

New Technical Notes

Macintosh



®

Developer Support

A/UX Q&As Platforms & Tools

Revised by: Developer Support Center
Written by: Developer Support Center

May 1993
October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As this month:
Compiling with MPW C under A/UX

Compiling with MPW C under A/UX

Date Written: 11/3/92

Last reviewed: 3/1/93

SillyBalls compiles without errors in an exclusively MPW environment, but we can't compile it in an A/UX environment (CommandShell) on our UNIX partition. Here's our command line to invoke the MPW C compiler, along with the message we get before MPW C fails:

```
mac1.borison 18 % /MPW/Tools/C -o SillyBalls.o SillyBalls.c
# /MPW/Tools/C - Fatal Error : 421
# can't allocate memory for global symbol table
#-----
#      File "OSUtils.h"; Line 173  # Compiling SillyBalls.c
#-----
# /MPW/Tools/C - Aborted !
```

How can we get this to compile from A/UX?

You're trying to execute the C compiler directly from the UNIX shell. That's a no-no, because it's not a standalone program, but an MPW Shell tool. Since those tools have to run inside the environment created by MPW Shell, when you attempt to run it directly, the environment it needs

isn't there and it blows up. You need to run MPW the normal way, and then it'll work.

A/UX 2.x and GetFrontProcess

Date written: 9/3/92

Last reviewed: 11/1/92

Under A/UX 2.x when I call `Gestalt(gestaltLaunchControl, &reply)` the value returned in `reply = 0x7F` indicating that the Process Manager is available. However, when I call `GetFrontProcess` I crash. Is this a known bug and if so what is the recommended way to check for the availability of `GetFrontProcess`? (This is only a problem under A/UX 2.x; A/UX 3.0 works fine.)

The response from `Gestalt` is unfortunately in error; the Process Manager is not available under A/UX 2.x, because it's primarily based on system software version 6.0.x.

We recommend that you introduce an explicit check for A/UX 2.0, using the standard `gestaltAUXVersion` selector ('a/ux'), and if the version < 3.0, assume the Process Manager is not available.

Setting file types and creators with A/UX's setfile utility

Date Written: 6/12/90

Last reviewed: 8/1/92

Using A/UX language tools such as `cc`, `ld`, and `make`, how do I set a file's type and/or creator? (I can use `ResEdit`, but I want to include the command to set type/creator in the A/UX makefile, as I do using `Link` under the Macintosh Operating System).

You can use the `setfile` utility under A/UX to set Macintosh types and creators. The syntax is the same as the MPW Tool of the same name, as follows::

```
SetFile [option...] file/folder...
-a attributes      # attributes (lowercase = 0, uppercase = 1)*
-c creator         # file creator
-d date           # creation date (mm/dd/yy [hh:mm[:ss] [AM | PM]])*
-l h,v            # ICON location (horizontal,vertical)*
-m date           # modification date (mm/dd/yy [hh:mm[:ss] [AM | PM]])*
-t type           # file type
```

Note: Period (.) represents the current date and time.

Note: The following attributes may be used with the `-a` option:

```
L   Locked
V   Invisible*
B   Bundle
S   System
I   Initiated*
D   Desktop*
M   Shared (can run multiple times)
A   Always switch launch (if possible)
```

Note: Options/attributes marked with asterisk (*) are allowed with folders

Accessing Macintosh Operating System from A/UX program

Date Written: 7/30/90

Last reviewed: 8/1/92

Is there a way that you may access a Macintosh Operating System file system from an A/UX program without going through the toolbox?

—

Currently there is no way to access the Macintosh file system from an A/UX program without going through the toolbox. It may not be exactly what you want, but it is possible to write a faceless Toolbox application that works like most UNIX tools. Its only requirement is that, although the application can run from any terminal, someone must be logged in from the console for this faceless application to work correctly because the Toolbox must be active for it to work.

Macintosh drivers under A/UX sometimes not locked in heap

Date Written: 10/23/90

Last reviewed: 8/1/92

Under A/UX MultiFinder, RAM-based device drivers (loaded from a resource) are sometimes installed by the system as ROM-based drivers. The undesirable side effect of this is that the driver does not get locked in the heap when opened, even if the `dNeedLock` bit is set in the driver's header flags.

Here's why it happens: As part of installing a Macintosh Operating System device driver, the ROM code compares the pointer to the driver to the value in the low-mem var `ROMBase`. If the driver pointer is greater than `ROMBase`, the driver is assumed to be in ROM. Its pointer, rather than `Handle`, is put in the DCE and a `dCtlFlag` bit is cleared to indicate a ROM-based driver. Since a driver in ROM can't move, the Device Manager ignores its `dNeedLock` bit and does not call `HLock` when opening the driver. Before making the comparison, the Device Manager calls `_StripAddress` on the driver pointer. Then, rather than call `_StripAddress` on the `ROMBase`, the code clears the high byte to zeros! (This happens at ROM address 40806A3C on my Macintosh IIx). This means that a driver loaded into the heap at any address higher than 0x800000 is installed as a ROM-based driver.

Under A/UX MultiFinder, it is possible to have heaps located above 0x800000. In fact, on my Macintosh IIx with 20 MB of RAM, virtually all my application heaps are above that address. When an application (actually, a printer driver called by an application) opens our spooling device driver, the driver does not get locked in the heap.

