It's <u>Still</u> Loading?

Designing an Efficient File System

Scott Bilas
Gas Powered Games

Introduction

- Me
  - Scott Bilas
- You
  - Win32 developers
  - Technical producers
  - Developers for other OS's don't leave! Win32 concepts should port over nicely.

Introduction (cont.)

- What is a “file system”?
  - Not particularly sexy, impressive, or difficult to engineer. (Sigh)
  - Executable code: find resources such as BMP’s or WAV’s, load or stream them into physical memory.
  - Content: the directory structure, naming conventions, and other process involved in content creation.
- Gabriel Knight 3 – the guinea pig
  - Over 36,000 unique files in 2 gig of raw data. No pathing, all files uniquely named.
  - 800 meg compressed ordered data over 3 CD’s.
  - File system 100% based on memory mapped files.

Four Basic Requirements

- My requirements for a “good file system”
  1. Easy to Use
  2. Fast and Efficient
  3. Use Package Files
  4. Solid Build Process

Requirement 1: Easy to Use

- This is the top priority!
  - Easy for *entire* team.
- Easy for engineers
  - Simple and safe to code for engineers (especially junior level) using familiar file access conventions.
  - They are probably used to fopen(), fread(), fclose() style functions – provide an API like that first.
  - Provide a second API that goes directly to the metal using file mapping (more on this later).

Requirement 1 (cont.): Easy to Use

- Easy for content developers
  - Easy to drop in and modify resources.
  - Easy to verify that resources work properly - just run the game and see if it works.
  - Best if this can be done without restarting the game.
  - Fastest development process is:

    1. Alt-tab away from game.
    2. Update content.
    3. Alt-tab back to game.
    4. Tell game to “rescan”.
    5. See results immediately!

Requirement 2: Fast and Efficient

- Stay close to the hardware
  - Exploit advantages inherent in a virtual memory OS by using memory mapped files.
- Support basic file access scenarios
  - Block: probably the most common – one-shot block read used to initialize something (bitmaps).
  - Random: pure random access (huge file paging).
  - Streaming: like Random except serial (sounds).

Requirement 3: Use Package Files

- Definition
  - Packaged file format is a huge file containing multiple resources, possibly compressed.
  - Examples are WAD, HOG, ZIP, BRN.
- Advantages
  - Use only a single file handle left open all the time. Opening and closing handles is expensive, especially on NT.
  - Maximize disc space usage efficiency.

Requirement 3 (cont.): Use Package Files

- Advantages (cont.)
  - Can be statically indexed for super-fast access.
  - Support easy versioning for future expansion and resource patching.
  - Requires a build process – a very good thing.
  - More professional, install is easier to manage and faster (one file vs. thousands).
  - Source data is more difficult to rip from the game.

Requirement 4: Solid Build Process

- Something needs to build the packages
  - This is a standalone tool that “compiles and links” all resources together into the big package file(s).
- Support “filtering” of the data as it goes in
  - Ideal opportunity to verify data integrity.
  - Easy to extend build tool to modify the data as it packages it.
- Streamline build process
  - One big batch file can do it all!

Six-Part Solution

- This should apply to nearly any type of game
  - Especially effective with games where frame-to-frame content requirements greatly exceed available system memory.
- This solution shipped with Gabriel Knight 3.
- A variation of this will ship with Dungeon Siege.

Solution Part 1: Organized Network Resources

- What does that mean?
  - Put resources out on the net and have the development team run exclusively from net data.
  - Source control systems make this easy – Visual SourceSafe offers “shadow directories”.
  - Choose sensible naming conventions, enforce from code if possible.

Solution Part 1 (cont.): Organized Network Resources

- Set it in stone
  - Before production begins and inertia takes over.
- Everybody is in sync
  - All developers guaranteed to run from the same data. Nobody should have outdated or incorrect content.
  - Minimizes the always mysterious “it works fine on my machine” phenomenon.

