The Macintosh as an Internet Server

David Peterson

Abstract

The Macintosh is appearing on networks alongside UNIX hosts more and more frequently. Users and network managers alike expecting these Macs to provide the network services common to these other machines. A few programs are appearing which handle some of these services, ftp and finger servers exist and discussions are going on about writing programs to provide other services. Although it would be preferable to keep all these programs running constantly in order to provide their services on demand, it is impractical to do so. What I have developed is an implementation of the UNIX inetd server daemon to run on the Macintosh. It listens for connections on specified TCP and UDP ports and when one is made a predetermined program is launched to service the request.

The Internet Server program itself is written as a faceless background application. It performs no idle processing and as such consumes CPU time only when it must launch a program to service a remote machine. When this occurs, the server looks up the appropriate program and launches it with the MacTCP stream pointer as the program's launch parameters. This makes the first event received by the program contain the stream pointer for the connection it must manage. There is also a mechanism through which the launched program can request that asynchronous notification events related to the stream be sent to it.

This approach gives the program that implements the service complete control over the IP connection just as if it had actually been created by the program. It also separates the listening for service requests from the actual handling of the service, yielding two advantages. First, there need only be a single program running on the Mac at any given time to listen for all service requests. Second, it allows the programs providing the services to be written by any one, in whatever style and language they prefer, making it easy to implement new services. A developer only needs to write an application, not a stand-alone code resource, that handles the specific protocol. The only requirement is that it must respond to a certain AppleEvent.

The Internet Server is managed by a control panel that allows users to specify which TCP and UDP ports to listen on, and to choose what program to launch when a request is received. The configuration process is presented in a familiar Mac-like way – there are no text files to edit or formats to remember.

The whole system is written as a series of programs and a control panel, there are no drivers or system patches to install and cause system conflicts. It can be easily expanded to provide other network services, and enables the Macintosh to provide common services to the other machines it is sharing the network with.

1. Introduction

Everyone who has a Macintosh on the Internet[1] or some other network that is shared with UNIX hosts has had to deal with this at some time or another. How do you get your Mac to be a good network citizen and provide some of the standard services that the other machines provide and expect?

Maybe you want to get files to or from your Mac, or get your Mac to do something else while you're working at a different machine. If it were a UNIX host it would be simple – copy the files over using rcp, or if you wanted to make it difficult, use ftp. If you want to start the machine processing something else just log in and start the program.

On the Mac, it's not quite that easy though. There are ftp servers that exist, but they are usually as part of another package. You start up the program, transfer the files then quit – this gets old after a while, not to mention that it is just plain cumbersome to use ftp all the time just to move a few files around. What if you want to use rcp? Well, you could write a program to listen on the specific port and handle the file transfers. What if you would like to have a finger server running on your Mac so that other people could tell if you were at your machine and using it? Or how about a talk daemon?

It's no problem to write all these little programs, but before long you have all these processes constantly running but their services are only being used occasionally. Obviously some way to manage all this is needed.

In comes the Internet Server. This is a single program that listens for requests for all services provided. When such a request comes in, the appropriate program is launched to handle the transaction. In this manner only one program needs to be running on the computer at any given time to provide all services. The advantages of this should be obvious.

The remainder of the paper will discuss this specific implementation of an internet server for the Macintosh. After an architectural overview describing the major components of the system there is a more in-depth look at some of the implementation details followed by a discussion of some of the problems with the system.

2. Overview

The Internet Server system consists of three main parts. The server itself, a control panel for setup and a remote shell server. The system also requires System 7 or later of the Macintosh OS because of the use AppleEvents and the *FindFolder* routine.

The Server

The concept of a single process to handle all service requests is nothing new. The UNIX *inetd* program was designed for this exact purpose. What is presented here is a version of the *inetd* server for the Macintosh.

The server process is implemented as a background only application kept in the Extensions folder and is launched automatically by the system during startup. The server reads a configuration file containing protocols and port numbers to listen on and the programs to launch when a connection is made.

Once the server is running it does no idle processing. Instead it uses completion routines and the *WakeupProcess* system call to determine when to launch a child process. The impact the server has on the computer is negligible.

Server Configuration

The configuration file mentioned above is created and managed by a control panel. It presents the information contained in the standard UNIX /etc/services and /etc/servers files. To provide a familiar interface it is modeled after the 'File Sharing Setup' control panel of System 7 fame.