To work around the problem, we set the locked bit for the 'DRVr' resource, ensuring the driver is locked when read into the heap, regardless of what the Device Manager does. In our case, this driver is stored as a resource in a print driver file. Once the print driver is closed, so is its file, and our driver is therefore removed from the heap. Otherwise, the driver would forever be locked in the heap, even when not open.

An alternative solution would be to have the driver lock itself. This is difficult because it would have to be done in the driver header code which is automatically prepended to the driver by the C compiler we use. (This header code loads a data resource, so the lock would have to precede that action). We could do our own header, but obviously that costs time and \$\$\$.

Please let me know if you can think of any problem with the first solution. I'm assuming that you will also follow up on having this 32-bit unfriendliness corrected in future system software.

—

Your analysis of the situation with Macintosh drivers under A/UX is excellent. I checked the ROM listings and it is doing exactly what you said. The good news on the Macintosh Operating System side is that the same code on the Macintosh IIci and IIfx has already been fixed in the ROM. Because these are the only machines in 32-bit mode under 7.0, it shouldn't be a problem.

On the A/UX side, I have filed a bug report. I'm not sure when it will be fixed, but your workaround should be OK.

Checking network address for software protection

Date Written: 10/30/90

Last reviewed: 8/1/92

We need to have security for our software so that it will run on only one machine. With Macintosh systems we are forced to use an Ethernet address because that is the only unique hardware number we are able to access. We need to know if there is some type of operating system software running on A/UX 2.0 that we can use or if there is a hardware code unique to a machine that we as developers can get to.

—

You could use the Ethernet address under A/UX, as long as you test that the system has an Ethernet card installed. There's no rock-solid serial number available on the Macintosh hardware platform. One of the considerations is that if the motherboard or the ROMs are repaired and switched, the serial number might change and cause problems with applications.

The pros and cons considering software protection are many, and I don't want to take any stands. Instead I will give you some other clues how to protect software, and I will also include at the bottom of this letter the official Apple DTS statement concerning software protection.

Anyway, one way (as you maybe know) is to use various dangles. We've had bad experience with ADB-based dangles and A/UX, so we suggest you use serial line-based (if you want to use the dangle method).

Another method of registering software is to register a unique name on the network (AppleTalk or TCP/IP), and have the software check if there's already someone else using this label. It assumes that the machine is connected to a network. Concerning AppleTalk solutions you might use a simple PRegister (Using the NBP Protocol), or in the case of a

TCP/IP solution maybe a background task that checks a port and reacts if someone with similar identity calls.

There are also many other half-good solutions for registering a label and checking for this on the system. Unfortunately crackers are able to break these copy-protection schemes; the nice thing is that the number of crackers among Macintosh users is very small.

So unfortunately I can't give you a definite answer of how to protect your application from illegal copying. I hope this resolution explains the current situation. If you want to comment or if you have any other questions, please let us know.

—DTS Official position concerning software copy protection—

DTS's official position is that software copy protection is a bad idea, so we don't support development of protection schemes. Here is a summary of arguments against copy protection:

1. You will have to put your valuable resources into engineering a scheme, which usually means disassembling the ROM to understand the disk drivers, HFS, MFS, etc., as well as fully understanding how the hardware for each machine your software runs on works. This extra work adds time to your development schedule.
2. You'll have to maintain your software when it doesn't work on future machines or system software releases. Virtually every copy-protection scheme devised so far (that we know about) has required revision for new Apple CPUs.
3. You'll have to test the software on all machines and products, including third-party hardware and file system software.
4. You'll have to support possibly angry users who have problems using the software because of the protection scheme, or have broken "key" disks, or other problems. Users may be frustrated by an inability to copy software to a hard disk, make backups, or immediately use the software with new hardware or software.
5. Every copy-protection scheme devised so far (that we know about) can be defeated by commercial products that remove or circumvent the protection.
6. Protected software frequently receives poor reviews or mentions in the press because of the protection scheme.

We feel that copy protection is not a worthwhile use of development time. We cannot help you develop a protection scheme, or provide support when protected software breaks with future hardware or software.

A/UX 2.0 video ioctls not available with Finder

Date Written: 11/30/90

Last reviewed: 8/1/92

In A/UX 1.1.1 there were video ioctls to get screen information (VIDEO_PIXSIZE, VIDEO_ADDR, and VIDEO_DATA), and one to map the physical screen into user address space (VIDEO_MAP). We used these to do documentation on our system in the form of

screen dumps. (The normal screen dump function cannot be used with a menu held down with the mouse.) Under A/UX 2.0 these ioctls are unavailable when the Finder is running, WHY? They work fine if the Finder is not running (using the console emulator only). Can you tell me how I can get the same information, specifically base address, bytes per line, screen rows, and map the screen in with the Finder running, BUT not be a toolbox application?

—

The video ioctls documented in System Admin Reference won't work with the Finder because Finder remaps video memory to other locations. In this case the documentation does not mention it, so we will try to make this point clear in the next documentation release (use only during console mode).

The suggested way is to play according to Macintosh rules when Finder is running, so many of the public domain screen dump utilities should work, even with menus. Beware of utilities that use FKEYs; they won't work with A/UX 2.0. If you want to write your own, check out the DTS Sample code FKEY, and trap the MenuHook for handling menus.

Code for determining if a Mac application is running under A/UX

Date Written: 12/4/90

Last reviewed: 8/1/92

How does our application know whether it is running under A/UX?

