Converting FMI/OS to the 9P2000 distributed file system protocol

Erik Dalén
edalen@kth.se

November 16, 2007

Contents

Abstract

FMI/OS is a microkernel OS inspired by Plan $9^{\mathrm{TM}1}$ and $\mathrm{QNX}^{\mathrm{TM}2}$. When FMI/OS development started (under the name VSTa[3]) most details about the 9P protocol used in Plan9 where unavailable, so it used it's own IPC[4] protocol instead [1]. This paper researches how to convert FMI/OS to the 9P2000[5] protocol while retaining its' microkernel design and having a high degree of API compatibility with Plan 9.

The benefit of this would be easier porting of new file system servers[6] and drivers from Plan 9. As Linux now has 9P support it would also make it easier to communicate with both Plan 9 and Linux systems. At the time being FMI/OS has quite few developers, so making it easier to reuse code from other OSes would allow us to develop the OS more quickly.

[1]`http://cm.bell-labs.com/plan9/`

[2]`http://www.qnx.com/`

[3]`http://www.vsta.org/`

[4]Inter-Process Communication

[5]9P2000 is the latest revision of the 9P protocol, in this text 9P and 9P2000 are used interchangably

[6]In this paper server means a microkernel server which in FMI/OS implements all device drivers, file system drivers and networking

1 Introduction

In Plan 9 the kernel translates all file operation system calls to 9P requests, using the kernel resident mnt device [2, p503]. The 9P requests are then sent over a file descriptor to a 9P server[7], this server can be either local or remote, as network connections are presented as files in Plan 9.

FMI/OS shares some of those aspects with Plan 9, but there's also significant differences. In FMI/OS, there are no system calls for file operations, they are instead handled by the applications in the C library. For each file operation the C library sends an IPC request to a server that performs the request and sends the results back. These requests use a special protocol described breifly below. This design allows the kernel to do much less and is thereby much simler than conventional kernels, the system is also more modular.

2 Method

To determine how it can be done I have ported the Plan 9 server library, lib9p, to FMI/OS. Changed it to communicate over IPC system instead of over files, and ported a simple server using that. I have also written a few test programs to interact with the server over the IPC.

The major piece of work remeaining is to convert the C library to use 9P for the standard file operations and porting the existing servers to use lib9p.

3 The FMI/OS IPC protocol

In FMI/OS servers open *ports* that the clients can connect to. A connection to a port is called a *channel*, a bit similar to TCP/IP ports and connections. The following syscalls are used for IPC:

3.1 Server functions

```
msg_port(port_name port, port_name *portp)
msg_close(port_name port)
msg_receive(channel_t port, struct msg *msg)
msg_accept(long who)
msg_reply(long who, struct msg *msg)
msg_err(long who, error_t errno)
```

3.2 Client functions

```
msg_connect(port_name port, uint mode)
msg_disconnect(channel_t channel)
msg_send(channel_t channel, struct msg *msg)
msg_portname(channel_t channel)
```

[7]Servers refer to processes that export a file system, such as device drivers and file system servers, which are all outside the kernel in FMI/OS. Not network servers.

msg_port() Is used by a server to open a new port, the number of the port can either be specified by the server or it can let the kernel pick a free one for it.

msg_close() Used by the server to close a port.

msg_receive() Used to listen for a message on a specified port, the server is blocked until a message arrives. A hande of the sender is embedded in the message, it is then used when replying to the message.

msg_accept() If the message is of type M_CONNECT, this is used to accept the connection.

msg_reply() Used by the server to reply to a message.

msg_err() Used by the server to reply with an error. Can also be used instead ot msg_accept() to decline a new connection.

msg_connect() Connect to a specified port, returns a channel or error.

msg_disconnect() Disconnect a channel.

msg_send() Used to send messages from a client to a server. The client is blocked until the server has replied.

3.3 Message structure

In FMI/OS each message has the following "C" data structure:

```
typedef
struct msg {
        long m_sender;
        int m_op;
        long m_arg;
        long m_arg1;
        seg_t m_seg[MSGSEGS];
        int m_nseg;
} msg_t;
```

m_sender is an identifier for the sender of the message which is added by the kernel. m_op is the requested operation, each file system operation has its' own value, but also special IPC operations like connecting and disconnecting. m_arg and m_arg1 are two arguments sent, usually used for small messages that don't need to send a segment of memory. m_nseg is the number of segments the message contains, and m_seg the actual segments, each with a pointer to the location in memory and the lenght of the segment. The number of segments is limited to four.

