Windows NT - A Linux Perspective

By Jeremy Allison

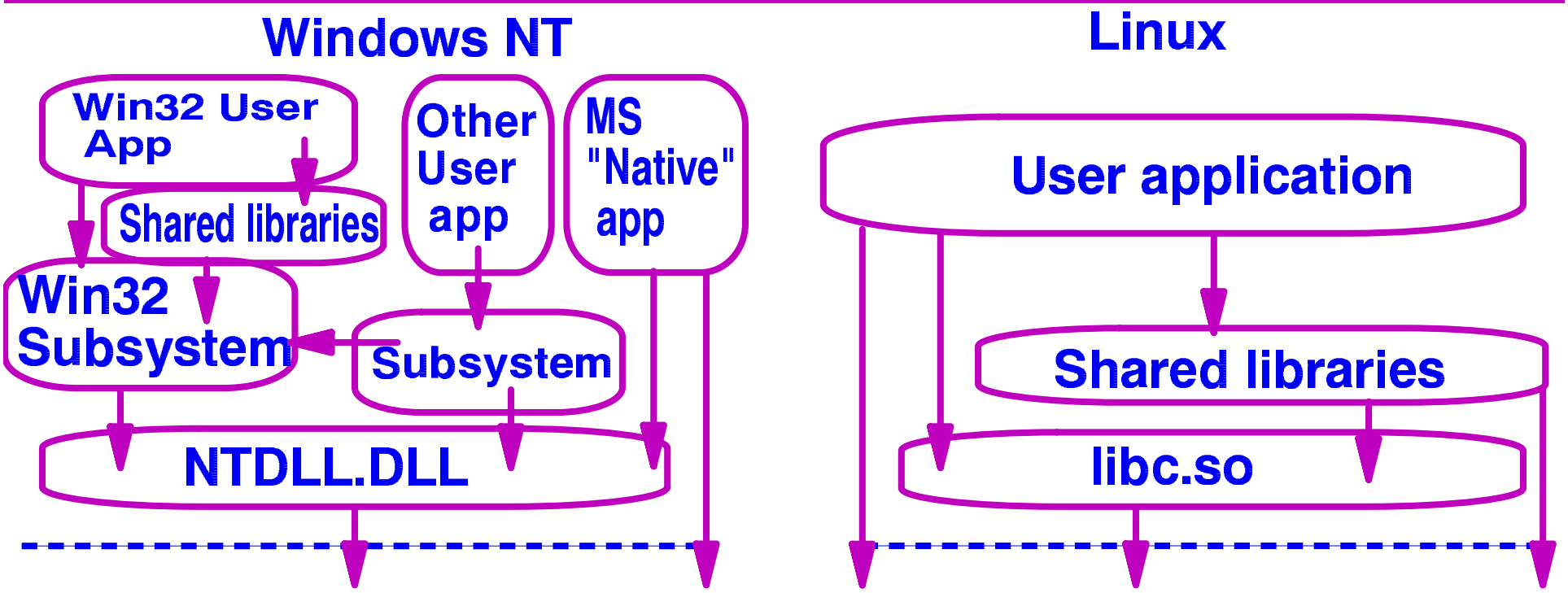
Development Team

email: jra@samba.org

Why is a Samba Guy doing this ?

- I've worked on all versions of NT, from the Win32 beta CD onwards.
- I have ported a lot of UNIX software to Windows NT.
  - The Vantive server, and a portable thread library.
  - The ONC/RPC library part of the Vantive client.
  - Kerberos 5.
  - One of the Cygwin32 developers.
- Samba interoperability teaches a lot about NT.
- We can learn from NT.
- SVLUG made me do it !

Comparison of Linux and Windows NT



Pageable SMP kernel with kernel threads

Non-Pageable SMP kernel with kernel threads

Little Known NT facts

- Windows NT is the only mass market OS with a completely undocumented system call interface.
- Win32 layer sits on top of this undocumented layer.
  - As do the other subsystems (POSIX and OS/2).
- Microsoft "system" applications call underlying kernel API (hidden in NTDLL.DLL) as needed.
  - Examples include event logger, winlogon process, the Windows shell.
- Most "general" Microsoft applications eg. SQLserver use Win32, most of the time.
  - Unless they need to do something no one else can.

