

Objects, Objects, Objects

by Dave Gilman

Message from the editor:

The articles in this issue of the DosGetNews() contain a lot of information about various projects happening in the OS/2JDA Group. It is suggested that you sit down and enjoy it, as opposed to trying to read it "on the run." Thanks to this month's contributors: Dave Gilman, Leif Pederson and Janine Harris.

As some or all of you have heard, Microsoft is embarking on a new technology called *object oriented programming*. The cynical amongst us would claim that it is nothing more than the '90's version of 'structured programming'. Those that have been deeply involved over the last few weeks believe that it is the only way to achieve Microsoft's long term goals:

1. Better end-user support through an advanced GUI (i.e. CUA 3).
2. True application interoperability.
3. Increased programmer productivity through the use of object oriented techniques.

Concepts

Three of the main concepts behind object oriented programming are *encapsulation* (data hiding), *inheritance* (code reuse) and *polymorphism* (overloading). The obvious question is, what do these three buzz words actually mean?

Encapsulation is basically a way of viewing the relationship between code and data. Today, code is written to operate on a known data format (e.g. an Excel spreadsheet); objects allow you to encompass that knowledge. The capabilities of an object are presented through an interface where you ask the object to do something (e.g. print itself) rather than operating on the data directly. For example, including an Excel spreadsheet in a Word document would involve asking the Excel spreadsheet object to include itself in the Word document. This is in contrast to today's process of Word having to be familiar with the format of an Excel spreadsheet file in order to read or display it.

Have you ever found yourself writing code only to discover that you're rewriting something for the nth time? Inheritance solves this problem! By carefully partitioning a problem you could reuse code by inheriting from another object. For example, consider the following objects (including their characteristics) and the relationships between them:

- *Microsoft Employee*
 - o *Well paid*
 - o *Nine to five work day*
 - o *Based in Redmond*
- *Developer*
 - + *Inherits from MS Employee*
 - o *Able to code in C*
- *Development Manager*
 - + *Inherits from Developer*
 - o *Writes reviews*
- *Sales Representative*
 - + *Inherits from MSEmployee*
 - * *Overrides Based in Rdmd w/,*
 - o *Based in New York*

Notice that in the case of the Sales Representative, we inherited from Microsoft Employee but then replaced (overrode) one of the characteristics (i.e. 'Based in Redmond' with 'Based in New York') rather than adding a new one.

Last, but certainly not least, we have polymorphism. This one is a bit trickier, but what it means is that you can apply a single operation to different objects and have them both react appropriately. If you want to 'add' object 'a' with object 'b', the method (procedure) that gets invoked to perform an 'add' will be different depending on the object type of 'a' and 'b' (e.g think about integers (addition) and strings (concatenation)).

The Vision

So, what is Microsoft going to do with all of this whizzy new technology? Simple: we are going to achieve the three goals mentioned above. That is what makes the objects project fascinating. It is a long term, cooperative effort between all areas of the company. The OS/2 group will implement the first design of the architecture. System languages will support it with C++ (and other) language technologies. And applications will make use of it in the next generation of Microsoft applications.

The Plan

Stay tuned to DosGetNews() for details as to how Microsoft is going to pull this off (you know, same BatTime, same BatChannel...).

PINBALL-IN-C

by Leif Pederson

Pinball is the High Performance File System for OS/2 1.2 and beyond. The chosen product name, High Performance File System (or HPFS), is indicative of Pinball's primary goal: to be fast! Performance together with long name and extended attribute support are probably Pinball's most important features. This article will briefly cover some history and some interesting features, look at performance, and describe Pinball internals. Lastly, it will cover differences in our build environment and give some project status.

History

The Pinball-In-C (PIC) product is based on Gordon Letwin's work. Gordon architected and designed Pinball and is currently working with a team of developers in the net group on a 386 assembler version initially targeted to be a LAN-MAN file server product. Last October, the performance of Pinball was compared to that of an IBM file system implementation leading to the decision that Pinball would be the installable file system shipped with OS/2 1.2.

