UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 6: UNIX and Windows Interoperability

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
Microsoft® Windows®
UNIX applications

The *patterns & practices* team has decided to archive this content to allow us to streamline our latest content offerings on our main site and keep it focused on the newest, most relevant content. However, we will continue to make this content available because it is still of interest to some of our users.
We offer this content as-is, without warranty that it is still technically accurate as some of the material is undoubtedly outdated. Note that the content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

Summary: Chapter 6: UNIX and Windows Interoperability focuses on technologies that enable Windows and UNIX Interoperability. It provides a high-level overview of the considerations that you must address when you have decided that interoperability best suits the needs of your organization. (19 printed pages)

Introduction

The UNIX and Microsoft® Windows® operating systems are very different. (For detailed information on the differences, see Chapter 2, UNIX and Windows Compared.) Despite the differences, UNIX and Windows-based systems must communicate or interoperate during a migration project.

This chapter presents the issues of Windows and UNIX interoperability and the tools available to

address the issues.

Three categories of interoperability issues and the tools to address them are presented in the following sections:

- "Windows to UNIX Connectivity" discusses how a user on one operating system can connect to and use the other operating system over a network.
- "User Authentication and Authorization" discusses the UNIX and Windows security models and how to integrate them so that users on one operating system can securely use resources on the other.
- "Resource and Data Sharing" discusses how to provide access to UNIX and Windows-based resources (specifically networked file systems) across the two operating systems.

The fourth section, "Choosing Interoperability Solutions," presents strategic considerations presented by Win32 and Interix environments and by cross-platform support for interoperability.

For information on interoperability in general, see the section "Further Reading" in this chapter.

For specific advice on choosing solutions for the development and live environments, see Chapter 7, Creating the Development Environment, and Chapter 13, Creating the Live Environment.

Windows to UNIX Connectivity

The primary interface from Windows-based systems to UNIX systems was traditionally a command-line shell accessed by using a terminal or terminal emulator. (For more detailed information, see Chapter 2, UNIX and Windows Compared.) Modern terminal access to UNIX systems is either from the command line or from a graphical user interface (GUI).

- Command-line access tools include telnet, Berkeley "r" access commands (**rsh**, **rexec**, **rcp**), and secure shell (**ssh**).
- The telnet client allows the user to operate in the multi-user environment provided by the host, with access provided by the telnet server.
- The **rsh** command allows the user to run commands remotely from a local environment.
- The GUI access tool is X Windows.
- An X Windows server provides a graphical interface to local or remote applications.

Developers and users of UNIX applications typically use one or more of these access methods. The access method used affects both the migration strategy and the migration process itself. While migrating, developers need access to the UNIX development servers from their Windows desktops. Even during a rewrite to Win32, which replaces the UNIX environment permanently, developers must have access to the UNIX environment.

Developers need access for operations such as:

- Running the UNIX application to check the original functionality
- Referring to UNIX code, the build configuration, or management scripts
- Migrating any missing portions of code or scripts

Users require UNIX connectivity when the migrated application continues to use UNIX-style interfaces, such as in some migrations to the Interix environment.

The Windows operating system includes a basic telnet client and supports the **rsh**, **rexec**, and **rcp** client commands.

With an X Windows server on the desktop, developers can also use a GUI to connect to UNIX-based applications. Windows operating systems do not provide X Windows. For X Windows connectivity, developers need a third-party X Windows server.

The following sections describe the main connectivity solutions available.

> **Note** In X Windows terminology, *server* refers to the software running on the user's desktop that provides the user interface, and *client* is the application running on the UNIX server.

Terminal Emulation and Command-Line Connectivity

UNIX developers use the tools described in this section to access UNIX development servers from the command line. If the UNIX application has a command-line interface, these tools are also helpful to users.

Windows telnet client

The Windows telnet client provides basic command-line access to UNIX systems. Its command syntax is identical to the telnet clients found on UNIX operating systems. The Windows telnet client runs in a window. Users can cut and paste text between the UNIX command line and Windows-based applications. Users run the client from the command prompt or by using **Run** on the Start menu.

The Microsoft HyperTerminal application also includes a telnet client. HyperTerminal offers more configuration options than the command-line telnet client. Terminal types for the client range from ANSI to VT52 to VT100. It is straightforward to cut and paste text between HyperTerminal and Windows-based applications.

A third-party telnet client may be required for command-line tools that use terminal characteristics beyond those provided by HyperTerminal, such as function keys or full-screen operation.

Windows telnet server

Developers who need to connect from UNIX to Windows-based servers can install the telnet server included in the Windows server operating systems.

To connect from UNIX to a Windows-based server

1. Use the **tlntadmn** command to modify the NTLM authentication registry option. You need to change the default value from 2 to 1 as follows:
    a. Run the **tlntadmn** command utility.
    b. Choose option 3 to configure registry settings.
    c. The choose option 7 for NTLM settings.
    d. Change the value to "2".
    e. Exit the utility.
2. Start the server by typing **net start tlntsvr**.