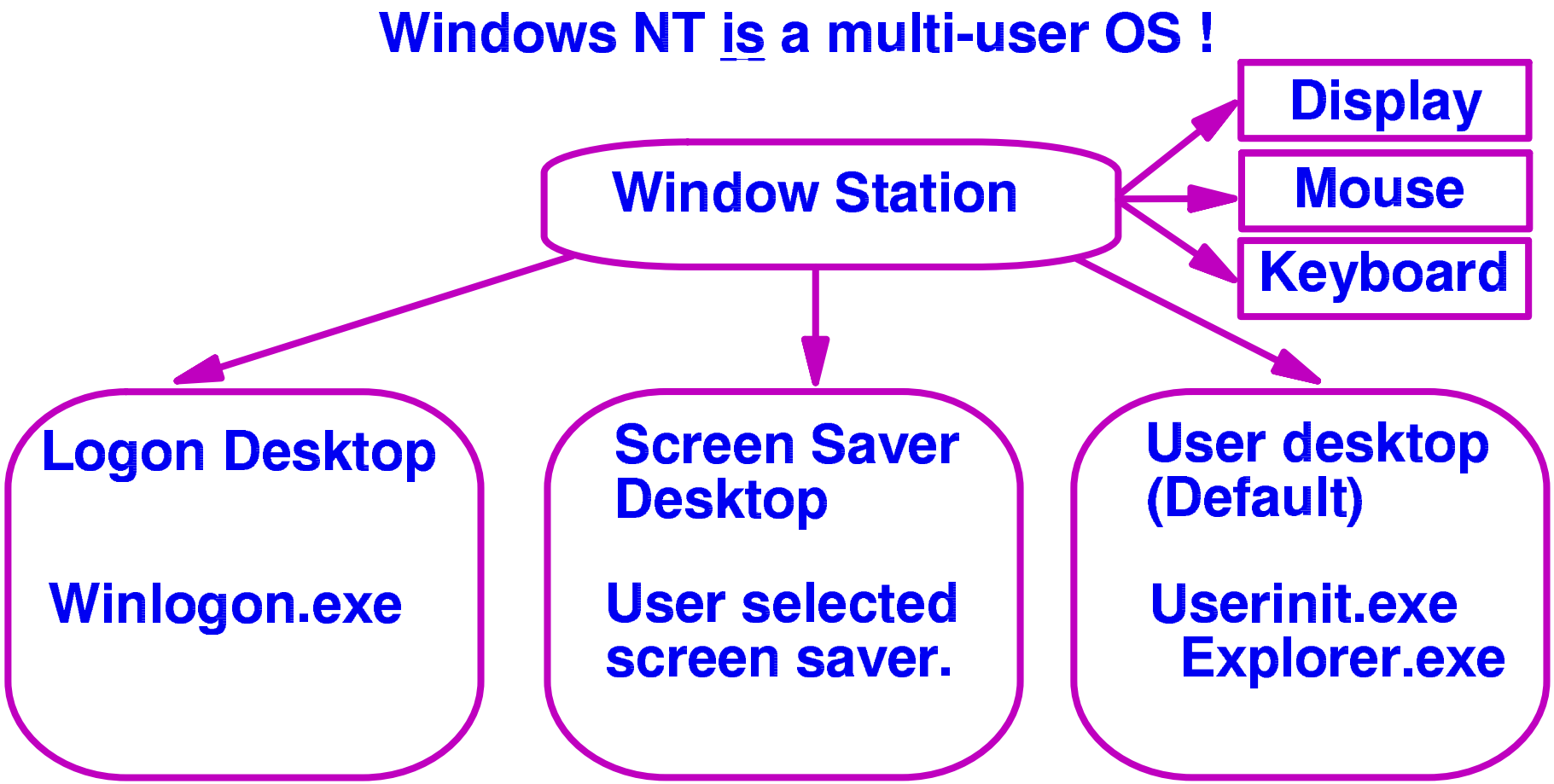
A brief comparison of the Win32 and POSIX API set.

- Win32 is very complex.
  - Changes with every service pack release.
  - Inconsistent design (error returns from API's are different in many cases, NULL or -1 meaning an error for example).
  - Provides rich functionality (multimedia API's and high level Internet API's for example).
  - Poorly documented (no real "spec" - the help pages do not cover many "corner" cases).
  - Is designed to be a "one-way" porting process. Difficult (although not impossible) to take Win32 code and move to other API sets.
  - Non portable between different Windows implementations.