—

Below you will find a piece of sample code that will enable you to know whether or not your application is running under A/UX:

A/UX Sample Code:

```
#define BTstQ(arg, bitnbr)      (arg & (1 << bitnbr))
/*
 *   GetGestaltResult returns the result value from Gestalt for the specified
 *   selector. If Gestalt returned an error GetGestaltResult returns 0. Use
 *   of this function is cool only if we don't care whether Gestalt returned
 *   an error. In many cases you may need to know the exact Gestalt error
 *   code so then this routine would be inappropriate. See GetAUXVersion, for
 *   example.
 */
long   GetGestaltResult (OSType gestaltSelector)
{
    long   gestaltResult;

    if (Gestalt (gestaltSelector, &gestaltResult) == noErr)
        return (gestaltResult);
    else
        return (0);
}

#define HWCfgFlags    0xB22    /* Low memory global used to check
                                if A/UX is running */

/*
 *   getAUXVersion -- Checks for the presence of A/UX by whatever means is
 *   appropriate. Returns the major version number of A/UX (i.e. 0 if A/UX is
 *   not present, 1 for any 1.x.x version 2 for any 2.x version, etc.)
 *   This code should work for all past, present, and future A/UX systems.
 */
short GetAUXVersion ( void )
{
    long   auxVersion;
    short   err;
    short   *flagPtr;

    /*
```

```
*      This code assumes the Gestalt glue checks for the presence of the
* Gestalt trap and does something intelligent if the trap is unavailable,
* i.e., return unknown selector.
*/
auxVersion = 0;
err = Gestalt (gestaltAUXVersion, &auxVersion);
/*
*      If gestaltUnknownErr or gestaltUndefSelectorErr was returned, then
*      either we weren't running on A/UX, or the Gestalt trap is
*      unavailable so use HWCfgFlags instead.
*      All other errors are ignored (implies A/UX not present).
*/
if (err == gestaltUnknownErr || err == gestaltUndefSelectorErr) {
    /* Use HWCfgFlags */
    flagPtr = (short *) HWCfgFlags;
    if (BTstQ(*flagPtr, 9))
        auxVersion = 0x100;      /* Do Have A/UX, so assume version 1.x.x */
}
/*
*      Now right shift auxVersion by 8 bits to get major version number
*/
auxVersion >>= 8;
return ((short) auxVersion);
}
```

A/UX technical information sources

Date Written: 12/3/90

Last reviewed: 8/1/92

Is there a special AppleLink group address to send A/UX-related questions to rather than the general MACDTS address?

—

There's a folder under Developer Services for A/UX-related (developers ask each other) questions. You'll find a lot of good information also from USENET, comp.unix.aux.

Use a device driver to speak with ADB device from A/UX

Date Written: 12/11/90

Last reviewed: 8/1/92

Does A/UX do a Global Reset on the Apple Desktop Bus (ADB) when it initializes? That is, if an ADB device's handler were changed by an INIT during the Macintosh Operating System stage of A/UX startup, would that change persist in A/UX?

—

A/UX unfortunately does a reset on the ADB side when it boots. This means that any ADP opcodes that are sent to the devices are not recognized after the boot to the A/UX side. The only way for the moment to speak with the ADB device is by writing a device driver from the A/UX side.

GetGamma and SetGamma support after A/UX 2.0.1b2

Date Written: 12/14/90

Last reviewed: 8/1/92

Are GetGamma (csCode 4) and SetGamma supported in A/UX 2.0.1?

—

GetGamma (csCode 4) is not supported. SetGamma will be supported, starting with A/UX 2.0.1.

A/UX symbolic debugger

Date Written: 1/10/91

Last reviewed: 8/1/92

Is there a symbolic debugger for “hybrid” applications under A/UX? MacsBug is not a solution for looking at data structures 1K in size.

—

Unfortunately I’m not familiar with any A/UX debuggers. I have heard of one, but I can’t say that I’ve ever used it. It’s called UDB—OASYS Universal Remote Debugger. It’s a multi-targeted, window-oriented, source-level debugger for single and multiple embedded microprocessor systems. If you’d like to contact OASYS and talk about it, they can be reached at (617) 890-7889.

As for further information, you might have better luck calling the A/UX Hotline. Their number is (408) 974-5091.

Use ps -efa to see Macintosh Operating System processes

Date Written: 1/31/91

Last reviewed: 8/1/92

Can I use the ps -efa command to tell if a Macintosh program is running?

—

You should be able to see the Macintosh Operating System processes with the ps -efa command (as you tried). It will be listed as Mac24 or Mac32, depending on whether you’re running in 24- or 32-bit mode. You can even launch (start) these processes from the command shell.

Use kill -9 to kill Mac OS but not an individual application

Date Written: 1/31/91

Last reviewed: 8/1/92

Can I use the “kill” command to kill a Macintosh program running under A/UX? Is it even possible to kill a Macintosh Operating System–based program from the Command shell

window?

The kill -9 command should be able to kill the Macintosh Operating System. It obviously won't be able to kill an individual Macintosh program, however, since all Macintosh programs run under the framework of the one Macintosh Operating System process.

A/UX 2.0.1 fixes console log-in terminal control bugs

Date Written: 3/8/91

Last reviewed: 8/1/92

We are running A/UX 2.0 on both a Macintosh IIfx and a Macintosh IIfx. When a console log-in session uses “set -o emacs” (emacs editing keys as opposed to vi) in the korn shell (ksh), the session hangs, whether the set command is in .kshrc or typed in. A remote connection as root can log in and kill the console process and return the system to the log-in program. Is this a A/UX ksh bug?