> **Note** Standard telnet is inherently insecure in that passwords are transmitted over the network in clear text. Some UNIX telnet servers and clients make use of Kerberos authentication, and the Windows client and servers can use NT authentication for added security. If the application to be migrated makes use of standard telnet, consider migrating it to the more secure Windows version of telnet.

Remote shell (rsh), remote copy (rcp), and remote execute (rexec)

Using the Berkeley "r" commands for remote access into UNIX servers simplifies access. Authentication occurs only once on the logon UNIX system, and then a user does not need to log on again to connect to other systems.

Because of this, it is easy to run remote shells (**rsh**) and remote commands (**rexec**) and to copy files between remote machines (**rcp**).

The remote clients are included in Windows. The syntax for the Windows commands is:

```
rsh [Host] [-l UserName] [-n] [Command]

rcp [{-a | -b}] [-h] [-r] [Host][.User:] [Source] [Host][.User:]
[Path\Destination]

rexec [Host] [-l UserName] [-n] [Command]
```

On Windows-based clients, the *UserName* parameter is required if user names on UNIX and Windows are different.

The Windows 2000 Resource Kit includes an **rsh** tool, Rshsvc.exe.

> **Note** Authentication for Berkeley "r" commands is based on the existence of a valid client host and username combination as command parameters. Because this information is revealed, these commands can present a security risk. For this reason, the commands are often disabled and replaced by the **ssh** command, as described below.

Secure shell (ssh)

The secure shell provides extra authentication and encryption features. It is often used to replace Telnet and the Berkeley "r" commands where security is an issue. The **ssh** command functionality is similar to the **rsh** command, except that **ssh** can be made very secure (client and user information is not revealed to third parties).

Windows operating systems do not include a secure shell client or server. However, third-party products do provide **ssh** functionality.

Windows

Users who require GUI access to a UNIX server from the Windows-based desktop normally use an X Windows server. Remote desktop management services are provided by Windows Terminal Services (and its corresponding implementations on UNIX and Linux).

X Windows

Windows operating systems do not include an X Windows server. However, the Interix environment does include X Windows clients, which can be used to develop applications that use an X Windows user interface.

Third-party products that provide Windows-based X Windows servers are well integrated with the Windows desktop. Users of the third-party products can cut and paste text graphics between X Windows and Windows. Third-party Windows-based X Windows servers include:

- Hummingbird Exceed. For more information, see the Hummingbird Web site.
- MKS Toolkit. For more information, see the MKS Web site.
- WRQ Reflection(r) X. For more information, see the WRQ Web site.

These products provide support for most X Windows standards. Each product also includes extras. For example, MKS Toolkit also includes more than 300 UNIX commands that can be used within Windows.

Remote desktops

Windows operating systems handle multiple users in a different way than UNIX does. A single user at a time logs onto a Windows machine. However, using Windows 2000 Terminal Services is analogous to using X Windows on UNIX. By using Terminal Services, a Windows-based server can provide a desktop to remote users on clients spread over a network. This is often referred to as *thin client* as the client only has to handle input and output whereas the application runs on the server.

For UNIX to Windows connectivity, UNIX and Linux implement the RDP protocol used by Terminal Services. (At the time of writing, these products are new. It has not been determined whether they are appropriate for a commercial environment.)

To connect a remote desktop from a UNIX client to a Windows-based server, implement Citrix MetaFrame. Citrix MetaFrame has clients for a wide range of operating systems, including UNIX and Linux. (For more information about Citrix MetaFrame, see the Citrix Web site. An X Windows interface to UNIX is required.

User Authentication and Authorization

This section discusses security issues during UNIX to Windows migration, and potential solutions.

In any heterogeneous network—such as a network migrating to Windows—users need to work across systems and the systems themselves need to interoperate. Because of this, a migration project needs to take into account the differences between the UNIX and Windows security models.

Migration planners need to resolve these security questions:

- Do users require two sets of user names and passwords, one for UNIX and one for Windows?
- How do users keep their passwords synchronized?
- Do administrators need to manage two user databases?
- How do administrators manage user security on shared resources, such as network file systems?

The greater the need for interoperability during the migration, the more important these questions become. The rest of this section presents information about Windows and UNIX security and how to integrate them. This information determines which of the security questions must be addressed in a

particular migration project, and how to address them.

Chapter 2, UNIX and Windows Compared, describes and compares the UNIX and Windows security models. Security models for both operating systems require:

- Authentication to verify the user's identity
- Authorization to control access to resources and to limit what a user can do

This section describes how UNIX handles authentication and authorization, how Windows handles them, and then how to integrate UNIX and Windows security.

Authentication and Authorization in UNIX

UNIX security relies on user credentials and file permissions. (For more information, see Chapter 2, UNIX and Windows Compared.)

In UNIX, user credentials consist of two unique identifiers for each user: a user identifier (UID) and a group identifier (GID). These identifiers apply to every UNIX process. UIDs and GIDs are integers assigned by the administrator. They are not guaranteed to be unique across a network.

Users are listed in the "/etc/passwd" file. This file contains a single line of text for each user. The line has fields for the user name, user ID, password, home directory, default shell, and others. The password is encrypted; the administrator cannot read it.