The top half of the control shows a message telling the current state of the system plus a button to toggle it. The bottom half of the control has a scrolling list with buttons to add, remove and change entries. Clicking on the 'New' button will bring up a choose file dialog with a couple of additions to it. Use the radio buttons to choose the transport protocol and enter the port number

[1]No, its not a service as some journalists would have you think, The Net is an entity unto itself.

that the service will use. Then, just choose the program

that will handle the service, both background only applications and normal applications are allowed. Be sure to choose a program written for the specific protocol. The control panel is more than happy to let you choose Microsoft Word to handle network services, however, this is not advised.

The start/stop button takes effect immediately, if any changes are made to the list of services the server will be notified when the control panel is closed. No restarts are necessary.

Remote Shell

The remote shell server follows the protocol used by the UNIX rshd. When connections come in on TCP port 514 rshd is launched and the command string generated by the peer process on the remote machine is parsed and the command executed.

In this Macintosh implementation, remote shell services are provided in two different ways. First rshd looks for a stand-alone program in its folder with the same name as the command passed to it. If such is found, it is launched and its arguments are passed to it via the same method as the stream pointer. If no program by that name is found, it is assumed that the command should be handled by ToolServer.

By using the standard script and diagnostic AppleEvents described in the ToolServer documentation the command string is given to ToolServer for execution and its output is returned to the remote host.

If ToolServer is not running, an error is returned to the remote host.

3. Implementation Details

In this section, some of the more important mechanisms used by the Internet Server are examined in detail. Items such as the location and size of the MacTCP streams, passing the stream from the server to a child process, the handling of asynchronous notification routines, and clean-up after a child has exited. The AppleEvents used by the system are also described in detail.

MacTCP Streams

Since there is no concept as the inheritance of file descriptors or the use of data structures by child and parent after a fork system call, communicating the existence of a stream to a child process is a major issue. The only available solution on the Macintosh is to pass the absolute address of the MacTCP stream pointer created by the server on to child process, but this brings up a rather incestuous situation. Creating the streams in the server’s heap would appear the most straightforward, but having one process accessing the heap of another seems like a Bad Thing. In addition, if protected memory ever makes an appearance on the Macintosh this solution would certainly break.

The use of temporary (Multifinder) memory would be inappropriate in this situation. The MacTCP streams are by no means temporary, and worse yet, must be non-relocatable blocks of memory. Using temp memory would eventually result in physical memory being fragmented into small unusable blocks.

The only other solution then is to use the system heap. This has two main advantages, since many applications use the system heap for shared memory it is unlikely to break under protected memory, and, more importantly, the system heap was suggested by Apple DTS.

Contrary to popular opinion, the size of the buffer given to MacTCP in the original create call has little effect on performance. I experimented with sizes ranging from 4 to 32 kilobytes and performance seemed to top out between 8 and 12 even with high traffic protocols such as rcp.[2]

Passing Stream Pointers

AppleEvents are used to pass the stream pointer from server to child. When a child process starts up, it must register an AE handler to receive an event that will contain the stream pointer it must manage.

To simplify the launching of a child process and

[2]However, the size of buffer used is stored in a resource in the server for those who would like to experiment on their own.

delivery of the stream pointer to it, a feature of the *LaunchApplication* trap is employed. It is possible to stuff an AppleEvent into the application parameters

field of the launch block[3], the system guarantees that this will be the first event received by the new application. This insures that the child process receives all the information it needs to begin processing the service right away, thereby improving response time.

Asynchronous Notification Routines

ASRs are used my MacTCP to let an application know about certain events, including remote termination and data arrival, related to a particular stream. If an application wants to receive such events, it registers a routine with MacTCP at the time of stream creation and then the routine is called by the system when such events occur.

There is no way to change an ASR for a given stream once it has been registered, and since the Internet Server is responsible for all stream creation, it makes it difficult for child processes to access this feature of MacTCP. The information can't be bundled into an AppleEvent and handed to the child for a couple of reasons. First, the ASR is called at interrupt time which means that it can't allocate or move memory, hence it can't create or send AppleEvents. Also, if it were possible to use an AppleEvent, the notification would show up in the application's event queue instead of being processed immediately which could possibly introduce timing problems.

Instead, this system uses a really ugly hack. If a child process wants to register an notification routine, it must do so with the server. It does this by sending an AppleEvent containing the address of its ASR, its own process serial number, the MacTCP stream it's related to, and an optional user value that will be passed into the routine every time it is called. The server builds a list of ASRs using this information. When the servers notification routine is called it performs a lookup based on stream pointer and calls the appropriate routine in the child process.

Child Exit and Cleanup

It is up to the individual child processes to release the MacTCP stream they were handed and to dispose of the memory allocated for it. This is mandatory, once the server has passed a stream pointer to a new process it forgets about it completely. The release call to the MacTCP driver returns a pointer to the buffer originally given to the system in the create call.