A brief comparison of API's (continued).

- POSIX API's are limited.
  - Design by committee means lowest common denominator API's.
  - Some amazingly stupid decisions written into the spec (locking with multiple file descriptors for example).
  - Very slow to accept change (is this good or bad ? :-).
  - Fragmented implementations means autoconf needed.
- Most new design done in libraries (X, GTK, Qt).
  - Not all libraries available on all POSIX platforms (the horror of Motif, for example).
  - Vendor "holy" wars over library interfaces (GTK vs Qt vs Motif).

More little known NT facts.



Window Stations

- A "Window Station" is like a DISPLAY device in X
- Theoretically multiple "Window Stations" could be attached to a Windows NT box to provide multi-user service.
  - Cytrix and others hacked the NT kernel to do this.
  - API's to control desktops and Window Stations live in NTDLL.DLL. Not exposed. VNC could use these.
- Windows NT services (equivalent to daemons) also have their own Window Stations.
  - This is poor design due to Windows "message pump" programming. Should not be needed for daemons (daemons don't use X for messages).

Little known NT facts (continued).

- The Windowing system was originally designed to be remotable, in NT 3.1 -> NT 3.51.
- User threads making graphics requests were paired with a thread in the CRSS.EXE process, that would access the hardware on their behalf (like a MT-X Server).
- Shared memory calls (LRPC) were used to transfer graphics requests to the paired CRSS.EXE threads.
  - Context switch still required.
  - There is a Win32 batch flush call, like XFlush().

Windowing in the kernel - a good idea ?

- NT 4.0 put large parts of the CSRSS.EXE into the kernel for desktop speed.
  - This was the point that NT stability became a real issue.
- "Inside NT", 2nd Edition claims this had no effect on stability, as a CSRSS.EXE crash always caused a system reboot, as the Win32 display is essential even for services (daemons).
  - This is untrue. NT prior to 4.0 could have a display problem and file and print services would keep on running, NT4.0 and above will blue screen.
  - Samba became very popular around NT4.0.....

Windows NT configuration - the Registry concept.

- The Registry concept (IMHO) is a very powerful one.
  - Allows all apps to get/set name/value pairs that control system configuration using a (reasonably) simple API.
  - Security (complex ACLs) can be set on each value individually. User account info (passwords) stored here.
  - Maintanence of UNIX /etc files programatically is a nightmare in comparison. No common API, no common data format. Historical reasons.
  - Look at Linuxconf and the KDE configuration editors. They are trying to map the registry complex onto a nest of files that have different formats.

The NT registry (continued).

- NT equivalent of /proc file system appears in a registry area. Name/value pairs can be created on query (this is how NT perfmon works).
- The problem with the registry is that the implementation sucks !
  - Uses MS Jet database engine. Same engine as used in MS-Access that regularly corrupts Exchange and WINS databases.
  - Each registry "hive" (chunk-o-data) is a single Jet db file stored in WINNT\System32\config.
  - Old Windows 3.1 '.INI' file mechanism actually was more robust, as problem files could be fixed by hand.

UNIX User and Group account mechanisms.

- UNIX has a simple 32-bit user id to represent any user. Likewise a 32 bit group id to represent a group. User and group number spaces are disjoint.
- "Foreign" users (from another machine) cannot be distinguished from a local user if their uid is the same.
  - YP/NIS and NIS+ are simply a way of getting a group of UNIX machines to agree on a common mapping for these numbers to user/group names.
- On-disk storage of ownership is only the owning uid and gid.

Windows NT User and Group account mechanisms (SIDs).

- NT uses a "SID" (security identifier) to store user and group identities. NT machines also have SIDs.
- SID number space is flat (users / groups / machines) all allocated from the same number space.
- Unlike uids, SIDs have a complex structure. A typical SID looks like :
  - S-1-5-21-<32 bits>-<32 bits>-<32 bits>-<32-bits>
  - The first '1' is the revision level of the SID.
  - The '5' is the "indentifying authority" (ie. who created it). 5 means an NT system (Samba uses this also).

NT SIDs (continued).

- The '21' is the sub-authority. 21 means accounts created by the Administrator (ie. not built in accounts). Well, mostly :-).
- The next 96 bits are a Domain or Machine ID.
  - All NT machines have a 96 bit unique identifier. This fact is significant and will be covered in a later slide.
- The final 32 bits are a "RID" (relative ID). This is the actual user or group ID within the 96-bit unique identifier.
- The overall SID design is unusably complex (and almost no programmers understand it).