Groups are listed in the "/etc/group" file, which lists the names of the users in each group as well as the groups themselves.

UNIX assigns permissions to every file, directory, and device. The permissions are for the user (owner of the file), for the group, or for everyone else.

When a process seeks access to a file, UNIX uses the UID and GID of the process to select which permissions to apply to that process.

To authenticate a user at logon, UNIX uses techniques such as username/password pair, Kerberos V4/V5, and proprietary schemes. After logon, every process a user creates has the same UID and GID, and thus the access rights of the user referenced by the UID.

To secure stations, some UNIX installations also use a host.equiv or .rhosts file containing host information based on the host name and IP address. These files define the set of trusted hosts, users, and host-user pairs that certain UNIX processes use to verify access. For example, **rlogin** (remote logon) and **rsh** (remote shell) bypass the normal logon and password security mechanisms of /etc/passwd or Network Information Service (NIS). The /etc/hosts.equiv file contains either trusted hosts and/or trusted host-user pairs.

The UNIX user database can be centralized for a network of UNIX machines using NIS. NIS maps the /etc/passwd and /etc/group files to a central database held on the NIS servers. All other aspects of UNIX security continue to work in the same manner, but use UIDs and GIDs from the NIS server rather than local files.

Authentication and Authorization in Windows

Windows security relies on user credentials and object permissions. (For more information, see Chapter 2, UNIX and Windows Compared.)

When a user logs on to a Windows-based machine or domain, Windows authenticates and then identifies the user by using a unique security identifier (SID).

Windows objects (including files and directories) have a security descriptor (SD) associated with them. The information in the security descriptor includes the owner of the object and a discretionary access control list (DACL). The DACL in Windows allows a much greater range of permissions than UNIX does, and those permissions can be assigned more selectively than those provided by UNIX.

Integrating UNIX and Windows Security

Figure 1 illustrates the differences between the UNIX and Windows security models. Although the differences in this example focus on NIS and Active Directory domains, the differences are identical for UNIX and Windows-based computers that are not part of a security domain.

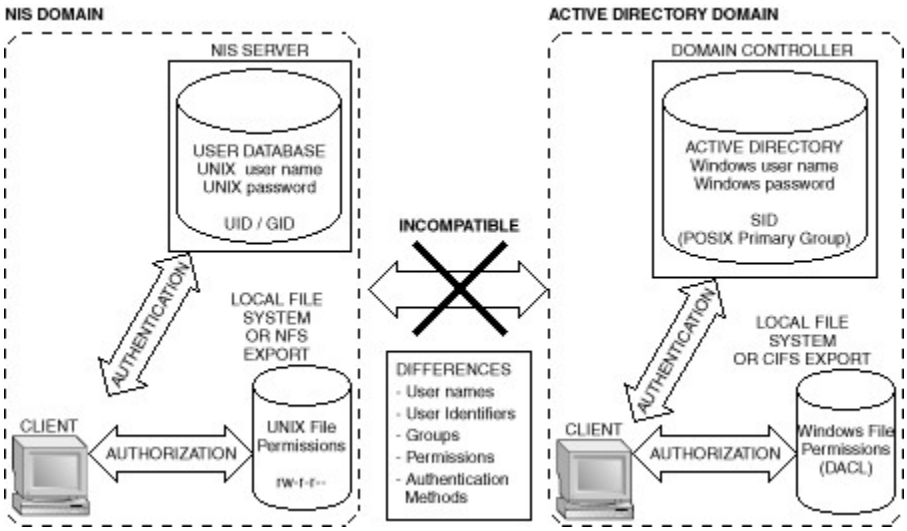

Figure 1. UNIX and Windows security model differences

Table 1 summarizes the differences illustrated in Figure 1.

Table 1. Security model differences

Security feature		UNIX	Windows
User names			
	Length	Typically 8 characters	20 characters
	Case sensitive	Yes	No
	Can be same as group names?	Yes	No
User identifier		UID 0-65353	SID unique
POSIX groups		Yes	Yes—but not default
File permissions		Simple bit mode rwxrwxrwx	Complex Access Control Lists DACL
Network user database		NIS	Active Directory

These differences raise a number of interoperability issues:

- Mapping UNIX-style user names to Windows-style user names
- Mapping UIDs to SIDs and SIDs to UIDs to control access to resources such as files
- Converting incompatible file and resource permissions
- Centralizing the management of UNIX and Windows users

The following sections describe some tools for integrating UNIX and Windows security.

User account management and synchronization

There are three approaches to managing users across UNIX and Windows:

- **Create separate and unrelated accounts for UNIX and Windows**.

  This solution is simple, but requires the most effort for users of both platforms. It complicates cross-system authentication and security permissions. This solution is appropriate when the need for interoperability between UNIX and Windows is limited and brief.

- **Create duplicate, identical accounts on UNIX and Windows**.

  Duplication of accounts simplifies interoperability because UNIX and Windows accounts can relate to each other. It also simplifies the conversion of permissions. However, it does double administration and can result in users having two sets of passwords.

- **Integrate UNIX and Windows account management**.

