

Max OS X 10.4's Spotlight Desktop Search Tool

Mac OS X 10.4 Tiger features a new search tool called Spotlight. Similar to Google Desktop, for example, Spotlight is capable of searching through a variety of local objects: the usual files and folders, plus e-mails, contacts, calendar events, etc. Furthermore, searches can be quite complex (albeit not by default) and be based on a plethora of attributes. However, the most interesting aspect of Spotlight is the means by which it achieves performance by integrating with the kernel.

In order to perform efficient searches, Spotlight maintains indexes on the relevant objects' metadata. When a search query is executed, it can consult the appropriate index instead of traversing the catalog or, worse yet, the actual data. However, this approach introduces the problem of a possibly "stale" index. Changes may be made to the file system's contents but the affected files may not have been re-indexed prior to executing another query.

One approach is to address the issue at the user-level. This may seem natural since the indexes themselves are first created by a Spotlight indexing process. Thus, perhaps we could implement a separate, persistent process that monitors for changes and updates the indexes accordingly. However, this does not genuinely solve the problem. Assuming that the process makes it check every certain interval of time, the staleness problem still remains. Furthermore, CPU cycles are wasted each time we run the check.

Rather, in order to provide efficient, real-time updates to the indexes, Spotlight also works at the kernel level. In particular, the metadata are gathered by "importer plug-ins" that are associated with each file/object type. When a file I/O occurs, it normally goes through the kernel. In addition to handling the actual file I/O, the kernel also calls the appropriate importer plug-in, which handles any changes and updates the index. Thus the indexes are generally kept up-to-date in real-time. Other changes that do not go through the Tiger kernel are updated via an indexing process.

This scheme of integrating the indexing with the kernel was in response to a desire for performance when searching/indexing and maintaining up-to-date indexes. Indeed, doing so provides both good performance and a "consistent" set of indexes (in the sense that they always reflect the actual state of the file system). We do lose some performance when doing file I/O (in particular, any modification) since the kernel also has to invoke the importer plug-in. However, that is probably negligible compared to the cost of doing the indexing via a background process. Accordingly, we thus maintain more resources for actual user-level applications.