The list of notification routines is managed automatically by the server by listening for 'Child Died' events which are sent to it by the system when a child process has exited. Upon receiving such an event, the server looks up the ASR by using the process serial number of the child as a key and then removes that entry from the list. However, this doesn't work for children sub launched by the remote shell process because the server isn't sent 'grandchild died' events. Instead, they must manufacture their own child died event and send it to the server.[4] I admit, this is gruesome, but the only other alternative was to keep the remote shell process alive maintaining its own list of ASRs much like the server does. This would keep the system transparent, but seemed like needless overhead.

Programming New Children

There are very few requirements imposed on a programmer who wants to make new servers for the system. In fact, there are only two: the program must be able to communicate with the MacTCP driver[5] and must be able to receive at least one AppleEvent, and send another if it wants to register an ASR.

The specific AppleEvents are described in the figures below, their format is copied from the AppleEvent chapter of *Inside Macintosh Volume VI*. Full source code for the Internet Server and several example daemons is provided.[6] Included is a C++ library implementing a generic background only application which can easily be subclassed to implement a specific service. There is also a small library of TCP and UDP calls.

The following figure represents the AppleEvent

[3]By using the AECoerceDesc routine you can coerse the data contained in an AppleEvent into a different type. See *Inside Macintosh, Vol VI* pp 101.

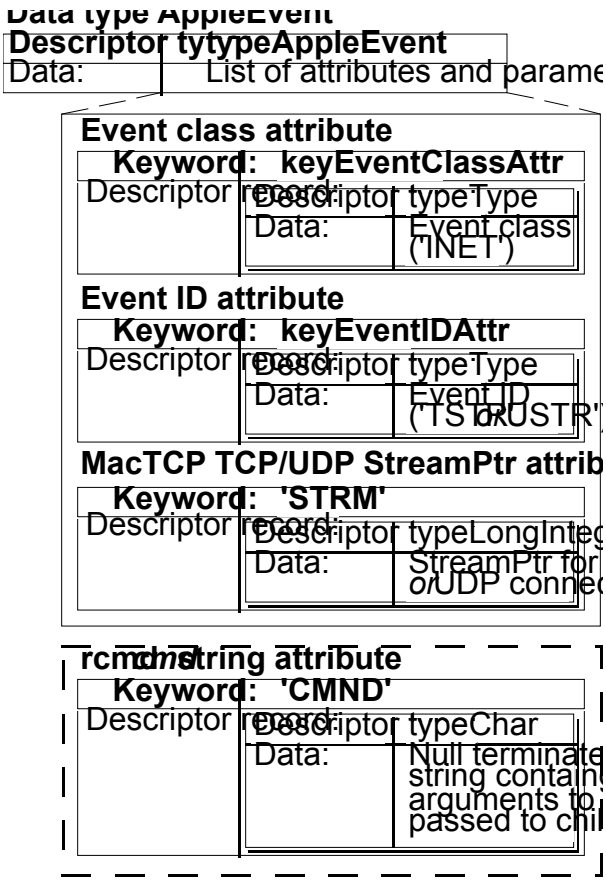
[4]Only if the process has registered an ASR with the server.

[5]For information on MacTCP programming, please consult the *MacTCP Developers Kit,* APDA #M0217LL/A

[6]If not on the same disk as this paper, please send mail to davidp@cst.usc.edu requesting it.