  This long-term solution accommodates a long-term migration project or a target environment that includes both UNIX and Windows-based servers. In this case, administrators should centralize user management into one place, such as Windows Active Directory. Management from Windows Active Directory is consistent with a migration to the Windows platform. Also, Windows Active Directory is a well-integrated product with built-in management functionality.

Many third-party and Microsoft interoperability products provide UNIX-to-Windows user mapping. For example, Samba provides its own methods of mapping user names. (For more information about Samba, see "Samba" and "Resource and Data Sharing" later in this chapter.) Most products provide different methods of mapping user names, which creates a need for additional administration.

Windows Services for UNIX Server for NIS

Windows Services for UNIX includes a Network Information System (NIS) server called Server for NIS. This service uses Windows 2000 Active Directory to implement NIS.

Administrators can use Server for NIS to:

- Store NIS maps by using Active Directory.
- Integrate NIS data (such as user data) with corresponding Active Directory objects.
- Use Windows 2000-based servers as NIS servers.
- Migrate NIS domains to Active Directory.
- Merge multiple NIS domains.

- Synchronize passwords. (See "Windows Services for UNIX Password Synchronization" in this chapter.)

Figure 2 shows the architecture of Server for NIS. Notice that although UNIX slave NIS servers are still usable, the master NIS server now resides on Windows.

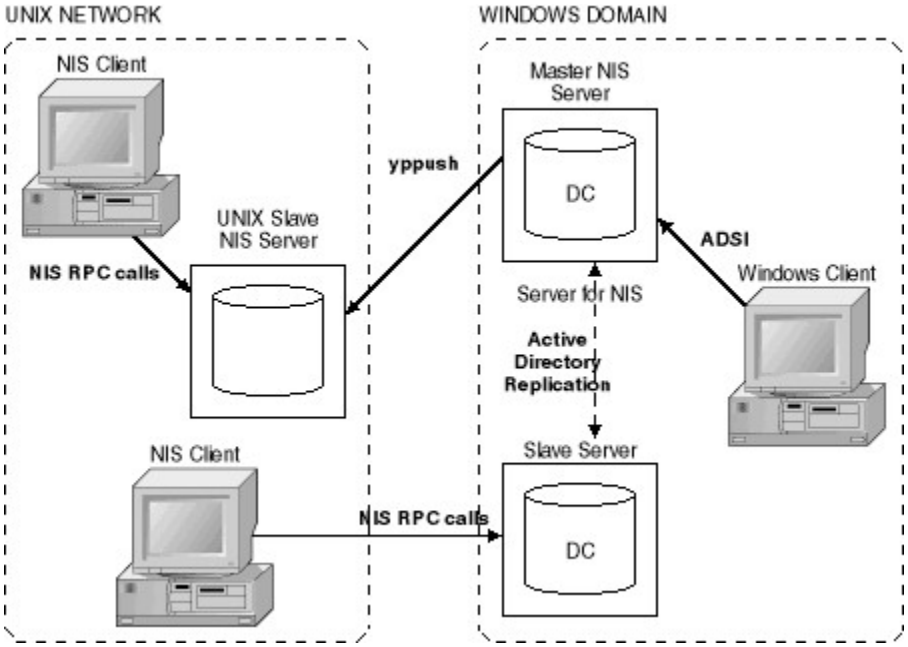

Figure 2. Architecture of Server for NIS

Server for NIS is especially useful for large networks that already have NIS and Active Directory. However, administrators need to be aware of how Server for NIS interoperation works:

- Server for NIS must be installed on a Windows 2000 domain controller. The Windows-based server uses Active Directory to store the NIS information. Normal Active Directory replication is used to propagate the NIS data.
- The computer running Server for NIS must be the master NIS server. This means that all the other Windows-based and UNIX NIS servers are subordinates.
- The Active Directory tools on the master computer administer both the NIS and the Active Directory domains.
- An NIS to Active Directory Migration Wizard assists in moving the existing NIS domain to Active Directory.

Windows Services for UNIX password synchronization

Windows Services for UNIX includes two-way password synchronization between UNIX and Windows.

To synchronize from UNIX to Windows, a single sign on daemon (SSOD) must be running. Windows Services for UNIX supplies precompiled SSODs for popular UNIX versions. For UNIX versions without a precompiled SSOD, Windows Services for UNIX supplies the source to create one.

Password synchronization occurs only between systems that are configured to do so by an administrator. A file on the UNIX side contains a list of computers to notify of password changes. Windows-based

tools on the Windows side maintain the list. In addition, administrators can specify which users have password that must be synchronized.

To synchronize passwords for Windows 2000 domain accounts, all domain controllers on the network must be running the synchronization service, and all participating UNIX computers must be running the SSOD. When synchronizing passwords for local user accounts, the service must be installed on all Windows 2000-based computers.

Because password synchronization does not use user name mapping, the user names must be exactly the same for both UNIX and Windows. Therefore, user names must conform to the most restrictive definition of length and structure. For example, because Windows user names are not case sensitive when used on UNIX, all Windows user names must be in lowercase.

