

[0048] It is important to characterize the base similarities and differences, as they exist for each device driver class, to ensure the present invention can correctly function. It is not truly desired to load and unload device drivers for system hardware that is constantly present. It should be understood that although this is not a preferred embodiment in terms of programming ease, it is within the scope of the present invention and may be required for specific reasons, such as the restriction in licensing agreements for applications that are delivered and run using the present invention.

[0049] On non-Microsoft platforms, device drivers are typically handled very differently. Macintosh systems support both static and dynamic drivers, but they are all installed and removed through the same method. Linking with the Macintosh system folder will provide the necessary support. For UNIX systems, device drivers most typically require a modification to the running UNIX kernel, followed by a reboot. This process can be very complex. In the preferred embodiment, this process is automated; including resetting the kernel once the application is complete. The general parameters of the process are the same as that described above for Windows applications, the actual process steps of compilation and persons familiar with such operating systems can carry out reboot.

[0050] Finally, those of skill in the art will understand that it is desirable to be able to recover and remove drivers across system failures. Whatever data or processes necessary to retain system integrity are therefore a preferred embodiment of the present invention. Those of skill in the art will also appreciate that all types of device drivers might not be conveniently or efficiently provided via the present invention, most particularly those associated with permanent hardware attached devices.

[0051] Other Items

[0052] In the present invention, it is recognized that there are several components of the invention, the behavior or presence of which is different on alternate operating systems. These components include fonts, processes, environment variables, and others.

[0053] Some applications require fonts to be installed in order to perform correctly. Any fonts required will be specified in the Operating System Guard's configuration file. The Operating System Guard will enable these fonts prior to running the application and if necessary remove them afterwards. Most systems have a common area for storage of fonts in addition to a process for registering them or making the system aware of their presence, the Operating System Guard will utilize these available methods.

[0054] On Windows, a font is copied to the \WINDOWS\FONTS directory. This however does not guarantee that the font is available to the running program. In the preferred embodiment, if the program uses the Windows API to access fonts, the font will need to be registered with a Win32 API call such as CreateScalableFontResource/AddFontResource. This will insert the font into the system font table. Once complete, the Operating System Guard can remove the font with another appropriate API call like RemoveFontResource, then remove the file from the system. As an alternate embodiment, the Operating System Guard could hook the API functions as described in the virtual registry method. In addition, the Operating System Guard can use its File subsystem to avoid placing the actual font file in the running system.

[0055] On Macintosh, the process is extremely similar and based on files in the Macintosh system folder and registration activation. On UNIX, however, the process is dependent upon the application. Most typically, font resources are added to the system as regular files resolved in the proper location, so they can be accessed by name. With many Motif systems, a font description needs to be placed into a font resource file, which will allow the font to be resolved. The Motif or X application can invoke the font either through the resolution subsystem or by a direct call. Recently, many Motif and CDE based systems utilize Adobe scalable postscript fonts. These fonts need to be managed through the Adobe type management system. There are exceptions, however, and as stated above, there are alternates to the Windows or other operating system default font management systems. The Adobe Type Manager provides some alternate interfaces for this process, as do other third party type management systems. In most cases it should be decided whether to support the interface or ignore it. The purpose of Operating System Guard is not to provide a universal layer for all these systems, only to do so for the operating system's own subsystem.

[0056] Many applications require environment variables to be set. This is most common on UNIX systems, but is also heavily used by software, which was originally written on UNIX and ported to the Windows operating systems. Applications on the Windows operating systems heavily rely on the DOS PATH environment variable and often set their own application specific entries. On the Windows 9x/Me environments, there are many environment settings, which are applicable as at its core is the DOS subsystem. If an application requires the presence of specific variables, or values to be set in existing environment variables, the required environment variables will be specified in the Operating System Guard's configuration file. The Operating System Guard will set these variables for the application's main process when it is launched. As applications do not typically change environment settings as they operate, the virtual environment will not trap these calls, nor will it provide the full complement of functionality that the registry and configuration subsystem does.

[0057] Recovery

[0058] In some cases shown in the previous sections, actual modifications must be made to the operating system. This is frequent with device drivers and fonts. In addition, changes can be made to the virtual environment that need to be persisted and available the next time an application is run. It is required that the Operating System Guard system be able to recover from changes to the system, removing the change from the system at its earliest possible opportunity. Alternately, if the system crashes during an application's execution, the Operating System Guard should track enough information to remove any change to the system if it is rebooted or otherwise, and should track the changes made to the virtual environment. In the preferred embodiment, this is implemented as a transaction log, but can in other embodiments be done as some other similar component, which can be read on system startup so that changes can be backed out.

[0059] Controlling Virtualization

[0060] An important aspect of the invention relates to control of the many facets of virtualization which the Operating System Guard is capable of. In the preferred