Windows NT Domains.

- An NT Domain is a collection of machines that can share a single account database.
  - Almost identical to a YP/NIS setup. NIS+ is a closer match due to the crypto and machine accounts.
- The machine holding the primary account database is called a PDC (Primary Domain Controller). This is close to an NIS master.
  - BDCs (backup domain controller) contain read-only replicas of this database. This is similar to an NIS slave.
- PDCs and BDCs have no separate local account database (ie. no local /etc/passwd or /etc/group).

Windows NT Domains (continued).

- The 96-bit machine identifier in a SID is used as the Domain identifier when the machine issuing the SID is a PDC.
  - NT accounts are usually written in text form as : DOMAIN_NAME\user_name.
  - The "DOMAIN_NAME" part represents the 96-bit domain identifier.
  - The "user_name" part represents the 32 bit RID within that domain.
- Non PDC machines can also have local accounts and groups: MACHINE_NAME\user_name
  - Equivalent to a local /etc/passwd or /etc/group.

Windows NT Domains (continued).

- A users SID completely identifies the creating machine/domain, as well as the user within it.
  - As SIDs are stored in NT ACLs and as the owners of files/directories (in NTFS) then there are no problems with uid/gid collisions from different machines.
  - Due to the flat number space a Group SID can be the owner of a file (ie. groups can own files instead of users).
  - To perform a SID to name lookup for an unknown SID, the machine/domain identified by the 96-bit machine id must be on line.
- The passwords associated with Machine SIDs are used to create a secure channel (more later).

Windows NT Security Model.

- When a user logs onto an NT machine, their SID, as well as a list of the group SIDs they belong to, are stored in a kernel data structure known as an "access token".
  - An access token is associated with every process.
  - Threads share their owning processes token, unless they are impersonating a user (more on this later).
  - This is identical to the process credentials in the Linux task_struct structure.
  - This group list is fetched from the account database (local account) or from the PDC/BDC (domain account).

Windows NT Security Model (continued).

- Win32 has no generic setuid call.
- The MS security model is predecated on a server process receiving an IPC communication from a process running under another user context, and allowing the server process to impersonate the caller.
- MS claim is that the super user (Administrator) cannot arbitrarily take on another users context.
  - This is untrue. Given access to the users hashed password stored in the registry (the equivalent to the password field in /etc/passwd or /etc/shadow), a user can be arbitrarily impersonated.

Windows NT Security Model (continued).

- All (NT4.x and below) NT security is based upon the NTLMSSP challenge/response protocol.
  - This includes SMB/CIFS, RPC (including all remote admin services for NT and BackOffice apps such as Exchange, IIS, SQLserver), MS-CHAP, the Outlook POP/IMAP services, their PPTP VPN solution and DCOM.
- NTLMSSP consists of an 8 byte server challenge being DES encrypted by the hash of the users (or machines) password and returned.
- This protocol is tunneled onto other protocols (IMAP) using SASL and other Internet standards.

Windows NT Security Model (continued).

- Once the NTLMSSP exchange is completed, the receiving kernel creates an access token representing the 'remote' user.
  - The Domain/Machine and user names are included in the NTLMSSP exchange so the user SID and group SID list can be looked up by the receiving machine and set inside the kernel access token.
  - This is the token that can be attached to a thread by use of the "ImpersonateXXX" Win32 API calls.
  - Under Linux, a kernel thread can also do a seteuid that will only affect itself (not the owning process). However this is not defined by the pthreads standard and should not be relied upon for portable code.

Windows NT Security Model (continued).

- NT Domains can be a 2 level heirarchy (one domain "trusts" another).
  - NTLMSSP authentication over encrypted MS-RPC is used to pass the challenge sent from the issuing server and response from the client up to the PDC/BDC, which then forwards it to the trusted PDC/BDC.
  - This is most similar to Kerberos 5 (which explains the Win2000 choice).
  - Fundamental flaw designed into the NT Domain security model (known initial password) makes NT Domain security very weak.

Comparing UNIX and NT "pluggable" authentication.