Samba

Samba is an open-source freeware implementation of the common Internet file system (CIFS). (For more information on Samba, see "Resource and Data Sharing" later in this chapter.)

Samba is designed to integrate with Windows-based networks. Therefore, it must convert between UNIX and Windows user identities and between UNIX and Windows file permissions. To do this, Samba maps UNIX to Windows and Windows to UNIX users and it converts between UNIX and Windows file security schemes.

In Samba, user names can be different on UNIX and Windows. Thus, Samba provides the flexibility to use UNIX style user names on UNIX and Windows user names on Windows. This feature comes at the cost of additional administration.

Resource and Data Sharing

Any migration from UNIX to Windows requires the two environments to share data and resources. In the simplest case, UNIX servers migrate files to Windows-based servers. At the other end of the spectrum, UNIX servers and Windows-based servers share data and resources, such as printers, during and after migration.

This section presents the options for sharing resources between the two environments, concentrating on networked file systems. By using networked file systems, users of Windows or UNIX can manipulate files as though they were stored on a local file system.

Less integrated methods exist to transport files between systems, such as the File Transfer Protocol (FTP) and removable media (for example, tape). These methods are not considered here because they do not provide interoperability. However, removable media and FTP can be very useful during the migration. Tape is particularly effective to transfer volumes of data too large to feasibly transfer across a network.

Developers or application users should be able to use the files they need in their target environment (Windows or Interix) transparently; that is, they should not need to know whether the files are on a UNIX server or a Windows-based server. However, UNIX systems and Windows-based systems use different network protocols, functionality, and naming conventions for file sharing.

In Windows-based networks, the most common file-sharing protocol is server message block (SMB),

also known as the common Internet file system (CIFS). The most common network file system on UNIX systems is the network file system (NFS). NFS and SMB protocols do not interoperate.

The following sections review the file-sharing environments in UNIX and Windows and the options for providing interoperability.

UNIX Data Sharing Environment

The network file system (NFS) protocol is a UNIX-style file system that can be shared over the network. NFS uses UNIX user identification, group identification, and permissions.

NFS uses the *export* naming convention, in which an exported file system is referred to by the host name and the export name. For example,

```
server:/usr/local/pub
```

Where *server* is the host name and */usr/local/pub* is the export name.

An NFS client mounts the NFS export into its file system tree just as it mounts any other file system. To the user, the network file system looks like another directory under the root of the file system (/).

For example, a **mount** command might look like this:

```
mount -t nfs server:/usr/local/pub /pub
```

These features make NFS transparent to UNIX users.

NFS is used in many ways in UNIX environments. Servers can share common data by using NFS. Desktops or workstations can streamline administration by centralizing user data on an NFS server.

Windows Data Sharing Environment

The server message block (SMB) protocol—or the common Internet file system (CIFS) as it is now known—was designed to integrate transparently into Windows operating systems. It uses Windows security control identical to the features found on the NTFS or FAT file system. Available networked file systems are called *shares*. A Universal Naming Convention (UNC) name identifies the networked file. The UNC name consists of a server name and a share name, such as \\SERVER\SHARE.

On a client, a share maps to a drive letter in the same way as a disk partition or floppy disk; for example, X:\. Drivers can be mapped in a number of ways. The most basic way is to use the command-line utility **net use**. For example:

```
net use X: \\SERVER\SHARE
```

It is also possible to mount shares just as UNIX mounts them. In this case, there is no need for a drive letter reference.

File-naming conventions and other features of the networked file system are the same as for other Windows file systems.

Like UNIX, Windows uses file system sharing. Servers can share and replicate file systems. Desktop clients can store all their data on a centralized server, thus simplifying administration.

Network File System Interoperability

The Windows and UNIX networked file systems are incompatible, but there are two ways in which they can interoperate. They can be configured with the other type of networked file system, or they can use a gateway to interoperate. Both Windows and UNIX can be configured to use the NFS and SMB protocols and to provide gateways between the two protocols. Additional software is usually required.

For interoperability, network file system software must provide basic functionality:

- It must provide connectivity between the network client and the file server.
- It must translate between environments.
- It must provide for security mapping between the two systems. (For more information, see "User Authentication and Authorization" earlier in this chapter.)
- It must convert file system features between UNIX and Windows, for example, links and file locking.

File sharing between UNIX and Windows-based systems can be implemented in many ways. Figure 3 provides an overview of the options. Each of the interoperability solutions can be implemented either on a UNIX platform or on a Windows platform.

