N/A   T. Keiser
Internet-Draft   Sine Nomine
Intended status: Informational   S. Jenkins
Expires: March 14, 2013

                                                        A. Deason, Ed.
                                                          Sine Nomine
                                                   September 10, 2012

AFS-3 Protocol Capabilities Query Mechanism
draft-keiser-afs3-capabilities-00

Abstract

   AFS-3 is a distributed file system based upon prototypes developed at
   Carnegie Mellon University during the 1980s. AFS-3 heavily leverages
   Remote Procedure Calls (RPCs) as the foundation for its distributed
   architecture. In 2003, new RPCs were introduced into AFS-3 that
   provide for capability querying between file servers and cache
   managers. This memo provides a formal specification for that
   functionality, and provides analogous extensions to the volume server
   RPC interface.

Internet Draft Comments

   Comments regarding this draft are solicited. Please include the
   AFS-3 protocol standardization mailing list
   (afs3-standardization@openafs.org) as a recipient of any comments.

AFS-3 Document State

   This document is in state "draft", as per the document state
   definitions set forth in [I-D.wilkinson-afs3-standardisation].

time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 14, 2013.

Copyright Notice

Table of Contents

1. Introduction

   AFS-3 [CMU-ITC-88-062] [CMU-ITC-87-068] is a distributed file system
   that has its origins in the VICE project [CMU-ITC-84-020]
   [CMU-ITC-85-039] at the Carnegie Mellon University Information
   Technology Center [CMU-ITC-83-025], a joint venture between CMU and
   IBM. VICE later became AFS when CMU moved development to a new
   commercial venture called Transarc Corporation, which later became
   IBM Pittsburgh Labs. AFS-3 is a suite of un-standardized network
   protocols based on a remote procedure call (RPC) suite known as Rx.
   While de jure standards for AFS-3 fail to exist, the various AFS-3
   implementations have agreed upon certain de facto standards, largely
   helped by the existence of an open source fork called OpenAFS that
   has served the role of reference implementation. In addition to
   using OpenAFS as a reference, IBM wrote and donated developer
   documentation that contains somewhat outdated specifications for the
   Rx protocol and all AFS-3 remote procedure calls, as well as a
   detailed description of the AFS-3 system architecture.

   Unlike most network file systems (e.g., NFS), where protocol updates
   are traditionally handled in large batches, AFS-3 aims for
   incremental, evolutionary change to its wire protocol. Because of
   this lack of RPC interface major versions, it is the responsibility
   of calling peers to ascertain what capabilities (i.e., available RPC
   interfaces, call semantics, etc.) are supported by the peer.
   Naturally, this is a best-effort mechanism since the capabilities are
   assumed to be malleable by, e.g., transparent code upgrades at the
   remote peer.

   To this end, the afsint [AFS3-FSCM] and afscbint Rx RPC service
   endpoints--RXAFS (UDP port 7000, Rx service id 1), and RXAFSCB (UDP
   port 7001, Rx service id 1), respectively--were (circa 2003) each
   extended with a capabilities query RPC. These RPCs return (among
   other things) an OUT parameter containing an XDR [RFC4506] variable-
   length array of reserved fields, which future extensions could
   utilize to advertise support of new protocol extensions.

1.1. Purpose

   This memo serves several purposes:

   1. it serves as a historical record of the interfaces and semantics
       (of the file server and cache manager capabilities query
       interfaces), as they were designed and implemented circa 2003,
       and informally specified in 2006;

   2. it specifies the general data encoding and semantics for all
       present AFS-3 capabilities interfaces (thus, hopefully, serving

as a normative reference for future capability drafts); and

3. specifies a new capabilities interface for the volint Rx RPC
    service endpoint--AFSVol (UDP port 7005, Rx service id 4)--in
    order to permit future extensions to the AFS-3 volume server.

1.2. Motivations

The legacy method for extending AFS-3 protocols has been to define
new RPCs with prototypes that augment existing RPC interfaces with
additional arguments, or that supplant legacy data structures with
new data structure definitions (and associated new RPCs that
references those definitions). While this solves the XDR decoding
backwards compatibility problem, it does so at the expense of
requiring an O(n) search to find out which RPC version is implemented
on a given endpoint: by iterating backwards from the newest to the
oldest implementation of a given interface--until an invocation does
not return the RXGEN_OPCODE error. Consequently, the "get
capabilities" mechanisms specified in this document reduce the worst-
case capability probing from N round trips to 2--at the potential
expense of 1 additional round-trip, occasionally, to prevent
capabilities cache staleness.

