

[0063] Actually, after the domain is identified, the client can interact directly with the service provider associated with the appealed domain, i.e. without any intermediation.

Independent User Naming System

[0064] According to the prior art, a hosting computer that hosts the domains domain1.com, domain2.com, and so forth, cannot have two or more users with the name sysadmin, for example.

[0065] According to one embodiment of the invention, in order to enable an independent user naming system for each domain hosted on a hosting computer system, the domain name is embedded in the user name. Referring to the previous example, instead of identifying himself by the user name, i.e. xxx, a user identifies himself as xxx%aaa.com, xxx%bbb.com, and so forth.

[0066] Practically, on issuing a USER command according to the standard FTP protocol, the user identifies himself by a name wherein the domain is embedded, which according to this example is USERNAM%DOMAN, i.e. the user name followed by the character “%” and the domain name.

[0067] Of course, this syntax does not suit the standard protocol of the FTP as defined in RFC 959, and therefore an additional step should be carried out, as described herein-after.

Hosting a Plurality of Domains by One Computer System

[0068] According to one embodiment of the invention, the FTP process is modified in order to support this format.

[0069] According to another embodiment of the invention, a component that handles the communication part is added to the communication chain. This component handles the connection to a third party’s FTP server (i.e. the owner of the computer can use any FTP server—open source, commercial, or even homemade) when the user is identified. For the sake of brevity, it is assumed that this component is a process. This process is hereinafter called FTP-wrapper or wrapper.

[0070] When a connection to the FTP port arrives to the hosting computer, the daemon creates an FTP-wrapper process. The FTP-wrapper “negotiates” with the client as an FTP server, i.e., receives commands from the client and relays answers to the client as the FTP server. The negotiation is carried out until the client issues the “USER” command.

[0071] It should be noted that the wrapper might support only a subset of the FTP commands, as most commands are not supported before the user logged into the FTP server. Once the FTP-wrapper received the “USER” command, it confirms that it includes the user name and a valid domain name, and starts an FTP server for that domain.

[0072] It should be noted that in order to achieve better performance, the wrapper might be integrated into the daemon, so the creation of the wrapper process for each connection is unnecessary.

[0073] Once the original FTP server is created, it expects to have a communication from the beginning, including the “USER” command, and any prior commands that the user issued.

[0074] According to one embodiment of the invention, the wrapper process is kept active for the entire FTP session, thus acting as an interface between the client and the FTP server—every request for service is passed to the FTP server, and any result from the server is passed to the client.

[0075] A Shared library (or DLL in Windows operating system) is a collection of functions, usually related to some specific subject, that are kept in an independent file accessible to all the processes at run-time.

[0076] According to a preferred embodiment of the invention, the solution to the above-mentioned problem is based on that fact that most Unix-based processes are linked to shared libraries. The benefits of this approach are:

[0077] Each process is smaller, as the common functions are not part of the process itself. This requires less disk space for storing the program, and less disk space for distributing it.

[0078] Since the operating system can load a single copy of the library into memory and use it for several processes, the amount of memory required for several processes that use shared libraries is less than the amount needed for the same processes when executed without shared libraries.

[0079] In order to include new features in an existing function, correct bugs or correct security holes, new versions of the functions are developed. Upon installing a new version of a function in a shared library, processes being loaded at run time refer to the new version of the function, while other processes do not.

[0080] According to one embodiment of the invention, a new shared library that replaces the relevant functions of the original socket-library is activated, but the original socket-library is retained in order to be used later. A function of the new shared library performs some additional operations (which are not a part of the original function), and then activates the original library’s function with the same name.

[0081] This mechanism is well-known in the art, and is referred to as hooking. A hook is a place (and usually an interface) provided in packaged code that allows a programmer to insert customized programming, such as additional features.

[0082] According to one embodiment of the invention, the hooking is carried out as follows:

[0083] A buffer is provided to each socket, for retaining temporarily the information received from the client.

[0084] During the operation, if the buffer is not empty, “read” commands read the data from this buffer, and if the buffer is empty, then the “read” command retrieves the data from the socket.

[0085] Any “write” command ignores the data until the “read” buffer is empty. After that, all the information is transferred to the socket. Whenever the process using the library performs a “write” command, the library checks the status of the internal buffer. If the internal buffer comprises any information, the information passed to the “write” command is ignored and a “success” status is returned to the caller, as this information was already handled by the wrapper. If no more data is present in the internal buffer, the information is passed to the normal sockets library.