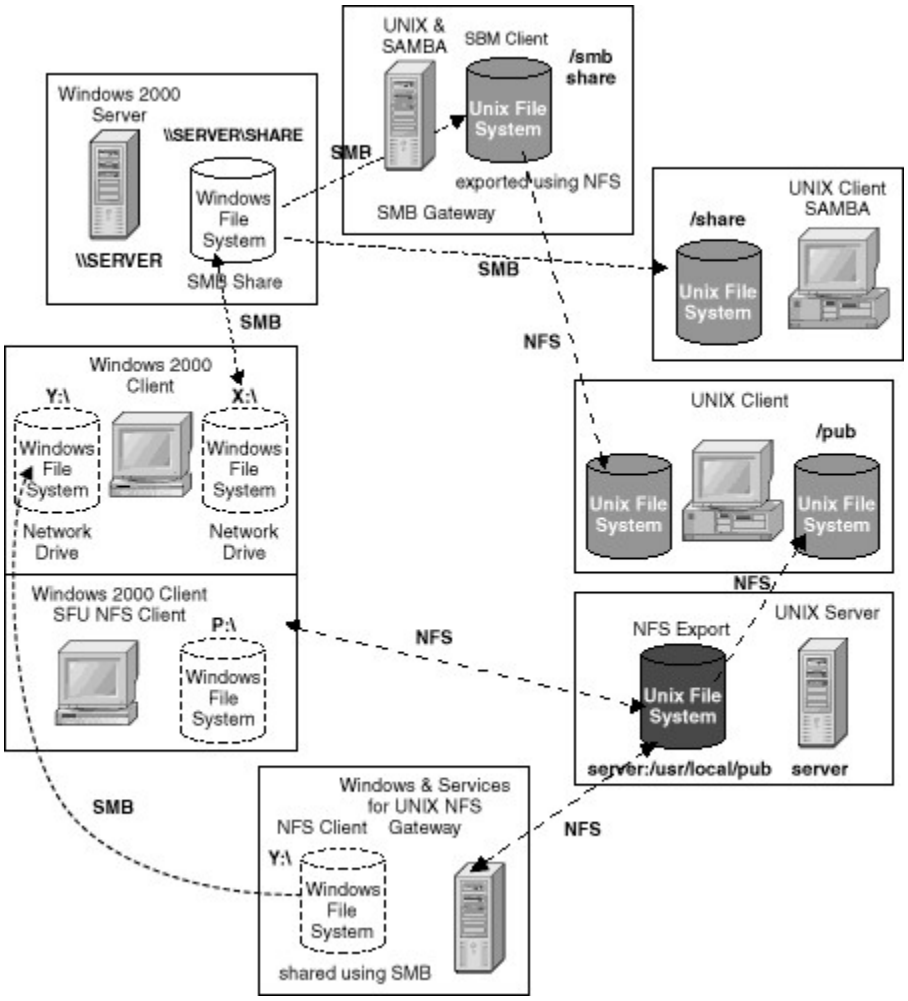

Figure 3. File-sharing options for UNIX and Windows

The following combinations are commonly used (shown in Figure 3):

- UNIX server provides SMB file shares to Windows clients and NFS file shares to UNIX clients. (Samba)
- UNIX server provides NFS file shares to both Windows and UNIX clients. (For example, Windows Services for UNIX: Client for NFS)
- Windows-based server provides SMB file shares to Windows clients and NFS file shares to UNIX clients. (For example, Windows Services for UNIX: Server for NFS)
- Windows-based server provides SMB file shares to Windows and UNIX clients. (Samba)

Samba for Data Sharing

Samba is a suite of programs that emulate Windows networking (CIFS) on other operating systems. Samba can make UNIX act as a CIFS server or client.

In addition, Samba features can ease integration of the two environments. Samba can share printers and file systems, act as a WINS server, or integrate into Windows NT domains and Windows Active Directory.

Samba is open source software, available on UNIX and many other platforms.

> **Note** Samba is not the only product that emulates Windows networking on UNIX systems, but it is widely used. For detailed information on Samba, see the Samba Web site.

Samba as a file server

By using Samba as a file server, UNIX file systems can be shared to Windows clients and servers as though they were hosted on a Windows platform. During a migration, users can then transparently gain access to UNIX resources from the new environment.

Samba as an NFS to CIFS gateway

NFS exports in a development or live environment can be converted into CIFS shares by using Samba. Samba can share any UNIX file system, including NFS mounted exports. This hides the NFS and UNIX nature of file systems from users in the Windows environment. Developers and users view the network using a Windows interface.

When Samba is used for data sharing, all workstations use native authentication protocols: that is, Windows Challenge/Response for the Windows-based workstation user, and NIS for the UNIX workstation user.

Samba user and password integration

To integrate, Samba maps credentials from the Windows to the UNIX environment, thereby obtaining NFS file and folder access based on UID and GID credentials, the native UNIX authorization mechanism. In other words, Samba takes credentials from the Windows client, validates them with a Windows domain controller, and then looks up the user name in the passwd or NIS map file by doing an NIS passwd query. If a match is found, Samba uses the UID and GID for the UNIX user to access files

on the UNIX NFS server, just as if the user were local.

The advantage of this arrangement is that the passwords on the UNIX and Windows systems do not have to be the same. Only the user name must be the same on both systems. (Samba does provide tools for synchronizing passwords between UNIX and Windows.)

All users must have UNIX accounts on the server (in the /etc/passwd file) or accounts supplied by an NIS domain server of which the Samba server is an NIS client. If a user does not have an account and guest privileges are not enabled, access is denied to data on the UNIX servers. Therefore, all users require both a Windows and a UNIX account.