Pinball-In-C started back in June of '88 as a translation of the ASM code that existed at that time. Because it is our primary goal, we have carefully monitored performance on a regular basis. After intensive performance tuning just before we entered public build in November, we were only 2 to 16 percent slower than the ASM version's elapsed times on a suite of six scenarios—the very same “fly-off” tests used earlier. The performance hit was probably due to an overhead of writing in C (but the compiler is pretty good), and the fact that

our compiler is generating 286 code! Needless to say, PIC came out much faster than FAT.

Performance

The fly-off tests mentioned above were pulled together quickly in order to get some feeling on how Pinball would do under some real-life scenarios. Measurements of PIC vs. FAT have been done periodically and the most recent available are shown below, courtesy of Russ Blake's group. The tests are listed with an indication of what the test does along with the elapsed time relative to FAT. Numbers were taken on build 12.71 except for MakeC which was on 12.60 using a PS/2 model 80-071, with a 400 KByte cache.

Editor (file copy, delete, rename)	77%
Attrib (FindFirst/Next, get attribs)	63%
Xcopy (directory tree copy)	47%
Rbase (random read/write)	45%
Lanmc	55%
MakeC (C compiler/make)	86%

Additionally, we have some favorable results from a test which performs random I/O in a 3 megabyte file. For random writes using the 12.71 build, PIC's average time for DosWrite of 256 bytes compared to FAT looks favorable: 34 ms for PIC vs. 61 ms for FAT.

In the course of the next few weeks Russ Blake's group will be collecting more data on PIC's performance in a variety of tests.

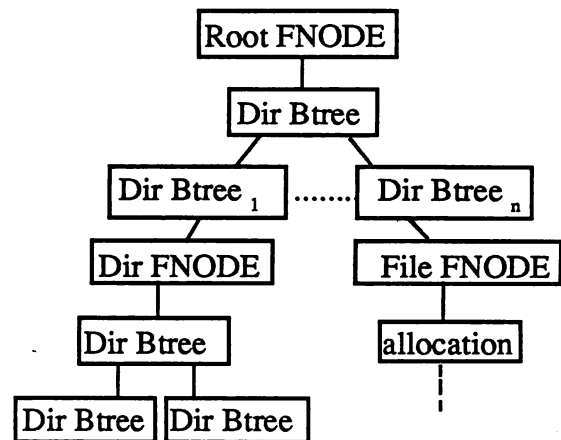
Features

Besides performance, extended attribute support and long names are important aspects of Pinball. Pinball allows a file name to be up to 254 bytes long and supports the same set of allowed characters in file names as FAT. However, the '.' character has no special meaning in Pinball. For example, "MY_PERSONAL_FINANCES.1-1-89.XLS" and "CRACK_PATH_NAME.C.DIFF" are perfectly valid file names.

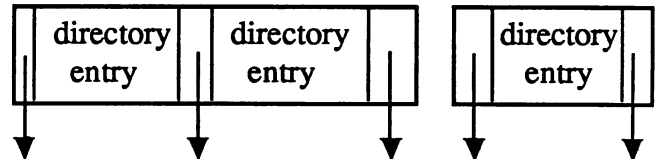
A file's extended attributes are a second data stream that is stored logically as name, value pairs. The total length of all extended attributes for a file can be up to 2 GBytes. APIs exist that allow a program to set or retrieve an extended attribute by name. Uses for EAs are up to the application but typically might be to store information that describes the file (such as an icon, or a description of the contents of the file or author); the name of a program (or the program itself) that allows the file to be viewed; or possibly even digitized voice that corresponds to certain sections of a document used in a voice-annotated-text application.

Internals Overview

In Pinball, every file and directory (including the root) has a disk data structure, known as an FNODE, that contains information such as extended attributes, and a pointer to either the file allocation or the B-tree holding the directory entries. The layout is shown below.



Logically, a B-tree node looks like this:



Each down pointer points to another B-tree node. Directory entries are ordered such that entries in the node pointed to by the left down pointer are lexically less, and entries in the node pointed to by the right down pointer are lexically greater than the entry itself. This results in an ordered set of records returned from FindFirst/Next calls. For file names of length 13, over 175,000 files would have to be created in the same directory before the B-tree would grow beyond 3 levels! Directory entries not only contain the name of the file or subdirectory, but hold the attributes, modification and access times, and extended attribute size as well.

Much of the performance in Pinball is achieved through its caching. There are, in fact, three levels of caching. At the lowest level, of course, is the disk-block cache consisting of a collection of 2KByte buffers.