In FMI/OS each filesystem operation sets m_op to the corresponding value and in the case of for example a write message or the return of a read message the segments are used for the actual file data.

4 Implementation

To implement 9P in FMI/OS I will use a special value in m_op to designate that the message is a 9P message, and then send the entire 9P message in the first segment, leaving m_arg and m_arg1 unused. I think this is the easiest solution and will also enable changes to the IPC system in the future without much change to the 9P code.

The Plan 9 file server library, lib9p, which most servers are using uses a data structure called **Srv**[2, p210] in which the server registers callback functions for file operations that the client performs. The server also registers file descriptors to use for communication in it. The server finishes by calling the srv() function, and then the library reads 9P messages and gives hands the requests to the appropriate callback function which returns the reply which the library sends back.

In the FMI/OS port I had to change the **Srv** structure to record a message port to communicate over instead of file descriptors, and also add a new callback function for IRQ messages which is needed to write device drivers who are also file servers outside the kernel in FMI/OS.

On each message port several clients can connect, so I also had to add support for multiple clients to lib9p, but that is handled by the library should be fairly transparent when writing a server. The **Srv** struct contains a list of clients, and when a new client connects or disconnects the library adds and removes the client from the list. Each **Req** structure that is passed to the callback functions also contain a pointer to the Client struct in case the server wants to handle each client differently. The pool of active file ids[2, p214] also had to be moved from the **Srv** structure to the Client structure, as that is a per-client state.

4.1 Plan 9 compatibility

Even though several changes had to be done internally to lib9p, the API has remained mostly the same. So far one server has been ported from Plan 9 to FMI/OS, and it required no changes to the callback functions in it and only minor changes to the initialization routines, it had to open a port instead of file descriptors and advertising the server in /srv isn't implemented yet, so that option was removed.

Some changes to the layout of the structure of various C structs used in the API also had to be changed due to incompatibilities between the compilers used in FMI/OS and Plan 9[8]. Most notably the per file struct **File**[2, p216] had to be changed slightly as kencc allowes unnamed members of structs to be addressed automatically which standard C doesn't allow. Detecting changes needed in servers when porting them due to this change in API is generally easy though as the compiler will give warnings and errors indicating them.

So far communication between FMI/OS and other systems capable of 9P such as Plan 9 and Linux has not been attempted as the changes needed to be done in the C library and network proxying server are yet to be done, but once they are I don't foresee any compatibility problems in that aspect as the 9P protocol has remained unchanged in the port to FMI/OS.

[8]GCC in FMI/OS and kencc in Plan 9

4.2 Forking and sharing connection

POSIX states that a child process should inherit all file descriptors of the parent process[9]. In the C library each file descriptor is stored in a structure containing the channel it is on, the file id on that channel and the position in the file, the file descriptors are then indexed by a number. When a process forks the child process needs to create a copy of the file descriptor table that it can use for further file operations.

A problem with the FMI/OS IPC system is that it does not allow two processes to share a channel to a server. Instead there is a special, M_DUP, message a client can send to ask the server to duplicate the channel, creating a new one with the exact same state. In 9P there is no such M_DUP message, there are three possible solutions to this problem:

4.2.1 Solution one

It would be possible to make lib9p recognise M_DUP messages and create a copy of the client in its' internal structures. But this solution would not work if the servers stored additional per-client data apart from that stored by lib9p, which could make porting servers from Plan 9 more difficult. Additionally it would only work on local servers as there is no M_DUP message in the 9P protocol, so another solution would be required for network mounted file systems.

4.2.2 Solution two

It would also be possible to make the C library reconnect to all servers after a fork and reopen all open files. There are two problems with this however, first this would require reauthenticating to all servers, which could force the C library to ask the user for a password if it wasn't saved, clearly not something that you would want to do on every fork(). It would also be impossible to reopen files that have been renamed as the C library wouldn't be able to find them.

