OSF DCE SIG
Request For Comments: 51.2
June 1995

S. Strange (Digital)

A 32-BIT/64-BIT INTEROPERABILITY SOLUTION FOR DFS

1. INTRODUCTION

This document discusses DFS code changes to address the 32/64-bit interoperability problems discussed in [RFC 51.0]. The changes have been made to the DFS code at both Cray Research and Digital Equipment Corp., and both companies are currently shipping DFS products that include these changes. We would like to see these changes integrated into the DFS code base for DCE R1.2, along with the 64-bit cleanliness changes proposed in [RFC 51.1]. A description of the code changes in source-level detail is available from the author to those who have the appropriate DFS source license.

The code changes proposed here were motivated by two primary issues. First, as the two major 64-bit vendors shipping DFS products, it was important to Cray and Digital that their DFS implementations interoperated properly to support full 64-bit file access. Second, given that all the 32-bit DFS implementations currently do not handle 64-bit interoperability issues correctly, we needed to provide some level of backward compatibility. Not doing so could cause major problems, such as data loss or corruption, when a 64-bit DFS implementation was introduced into a cell of 32-bit DFS clients and servers. Although we feel strongly that the 32-bit implementations need to address these problems as soon as possible, it seemed appropriate to add some special code to the 64-bit implementations to deal with the most significant problems in the interim.

2. PROBLEM STATEMENT

[RFC 51.0] describes the interoperability problems between 32-bit and 64-bit DFS clients and servers, and suggests one or more solutions to each of the problems. In this document, we will try to parallel [RFC 51.0] as closely as possible, calling out which solution we chose for each problem, and describing the code change in more detail. While implementing the changes, we uncovered some unanticipated problems that are not discussed in [RFC 51.0]. These problems, and how they were addressed, are discussed in the appropriate places.

We summarize the problem list from section 3 of [RFC 51.0] here. For those problems that are not resolved in [RFC 51.1], we provide a subsection in the SOLUTIONS section, below.

(a) The `SAFS_FetchStatus()` function should pass back a 64-bit-wide file size. This is addressed with the 64-bit clean changes in [RFC 51.1].

(b) The `SAFS_StoreData()` and `SAFS_FetchData()` functions should examine the full 64-bit file offset to determine which file block the client wishes to access. This is addressed with the 64-bit clean changes in [RFC 51.1].

(c) The cache manager's internal `Scache` structure should have the file length declared as `hyper` rather than `long`. This was not addressed in the original Cray/Digital implementation (`long` is 64 bits on both architectures), but should be included when the changes are submitted into DCE 1.2.

(d) The `SAFS_Readdir()` function on 64-bit servers needs to return an error if the current offset into the directory is too large for the calling client to handle.

(e) There are 32/64-bit interoperability problems associated with the byte ranges in ''optimistic grant'' tokens (tokens used to reference whole files). The current code defines the end of the byte range to be 0x7fffffff, which does not sufficiently cover 64-bit files. The solution here is to extend the optimistic grant range to 64 bits, but to accept the old 32-bit range as a whole-file token when it is received from 32-bit clients.

(f) There are interoperability problems with file locking, because the original DFS client code base requested a 32-bit range to lock a whole file, which is clearly insufficient for large files. Changing 64-bit clients to request a 64-bit locking range causes problems for 32-bit servers that ignore the upper half of the locking `endRange hyper`, but fortunately, this is solved using the same checking mechanism used for optimistic grant tokens.

In addition to these specific 64-bit deficiencies in the code, we will address each of the following more general interoperability concerns, and dedicate a subsection below to each solution. What we are particularly concerned with here is interoperability with the installed base of DFS implementations that do not incorporate any of the 64-bit changes.

(a) 64-Bit Client, 64-Bit Server: Because 64-bit DFS platforms don't necessarily support file offsets as large as 2^63, a mechanism that prevents DFS clients from over-committing themselves past the offset limit for a particular server is necessary. Fortunately, there should be no installed-base problems in this case because no 64-bit DFS implementations have shipped without providing the interoperability changes we present here.

(b) 64-Bit Client, 32-Bit Server: A 64-bit DFS client must not allow applications to write past the largest file offset allowed on the server. Because many 32-bit servers are already in the field running code that is not aware of the 64-bit enhancements, the 64-bit client will often need to handle 32/64-bit interoperability without the help of the server.

(c) 32-Bit Client, 64-Bit Server: A 64-bit DFS server is capable of serving files larger than what a 32-bit client can handle. Because the 32-bit DFS client may not contain interoperability code, the 64-bit server must handle this problem in a reasonable way.

In providing solutions to these problems, we have strived for full backward compatibility with 1.1-based (and earlier) DFS clients and servers. However, there are some caveats that are noted in the SUMMARY section of this document.