1.3. Abbreviations

AFS   - Historically, AFS stood for the Andrew File System; AFS
             no longer stands for anything

afscbint - AFS-3 Cache Manager RPC Interface Definition

afsint   - AFS-3 File Server RPC Interface Definition

AFSVol   - AFS Volume Server Rx RPC Implementation

CM   - AFS-3 Cache Manager

FS   - AFS-3 File Server

RPC   - Remote Procedure Call

RPC-L   - Rx RPC Interface Definition Language (fork of ONC RPC
             [RFC5531] .x file format)

Rx   - The Remote Procedure Call mechanism utilized by AFS-3
             [AFS3-RX]

   RXAFS   - AFS File Server Rx RPC Implementation

   RXAFSCB   - AFS Cache Manager Rx RPC Implementation

   TTL   - Time to Live for cached data

   volint   - AFS-3 Volume Server RPC Interface Definition

   volser   - AFS-3 Volume Server

   XDR   - eXternal Data Representation [RFC4506]

2. Conventions

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

3. Capability Query Mechanism

   Many AFS-3 services permit evolution by querying the capabilities of
   the service that they are contacting. This is accomplished via a
   special RPC code point that is defined for many AFS-3 RPC services.
   This RPC returns an opaque bit string to the caller. In terms of
   RPC-L, this bit string is defined as a variable-length sequence of
   32-bit unsigned integers [I-D.keiser-afs3-xdr-primitive-types]:

```
    const AFSCAPABILITIESMAX = 196;
    typedef afs_uint32 Capabilities<AFSCAPABILITIESMAX>;
```

   This XDR encoding permits the size of a capabilities payload to grow
   quite large, while simultaneously keeping the payload small--until
   more elements in the variable-length array are allocated. Moreover,
   it allows the semantics of the XDR variable-length array to be
   defined gradually. The cost for this flexibility is that the 32-bit
   value of all zeroes MUST be treated the same as if the array index
   had not been returned at all.

3.1. Capability Bit Vector (array index 0)

   The file server and cache manager both define the first array
   slot--of their capabilities arrays--to be a bit vector, where each
   bit position SHALL be advertised as zero, until such time as a bit
   position is:

   1. allocated by the AFS Assigned Numbers Registrar;

   2. its semantics are agreed upon by the AFS-3 standardization group
       (if applicable); and

   3. the attendant functionality is implemented by the callee of the
       capability query RPC.

3.2. Interpretation by caller

   Interpreting the capabilities bit string returned by a server is
   complicated by the fact that the client and/or the server may
   implement differing subsets of the AFS-3 protocol. For this reason,
   differing subsets of the standardized capability bit vector array
   indices may be supported by the caller and callee.

   For example: the client may know how to decode the contents of
   capability array indices 0, 1, and 3; the server may implement the
   encodings of indices 0, 1, and 4. In this example, the server would
   return a 20-octet array where indices 2 and 3 are all zero bits. As
   noted earlier, the client SHALL interpret the 32-bit value of zero
   for indices 2 and 3 to mean that the server does not support the
   standardized encodings for these indices, let alone the standards
   whose capability metadata is encoded therein. Of course, the client
   would ignore the contents of array indices 2 and 4, as it has no
   means of decoding those capabilities. Therefore, the client will
   decode and interpret the intersection of known capabilities: array
   indices 0, and 1.

4. afsint Capability Query Interface

   In 2003, the AFS-3 community agreed to define file server and cache
   manager capability RPC code points. The RPC-L definition of the file
   server's (afsint) code point is as follows:

       proc GetCapabilities(
         OUT Capabilities *caps
) multi = 65540;

   The first array element in this variable-length array was defined as
   a 32-bit bit vector of capability flags (See Section 3.1). Four flag
   bits were subsequently allocated:

VICED_CAPABILITY_ERRORTRANS = 1

> Advertise support for the UAE error code translation table
> mechanism. When this flag is asserted, the server may translate
> system errno codes into the UAE namespace to preserve accuracy.