4.2.3 Solution three

The third solution would be to change the kernel to allow several clients to share a channel to a server. This would require the C library to not try to close active file ids[9] that are shared with another process. It could also lead to file id collisions when trying to open a file using a file id that has already been used by another client on the same channel, but in this case the server should report an error and the client can try again with a new file id. Possibly the forked clients can create a new channel to the server and open all new files using that channel, which would avoid this problem.

I think the third option is the best alternative to implement, as it offers best compatibility when communicating with other operating systems using 9P.

4.3 Blocking

Another problem with the FMI/OS file protocol is that all IPC operations are blocking, so when sending a message on a channel the process is blocked until there is a reply from the server. This makes the code quite easy and also

[9]With the 9P clunk message.

allows for some optimizations in the kernel regarding the message passing. It doesn't need to queue messages as there can only be one message on a channel at a time, so it can copy the message directly from the sender to the recipent without needing to store it in the kernel memory space.

This wasn't a big problem in FMI/OS before as all files opened used a new channel for the communication, so blocking a channel only meant that file operations on that specific file had to wait until the previous operation was finished. However in the 9P version of FMI/OS all file operations on a server use a single channel to avoid redoing the authentication every time a new file is opened, the 9P protocol is also designed to allow several files to be accessed over a single connection. So when a file operation is done on a file system all other operations on that entire file system is blocked until the first operation is finished.

This is mainly a problem for multi-threaded applications that are quite IO intensive. After the entire 9P conversion is finished we will have to see how big problem this is, but we might be required to rewrite the IPC system to allow several outstanding messages at once.

5 Conclusions

There is still big parts left to implement and some problems outstanding, but I beleive that the effort required to solve that would be well invested as it allows for greater interoprability with other operating systems. The ease of porting file servers from Plan 9 should also give FMI/OS many new features such as improved hardware and file system support. The lib9p also offers a better API for writing entirely new servers than the current FMI/OS server API.

References

[1] Andrew Valencia,
An Overview of the VSTa Microkernel,
`http://www.vsta.org:8080/VSTa_2fDocumentation_2fOverview`
A paper introducing the VSTa OS on which FMI/OS is based. It has some motivation of the design decisions made.

[2] *Plan 9^{TM}Programmer's Manual, Volume 1*,
Computing Science Research Center,
Bell Laboratories,
Lucent Technologies,
Murray Hill, New Jersey,
Fourth Edition,
2002.
`http://cm.bell-labs.com/sys/man/`
Contains the man-pages describing the Plan 9 API and the 9P protocol in detail.

[3] *Plan 9^{TM}Programmer's Manual, Volume 2*,
Computing Science Research Center,
Bell Laboratories,
Lucent Technologies,
Murray Hill, New Jersey,
Fourth Edition,
2002.
`http://cm.bell-labs.com/sys/doc/`
A collection of research papers on Plan 9, mostly written by the authors of the operating system. Especially the ones in the Introduction section and Implementation section are interesting for this paper.

[4] Sape Mullender,
Dave Presotto,
Programming Distributed Applications using Plan 9 from Bell Labs
Bell Laboratories,
Murray Hill, New Jersey, 07974.
`http://www.cs.unibo.it/ersads/tutorials/mullender.ps`
A quite detailed introduction to 9P and the API Plan 9 uses.

[5] Francisco J Ballesteros,
Notes on the Plan 9^{TM}3rd edition Kernel Source
May 8, 2007.
`http://plan9.escet.urjc.es/usr/nemo/9.pdf`
A commentary on the Plan 9 source code. A lot of information on how different things were implemented are described.

[6] Francisco J Ballesteros,
Introduction to Operating Abstractions using Plan 9 from Bell Labs
Draft - 9/28/2007.
`http://plan9.escet.urjc.es/who/nemo/9.intro.pdf`
Chapter 13. Building a File Server contains some good info on 9P and the Plan 9 API.

[7] *The FMI/OS source code*
`https://www.fmios.org/wiki/UserDocs/GettingTheSource`

[8] *The Plan 9 source code*
`http://cm.bell-labs.com/sources/plan9/sys/src/`

[9] *The Open Group Base Specifications Issue 6*
`http://www.opengroup.org/onlinepubs/000095399/`
IEEE Std 1003.1, 2004 Edition
Copyright ©2001-2004 The IEEE and The Open Group
A description of the POSIX interface.