NFS for Data Sharing

For network file transfers that use NFS, NFS software must be running on the Windows-based server or an NFS gateway must exist to convert an NFS export into a Windows share.

One system must be the NFS file server and the other must be the client. Often, because it probably already runs NFS, the UNIX side becomes the server. In this case, the Windows-based machine must run an NFS client program.

If files must be shared from a Windows-based server to UNIX clients, then it is easiest to set up the Windows-based system as the file server. The Windows-based server needs to have an NFS server on the systems that need to have NFS server software running.

NFS software products, including clients and servers, are available from Microsoft and others. The next sections discuss these products.

Windows Services for UNIX

Windows Services for UNIX provides the tools needed to set up sharing between UNIX and Windows-based systems. In addition to shells and command-line tools, Windows Services for UNIX delivers a server and a client for NFS and an NFS gateway. In addition, Windows Services for UNIX delivers User Name Mapping, which allows NFS access by using the user's Windows credentials.

The most straightforward way to set up NFS sharing is to install an NFS client on each client machine that needs access to NFS shares. By using the NFS client, users have access to files saved on NFS shares just like they have access to files on Windows-based servers. The same mechanisms used to map Windows shares can be used to map NFS shares. The share can be assigned a drive letter or the client can access the share by using an UNC name. After the share is mapped, users have normal access to the files.

Authentication and permissions

The Windows Services for UNIX NFS system uses User Name Mapping to authenticate the user to the server, whether or not the names match on both systems. Actual name mapping makes it possible to set up different schemes for allowing access to the NFS shares.

UNIX user names are case sensitive and Windows user names are not. Name mapping can handle this discrepancy by mapping names with uppercase letters to the UNIX convention of all lowercase letters.

Because User Name Mapping runs as a service, it requires a server set up on the network.

When Windows Services for UNIX NFS client software is being used to provide file sharing integration, each workstation client uses its native authentication protocol: Windows Challenge/Response for the Windows-based workstation user, and NIS for the UNIX workstation user. But the Windows Services for UNIX NFS Client uses the Windows Services for UNIX User Name Mapping Server to map the authenticated Windows user (by using the user SID) to a corresponding UNIX user name. It obtains the UID and GID to use for authorization (for example, file access permissions) in an NFS request to the NFS server. The Windows Services for UNIX User Name Mapping server functionality is essentially identical to the Samba server mapping function described previously.

Windows Services for UNIX Gateway for NFS

Gateway for NFS is another Windows Services for UNIX feature. In this case, the gateway computer communicates with the client computers by using the usual Windows file-sharing protocol. This eliminates the need to install the NFS client software on each client.

However, the best environment in which to use the gateway has a limited number of NFS shares that need to be available to the Windows clients. The server makes shares available as though they were shares on the gateway computer. Each share must be mapped to an available drive letter on the gateway computer. It is possible to use more than one gateway computer, but any single gateway computer is limited to the number of available drive letters.

Hummingbird NFS

The Hummingbird NFS Maestro suite offers NFS solutions similar to those in Windows Services for UNIX. NFS Maestro includes an NFS client package, an NFS server, and an NFS gateway product. The NFS Maestro gateway also eliminates the need to install NFS client software on each client machine.

For more information, see "Host Access and Network Connectivity" on the Hummingbird Web site.

WRQ Reflection

WRQ Reflection products also deliver NFS functionality for Windows computers. WRQ Reflection NFS Client offers client functionality and WRQ Reflection Suite for X contains the NFS client, X Windows capabilities, and connectivity to UNIX and mainframe systems.

For more information, see the WRQ Web site.

Choosing Interoperability Solutions

A complex migration environment requires many tactical decisions on interoperability solutions. This section looks at the interoperability solutions strategy and outlines factors that affect tactical decisions. (For specific advice on solutions, see the discussion of development and live environments in Chapter 7, Creating the Development Environment, and Chapter 13, Creating the Live Environment.)

For example, an administrator might decide that developers require only telnet access to the UNIX servers from their Windows-based machines. However, there one specific X Windows application might need to be referenced during the migration. In this case, a limited tactical implementation of X Windows servers for specific developers might be appropriate, even though it is not in the strategic plan.

Here are some major strategic considerations to consider:

- **Ease of use**

  The solution should be appropriately easy to use. The appropriate level depends on target users and the target environment.

- **Ease of administration**

  The solution should be easy to administer. Ease of administration means different things to UNIX and Windows administrators.

- **Transparency in the target environment (Windows)**

  This is closely linked to ease of use. However, because the target environment is Windows, the interoperability solution should have the look and feel of Windows.

- **Lifetime of the UNIX environment**

  The UNIX environment does not disappear overnight. In some migrations, a cross-platform, heterogeneous environment may continue for years. In others, the UNIX environment phases out after the migration is complete. The level of integration required for short-term and long-term interoperability is quite different.

- **Cost**

  A cost-benefit analysis is always needed. Even freeware solutions such as Samba have costs for administration and configuration.

- **Functionality**

  The technical features of an interoperability solution must meet the requirements of the users.

