock equivalents.

Logical Record Locks

Many applications may find it simpler and more convenient to identify records logically rather than physically. A logical record is a name that represents network data. The data may include files or physical records. An application can assign logical records as it chooses, but it must consistently access that data by its record name.

Logical records are logged, locked, released and cleared the same as physical records. However, locking a logical record locks only the record name, not the data it refers to. When a workstation locks a logical record, other workstation will have to wait until the record is released before they can lock it.

Logical record locks serve more as coordinating devices than as security devices. Their effectiveness depends on the internal consistency of the application that creates them. A locked logical record cannot prevent another workstation from tampering with the locked data if the workstation knows the data's address.

Since files and physical records affect data directly, they invalidate logical record locks. For this reason, logical record locks should never be used in conjunction with file locks or physical record locks.

The following Synchronization Services control logical record locking:

NWClearLogicalRecord
NWClearLogicalRecordSet
NWLockLogicalRecordSet
NWLogLogicalRecord
NWReleaseLogicalRecord
NWReleaseLogicalRecordSet

Individual records are logged, locked, cleared and released by passing a character pointer to the logical record name. This contrasts with physical records, which are referenced by a file handle and a record address. Logical record names can be up to 100 bytes in length, including a null terminator.

As with physical records, an attempt to lock a logical record or record set is controlled by a time-out limit. When the limit expires, the file server aborts the attempt to lock the records. Another parameter, lockDirective, indicates whether the record should lock at the time it is logged.

Semaphores

Semaphores, like logical records, are labels that indirectly control network activity. In essence, a semaphore is an ASCIIZ string with an associated value. A semaphore name can be up to 127 bytes in length. Its value may be from 0 through 127.

It is up to an application to define the influence of the semaphores it creates. Generally, semaphores are used to control access to a network resource. For example, a semaphore can be used to limit a resource to one user at a time or to a specified number of maximum users.

The following Synchronization Services deal with semaphores:

NWCloseSemaphore
NWExamineSemaphore
NWOpenSemaphore
NWSignalSemaphore
NWWaitOnSemaphore

An application accesses a semaphore's resource by opening the semaphore. When a semaphore is opened its value is automatically decremented by one. An application can open a semaphore by making the call NWOpenSemaphore. If the semaphore does not exist, NWOpenSemaphore will create it.

NWOpenSemaphore passes a semaphore name and an initial value. The initial value is assigned to the semaphore only if the semaphore does not already exist. NWOpenSemaphore returns a semaphore handle and an open count. The open count indicates how many applications have the semaphore open.

When an application is finished using a semaphore's resource, it must make the call NWSignalSemaphore to increment the semaphore's value. The application must also call NWCloseSemaphore to decrement the semaphore's open count by one. When a semaphore's open count reaches 0, the semaphore is deleted. Both of these calls pass the semaphore handle.

The call NWExamineSemaphore allows an application to find out a semaphore's value and its open count without attempting to open the semaphore. The semaphore value can be positive or negative (from -127 through 127). A positive value indicates that the application can open the semaphore and access its resource. A negative value indicates the number of processes waiting to open the semaphore. Based on this value, the application may have to wait until another application closes the semaphore before it can access the resource.

An application can use the call NWWaitOnSemaphore to wait for a semaphore to open. NWWaitOnSemaphore

decrements a semaphore's value by one. If the value is greater than or equal to 0, then the application can access the semaphore's resource. If the value is a negative number, the application is placed in a queue.

Through a parameter, timeOutLimit, the NWWaitOnSemaphore call passes the amount of time the application wants to wait for the semaphore. When the time-out expires, the semaphore value is tested again. If it is positive, the application can access the resource. If it is negative, the application is removed from the queue and the semaphore value is incremented.