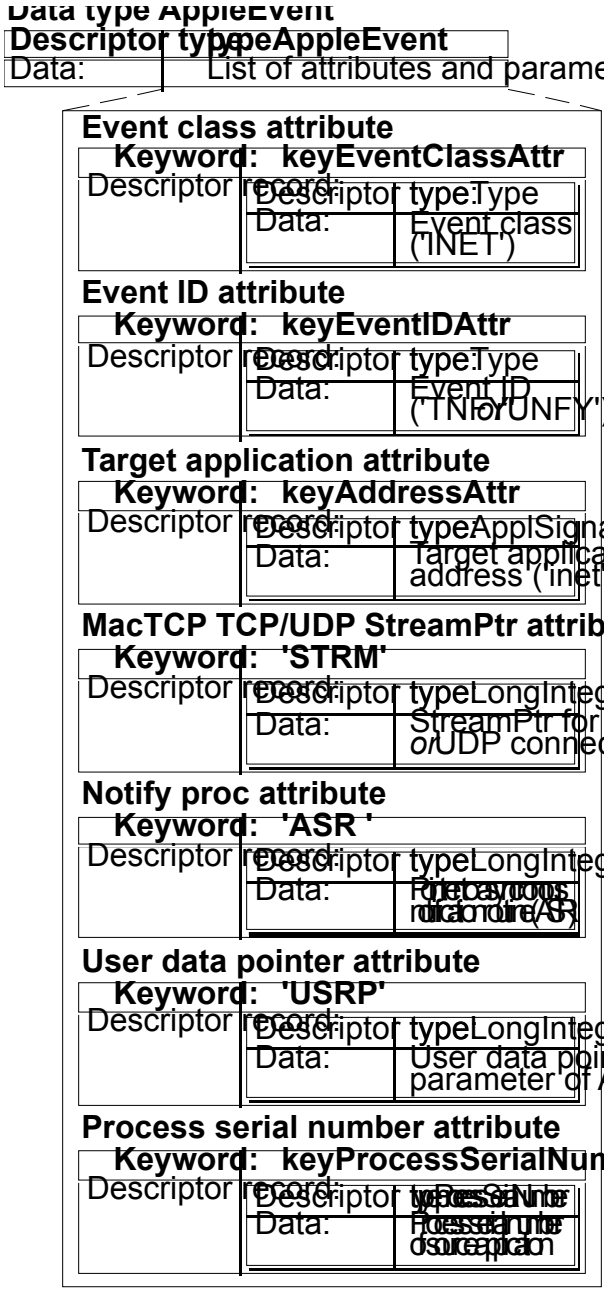
that a child must respond to on startup. The event ID determines the transport protocol being used, either TCP or UDP. The event contains a long integer which is to be interpreted as a stream pointer returned from the create call to the MacTCP driver.

If this is event is being sent to a child of the remote shell process it also contains the string generated by the peer process on the remote machine.



This figure shows the format of the event used to register an asynchronous notification routine with the server. The destination address of this AppleEvent is always ‘inet’ – the signature of the server process.

The event contains the child’s process serial number, the address of the child’s notification routine, a user value to be passed to the routine when it is called, and the MacTCP stream pointer this routine is related to.



4. Problems

There are several holes in the system, this section will look at some of the major problems with the current design of the Internet Server.

Security

There is none. This isn’t quite as bad as it sounds except in the case of the remote shell service.

Instead

of introducing yet another set of accounts and passwords for people to manage the issue was ignored. Hopefully with the introduction of A.O.C.E[7] there will be a user authentication service built into the system software that the Internet Server could take advantage of.

Port Numbers

Currently, the user must know the specific 'well known' port number of a service in order for the server to listen on it – an awkward complication in the server configuration. A cleaner solution would be to create an remote procedure call library that could be used to contact a server to obtain the proper protocol name to port number mapping. In essence an equivalent to the UNIX *getservbyname* library call. It would also be possible to keep a local copy of a services file, but that would require another piece of the system to be maintained by the user.

Notification Manager

There are some limitations on using the Notification Manager in conjunction with faceless background applications. There is no way to post a polite notification and then let the user bring your application to the front and deal with the problem on their terms. When using BOAs, you can only put up a dialog – immediately and in front of everything else. Background processing still occurs, so the machine isn't froze. Its just annoying.

Asynchronous Notification Routines

The current mechanism of jumping from one routine to another is just plain ugly. In addition, to work reliably it requires masking off interrupts while updating the ASR list. I couldn't think of another solution besides walking through the system heap looking for the previous ASR and changing it by hand though. Jumping from procedure to procedure seemed like the lesser of the two evils.

Conclusion

The software presented here goes a long way toward integrating a Macintosh into a heterogeneous network environment. Most of the system is hidden, and the one user interface point borrows heavily from current pieces of the operating system thereby providing such network services in a Macintosh-like manner.

The system is completely open and easily extended. The only requirements of a child server is that it respond to a certain AppleEvent. There are no libraries that must be used, or data structures to be aware of.

Also, this implementation presents almost no burden to the rest of the machine. No idle processing is performed, and because no patches or drivers are installed the possibility of system conflicts is minimized.

Bibliography

[1] Apple Computer, Inc. *Inside Macintosh,* Files, Processes, Memory, Volumes I-VI. Addison-Wesley, 1985-1992.

[2] Apple Computer, Inc. *MacTCP Documentation Kit*. 1991.

[3] Apple Computer Inc. *New Technical Notes: Background Only Applications*. December 1992.

[4] The Regents of the University of California, miscellaneous code from 386BSD source distribution. 1983, 1991.

[5] W. Richard Stevens. *UNIX Network Programming*, Prentice Hall 1990.

[6] The Net. Especially the contributors to comp.sys.mac.programmer.

[7] Apple DTS.

[7]Apple Open Collaboration Environment