Interoperability Solutions for Windows Environments

When Windows is the chosen target environment, there usually exists a plan to move the development and live environments to a Windows-based platform as well. Because of this, interoperability solutions should also be Windows-based. Wherever possible, the UNIX environment should be hidden from Windows users. In addition, interoperability solutions should not constrain or further complicate the administration of the Windows environment.

Here are specifics of the three major categories of interoperability issues to consider when choosing interoperability solutions for Win32 environments:

- **Connectivity**

  The live environment should not require any connectivity to the UNIX environment. Developers will need connectivity during the migration. Connectivity solutions should be Windows-based, but their cost must take into consideration the lifetime of the UNIX environment: for example, there is no point in procuring an expensive connectivity solution if the UNIX environment is to be

decommissioned within a matter of months.

- **Authentication and authorization**

  Security should be Windows-based and transparent to Windows users. It is not necessary to integrate accounts using Services for NIS (or a similar product) because the UNIX environment has a short lifetime.

  Security differences should be managed by using either separate or duplicate accounts for developers and others who need UNIX access, and by User Name Mapping.

- **Resource sharing**

  During a migration to the Win32 environment, resource sharing might be needed for access to UNIX-based files in the development environment or for access to migration data in the live environment. The resource-sharing solution should be transparent in the Windows environment.

  It is possible to use Samba servers or Windows Services for UNIX NFS clients and gateways. If the priority is to move administrators to the Windows environment, using SFU NFS moves administration to the Windows platform. However, by using a UNIX solution, the Windows environment is not complicated by non-native protocols and systems such as NFS. Which solution to choose depends on the relative advantages to a particular migration project.

Interoperability Solutions for Interix Environments

When the target environment is Interix, the application and its environment can retain many UNIX characteristics. Because of this, users, developers, and support staff sometimes want to emphasize UNIX-style solutions.

However, Interix creates Windows-based solutions by using UNIX-style source code and development tools. Because Interix can do this, a Windows environment is still desirable. For that reason, the discussion that follows assumes that the UNIX environment will be phased out.

Here are specifics of the three major categories of interoperability issues to consider when choosing interoperability solutions for Interix environments:

- **Connectivity**

  Both live and development environments probably still require UNIX-style connectivity. If the Interix application continues to use an X Windows or character-based user interface, then Windows-based terminal emulators or X Windows servers will be required.

- **Authentication and authorization**

  Security should still be Windows-based and transparent to Windows users. Interix provides account interoperability.

  However, if the UNIX environment lifetime is short, integration of accounts by using Services for NIS or another product might be unnecessary.

Separate or duplicate accounts for those who need UNIX access and user name mapping can be used to manage security differences.

- **Resource sharing**

  During a migration to the Interix environment, users might need to share resources for access to UNIX-based files in the development environment or for access to migration data in the live environment. The resource-sharing solution should be transparent in both the Windows and the Interix environments.

  As explained earlier in this chapter, Windows shares can be mapped to Interix paths by using Interix and Microsoft Windows Services for UNIX (similar to how UNIX mounts NFS shares). To retain native Windows file sharing, any NFS share required by the application either should be shared as an SMB share by using Samba or should be accessed by using the Windows Services for UNIX NFS client.

  An Interix solution will probably use NFS and SMB shared resources. However, it is still desirable to move to a native Windows environment.

Cross-Platform Support

The most important factor to consider when choosing solutions is the need for ongoing cross-platform support. A detailed analysis of this subject is beyond the scope of this guide, but his section introduces the main points.

Cross-platform support often requires an integrated environment across Windows and UNIX with common source code and development tools. In such an environment, it is important to make interoperability as transparent as possible to both UNIX users and Windows users. In a cross-platform scenario, the most difficult interoperability issue is security.

Here are specifics of the three major categories of interoperability issues to consider when planning cross-platform support:

- **Connectivity**

  In a cross-platform environment, Windows to UNIX connectivity is particularly important. All connectivity options probably need to be implemented on the Windows clients. Developers who want remote access to Windows-based servers can use either the Windows Telnet server or a remote desktop running Terminal Services or Citrix MetaFrame.

- **Authentication and authorization**

  Even in a cross-platform environment, security should be transparent in the Windows environment. In this case, it should also be transparent in the UNIX environment. Due to the potential for human error, not to mention the additional administration burden, user name mappings and password synchronization should not be done manually in a cross-platform environment. Depending on the level of interoperation of the two environments, the Windows Services for UNIX Services for NIS could be appropriate for handling user name mapping and password synchronization.

- **Resource sharing**

  Resource sharing is particularly important in a cross-platform environment, and probably requires a mixture of NFS and SMB solutions. Such a mixture means that both Samba and Windows Services for UNIX services must be implemented.

Further Reading

This chapter can only be an introduction to such a large and varied subject as interoperability. There is a wealth of information on the Web and in particular on the Microsoft Web site with details of UNIX and Windows Interoperability. Some useful resources are listed below:

- Migrating to Windows from UNIX and Linux
- Windows Interoperability
- Windows Services for UNIX
- Microsoft Interix
- Windows 2000 Professional in a UNIX Environment
- Windows 2000 Resource Kits

Send feedback to Microsoft