VICED_CAPABILITY_64BITFILES = 2

> Advertise support for 64-bit variants of the FetchData and
> StoreData RPCs (i.e., RXAFS_FetchData64, and RXAFS_StoreData64).

VICED_CAPABILITY_WRITELOCKACL = 4

> Advertise that RXAFS_SetLock and RXAFS_ExtendLock will permit
> ViceLockType of LockWrite (1) when the caller only possesses
> credentials conferring the PRSFS_INSERT privilege
> [afs3-stds-jaltman-2006-08-01].

VICED_CAPABILITY_SANEACLS = 8

> Hint to the client that volumes on this file server are, to the
> best knowledge of the system administrator, sane with respect to
> the lock ("k") permission bit [afs3-stds-jhutz-2006-07-19].

4.1. Cache Coherence

Clients SHOULD issue an RXAFS_GetCapabilities call frequently in
order to limit the capability incoherence time window. It is
RECOMMENDED that clients issue RXAFS_GetCapabilities where they would
have formerly issued RXAFS_GetTime (usually during periodic server
probing, and NAT table entry refreshing).

5. afscbint Capability Query Interface

In 2003, the AFS-3 community agreed to define file server and cache
manager capability RPC code points. The afscbint RPC-L definition is
as follows:

```
proc TellMeAboutYourself(
  OUT struct interfaceAddr *addrs,
  OUT Capabilities *caps
) = 65538;
```

The semantics of the addrs parameter are as defined for
RXAFSCB_WhoAreYou (a multi-home aware evolution beyond RXAFSCB_Probe
[AFS3-FSCM], which was implemented by IBM during the 1990s). The
first array element in the variable-length caps array was defined as

a 32-bit bit vector of capability flags. One flag was subsequently
allocated and standardized:

CLIENT_CAPABILITY_ERRORTRANS = 1

   Advertise support for the UAE error code translation table
   mechanism. When this flag is asserted, the server may translate
   system errno codes into the UAE namespace to preserve accuracy.

5.1. Cache Coherence

Clients SHOULD issue an RXAFSCB_TellMeAboutYourself call frequently
in order to limit the capability incoherence time window.

6. volint Capability Query Interface

This memo introduces a capabilities namespace, and GetCapabilities
interface to the volint service. The GetCapabilities interface SHALL
be be functionally identical to the previously-defined
RXAFS_GetCapabilities interface (see Section 4). Its RPC-L
definition SHALL be:

```
proc GetCapabilities(
    OUT Capabilities * capabilities
) = XXX;
```

                              Figure 1

The "Capabilities" type referenced here is the same one utilized by
the afsint interface. As with that interface, the first index in the
capabilities array MUST be interpreted as a 32-bit bit vector, where
all bits SHALL be set to zero--until such time as they meet the
requirements set forth in Section 3.1.

6.1. Cache Coherence

One important distinction between this capability querying interface
and the ones utilized by afsint is: afsint is a stateful circuit --
file servers can reset the cached state across themselves and clients
via the RXAFSCB_InitCallBackState, RXAFSCB_InitCallBackState2, and
RXAFSCB_InitCallBackState3 RPCs. Because volint is a stateless (with
the exception of rxkad and voltrans) client/server protocol, there is
no means of maintaining volint capabilities cache coherence. It is
RECOMMENDED that clients receiving RPC error codes, or extended union
legs that they cannot decode, perform a new AFSVolGetCapabilities
invocation to ensure that capabilities cache incoherence is detected.

Clearly, the above technique is open to races; volint clients SHOULD
try to limit race probability by minimizing the time window between
GetCapabilities calls, and invocation of capabilities-dependent RPCs.
All volint clients MUST flush cached capabilities at most two hours
after retrieving them via AFSVolGetCapabilities.

7. Acknowledgements

The author would like to thank, in particular, Jeffrey Altman, and
Jeffrey Hutzelman for their contributions to the 2006 mailing list
discussions regarding the capabilities RPCs; and Derrick Brashear for
the capabilities RPC language he wrote in
[I-D.brashear-afs3-pts-extended-names], which the author found quite
useful while writing this specification.

8. IANA Considerations

This memo includes no request to IANA.

9. AFS Assigned Numbers Registrar Considerations

This memo makes one registry allocation request of the AFS Assigned
Numbers Registrar.