—

Bugs found in the A/UX 2.0 console log-in terminal control code have been fixed in A/UX 2.0.1. The set command “set -o emacs” in vi works OK in A/UX 2.0.1.

A/UX 2.0.1 and third-party partitioned hard disk drives

Date Written: 3/11/91

Last reviewed: 8/1/92

I’ve partitioned into two equal-sized Macintosh volumes on my third-party hard drive, and A/UX 2.0.1 refuses to recognize the second partition at all. Can you tell me how to get this second volume to mount on the desktop?

—

The A/UX file system emulation doesn’t recognize multiple Macintosh partitions on the same hard drive. The A/UX engineering team is working on this known restriction.

A/UX & A/ROSE Macintosh Coprocessor Platform (MCP) protocol

Date Written: 3/31/91

Last reviewed: 8/1/92

Are there any catches in developing MCP-based (Macintosh Coprocessor Platform) cards to be used under A/UX? Or can they be developed under the Macintosh Operating System as usual and then used under A/UX?

—

A/UX doesn’t currently understand the A/ROSE MCP protocols, so cards that are using A/ROSE will not work under A/UX. The only way to get the cards working is to write a specific A/UX device driver that talks with the card from A/UX.

Use AppleTalk to print from Macintosh A/UX to a LaserWriter

Date Written: 5/7/91

Last reviewed: 8/1/92

To print from a Macintosh running A/UX to a LaserWriter, would I use AppleTalk or TCP/IP?

—

To print from an A/UX machine to a LaserWriter, you need to use AppleTalk. If you use your LocalTalk port as your AppleTalk connection, you will require no further hardware (other than your LocalTalk network). If you use your Ethernet as your AppleTalk connection, you will need some kind of router to translate these packets from Ethernet to LocalTalk, as LaserWriters are available only with LocalTalk connections.

A/UX and the Macintosh Deferred Task Manager (DTM)

Date Written: 7/11/91

Last reviewed: 8/1/92

The Deferred Task Manager (DTM) under A/UX is not supported at this time. A call to the DTInstall routine generates an unimplemented trap message.

The reason for this is that DTM should only be used for low-level driver functions, and for A/UX this is handled by device drivers talking with the NuBus cards.

To implement something with a similar functionality as the DTM provides, you'll need to write a separate UNIX device driver, because Macintosh OS NuBus device drivers as they are don't work with A/UX. For driver details, check the "A/UX Device Driver Kit," available from APDA.

SGetCString in A/UX archives should be SGetCString

Date Written: 7/25/91

Last reviewed: 8/1/92

The Slot Manager function SGetCString is called SGetcString, with a lower-case "c," in both libmac.a and libmac_s.a archives under A/UX 2.0 and 2.0.1. It should be SGetCString according to the rules of MPW and *Inside Macintosh*. This is another case where the case-sensitivity of UNIX bites.

If you have any similar problems with other calls to the Macintosh toolbox, try a script such as this:

```
strings use | lib | libmac.a | grep SGet
```

Also, check libmac_s.a if you are using shared libraries. The *A/UX Toolbox: Macintosh ROM Interface* manual has a list of supported Slot Manager calls.

Use A/UX libraries to emulate Macintosh Toolbox functions

Date Written: 7/30/91

Last reviewed: 8/1/92

How can I use the Macintosh Toolbox functions from the A/UX kernel? When I link my module into the kernel, ld can't resolve the references to toolbox functions.

Macintosh Operating System ROM Toolbox emulation under A/UX is done with the help of a device driver, `/dev/uinter0`, which handles the A-line trap interrupt emulation. This will not work in the kernel side, because you can't signal A-line traps suddenly to another device driver (without a lot of pain; anything is possible in software). For this reason, you should use the A/UX libraries that support most of the toolbox side concerning hardware. Slot Manager support, for example, is done with the slot libraries (`-lslot`).

Let DTS know if you need additional device driver support in the current hardware-related libraries. Otherwise, in the A/UX (or UNIX) device driver world, you should follow the rules and libraries of A/UX (most of the operating system-related traps are mapped to UNIX-equivalent calls anyway).

MacX client and reading Macintosh Command key

Date Written: 8/27/91

Last reviewed: 8/1/92

I need some help with MacX under A/UX: Is there any way to read the Command key from a MacX client? Pressing the Command key doesn't produce a keypress event.

The Command key is not activating a keypress because the Command key is used in the Macintosh side of the X world for the menu shortcuts and other things, and in order to avoid confusion, the design team made a deliberate decision not to trigger any key events on the X Windows side concerning the Command key.

If you have an important reason why this should be otherwise, please let DTS know.

MPW-like development environment under A/UX

Date Written: 11/15/91

Last reviewed: 8/1/92

Are there any tools or packages that give a programmer MPW-like functionality under A/UX? Can I use any of my MPW tools under A/UX?

You can't directly use MPW for UNIX development work at this moment. A new A/UX development product called the A/UX Developer's Tools is useful for developing applications on the A/UX platform. It provides you with a "toolset" to create UNIX and X Window Systems for Macintosh and other UNIX-based platforms, as well as hybrid Macintosh/UNIX applications on the A/UX platform. It also includes MPW 3.2 for writing pure Macintosh Operating System applications.