3. BASIS OF 32/64-BIT INTEROPERABILITY SOLUTION

The basis for our solution to 32/64-bit DFS interoperability is an exchange of information between a server and a client when they first establish a mutual communication path. This is initiated by the client passing a ''maximum client file size'' parameter to the server via `AFS_SetParams()`. The server places this value in its internal host structure for this particular client for later reference, then returns a ''maximum server file size'' parameter back to the client. The client then places this value in its internal server host structure for this particular server for later reference. If either the server or the client do not support this interoperability design (e.g., all 1.1-based [or earlier] DFS clients and servers other than Cray and Digital), the client or server that does support it will assume a maximum file size of 2 GB (2^31 − 1).

This parameter exchange mechanism is similar to that proposed in [RFC 51.0], but note that this implementation differentiates between maximum file size on the server and maximum file size on the client. This was done because not all physical file systems on 64-bit servers necessarily support file sizes up to 2^63 bytes, while the same system may indeed be able to handle up to 2^63 bytes as a DFS client. This happens to be true of the Digital UNIX DFS platform.

4. SOLUTIONS

The following subsections correspond to the list of interoperability concerns in the PROBLEM STATEMENT section that are addressed in this RFC.

4.1. SAFS_Readdir() Function on 64-bit Servers

The `SAFS_Readdir()` function is modified to return `E2BIG` if the client can't handle the offset being returned in `NextOffsetp`. For example, an unmodified 32-bit client will receive an error if an `SAFS_Readdir()` call causes the current directory offset to exceed 2^31 − 1.

4.2. Token Management — Optimistic Grant Tokens

[RFC 51.0] discusses the fact that the original DFS code defines a 32-bit (0x7fffffff) range as the default range for optimistic-grant tokens (i.e., a ''whole-file'' token), and that this must be changed for 64-bit file support. We have found that it is acceptable to leave the optimistic grant default range at 32 bits, and simply allow the client to request a larger range when it actually needs it (e.g., when a file read request from an application goes beyond the

32-bit boundary). The token code must still be changed to honor requests for ranges beyond 32 bits, but leaving the optimistic-grant at 32 bits avoids special-casing initial token requests from 32-bit clients to 64-bit servers.

A second approach to the problem is to change the optimistic grant range to a 64-bit range (0 → 0x7fffffffffffffff). This approach requires 64-bit clients to accept the original 32-bit optimistic grant range as a ‘‘whole-file’’ token when communicating with 32-bit servers. (This check is made in `cm_HaveTokens()`.) There is a performance advantage to this approach, in that it eliminates the need for extra token requests when accessing beyond the 32-bit optimistic grant range. In addition, it allows us to increase the default whole-file lock range to 64-bits and still maintain whole-file lock semantics between 64-bit clients and 32-bit servers (see File and Record Locks section below for more details). Because of these two advantages, this is the approach we would like to see adopted by all the DFS vendors.

4.3. 64-Bit Client, 64-Bit Server

Although the 64-bit clean changes described in [RFC 51.1] go a long way towards supporting 64-bit file access between two 64-bit DFS platforms, they do not solve all the problems. This is because 64-bit servers don’t necessarily support file sizes up to 64 bits — they may support some size greater than the 32-bit limit, but significantly less than the 64-bit limit. The exchange of file size limit information between client and server can be used to solve this problem.

Although 64-bit cleanliness ensures that the upper 32 bits of `hyper` (64-bit) quantities are taken into account, it does not ensure that file size limits between 2^31 and 2^63 are handled correctly. As an example, the underlying physical file systems in Digital UNIX(tm) DFS have a maximum file size of about 2^43 bytes. Therefore, there needs to be a mechanism that prevents 64-bit clients from trying to read or write beyond 2^43 bytes if the server is Digital UNIX. This mechanism must cause an error to be returned to the client application when the offending system call (e.g., `write()`) returns. This means the checking must take place completely within the DFS client, because writes back to the DFS server are asynchronous. In other words, relying on `AFS_StoreData()` to return the error is not acceptable, because the client application won’t be made aware of the failed write. In order to solve this problem on the 64-bit side, we add logic to the 64-bit client to return `EFBIG` if an application specifies a file offset larger than what the server had specified as its maximum file size in the `AFS_SetParams` exchange.

4.4. 64-Bit Client, 32-Bit Server

There are two types of 32-bit servers that we must consider — those that are 64-bit clean and have implemented our proposed changes, and those that are not 64-bit clean (currently all 32-bit servers fall into the latter category). For 32-bit servers that implement the 64-bit changes, interoperability problems are solved by the same changes that solved the 64-bit client/64-bit server problem.

The problem we face in supporting the existing 32-bit servers that have no 64-bit modifications is solved via the `AFS_SetParams` exchange. When a 64-bit client analyzes the returned parameters from the `AFS_SetParams` call to an unmodified 32-bit server, it will find the `OSI_MAX_FILE_PARM_SERVER` parameter to be invalid, and will therefore default to a server max file size of 2^31 – 1. Thus the 64-bit client will not allow applications to read or write past the 32-bit boundary to unmodified 32-bit DFS servers.