- Linux provides a replacable login program, and also replacable "PAM" modules underneath it.
- Linux also allows the user and group password database to be replaced via the nsswitch mechanism in libc (this is how YP/NIS is done).
- Microsoft claim that the "GINA" (Graphical Identification and Authentication) DLL mechanism is also a replacable module of equal power.
  - This is untrue.
  - GINA allows additional authentication to be done at "interactive" logon time only. You cannot replace the token creation function at the GINA level - an MS underlying mechanism must be in place.

The GINA shell game :-).

- The real authentication mechanism in NT is a .DLL known as the LSAauth (Local Security Authority authentication). This is also a pluggable mechanism.
  - It is intentionally undocumented.
  - There is a mandatory DLL here, called MSV1_0 subauthentication DLL. No documentation on replacing this exists.
  - My speculation on the reason for this is that an open LSAauth API would allow an NIS or NISplus authentication mechanism to be integrated into Windows NT, such that an NT PDC would no longer be needed in most UNIX environments.

Windows NT Access Control (NT ACLs).

- All securable 'objects' in NT can be protected with an Access Control List (ACL).
- Under UNIX all securable 'objects' are represented in the filesystem (mostly, SysV shared memory is an exception) and use the file system security user/group/world to control access.
- NT has separate namespaces for all securable 'objects' that may be used for IPC. Often, inconsistant Win32 calls are used to set/get security information on them (bad API design).

NT ACLs (continued).

- Securable 'objects' have a structure called a "security descriptor" attached to them. It contains an owner SID and group owner SID (for POSIX) and two ACLs, one to determine access (DACL), one to determine auditing (SACL).
- NT ACLs can be arbitrarily complex, but they have a canonical form (deny list first, followed by allow list).
- It is prgramatically easy to create file ACLs that the Explorer GUI cannot display.

NT ACLs (continued).

- Each entry in the ACL (Access Control Entry - ACE) contains a type (allow or deny), a bitmask specifying what permissions are covered, and a SID to which these permissions apply.
- As they can be any length dealing with them programatically is difficult.
  - They have two forms, an "in memory" form, and a linearized form known as "SELF RELATIVE" that is used to store them over the network or onto permanent storage.
  - There are two sets of bits in the bitmask, a "generic" set and a "object specific" set. The generic set is mapped to the specific set by an API call.

NT ACLs (continued).

- They are extremely flexible (maybe too much so) and can be set to be inherited on creation of an 'object' within a 'container' (this usually means a file or directory within a file system).
- POSIX ACLs are unfortunately not a standard over any of the UNIX varients. This makes writing portable code very difficult.
- However, NT ACLs are not implemented on Win9x or WinCE and mostly ignored by Microsoft applications (MS Office) so portable Windows ACL code is also very difficult.

Remote Access and Administration.

- The UNIX remote administration mechanism is designed around the philosophy of providing a remote tty interface.
  - This is extremely powerful - no GUI tools needed, remote administration can be done over very low bandwidth links.
- NT remote administration is all designed around MS RPC calls (more on these later).
  - By NT4.x I believe all needed administration can be done remotely (even loading of new services onto a remote machine).
  - GUI tools designed around permitted RPC calls - if no call was designed, the operation cannot be done.

Remote Access and Administration (continued).

- Problems with the NT model are that a large set of functionality needs to be working continuously for remote admin to work (networking stacks, RPC binding mechanism, RPC services).
- In practice NT remote admin fails to deliver.
  - The most useful NT remote admin tool is a car :-).
- Luke of the Samba Team is developing a UNIX command line remote admin tool, sending MS RPCs to NT servers.
  - He got re-boot working really early :-).

Comparing UNIX and Windows NT Remote Proceedure Calls.

- UNIX RPC is mostly based on Sun's ONC/RPC spec.
  - Uses "network" (big endian) data format.
  - RPC servers allocate a transient port and register it with a "portmapper" daemon.
  - Although security was designed in, it is little used and UNIX 'auth' security (uid/gid in packet) is usually used (ie. No security, such as NFS).
- MS-RPC is mostly based on the OSF DCE/RPC spec.
  - Uses receiver-makes-right data format.
  - "Extended" by MS to use NTLMSSP security.

Remote Proceedure Calls (continued).

- DCE/RPC had a Kerberos based security authentication mechanism that MS ignored.
  - Even in Win2000 with Kerberos 5, MS are ignoring the DCE/RPC security spec. and modifying Kerberos 5 to suit their own purposes.
- MS-RPC can be tunneled over many different transports.
  - "Raw" TCP/UDP on port 135.
  - Inside SMB/CIFS on port 139 (Samba implements this). This uses the \\SERVER\IPC$ connection followed by a named pipe bind.
  - NetBEUI, IPX/SPX etc.

