ABS Tech Note: AWS26 Hybrid Applications (3/95)

Revised: 3/31/95
Security: Everyone

ABS Tech Note: AWS26 Hybrid Applications (3/95)

==========

Article Created: 14 November 1994
Article Reviewed/Updated: 31 March 1995

TOPIC

This note describes some of the deficiencies of A/UX hybrid applications, and gives some suggestions as to alternative approaches that may be more robust and functional.

DISCUSSION

A/UX Hybrid Applications

A/UX supports several different types of applications, including pure UNIX applications, pure Mac applications (which run under Mac OS as well), and hybrid applications, which combine both UNIX and Mac programming models.

There are two types of hybrid applications -- UNIX COFF -- format hybrids (such as CommandShell) and Mac hybrids. Both of these types of hybrids have proven to be architecturally weak and difficult to program robustly. Specific weaknesses will be discussed below.

Due to these limitations, it is unlikely that either type of hybrid will be supported in future Apple UNIX systems. A/UX will continue to support the current hybrid models, with all their existing deficiencies. Alternative approaches will be discussed in section 5.

UNIX COFF Hybrid Architecture

This section reviews the architecture of UNIX COFF hybrid applications to provide context for later sections.

Processes and Memory Layout:

Under A/UX, all pure Mac OS applications run within a single UNIX process called startmac. Each UNIX COFF-format hybrid, such as CommandShell, runs in its own

separate UNIX process.

As part of its initialization, the startmac process creates a large shared memory segment that is set up to look like the memory of a regular Macintosh. This shared memory segment contains low memory globals, the system heap, and Macintosh application partitions. As part of its initialization, each UNIX COFF hybrid application also binds to this shared memory segment.

For pure Macintosh applications, the application partition in this shared memory segment contains the application's heap, its A5 world, and its stack. Thus, all of this memory is accessible to any other Macintosh or COFF hybrid application.

For a UNIX COFF hybrid application, though, the application partition contains only the application's heap and its A5 world. The stack is in UNIX virtual memory space, accessible only to that application.

Cooperative Multitasking:

The Macintosh uses a cooperative multitasking model. At any particular time, a single application has control of the processor (the "token of control"). That application will keep control of the processor (from the Macintosh viewpoint) until it voluntarily gives it up, by calling one of three Macintosh toolbox traps: GetNextEvent, WaitNextEvent, or EventAvail.

Note that though UNIX COFF hybrid applications are running in different UNIX processes than pure Mac OS applications, each is blocked until it receives the single "token of control".

Under A/UX, the user interface device driver (/dev/uinter0) receives notification from the Process Manager when the token of control should be passed to a UNIX COFF hybrid application. The user interface device driver then directs events to the appropriate UNIX process.

Macintosh Hybrid Architecture

Macintosh hybrid applications are pure Mac OS applications that are linked with libaux.a (supplied with the A/UX Developer's Kit) to make A/UX system calls. The A/UX Installer is an example of such an application. These applications run within the startmac UNIX process like any other pure Mac OS application.

Problems with A/UX Hybrid Applications

This section describes some of the problems that have been encountered with hybrid applications.

Can't Make Blocking UNIX Calls in a Cooperative Multitasking Environment:

A hybrid application must not make blocking UNIX calls. While waiting for the UNIX call to return, the hybrid application will be blocked. In addition, the entire Macintosh toolbox environment will be blocked, because the hybrid hasn't called one of the Macintosh toolbox routines (WaitNextEvent, GetNextEvent, or EventAvail) which allows control to pass to another Macintosh application.

Allocation of Data For Asynchronous Operations:

Various Macintosh calls result in asynchronous operations that can complete at any time. It is quite possible that these operations may complete when a different Macintosh application has the "token of control". If a hybrid application starts an asynchronous operation that completes while another application is active, it will complete within a different UNIX context than the one from which it was started.

If any memory is needed to complete the asynchronous operation, that memory must be accessible from startmac or any hybrid application process. For a COFF hybrid application, the only suitable memory for this is the application heap in the shared memory segment.

Thus, extreme care must be taken in a hybrid application to ensure that any data needed to complete an asynchronous operation is stored in the application heap, rather than on the stack or in UNIX virtual memory heap space.

It has proven difficult to ensure this for all possible asynchronous operations. In some cases, Macintosh toolbox calls internally call other routines that make asynchronous operations, and it is not always apparent that an asynchronous call will be made.

Examples of various asynchronous operations include:
• Any routine that allows asynchronous operation, such as the File Manager or Notification Manager. The code for the response procedure for such an operation must be in the application heap, rather than a hybrid application's .text COFF segment. CommandShell handles this for its Notification Manager response procedure by copying relocatable code from the .text segment into its application heap.

• File Manager routines for AppleShare volumes. Be sure to allocate all parameter blocks in the application heap rather than on the stack of a hybrid. This requires the use of low-level file manager calls, rather than high-level calls that internally allocate the parameter blocks on the stack.

• Resource Manager routines for AppleShare volumes. These routines internally call File Manager routines, but do not ensure that the parameter blocks are allocated in the heap. Thus, it is possible that these routines will fail for AppleShare volumes. There are no low-level routines for the Resource Manager to allow the application to force the parameter block to be in the heap. Calling Resource Manager routines from hybrid applications has not been fully tested for AppleShare volumes.

