LocalFile Implementor's Guide
Filed as [eris]<lispcore>internal>doc>localFileImpl.tedit
Written by Tayloe Stansbury 18-Feb-85
Modified by Tayloe Stansbury 15-Aug-85 12:27:46

The Dandlion/Dove local file system is implemented in two files: LocalFile and DskDisplay. User-level documentation for the file system can be found in the 1108 User's Guide or on [eris]<lispcore>doc>localfile.tedit.

The Dandelion/Dove local file system has two layers: the Pilot layer and the Lisp streams layer. The Pilot layer emulates a subset of the Pilot file system, as described in the Pilot Programmer's Manual. The Lisp streams layer implements the Lisp streams specification laid out in [eris]<lispcore>internal>doc>streams.tedit.

The Pilot layer is implemented by three modules in the file LocalFile:

1. LFALLOCATIONMAPCOMS, which keeps track of which pages have been allocated and which are free. LFAllocationMap provides the functionality of [idun]<apilot>11.0>pilot>private>volallocmapimpl.mesa, though its implementation is only very loosely based on that file.

2. LFFILEMAPCOMS, which keeps track of the mapping between file ID numbers and runs of disk pages. This mapping is stored in a specialized B-tree. LFFileMap provides the functionality of [idun]<apilot>11.0>pilot>private>volfilemapimpl.mesa, though its implementation was based more on [idun]<apilot100>pilot>private>volfilemapimpl.mesa, and later updated to be compatible with the Mesa 11.0 release of Pilot.

3. LFPILOTFILECOMS, which has a primitive notion of file, as embodied in its datatype FileDescriptor. LFPilotFile handles things like creating, extending, shrinking, and deleting files; reading and writing file pages; labels; and volume root directories (which map file types onto higher level directories -- e.g. Lisp file type -> Lisp directory ID, Mesa file type -> MFile directory ID, etc.). LFPilotFile does not emulate any particular Mesa file, but rather grew up as the gray area between the two layers became more well-defined during the evolution of the Lisp local file system.

The Lisp stream layer is defined by three more modules in the file LocalFile:

1. LFDIRECTORYCOMS, which implements the Lisp directory. The Lisp directory maps symbolic Lisp file names onto Pilot file ID numbers, and handles directory search and directory enumeration.

2. SCAVENGEDSKDIRECTORYCOMS, which implements a scavenger for the Lisp directory. It works by purging the old Lisp directory, creating a new one, using the BTree to figure out what Lisp files there are on the volume, using the leader page of each Lisp file to figure out what its name is, and then inserting an entry in the new directory for each Lisp file. There is no Pilot-level scavenger implemented in Lisp; for that we rely on the Othello Scavenge Logical Volume command.

3. LFCOMS, which implements all other operations of the local disk file device. LocalFile uses Pilot files as backing files for Lisp streams: page 0 of the Pilot file becomes the stream's leader page (containing stuff GETFILEINFO and the scavenger will be interested in), page 1 of the Pilot file becomes page 0 of the stream, etc. Pilot backing files may be longer than the Lisp stream they hold.

In addition, the file DskDisplay provides a window which displays file system status. This file is separate because it relies on the window system and therefore must be loaded considerably later in the loadup process than LocalFile need be.

Some future projects for the file system (apart from fighting off the stream of ARs):

1. Modify the READPAGES and WRITEPAGES methods to transfer contiguous pages all at once, and set the MULTIBUFFERHINT to be T.

2. Rewrite DiskDlion (which implements the Dandelion/Dove disk head) so that disk requests that cross cylinder boundaries are handled in runs rather than a page at a time (for both the Dandelion and Dove). Without this, large disk requests (which happen especially during deleting) can tie up the system for quite a while.

3. Modify DiskDlion to do a process switch while waiting for the disk. Currently it busy waits.

4. Currently too much of the file system is uninterruptable. Unfortunately, the primitive UNINTERRUPTABLY now prevents process switch in addition to preventing keyboard interrupts. What we really need is a construct that will do the latter but not the former. (There is now an AR for such a beast.) Given that, we should remove as many calls to UNINTERRUPTABLY as possible.

5. Currently files are allocated 20 pages at a time, regardless of whether the openfile request came with a size hint. This was originally because allocating long files sometimes took long enough that NS connections got dropped. Once #2 is accomplished, long allocations should be quite a bit faster though; and #s 3 and 4 will make it possible for other processes to sneak in while allocations are going on. Then it would make sense to change the file system to allocate files all at once when a hint is provided.

6. Currently, allocation map buffers and file map buffers are written out only once per stream creation or deletion. Without automatic built-in scavenging, that strikes me as a bit unsafe (although it does speed things up some). It would probably be better to write them out once every allocation or deallocation (and the performance penalty for doing so will not be too great if allocation requests are larger as a result of #5).

7. Longer term: rewrite the directory so that it uses some form of tree search. The linear search currently used gets unacceptably slow for large directories.

Should you have any questions, do not hesitate to contact me. My mail address is Stansbury.pa, and my extension is 4330. Good luck!