A/UX Developer's Tools work in conjunction with the A/UX operating system running on

an SE/30 or any Macintosh II series computer. The product is composed of three CD-ROMs (MPW, X11, and A/UX tools), as well as user and reference manuals. It includes the following features:

- A development environment for Macintosh, UNIX, X Window System and hybrid (Macintosh and UNIX) applications all on one platform.
- Macintosh development tools on UNIX. Complete MPW (shell, C, C++, Object Pascal, and assembler), ResEdit, SADE, and MacsBug are included.

- A/UX system call library for Macintosh-based applications that take advantage of the power of UNIX. XCMDs that can make A/UX system calls allow HyperCard stacks to use UNIX features.
- Toolbox application debugging from UNIX using dbx, with Apple-added extensions to allow source-level debugging of both UNIX applications and UNIX hybrids—applications that take advantage of the A/UX Toolbox.
- Industry-standard ANSI C with Macintosh enhancements. Includes aids in developing portable C code and access to routines written in Pascal.
- C++ Language System that enables object-oriented code development for producing powerful, easily maintainable code.
- Full X Window System development environment. Including X11 Release 4, Xlib library and Xtk toolkit, MIT's Athena Widget Set, desktop client applications, and utilities.
- Commando interface tools to add selected Macintosh user interface elements to standard UNIX applications.
- Complete, easy-to-read documentation for all product features.

If you're interested in this product, please feel free to call APDA at (800) 282-2732. Alternatively, you may also want to investigate some of the third-party A/UX development options by browsing through the Redgate Buyer's Guide library on AppleLink (in the Third Party Connection folder).

Mounting & unmounting a SCSI removable volume under A/UX

Date Written: 12/20/91

Last reviewed: 8/1/92

What commands do I enter to unmount a removable media drive with a HFS partition under A/UX, and then remount it or a different cartridge later? Sometimes when I drag it to the trash it pops back on the desktop. What commands would I use to mount and unmount an A/UX "Unreserved 1" partition?

—

The capability for ejecting SCSI volumes is not yet supported in A/UX; the only way to eject a volume such as yours is by using a manual eject, such as by pressing a button on the drive. Thus, if both Macintosh Operating System and A/UX partitions from a disk are mounted, your user needs to unmount the A/UX partition from the file system by using unmount, and then unmount the disk from the Macintosh Operating System by throwing it in the trash and ejecting the volume by pushing the eject button on the drive before the Macintosh Operating System detects and remounts it.

To mount and unmount A/UX partitions, you use the “mount” and “unmount” commands in A/UX, as described in A/UX’s manual pages.

Timeout versus delay call for A/UX driver

Date Written: 1/7/92

Last reviewed: 8/1/92

When the A/UX kernel calls our network driver's INIT routine, part of our initialization requires us to wait for approximately 1 second or so. To implement the delay we use the `delay(v.v_hz)` system call. This works for A/UX 2.x, but when the driver executes the system call `delay(v.v_hz)` under A/UX 3.0, the system never returns. Any ideas?

—

You should use the `timeout()` kernel library call instead of the `delay()` kernel routine. `delay()` will always do a sleep, and thus it's hard to get a wakeup state back from this.

Timeout wants a function pointer, which will call back the original thread after a defined time, like 1 second or so. It's surprising that `delay()` seemed to work OK in your driver in A/UX 2.x kernels. Maybe you were just lucky with the overall timing.

A/UX 2.0.1 and LaserWriter 7.0 compatibility

Date Written: 1/30/92

Last reviewed: 8/1/92

Is it safe to install the System 7.0 printer drivers under A/UX 2.0.1 systems? We have A/UX users, System 6.X users, and System 7.0 users all sharing the same printers.

—

There are no known incompatibilities between A/UX 2.0.1 and the System 7 print architecture. A complete LaserWriter installation on the A/UX machine should include three pieces of software: the LaserWriter driver, the PrintMonitor application, and Laser Prep. To make sure A/UX's native spooling operates normally, do not update or modify your Backgrounder system extension (INIT). If you follow these steps you'll have a signed treaty, no more laser wars.

A/UX 2.01 ignores enumerator field in NBP tuple

Date Written: 3/3/92

Last reviewed: 8/1/92

I am having problems with AppleTalk using A/UX 2.01. Only one NBP (Name Binding Protocol) name is allowed per AppleTalk socket; that is, it ignores the enumerator field in the NBP tuple and keeps only the first name for every socket. The same code under the Macintosh Operating System seems to work fine. Is there a known bug here?

—

There is in fact a bug with NBP and the enumerator field under A/UX 2.01. At this point there is no workaround. However, the A/UX group is aware of the problem and there should be a fix at some point in the future.

AppleSingle/AppleDouble file format

Date Written: 10/1/90

Last reviewed: 8/1/92

How can I find out more about the AppleSingle/AppleDouble file format specification?

—

The AppleSingle/AppleDouble file format specification was originally designed to facilitate storage of Macintosh files on A/UX. It has been generalized to allow storage across a variety of platforms. More information on this file format is contained in Chapter 6 of the A/UX 2.0 manual, *A/UX Toolbox: Macintosh ROM Interface*. It is also available as a separate document titled “AppleSingle/AppleDouble Formats for Foreign Files Developer’s Note.” This document is available in printed form from APDA, and in electronic form on the *Developer CD Series* disc and on AppleLink in the Developer Support folder.