9.1. AFSVol Capabilities Registry

This memo requests the allocation of a new registry with the formal
name "AFSVol Capabilities". This registry will be used to track
allocations of AFSVol capability bits. The capability bit namespace
contains 6272 bits, subdivided into 196 32-bit buckets. Allocation
requests for this namespace MUST be in the form of an RFC.
Furthermore, final approval for allocations SHALL be made by a
Designated Expert [RFC5226] to be nominated by the AFS-3 Working
Group. Should the AFS-3 Working Group be unable to assign a
Designated Expert, the AFS Assigned Numbers Registrar will be free to
appoint one or more Designated Experts to aid the registrar in the
process of vetting requests for this namespace. All allocation
requests for this registry MUST include the following information:

o capability name, and

o RFC section reference to definition of how this capability bit
   alters AFSVol protocol semantics.

In addition, an allocation request MAY include any of the following

   optional elements:

   o capability description,

   o desired capability bucket number and bit position,

   o RFC section reference to discussion regarding backwards
      compatibility, or

   o RFC section reference to relevant security considerations.

10. Security Considerations

   Given that these capability querying interfaces may be invoked over
   unprotected (e.g., rxnull) connections, application developers should
   be extremely careful when utilizing capability data to negotiate
   security-related mechanisms. When such functionality is required,
   the implementor should make every effort to access the required
   capability bits over an Rx connection whose security class guarantees
   the capability bits are at least integrity-protected.

11. References

11.1. Normative References

   [I-D.keiser-afs3-xdr-primitive-types]
              Keiser, T., "AFS-3 Rx RPC XDR Primitive Type Definitions",
              draft-keiser-afs3-xdr-primitive-types-00 (work in
              progress), June 2011.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
              IANA Considerations Section in RFCs", BCP 26, RFC 5226,
              May 2008.

11.2. Informative References

   [AFS3-FSCM]
              Zayas, E., "AFS-3 Programmer's Reference: File Server/
              Cache Manager Interface", Transarc Corp. Tech. Rep. FS-00-
              D162, August 1991.

   [AFS3-RX] Zayas, E., "AFS-3 Programmer's Reference: Specification
              for the Rx Remote Procedure Call Facility", Transarc Corp.

Tech. Rep. FS-00-D164, August 1991.

[CMU-ITC-83-025]
Morris, J., Van Houweling, D., and K. Slack, "The
Information Technology Center", CMU ITC Tech. Rep. CMU-
ITC-83-025, 1983.

[CMU-ITC-84-020]
West, M., "VICE File System Services", CMU ITC Tech.
Rep. CMU-ITC-84-020, August 1984.

[CMU-ITC-85-039]
Satyanarayanan, M., Howard, J., Nichols, D., Sidebotham,
R., Spector, A., and M. West, "The ITC Distributed File
System: Principles and Design", Proc. 10th ACM Symp.
Operating Sys. Princ. Vol. 19, No. 5, December 1985.

[CMU-ITC-87-068]
Howard, J., Kazar, M., Menees, S., Nichols, D.,
Satyanarayanan, M., Sidebotham, R., and M. West, "Scale
and Performance in a Distributed File System", ACM Trans.
Comp. Sys. Vol. 6, No. 1, pp. 51-81, February 1988.

[CMU-ITC-88-062]
Howard, J., "An Overview of the Andrew File System"",
Proc. 1988 USENIX Winter Tech. Conf. pp. 23-26,
February 1988.

[I-D.brashear-afs3-pts-extended-names]
Brashear, D., "Authentication Name Mapping extension for
AFS-3 Protection Service",
draft-brashear-afs3-pts-extended-names-09 (work in
progress), March 2011.

[I-D.wilkinson-afs3-standardisation]
Wilkinson, S., "Options for AFS Standardisation",
draft-wilkinson-afs3-standardisation-00 (work in
progress), June 2010.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard",
STD 67, RFC 4506, May 2006.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol
Specification Version 2", RFC 5531, May 2009.

[afs3-stds-jaltman-2006-08-01]
Altman, J., "Locking, ACLs, and Capabilities", AFS-3 Stds
List 000067, August 2006, <http://lists.openafs.org/

              pipermail/afs3-standardization/2006-August/000067.html>.