Remote Proceedure Calls (continued).

- MS-RPC uses NTLMSSP to provide "signing" (verification) and "sealing" (encrypting of packets). Usually 40 bit only for international use.
- RPCSS.EXE is the NT equivalent of the UNIX ONC/RPC portmapper (provides end-point mapping services).
- DCOM communication is done over an MS-RPC transport.
- As the standard NTLMSSP mechanism is used RPC servers can impersonate the client security context in the same way as file serving is done.

Comparing UNIX and NT Remote file access.

- UNIX uses the familar NFS - love or hate it.
  - Uses ONC/RPC packet format.
  - Server exports an area of the file system, client mounts it by asking for a pathname.
  - Automounter heavily used to provide location transparancy (why should I care what server my home directory is on).
  - Simple design - no security, "stateless".
- Windows NT uses SMB/CIFS.
  - Uses packet format from hell (39 byte header !).
  - Server associates a name with an exported area of the file system, client mounts it by name.

UNIX and NT remote file access (continued)

- SMB/CIFS uses (you guessed it...) NTLM security.
- Microsoft DFS is the equivalent of the automounter. Not yet widely used.
- Remote access directly achieved by use of UNC pathnames : \\SERVER_NAME\share_name\path.
  - Problem with this is "SERVER_NAME" embedded in the path. What happens if the data is moved ?
  - Samba neatly solves this problem with NetBIOS aliasing.

UNIX and NT Remote file access (continued).

- SMB/CIFS is very stateful. Clients are expected to remember enough state to reconnect.
- UNIX (POSIX) locking semantics are far richer than Windows NT.
  - This means UNIX can easily serve SMB/CIFS locking, Windows NT cannot possibly serve POSIX locking.
  - POSIX locking allows lock range splits/merges, lock upgrades/downgrades. Windows locking is simple exclusion.
- UNIX NFS locking is usually so broken that no applications depend on it, so it's usually not an issue.

Location transparancy (NT Profiles).

- In UNIX, user configuration information is stored in dot files and directories in the home directory.
  - Automounter is used to give location transparency on login (if I can get to my home directory, things should just work....).
- In NT user configuration information is stored in files on a server called the user "profile". The path to this (UNC) is stored in the user account database.
  - One of these files represents the HKEY_LOCAL_USER Registry hive. It is loaded into the computer the user is logging onto over the network.

NT profiles (continued).

- Any changes made are shipped back over the network the "profile path" on logout.
- What is worse, is that anything stored on the users desktop is stored in total back onto the profile server.
  - Think about what happens if a user drag-and-drops an MS Word icon onto their desktop, rather than creating a "shortcut" (symlink) to it.....
- A Windows desktop environment is extremely fragile. Many desktop technicians needed to maintain a large Windows NT network (high cost of ownership).

Internationalization (I18N).

- Windows NT has a very good internationalization model.
  - Internally the kernel is all 16-bit MS-Unicode.
  - All system calls are actually Unicode with a codepage to Unicode translation layer for legacy applications.
  - All remote RPC strings are Unicode.
  - NTFS File system stores all names as 16-bit Unicode on disk (this doesn't help with FAT of course).
  - All users/groups/passwords are in Unicode.
  - NT-version of SMB uses Unicode over the wire. Allows a true multi-lingual file server.

I18N (continued).

- Linux I18N model is poor by comparison.
  - Kernel is 8-bit clean - left to applications to determine what meaning filenames have.
  - Locale model allows different systems/environments to be in different languages, but the global
  - Users/groups/passwords are in locale format.
  - UTF-8 (Unicode into multiple 8-bit characters) is meant to solve these issues.
- The main problem comes from the mixing of "legacy" apps and systems that use local character sets and newer, UTF-8 based systems.
  - No good solution proposed as yet. NFS ignores this issue.

References.

- Inside Windows NT (2nd Edition): Microsoft Press.
- Windows NT Security guide: Stephen A. Sutton.
- Inside the Windows NT File System: Microsoft Press.
- Microsoft RPC programming guide: O'Reilly.
- Distributing Applications across DCE and Windows NT: O'Reilly.
- And lastly... The Samba source code (http://samba.org).