• Any Macintosh call that internally allocates window records or dialog records on the stack, such as the Standard File package routines or PPCBrowser. The A/UX toolbox patches the Standard File routines for hybrid applications to move the stack temporarily into the application heap. However, the PPCBrowser and other such routines have not been tested from hybrid applications and may have problems. As the Macintosh toolbox evolves, it is difficult to ensure that new toolbox routines will operate correctly within the context of A/UX hybrid applications.

COFF Hybrid Application Development Tools:

UNIX tools are used to develop COFF hybrid applications. This limits developer options. For example, MacApp or the Think class libraries can't be used to create COFF hybrid applications.

In addition, it is difficult to debug COFF hybrid applications. There are no Mac source level debuggers that understand COFF formats or symbol tables. It is possible to use MacsBug, but again without application symbols. Given an address in the COFF application, it is possible to use adb to determine which routine that address is in, but that's a kludge at best.

UNIX source level debuggers are also problematic. First, they must be used over a serial port or a remote login, because the entire Mac environment, including CommandShell, will be hung while at a break point in a hybrid application. In addition, the UNIX source level debugger has no inherent knowledge of Macintosh data structures.

Macintosh Hybrid Applications:

The Macintosh hybrid application development environment and libaux.a have not undergone intensive testing, and there is not a lot of experience with their use. Some problems which have been encountered in this domain include:
• Include file incompatibilities. There may be a need to include both Mac and UNIX include files, some of which conflicted (for example, types.h).

• Limited set of A/UX routines supported by libaux. This library supplies mainly system calls. Many A/UX library routines are not system calls, and therefore are not available through libaux. Although some of these routines, such as the printf family, are available through libraries such as those provided by MPW, many other library routines are not available.

• Certain system calls are risky from Macintosh hybrid applications. Due to the usage of various file descriptors, signals, stacks, and so on, fork'ing and exec'ing new UNIX processes from a Macintosh hybrid application running within startmac must be done carefully. In addition, pipes, waits, and exit statuses are tricky.

UNIX Security Concerns:

Hybrid applications that need any special UNIX privileges are quite risky. This is because it is quite easy to patch any Macintosh trap from an INIT. If, for example, a hybrid application is running setuid-root, any Macintosh traps that it calls could be patched by any INIT to do unknown and dangerous things.

Alternatives to Hybrid Applications

Client-Server Applications:

Rather than developing an A/UX hybrid application, consider using a client-server model instead, with a Mac client half communicating with a UNIX server half. Various technologies can be used for communication, including ADSP or TCP/IP.

Using TCP, for example, the UNIX server portion could be written in a standard UNIX manner, such that it could be launched automatically by the inetd daemon when a client tries to establish a new connection. The Mac client portion would use MacTCP to establish the connection and communicate with the server.

The fact that the application is a client-server application can be hidden from the user, so that the user perceives only a Mac user interface running in the A/UX Macintosh environment. Alternatively, the client-server aspects can be exploited, by allowing the Macintosh client application to run on any Macintosh on the network, including remote Macintosh computers connected via ARA.

Advantages of Client-Server Applications:

This client-server model offers several advantages, including:
• Development is simplified. In each half of the application, all the standard programming rules for that type of application hold true. There are no additional restrictions as there are with hybrids (such as not making blocking UNIX calls).

• Better development tools are available. Both halves of the application can be developed with the best available tools. For example, the Mac half could be written with MacApp and debugged with SourceBug. The UNIX half could be written using standard UNIX development techniques, and debugged with a UNIX source-level debugger such as sdb.

• Better application design is encouraged. The client-server programming model encourages a clean separation of the user interface from the underlying "engine" of an application.

• As discussed above, the client-server model offers additional functional capabilities, by allowing the Macintosh client application to run on a different machine on the network than the UNIX server application.

• The UNIX server portion may be portable to other UNIX systems which do not support the Macintosh environment.

Converting a Hybrid Application to a Client-Server Application:

An existing hybrid application can be converted to a client-server application. This involves identifying those portions of the application which are specific to the Macintosh side (generally user interface code) and those which are specific to the UNIX side (anything that directly utilizes UNIX OS facilities). Some functionality, such as data crunching, could occur on either the Macintosh client side or the UNIX server side.

One way to identify UNIX-specific code in a UNIX COFF hybrid is to re-link each object file as if it were a separate executable. This will report all unresolved external symbols, including calls to UNIX library and system calls. Those files with many such calls may contain functionality which should be in the UNIX server portion, while files with few such calls may contain functionality which should be in the Mac server portion.

After identifying the Mac client-side functionality and the UNIX server-side functionality, the next step is to define the messages which will be passed between the two sides. Then, re-code the application using Macintosh tools to create a pure Macintosh client application and UNIX tools to create a pure UNIX server application. Try to unit test each side of the application as much as possible, then test to ensure that the proper communication is occurring between the two sides. It may be helpful to bootstrap the application, by getting limited client-server functionality running first, and adding more advanced capabilities once the basic communication mechanism is stable.

Article Change History:
31 Mar 1995 - Made minor corrections.

Support Information Services

Tech Info Library Article Number:16749