kAFS

Overview
What is it?

An implementation of the AFS filesystem in the Linux kernel
Still needs the userspace utilities
Who?

David Howells <dhowells@redhat.com>

GSoC contributions

Jacob Thebault-Spieker <summatusmentis@gmail.com>
Wang Lei <wang840925@gmail.com>
Why write kAFS?

Red Hat needed an AFS solution for a customer
OpenAFS cannot be included in the Linux kernel
  IPL vs GPL
    Runs on everything: #ifdef hell
Disliked Arla's architecture
  Had a potential fundamental deadlock flaw at the time
  Userspace transport: potential OOM problems
Possible to write our own
  Wrote kAFS initially as a prototype
  Can take advantage of Linux kernel-only integration
Why write kAFS?

Most of all:

   It's a challenge!

    It's *fun*!
Goals of kAFS

Write a complete kernel filesystem that can drop-in replace OpenAFS's kernel module
Able to use OpenAFS userspace tools
The ultimate goal

Fedora/RHEL integration
Install by default
Set up at system installation
kAFS Architecture

kAFS comprises a number of components within the kernel:

- Keys/keyring management
- The AF_RXRPC socket family
- FS-Cache local cache
- An AFS filesystem driver

All within the kernel: no userspace component
Keys

Data required by kernel services for secure operation:
  Identities
  Authorisation tokens
  Encryption keys
May be used by userspace too
Keys: Keyrings

Keyrings are special keys that hold other keys
Contents can be added and deleted
Processes have special keyrings that can be inherited and shared

Session keyring
Set up by PAM upon login
Inherited across fork()
Keys: Use

Allow kernel filesystems and drivers to look up and cache keys
Allow user programs to propose and look up keys
*add_key(), request_key() and keyctl() system calls*
*pam_keyinit module*
*keyctl program*
Keys: After login

After logging in, I see:

[dhowells@andromeda ~]$ cat /proc/keys
17517315 I--Q-- 1 perm 1f3f0000 4043 -1 keyring _uid_ses.4043: 1/4
1bac056e I--Q-- 5 perm 1f3f0000 4043 4043 keyring _ses: 1/4
2ea802a8 I--Q-- 3 perm 1f3f0000 4043 -1 keyring _uid.4043: empty

[dhowells@andromeda ~]$ keyctl show
Session Keyring
-3 --alswrv 4043 4043 keyring: _ses
782762664 --alswrv 4043 -1 \_ keyring: _uid.4043
AF_RXRPC

socket() interface family:
  (fd = socket(AF_RXRPC, SOCK_DGRAM, PF_INET));
Can be used from the kernel or from userspace
Does RxRPC remote procedure call transport in the kernel
   Uses recvmsg() and sendmsg()
Secure
   Uses keys to pass security tokens
   Can do authenticated and encrypted transfer
   Uses setsockopt() to set parameters
Can be a client or a server
   connect(), bind() and listen() are available
**AF_RXRPC**

RxRPC connections are shared between callers with same:
- Source, Destination, Direction, Service ID and key
New connections are created automatically to expand call slots
UDP port belongs to AF_RXRPC and so can be shared
Userspace and kernel can share
AF_RXRPC state

AF_RXRPC state can be viewed through /proc:

```
[root@andromeda ~]# cat /proc/net/rxrpc_calls
<table>
<thead>
<tr>
<th>Proto</th>
<th>Local</th>
<th>Remote</th>
<th>SvID</th>
<th>ConnID</th>
<th>CallID</th>
<th>End Use</th>
<th>State</th>
<th>Abort</th>
<th>UserID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000002</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000003</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000004</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000005</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000006</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000007</td>
<td>Clt</td>
<td>1</td>
<td>Complete</td>
<td>00000000 ffff880002c31058</td>
</tr>
</tbody>
</table>

[root@andromeda ~]# cat /proc/net/rxrpc_conns
<table>
<thead>
<tr>
<th>Proto</th>
<th>Local</th>
<th>Remote</th>
<th>SvID</th>
<th>ConnID</th>
<th>Calls</th>
<th>End Use</th>
<th>State</th>
<th>Key</th>
<th>Serial</th>
<th>ISerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>34</td>
<td>00000008</td>
<td>00000001</td>
<td>Clt</td>
<td>0</td>
<td>Client</td>
<td>2bed31d2</td>
<td>00000003</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>00000008</td>
<td>00000007</td>
<td>Clt</td>
<td>6</td>
<td>Client</td>
<td>2bed31d2</td>
<td>00000011</td>
</tr>
<tr>
<td>UDP</td>
<td>0.0.0.0:7001</td>
<td>90.155.74.22:7000</td>
<td>1</td>
<td>284bd9a0</td>
<td>00000000</td>
<td>Svc</td>
<td>0</td>
<td>SvUnsec</td>
<td>00000000</td>
<td>00000001</td>
</tr>
</tbody>
</table>
```
My test klog adds a key for AF_RXRPC to the session keyring:

```
[dhowells@andromeda ~]$ klog
[dhowells@andromeda ~]$ cat /proc/keys
17517315 I--Q-- 1 perm 1f3f0000 4043 -1 keyring _uid_ses.4043: 1/4
19755aa7 I--Q-- 1 1d 39390000 4043 4043 rxrpc afs@CAMBRIDGE.REDHAT.COM
1bac056e I--Q-- 5 perm 1f3f0000 4043 4043 keyring _ses: 2/4
2ea802a8 I--Q-- 3 perm 1f3f0000 4043 -1 keyring _uid.4043: empty
```

This is then picked up by AF_RXRPC based on the name of the key. The kAFS filesystem, by using AF_RXRPC, makes use of this key too.
FS-Cache

Local disk cache for network filesystems
Persistent across reboots
Designed for AFS, but also usable by NFS and others
Transparent to user
FS- Cache

NFS

AFS

ISOFS

FS- Cache

CacheFS

CacheFiles

OtherCache
Red Hat: kAFS

AFS filesystem

Supports:
- Reading and writing
- File locking
- Security
- Automounting of mount points
- Local caching
- Failover
**AFS filesystem**

Does not yet support (or in progress):

- `pioctl()`
- `ioctl()`
- `inotify()`, `dnotify()`
- AFS userspace tools
- In-DNS AFSDB VL server records
- Tuning
- Disconnected operations
PAGs

kAFS does not currently support PAGs
  Keys go in the session keyring
Standardise PAG support for OpenAFS and kAFS
  Should go in main Linux kernel
Represent each PAG as a keyring
  Keyring name is PAG ID
  Should not modify supplementary GID list
Common OpenAFS/kAFS keys
  Go in PAG keyring
OpenAFS and kAFS coexistence

Would like for OpenAFS and kAFS to be able to coexist on a machine

Ought to be trivial:
- Give them separate local Cache Manager ports
- Give them different mountpoints
- Shared keys

However...
- Pathless `pioctl()` calls are a pain
Future Technologies

Support for future technologies needs considering:

- IPv6
- Non-UDP socket types
- Newer versions of AFS protocol
- Weirder security types
AFS filesystem

Set up by:
insmod kafs.ko rootcell=cambridge.redhat.com:192.168.1.1

Mounted by:
mount -t afs \\root.afs. /afs