A/UX 2.0 and text file CR-to-LF conversion

Date Written: 5/1/90

Last reviewed: 8/1/92

When and why are carriage return characters translated to linefeed characters by A/UX 2.0?

—

To understand the significance of this question you must realize that Macintosh applications expect lines in text files to be terminated by a carriage return character (CR, 0x0D) while A/UX applications expect the lines to end with a linefeed character (LF, 0x0A). To allow the same files to be read and written by both kinds of applications, the A/UX Toolbox performs some magic translations when Macintosh applications read and write these files. While every attempt has been made to make this translation completely transparent to both the user and the application, some situations can arise where unexpected results ensue.

First, translation NEVER occurs except for files of type 'TEXT', and then only when the file is written to the “/” UNIX partition. Translation does not occur if the file resides on a Macintosh file system, and binary files are not affected. But, if you have a file of type 'TEXT' and move that file to the UNIX partition, THEN change the file type to something other than 'TEXT', translation will have already taken place and the file will have bad data.

When does this translation occur? Translation takes place on _Write and _Read calls. At that point, the A/UX toolbox looks at the file type and the destination/source file system and determines if translation is appropriate. This means, for example, if you create a file of type 'TEXT' and write some data to it, then change the file type, translation will have already occurred, so you should always set the correct file type before writing any data.

Finally, how does the toolbox deal with “ordinary” UNIX files transferred to an HFS volume? For a more detailed discussion of the exchange of files see the question: “What assumptions are made when the A/UX Toolbox moves a file between an HFS partition and an A/UX partition?” With respect to CR/LF translation when transferring ordinary UNIX files, the Toolbox tries to determine if the file is a text file in the UNIX sense. It does this in the same way the “file” utility does it (man (1) file). For more information on how it does this, see the question: “How A/UX 2.0 handles file and creator information.” If the file looks

like an ordinary text file, like a shell script or source code, translation occurs; otherwise it does not. As long as the A/UX Toolbox can identify the UNIX file type and the toolbox can handle transfer both to and from the HFS partition, the process is completely reversible and no ambiguity occurs. But beware of transferring data only in one direction with the Toolbox. Suppose, for example, you transfer a file from A/UX to a Macintosh using a serial communication program, but then use the A/UX Finder to copy it back to the A/UX partition. You may not end up with an identical file because translation has taken place in one direction, but not the other.

A/UX Toolbox and HFS-to-A/UX file conversion

Date Written: 10/1/90

Last reviewed: 8/1/92

What assumptions are made when the A/UX Toolbox moves a file between an HFS partition and an A/UX partition?

—

Implicit carriage return/line feed translation occurs for text files. See the question: “A/UX 2.0 and text file CR-to-LF conversion.”

There are some other assumptions made regarding text files. Macintosh files, in addition to having the data in the data fork, often have additional information stored in the resource fork. UNIX has nothing corresponding to a resource fork, nor does UNIX have information such as the Finder maintains for files like type and creator information; so this information is not lost when the file is transferred to an A/UX partition. A/UX defines some special UNIX file types.

The simplest form is the AppleSingle format. This includes a header indicating the size and location of data within the file; various entries for file Finder information; and resource and/or data fork (if appropriate). This is fine for binary Macintosh files such as applications or word processor and graphics files, but in the case of plain text files, this presents a problem for UNIX utilities that expect only to see the text (data) in a file, so A/UX defines a second format referred to as AppleDouble.

AppleDouble format stores the Macintosh file in a pair of separate UNIX files. The data fork resides in a UNIX file all by itself, and all the other parts of the file go into a separate file under the same name but with a “%” character prepended. For example, if you have a Macintosh text file named “Foo,” which also has a resource fork, and you copy that file to a UNIX partition, it still appears as a single file to the A/UX Finder and other Macintosh applications, but to UNIX process there are two separate files named “Foo” (the text) and “%Foo” (the resource and other data).

Creation of an AppleSingle/AppleDouble file is NOT performed for files of type 'TEXT' which have no resource. Only a plain UNIX text file is created and all Finder information is lost. This simplifies working with text files between Macintosh and UNIX applications, avoids having the excess baggage of “%filename” files floating around and getting separated and allows UNIX text files to be transferred to HFS partitions and back without change. Explicit conversion between AppleSingle, AppleDouble, and plain UNIX files is possible using the fcnvt utility.

How A/UX 2.0 handles file and creator information

Date Written: 10/1/90

Last reviewed: 8/1/92

How does A/UX 2.0 associate file and creator information with ordinary UNIX files?

—

File and creator information has no counterpart in the A/UX file system. All files under UNIX are just streams of bytes, and some conventions have developed over the years to help applications determine the kind of files they are dealing with. Many of these conventions are exploited by the A/UX “file” utility to identify files.

The first few bytes of a file are often used to store a special number or string referred to as a “magic number.” Various magic numbers for files are contained in the file `/etc/magic`. For example, a file beginning with the string `#!/bin/sh` is a Bourne shell script. This is just a text file and the default for plain text files is to be owned by the TextEditor application. Plain text files will show up in the A/UX Finder with the TextEditor document icon. If you double-click the icon, TextEditor is launched to edit the file, but if you take that same file and make it executable (in the UNIX sense) using `chmod`, it suddenly becomes executable by the CommandShell and has a CommandShell document icon associated with it. If you double-click the icon, the CommandShell is made the foreground application and `command` is launched for the script.