4.5. 32-Bit Client, 64-Bit Server

From [RFC 51.0]: ‘‘What should the DFS client do when presented with a file whose size is larger than the size which the DFS client can handle? How should the size be represented to the client’s OS, for example, in a `stat` operation?’’ Two possible solutions were presented:

(a) Don’t allow 32-bit clients to access large files.

(b) Allow clients to see even the large files, but allow them to only modify or access data up to the limit which the client supports.

The Digital implementation of DFS chooses the second option, but implements it on the server, rather than on the client as described in [RFC 51.0]. This was accomplished by modifying `px_xvattr_to_afsstatus()` in `px_subr.c` to limit the file size returned to the maximum file size that the client can handle. Thus if a client’s maximum file size is 2^31 – 1, any file of that size or larger will appear to be of size 2^31 – 1 from the perspective of the client. This strategy was chosen primarily because it matches the way in which NFS V3.0 handles the same situation.

The Cray DFS implementation chose the first option. Lookups of files larger than 2^31 − 1 are failed if the client cannot handle file offsets greater than 2^31 − 1.

This topic should be revisited to decide which solution should ultimately be adopted by the DFS vendors, or to decide that this decision can be made by the individual vendors.

4.6. getattr, lseek Operation

[RFC 51.1] stated: ‘‘The `cm` will return −1 for the file size for large files for the ‘‘get attributes’’ (`stat`) operation, as specified by the POSIX 1003.8 TFA draft.’’ Because Digital chose to match the behavior of NFS V3, the `stat` operation involving a 32-bit DFS client and a Digital UNIX DFS server will return the maximum file size supported by the DFS client if the file exceeds that maximum file size. It is not clear what the final POSIX 1003.8 specification will say about this situation, so this is an area that should be revisited and discussed among the DFS vendors. The same is true for the `lseek()` operation — we do not propose any changes to `lseek()` here.

5. File and Record Locks

The original DFS code uses the range 0 → 0x7fffffff to indicate that the entire file is being locked. This must change to 0 → 0x7fffffffffffffff to support 64-bit file locking. Therefore, a 64-bit DFS client implementation requests and expects lock tokens with ranges exceeding that which can be offered by unmodified 32-bit servers. This problem is handled by the same code in `cm_HaveTokens()` that handles the 64-bit optimistic grant interoperability problem. If the range obtained from the 32-bit server is 0 → 0xffffffff (the upper half of the `endRange hyper` is ignored because it may contain garbage), the 64-bit client converts the token range to the 64-bit maximum range. Note that this strategy does not fully solve the locking problem with 32-bit servers that ignore the upper half of the `endRange`, because 64-bit clients that request specific ranges greater than 2GB will not obtain the correct token range. This problem will need to be addressed by the 32-bit servers by implementing the 64-bit clean changes specified in [RFC 51.1].

As mentioned in [RFC 51.0], there is a second issue with file locking, namely what to do when the 32-bit client requests a lock on the entire file, when the file's size is greater than the client's maximum file size. We recommend that client obtain the 64-bit lock range, but should not be allowed to read or write beyond its maximum file size.

6. SUMMARY

Reasonable interoperability among a heterogeneous network of 32-bit and 64-bit DFS clients and servers can be accomplished initially with code changes made only to the 64-bit DFS platforms. No change to the DFS IDL-defined protocol is necessary. The interoperability solution involves an exchange of information between client and server when the first contact between the two is made. This information consists of two values, the maximum file size supported by the client, and the maximum file size supported by the server. If either the client or the server do not yet support the exchange of this information, the system that does support it makes an assumption that the other is a 32-bit-capable machine running R1.1-derived (or earlier) OSF reference code. Once the client and server have exchanged this information, they can behave reasonably when dealing with large files (files greater than 2 GB in size).

7. 32/64-BIT INTEROPERABILITY SUPPORT FOR ADVANCED DFS FEATURES

This proposal does not address 32/64-bit interoperability problems that will arise with the advanced features of DFS. Fileset replication and migration, as well as the integrated DFS on-line backup mechanism, may not currently work properly between 32-bit and 64-bit DFS platforms. These issues were not initially addressed in the Cray and Digital DFS implementations because neither platform currently supports the advanced DFS features.

REFERENCES

[RFC 51.0] D. Delgado, ‘‘DFS Interoperability Issues for 32-bit and 64-bit Architectures’’, November 1993.

[RFC 51.1] S. Strange, ‘‘DFS Source Code Cleanup To Support Both 32-bit and 64-bit Architectures’’, February 1994.

AUTHOR'S ADDRESS

Stephen H. Strange
Digital Equipment Corporation
110 Spit Brook Rd.
Nashua, NH 03062
USA

Internet email: strange@zk3.dec.com
Telephone: +1-603-881-0595