Solution Part 2: Simple File Access API

- “FileHandle” API
  - Provides Open, Read, Close functions.
  - Familiar to all programmers, easy to port code.
  - Base it on memory-mapped files underneath.
- “MemHandle” API
  - Provides Map, GetPointer, UnMap functions.
  - Directly uses memory mapped files.
  - Easy to understand and use.

Solution Part 2 (cont.): Simple File Access API

- Need abstracted file system
  - Requirement 1 says to run from network resources. This requires a system that works with raw files found on local or remote paths.
  - Requirement 3 says to use package files. This requires a system that knows how to pull resources from one or more package files.

Solution Part 2 (cont.): Simple File Access API

- Need abstracted file system (cont.)
  - Abstract both the FileHandle and MemHandle API's in a base class and derive both file manager types from it.
  - Allow simultaneous use of multiple file systems.
- Solution affected by naming conventions
  - Must choose "unique" or "relative unique" convention for names (this affects indexing).

Solution Part 2 (cont.): Simple File Access API

- Unique naming (Gabriel Knight 3)
  - Data set is treated as a flat array of resources with no path information.
  - Example: OpenFile(“wood.bmp”) where file may exist anywhere on the available path trees.
  - Advantages: simple to use, packaged files are easy to implement, packaged file index lookups are fast.
  - Disadvantages: requires indexing of paths in development builds (slow), can have problems with duplicate filenames (big problems there), harder to enforce naming conventions, takes longer to implement.

Solution Part 2 (cont.): Simple File Access API

- Relative unique naming (Dungeon Siege)
  - Data set is treated like a directory tree. Files are accessed through relative paths.
  - Example: OpenFile(“art\maps\wood.bmp”) where relative path to file must always be specified.
  - Advantages: no indexing required in development builds, duplicate names are not possible, easy to enforce file location standards, quick to implement.
  - Disadvantages: implementation of package file system more difficult, lookups in package files may be slower.
- Which is better?

Solution Part 3: Path-Based File System

- Definition
  - An interim file system needed for development.
  - Give it a set of root paths and it pulls files from those trees.
  - Optionally support absolute and override pathing.
- Features
  - Minimize tree iteration across the network by using indexes.
  - Should be able to use new or changed resources without restarting the game.
  - Don’t just limit it to development builds?

Solution Part 4: Flexible Package File Format

- Features
  - Versioning (plan for content patches and expansion packs).
  - Special data formats (such as compression or encryption) transparent to client code.
  - Use the Win32 Portable Executable file format and put a VERSIONINFO resource in there.
- Check your project requirements
  - If multi-CD game, will require cross-package indexing.
  - May want to support adding new packages on the fly (via Web download?)

Solution Part 5: Package File Builder

- It's like WinZip or link.exe
  - Everybody is familiar with the "package file builder concept"
  - Process is: 1. choose files, 2. do something to them, 3. pack them end to end in the output file.
  - Keep it as simple as possible.
- Command line or GUI?
  - Command line! Easier to implement and can be batched for auto-generating packaged files.

Solution Part 5 (cont.): Package File Builder

- Support filters
  - Use a plug-in format to make it easy to add new filters later.
  - Examples of mutation filters: PSD-to-raw bitmap conversion, script precompiler, MP3 encoder.
  - Examples of verification filters: naming convention checker, zero file size checker, content syntax verifier.
  - Probably don’t want to do anything really advanced.

Solution Part 5 (cont.): Package File Builder

- Support data ordering
  - Will speed up loads by minimizing slow CD seeks.
  - Will speed up transfers by maximizing locality of reference for caches. The system and drive hardware will read ahead anyway – don’t throw that data away!
  - Implement in game’s EXE code via journaling.

Solution Part 5 (cont.): Package File Builder

- Support compression
  - Games keep getting more files, more content.
  - Reading data from CD and decompressing it may be faster than reading uncompressed data.
  - zlib is high compression ratio, supports streaming, good decompression rate.
  - LZO is good compression ratio, does not support streaming, but ultra-fast decompression rate.
  - GK3 used zlib for streaming files, LZO for all others.