The apparent type and creator is a fiction maintained entirely by the A/UX Toolbox; it is returned to Macintosh applications just like real type/creator information, using the `GetFileInfo` call, but it does not exist in the A/UX file system. If you want explicit type/creator information to be associated with a file, it must be an AppleSingle/AppleDouble format file.

Altering MultiFinder memory under A/UX 2.0

Date Written: 10/1/90

Last reviewed: 8/1/92

How can I get A/UX 2.0 to increase the amount of virtual memory available to MultiFinder?

—

Because A/UX is a virtual memory environment, A/UX can allow MultiFinder to run in a memory partition larger than physical RAM. There currently are some limitations based on the MultiFinder environment you are using, and there are of course tradeoffs when creating a virtual memory partition much larger than physical RAM, but here is the scoop:

In the 24-bit environment, A/UX limits MultiFinder to 8 MB. In the 32-bit environment, the limit is 16 MB. You can alter the default size by modifying the scripts `/mac/bin/mac24` or `/mac/bin/mac32` and adding the `-m` option to the `startmac24` or `startmac` command lines. For example,

```
/mac/bin/startmac -m 10M
```

would create a 10 MB partition. Alternatively, you can alter the default size on a user-by-user basis by setting the environment variable `TBMEMORY` in the user's .log-in script. For example,

```
setenv TBMEMORY 10m
```

will accomplish the same thing.

A/UX 2.0 recognizes AppleTalk Phase 2 (but not Phase 1) routers

Date Written: 10/1/90

Last reviewed: 8/1/92

Why does A/UX 2.0 seem not to find all the AppleTalk zones in my network?

—

The A/UX 2.0 AppleTalk implementation is AppleTalk Phase 2, and Phase 2 nodes do not recognize Phase 1 nodes and routers. You need to upgrade your routers to Phase 2. Alternatively, you can connect your A/UX machine to LocalTalk, which does not have this limitation.

Root has more free space than ordinary users

Date Written: 10/1/90

Last reviewed: 8/1/92

Why does root have more apparent free space than ordinary users?

—

When using Berkeley file systems, the system reserves a portion of the partition (called the “free-space reserve”) and makes it unavailable to user processes. For example, if you log in as Guest and issue the following commands, here is what you will see:

```
Guest 1 % df -t /          /dev/dsk/c5d0s0      8640 blocks    18880 i-nodes
                                total    102204 blocks    23040 i-nodes
Guest 2 % su Password:
Guest 1 # df -t /          /dev/dsk/c5d0s0      13752 blocks    18880 i-nodes
                                total    102204 blocks    23040 i-nodes
```

Although the total number of blocks are the same, the number of free blocks differ: 8640 for Guest, but 13752 for root. About 5% of the total number of blocks are reserved and unavailable to user processes. If an ordinary process were to write out more than 8640 blocks, it would get a file system full error, so df is accurately reporting the situation.

In the event that an ordinary process fills up the file system, root can still get in and take appropriate action because root still has access to the remaining free space. The free space reserve also helps reduce disk fragmentation because there are always free blocks (really) available, so the system does not have to scatter blocks all over the disk when creating or extending files. Disk fragmentation can have a drastic impact on performance; according to the main entry for tuneufs, the free-space reserve “value can be set to 0 for the file system, although up to a factor of three in throughput is lost over the performance obtained at a 10% threshold.”

As the previous discussion implies, free-space reserve is a tuneable file system parameter that initially can be set using the newfs command, or later controlled using tuneufs. (tuneufs can only be performed on an unmounted file system; this means you can’t tune the “/” partition currently in use. You must be able to boot A/UX separately so you can tune such a partition, so it is best to plan ahead and set such parameters when creating the partition.)

System V file systems do not have this feature, so df will report the same totals for both root and ordinary users.

How to recover if A/UX 2.0 Toolbox “hangs”

Date Written: 10/1/90

Last reviewed: 8/1/92

Why does A/UX 2.0 appear to “hang” frequently?

A/UX 2.0 running Macintosh applications is subject to the same problems caused by those applications under the Macintosh Operating System. If an application is not “well behaved” according to the MultiFinder guidelines, it can cause random behavior such as hanging the Toolbox. But this is usually not as fatal as under the Macintosh Operating System. Usually when A/UX appears to “hang,” it is the A/UX Toolbox, not A/UX itself, which is “hung.” It is usually possible to kill off the A/UX Toolbox without having to resort to the drastic step of punching the reset button (which is very UNIX-unfriendly and means you should check the file system with fsck when you reboot).

So, what is the best way to kill off a runaway Toolbox? Well, if you have installed MacsBug for A/UX, you may be able to drop into the debugger using the key combination Control-Command-I (using the hardware programmers switch doesn’t work). Once in MacsBug, you may be able to kill off the errant application with the es command, or reboot with the rb command. As a more drastic measure, you can try the key combination Control-Command-E, which should kill off the toolbox completely, log out the console and restart the log-in process on the console (in the absence of MacsBug, Control-Command-I is the same as Control-Command-E). Doing this is, of course, every bit as drastic as rebooting the Macintosh Operating System; any work not saved by toolbox applications will be lost, but A/UX is still safe. If you have access to A/UX via a serial or network terminal, you can log in as root and kill off the startmac process (essentially equivalent to Control-Command-E).

You should always try these methods before resorting to doing a hardware reset.

A/UX debugging tools

Date Written: 10/1/90

Last reviewed: 8/1/92

What debugging tools are available for A/UX?

The debugging tools available depend on the task.

