Tech Info Library

Power Macintosh: Run Time Environment (3/95)

Article Created: 10 March 1994
Article Reviewed/Updated: 6 March 1995

TOPIC ---

This article provides a technical description of the Run Time Environment and the switch from the 680x0 segment loader to the new Code Fragment Manager.

DISCUSSION --

A run-time environment is a set of conventions that determine how code and data are loaded into memory and managed. The PowerPC run-time environment is significantly different from the one provided in 680x0-based Macintosh System Software. Fortunately, this change is for the better. The most significant change is the use of fragments to organize code and data in memory. Fragments can be an application, import library (such as system software), or extension. Fragments provide the following benefits:

• Fragments let software in import libraries be easily shared with other programs.

• Fragments provide a simplified means for all software components to address global data. With 680x0 systems, it was provided only for applications.

• The use of fragments is one of the elements that allow future modernization of the Macintosh operating system. It lets new system features such as a microkernal be easily implemented.

Fragments
=========

A fragment is a method of organizing code and data. Fragments can loosely be categorized into three areas:

• An Application is a fragment launched by the user.

• Extensions are fragments which extend the capabilities of other fragments.

• Import library -- These fragments can be shared with other fragments. System software is an example of this type of fragment. Since this fragment can be shared, an import library is also referred to as a shared library. In addition, since import libraries are linked dynamically at application launch time (that is, not when the application was created or installed), they are also referred to as dynamically linked libraries.

At application launch time, the Code Fragment Manager makes sure all the necessary fragments are loaded to memory and prepared for execution. This manager uses the Code Fragment Loader to perform the loading into memory.

Fragment Storage

The physical storage for a fragment is called a container. A container can be any logically contiguous memory location such as a data fork of a file, Macintosh ROM, or a resource.

PowerPC executable code is stored in the data fork of the application file. A 'cfrg'0, is added to the resource fork to indicate that this application is a PowerPC application. In contrast, 680x0 applications place executable code as 'CODE' resources in the resource fork.

This arrangement of code and resources allows the creation of a single application that can run on both 680x0- and PowerPC-based Macintosh computers. By combining the two structures together, you get what is called a fat application.

Fat Applications

If a fat application is executed on a 680x0-based Macintosh computer, the system software ignores the 'cfrg'0 resource because it is an unknown type and loads just the 680x0 'CODE' resources. If executed on a Power Macintosh, the system software recognizes the 'cfrg'0 resource and loads the PowerPC code from the data fork.

A fat application is very useful when supporting mixed 680x0 and PowerPC environments. However, it may not be desirable to have a fat application. For example, there may be cases where the 680x0 performance is insufficient to run the application. In this instance, a 'CODE' resource should be installed so when the PowerPC-only application is run on a 680x0 system, a dialog box shows up to explain the situation.

A developer can distribute a single fat application. With Installer 4.0, the user has the option of installing only PowerPC or 680x0 executable application or both.

Import Library

An import library (shared library or dynamically linked library) makes applications be smaller. Since import code is loaded dynamically at application launch time, this code does not have to be part of the application. If only one application utilizes this library, there is little advantage unless the import library is stored in ROM (like system software).

The big advantage of an import library is when multiple applications utilize the same import library since a library is loaded into memory only once. Thus, several applications can have access to the library, without each having to have the RAM overhead to support the functionality.

In addition, centralizing functionality lets updates be made easily. By changing only one set of code, many applications are updated simultaneously.

An import library can be created for such sharable functions such as a spell checker. In addition, it can be used to support infrequently used application functions. The Code Fragment Manager can load these functions on demand after application launch. This results in smaller RAM usage and quicker launching.

Memory Organization
===================

The organization of memory in the PowerPC run-time environment is reasonably similar to the 680x0 environment. The system partition occupies the lower memory addresses while applications take the remaining space starting at the higher memory addresses. As with the 680x0 application partition, the Process Manager allocates space for a stack, heap, and global variable space.

The main difference between the PowerPC and 680x0 memory organization is the location of an application's code section and global variables. This difference varies depending on whether the Power Macintosh has virtual memory turned on or not.

In 680x0 environment, virtual memory uses a single monolithic file on a single volume as backing store. This backing store is mapped to a virtual memory address space. Its size is set via the Memory Control Panel. Unused pages of memory are written to this backing store and loaded into main memory as needed.

This backing store is necessary because the Resource Manager is not re-entrant. Since 68K code is found in code resources in the resource fork of an application, it is accessed by the Resource Manager. However, since it is not re-entrant, the manager cannot be accessed at an interrupt level. Therefore when a page fault occurs, the code must be written to the system wide backing file because it cannot be retrieved from its original location.

A backing store sometimes prolongs the launch of an application. This occurs when an application’s code is loaded into main memory only to be immediately written out to the backing store file.

This virtual memory delay is eliminated by the use of a scheme called file mapping. The Virtual Memory Manager uses the data fork of the application as the paging file. The entire code fragment is mapped to logical address space. Assuming the application code does not change (this is a guideline), it can be presumed to be read-only. As such, only the needed portions of code are actually loaded into physical memory.

NOTE: This file mapping only applies to native applications. These applications have code stored in the data fork.

As a result of this file paging scheme, native applications have a special note attached to their Get Info boxes. At the bottom of Get Info window, there is a message indicating that if virtual memory is turned on, the application’s RAM requirements change.

The same application's memory requirement changes when Virtual Memory is turned on (which is indicated in the note). This change in memory requirements is made automatically by the system.

If virtual memory is not turned on, code is loaded into the application heap. The Finder and Process Manager will automatically expand the memory as necessary. This is similar to the 680x0 run-time environment.

Article Change History:
06 Mar 1995 - Added keyword; made multiple technical updates.
11 May 1994 - Changed Title to better reflect the article.
21 Mar 1994 - Made various corrections to the Fragment section.

Keywords: kppc,kalley

===

19960215 11:05:19.00
Tech Info Library Article Number: 14865