   [afs3-stds-jhutz-2006-07-19]
              Hutzelman, J., "Locking, ACLs, and Capabilities", AFS-3
              Stds List 000063, July 2006, <http://lists.openafs.org/
              pipermail/afs3-standardization/2006-July/000063.html>.

   [openafs-delta-capabilities-20030304]
              Brashear, D., "DELTA capabilities-20030304", OpenAFS
              Git 2712c1202ab17436ced8b466575c8bebdd9f68b7, March 2003,
              <http://git.openafs.org/
              ?p=openafs.git;a=commitdiff;h=2712c1202ab17436ced8b466575c
              8bebdd9f68b7>.

Appendix A. Sample RPC-L for afsint Capabilities Mechanism

   The following is excerpted from an OpenAFS change-set implementing
   the RXAFS_GetCapabilities functionality
   [openafs-delta-capabilities-20030304].

```
      /*
       * Copyright 2000, International Business Machines Corporation
       * and others.  All Rights Reserved.
       *
       * This software has been released under the terms of the IBM
       * Public License.  For details, see the LICENSE file in the
       * top-level source directory or online at
       * http://www.openafs.org/dl/license10.html
       */

      const AFSCAPABILITIESMAX = 196;
      typedef afs_uint32 Capabilities<AFSCAPABILITIESMAX>;

      /* Viced Capability Flags */
      const VICED_CAPABILITY_ERRORTRANS   = 0x0001;
      const VICED_CAPABILITY_64BITFILES   = 0x0002;
      const VICED_CAPABILITY_WRITELOCKACL = 0x0004;
      const VICED_CAPABILITY_SANEACLS     = 0x0008;

      proc GetCapabilities(
        OUT Capabilities *caps
      ) multi = 65540;
```

Appendix B. Sample RPC-L for afscbint Capabilities Mechanism

The following is excerpted from an OpenAFS change-set implementing
the RXAFSCB_TellMeAboutYourself functionality
[openafs-delta-capabilities-20030304].

```
/*
 * Copyright 2000, International Business Machines Corporation
 * and others.  All Rights Reserved.
 *
 * This software has been released under the terms of the IBM
 * Public License.  For details, see the LICENSE file in the
 * top-level source directory or online at
 * http://www.openafs.org/dl/license10.html
 */

const AFSCAPABILITIESMAX = 196;
typedef afs_uint32 Capabilities<AFSCAPABILITIESMAX>;

/* Cache Manager Capability Flags */
const CLIENT_CAPABILITY_ERRORTRANS  = 0x0001;

const AFS_MAX_INTERFACE_ADDR  =  32;
struct interfaceAddr {
    int         numberOfInterfaces;
    afsUUID     uuid;
    afs_int32   addr_in[AFS_MAX_INTERFACE_ADDR];
    afs_int32   subnetmask[AFS_MAX_INTERFACE_ADDR];
    afs_int32   mtu[AFS_MAX_INTERFACE_ADDR];
};

proc TellMeAboutYourself(
  OUT struct interfaceAddr *addrs,
  OUT Capabilities *caps
) = 65538;
```

Appendix C. Sample RPC-L for volint Capabilities Mechanism

```
       /*
        * Copyright 2000, International Business Machines Corporation
        * and others.  All Rights Reserved.
        *
        * This software has been released under the terms of the IBM
        * Public License.  For details, see the LICENSE file in the
        * top-level source directory or online at
        * http://www.openafs.org/dl/license10.html
        */

       const AFSVOL_CAPABILITIES_MAX = 196;
       typedef afs_uint32 AFSVolCapabilities<AFSVOL_CAPABILITIES_MAX>;

       proc GetCapabilities(
         OUT AFSVolCapabilities * caps
       ) = XXX;
```

Authors' Addresses

   Thomas Keiser
   Sine Nomine Associates
   43596 Blacksmith Square
   Ashburn, VA 20147
   USA

   Email: tkeiser@gmail.com

   Steven Jenkins

   Email: steven.jenkins@gmail.com

   Andrew Deason (editor)
   Sine Nomine Associates
   43596 Blacksmith Square
   Ashburn, Virginia 20147-4606
   USA

   Phone: +1 703 723 6673
   Email: adeason@sinenomine.net