Very low-level kernel debugging: A kernel-level debugger ships with every system, although it is not included in the configuration. Documentation on the kernel debugger is in the release notes for the A/UX 2.0 Device Driver Kit.

Assembly debugging: The standard UNIX assembly debugger, adb, is provided.

Source level debugging: The standard UNIX source debugger, sdb, is provided. Additionally, an alternative source level debugger, cdb, is available through APDA.

Macintosh application debugging: Macintosh binaries can be debugged using MacsBug. MacsBug version 6.2 is compatible with A/UX. Place MacsBug and the Debugger Prefs file into the currently active A/UX System Folder, log out and then log back in. Once it is installed, you can drop into the debugger using the key combination Control-Command-I (using the hardware programmers switch doesn't work). Contact APDA for the latest information on MacsBug.

A/UX Toolbox application debugging: Use dbx, available from APDA.

A/UX 2.0 doesn't support IP encapsulation inside DDP packets

Date Written: 12/7/90

Last reviewed: 8/1/92

Can I get A/UX 2.0 to talk TCP/IP over the LocalTalk network? MacTCP seems to support this but I can't pummel A/UX into doing it.

—

IP encapsulation inside LocalTalk DDP packets is not supported under the current A/UX 2.0.

Cloning A/UX

Date Written: 5/1/90

Last reviewed: 8/1/92

What is the easiest way to make a complete copy of A/UX from one SCSI hard disk to another?

—

The dd command is the simplest way to copy A/UX. The following script documents the dd command usage and gives a sample dd command. If you use dd to copy slice 31, as in the example, the entire drive is duplicated, including the Macintosh Operating System partition.

```
# To clone an entire SCSI disk drive, issue the following command:
#
# dd bs=32b if=/dev/rdisk/cXd0s31 of=/dev/rdisk/cYd0s31
#
# where X is the SCSI id of the input device and
#   Y is the SCSI id of the output device.
#
# if you have lots of memory, bs can be larger than 32b to allow
# faster copying. bs=512k works well on machines with 4MB or
# more.
#
# For example, to clone SCSI drive 4 onto SCSI drive 5 do:
#
# dd bs=32b if=/dev/rdisk/c4d0s31 of=/dev/rdisk/c5d0s31
#
# The 31 at the end of the device name is a partition number
# (31) referring to the entire drive. Substitute an appropriate
# partition number to copy just a single partition.
#
# This is set up to clone 4 onto 5.
#
dd bs=32b if=/dev/rdisk/c4d0s31 of=/dev/rdisk/c5d0s31
```

A/UX 3.0 and FSMakeFSSpec with null filename

Date Written: 5/1/92

Last reviewed: 8/1/92

I believe I've run across a bug (alternate behavior) in A/UX 3.0. When calling `FSMakeFSSpec` with a null filename (meaning that we're specifying a directory), System 7 thinks it's cool, but A/UX 3.0 returns a `fnfErr`. According to *Inside Macintosh* Volume VI, the `fnfErr` is OK in the sense that the `FSSpec` is filled in, but it just doesn't match an existing file/folder. If one then makes a call to `NewAlias`, one gets `noErr` in System 7, and `fnfErr` in A/UX. The cure is to call `PBGetCatInfo` to get the directory's name and parent `dirID`, and supply those in the call to `FSMakeFSSpec`. *Inside Macintosh* Volume VI states that empty file names could be used with `FSMakeFSSpec` for creating special `FSSpecs`.

Yes, it's a bug in A/UX 3.0 final. It probably will be fixed in the next release of A/UX. Meanwhile, you'll need to continue using your workaround.

Mixing `calloc` and `NewPtr` in a Macintosh program

Date Written: 5/7/92

Last reviewed: 8/1/92

In the past we had heard of a problem using `calloc` and `NewPtr` in the same program. Is this true?

There are a few difficulties, which you can deal with if you need to. The primary problem is that `calloc` and all the other `malloc` routines weren't designed for the Macintosh platform. Macintosh memory management is designed around trying to squeeze as much as possible out of a limited memory area, which is why handles are the primary storage scheme in a Macintosh; they can move, and so greatly reduce memory fragmentation. Because the `malloc` tools return a pointer, they have to be located in a locked block, so they tend to lead to fragmentation if used with any other memory allocation calls (such as `NewPtr`). For this reason, any use of the `malloc` suite of allocation calls isn't recommended for Macintosh programs. The only good reason to use them is if you're porting a large body of code from other platforms; in this case, it may be a reasonable tradeoff to keep the old allocation code.

You should also be aware that most of the Macintosh `malloc` routines never free up memory. When you `malloc` some space, the routine must first allocate it from the Memory Manager. It allocates a large block of space using `NewPtr` and divides it internally for distribution in response to `malloc` calls. If, however, you eventually free all the blocks you allocated from this `NewPtr` block, the block won't be released to the Memory Manager with the `DisposPtr` call. This means that once you allocate some memory with `malloc`, you won't be able to free it and then use that memory from a Macintosh allocation call. Thus, if you had two phases to your program, one of which used the `malloc` calls extensively and the second which used Toolbox calls, the second phase wouldn't be able to use memory freed by the first phase. That memory is still available in the `malloc` pool, of course; it simply can't be used by `NewPtr` or `NewHandle`. The `malloc` routines supplied by THINK C work similarly, as described in their Standard Libraries Reference. Thus, mixing the C and Macintosh

allocation routines requires special care.