When cache buffers are written, they are marked dirty—an indication to the lazy writer that they should be written to disk in the near future. The lazy writer periodically wakes up and traverses the chain of dirty buffers looking for buffers to write. It wakes up when either the disk is idle for a specified length of time, the data in a cache block is older than a certain time, or the number of dirty buffers exceeds some high water mark.

The next level of caching involves maintaining in-RAM data structures linked together so that it mirrors the hierarchical nature of the file system. The first time a directory is touched, a new directory data structure is linked in. Each of these data structures contains the logical sector number (LSN) of the FNODE and a hint on the location of the topmost B-tree node. The hint may provide a low cost short cut to get to the directory—it contains the LSN of the topmost directory B-tree node and a pointer to a cache block that recently contained this disk data structure. If we're lucky, the buffer was not reused and still contains the B-tree node, so we save ourselves a potential disk hit. If not, then we must look in the hash chain to see if the desired sector exists elsewhere in the cache or, in the worst case, we must read the disk. At least we didn't have to traverse the entire directory structure where each directory is a B-tree in itself to find our target directory. Thus, in the typical case for a filesystem request, we'll use this directory "cache" to get to the topmost B-tree node for the target directory—and hopefully we get this B-tree node out of the cache. These directory structures also contain a semaphore providing a convenient way to serialize access to the directory when splitting, creating and deleting entries.

The highest level of caching involves maintaining a short list of complete path names, the name

checksum and the pointer to the corresponding directory cache structure. If we match on the path name, then we have a pointer to the in-RAM directory structure without having to traverse the complete tree. The name compare is fast since we first do a numerical comparison of the checksums.

Build Environment

We did something different in organizing our source code. The project is divided up into modules and a directory in the source tree exists for each of these modules. Within each directory, we have a convention of one function per file with the file name being the same as the function name. Each directory's makefile will build a library and the makefile for PIC's "root" will link PIC's entry point functions with the list of libraries created for each directory.

This has provided a good way to manage the code in an environment where summer students have had to come on board quickly, and prevents most situations where two people have to change the same file concurrently. Also, making a small change in one function will not cause the entire module to be recompiled.

Status

During the first week of February, PIC entered system test—whew! There's a minimal amount of new functionality that is currently under development and a lot of debugging work ahead of us. We also anticipate making some performance related improvements as we learn results of benchmarks and identify "hot-spots" in the code. Although we have a lot of work left, we are planning on meeting our June milestone for completing Pinball-In-C for 1.2.

Development Documentation

by Janine Harris

Quite a lot has been happening with documentation lately. For 1.3/2.0 there will not be a spec as we've had for 1.1 and 1.2. Instead we are going to use the Design Workbook (DWB) and generate a spec/technical reference from it. To facilitate this, a disk has been established on the PCV machine in Boca which contains all of the DWB files. Microsoft, Boca and Hursley share access to it.

The goals in establishing this disk are to provide ready access to the files, help to ensure more up-to-date documentation, and provide greater communication between the development sites. It should also solve the problem we have had with not being able to obtain documentation on Presentation Manager (PM). With Hursley an active partner in the DWB, we now have ready access to their information.

The system used to store the files on PCV, called "TOOLS" has several nice features. The major ones are "Inform" and "Subscribe". If you set a file

to the "Inform" status, any time the file is updated, a message is sent telling you the file has changed. "Subscribe" works in a similar fashion. It will tell you of a change and send a copy of the updated file to you.

Madeline Weise has had the job of building the workbooks and sending all of our DWB files to Boca once a week. Boca would send her back all of their files and she would rebuild the workbooks. With the advent of the TOOLS disk, Madeline is "informed" of any changes made by Boca or Hursley and no longer has to receive and then download all of their files. The process now is to get any IBM files from the TOOLS disk which have changed and then, together with the latest doc files from the Oatbran source tree, build the workbooks.

To view any of these documents on-line, look in the directories in \\OS2DOC\\os2.3\\workbook.

Directory	File	Volume
32API	apicomp.ppr	32API/DosCalls
CRUISER	cruiser.ppr	Kernel
VDM	mvdm.ppr	MVDM
SECURITY	secur.ppr	Security
PM	pmdoc.ppr	Presentation Mgr