Solution Part 5 (cont.): Package File Builder

- Data drive the builder
  - Using a scripting language is probably bad judging from GK3 experience.
  - Use a configuration (INI style) language instead.

Solution Part 6: Package-Based File System

- Definition
  - File manager that owns one or more packaged files.
  - Can quickly search through index(es) to find a resource in one of those packages.
  - A resource is identified through an entry giving its offset and size within the package.
  - Exclusively uses memory-mapped files.
- Easy to implement
- But first, what are memory-mapped files?

Memory-Mapped Files: Overview

- Definitions
  - File map: an object attached to a file from which views can be created. A file mapping object is just a memory structure and does not actually allocate any memory or address space. Create one per file.
  - View: a region of virtual address space “backed” by the file. Create many per file.

Memory-Mapped Files: Usage

- To open a memory mapped package file
  1. CreateFile() read-only on your packaged file.
  2. CreateFileMapping() read-only on the entire file.
- To access a packed resource
  1. Look up the resource name in the packed file index. Get its size and offset.
  2. MapViewOfFile() to the offset and size where it's located (aligned to system allocation granularity).
  3. Use the returned const void* pointer.

Memory-Mapped Files: How a Mapped View Works

- Built on a “section” object in OS memory manager
  - Pages in virtual memory are directly mapped onto a file in 4K chunks.
- Lazy evaluation is key
  - Mapping a view only reserves contiguous address space, does not take up any physical memory at first.
  - Accessing a page that has not been assigned physical memory is called a “page fault”.
  - File chunks are only brought into memory (“paged in”) on demand.

Memory-Mapped Files: Advantages

- Important: *memory-mapped files are your friends*.
  - If you’re not using them, you should. They’re fast and easy!
  - Can be used for many things – here we only use them for accessing read-only data. See docs for other uses.
- Advantages
  - Ideal for read-only package files. Files mapped read-only are “backed” by the file itself. Old pages will just be tossed, not swapped out to the page file.
  - Eliminates unnecessary layers of code.
  - Eliminates typical read/process/discard file procedures. Just get a pointer and dereference it.
  - Usage is more intuitive: it’s just read-only memory.

Memory-Mapped Files: Error Handling

- Win32 Structured Exception Handling
  - Off the end of the view? Dangling or uninitialized pointer? No surprise: access violation!
  - Special case: damaged files, dirty CD? Not a read error, but an “in-page error”.
- Write a top-level SEH handler for in-page error
  - Not much you can do about it.
  - Win95 will probably blue-screen (but not BSOD).
  - Just reformat the error so end users don’t freak out.

Some Potential Issues

- Multi-CD games
  - Separate out your data set, enforce with code.
  - Possibility of emergency CD swaps.
  - Detecting CD changes, avoiding AutoPlay problems.
  - Complications with deciding what to install.
- Running from network resources
  - Files will be locked while accessed by game.
  - Should special-case file system code to work around this. Detect files accessed from network and copy them to local memory buffers.

Some (More) Potential Issues

- Win9x
  - There are many – read the knowledge base articles referenced at the end of this talk.
  - It’s “emulated” and so certain features are not supported (this wasn’t a problem for GK3)
  - 1 gig shared memory region

Future

- Extensions
  - “File mapping” over the Internet.
  - Local hard drive caching of CD data.
  - Possibility of shipping the package builder as part of the installer.
  - Incremental builder rather than straight compile.
- Q&A.

Contact Info

Scott Bilas

http://scottbilas.com

References

- Microsoft Developer Network (MSDN) http://msdn.microsoft.com
  - KB articles Q125713, Q108231
  - Working sample FASTFILE.C in Platform SDK
  - Article: “Improving the Performance of Windows 95 Games”
- Inside Windows NT (2nd Ed.) by David Solomon Microsoft Press
- zlib, a free compression library http://www.cdrom.com/pub/infozip/zlib/
- LZO, a free compression library http://wildsau.idv.uni-linz.ac.at/mfx/lzo.html