AutoFS on FreeBSD 6.x

An Automounting File System Implementation for FreeBSD

ADAM David Alan Martin,

and Erez Zadok

Stony Brook University

http://www.fsl.cs.sunysb.edu/

STONY BROOK
STATE UNIVERSITY OF NEW YORK

About Me

- ADAM David Alan Martin
  - Former Physics student
  - Majoring in Computer Science, and Applied Mathematics and Statistics
  - Long-time UNIX user
- Programming
  - Strong knowledge of C and many assembly languages
  - Interest in File Systems, boot loaders, device drivers, and lower-level programming
- FreeBSD
  - Recently (around 2003) started using FreeBSD
  - Even more recent (around 2006) FreeBSD Kernel programming

Background

- Status Quo
  - Berkely Automounter
    - /usr/ports/sysutils/am-utils
    - Maintained by Erez Zadok
    - Uses SUN RPC based NFS simulation to behave like a filesystem.
      - NFS is slow. Bouncing off of userspace and RPC is even slower!
  - Alfred Perlstein made an AutoFS
    - Legal issues caused this to be abandoned
    - Erez Zadok got involved and AutoFS became a GSoC for 2006.

Basics of Automounting

1. User makes a request to a file (which is unavailable)
2. An interception layer interprets this request, and causes the user process to block
   A. The automounter is instructed to mount a file system
   B. The automounter mounts the file system
   C. The automounter notifies the interception layer of successful mount
3. The interception layer unblocks the process and it goes on its way.

AM-UTILS Automounting

- Uses NFS emulation to simulate a file system for mount point services
  - RPC services for NFS are slow and baroque
  - NFS emulation opens possible security risks
  - NFS emulation causes a massive number of context switches for each RPC call
    - User->Kernel and Kernel->User transitions for each RPC "packet"
  - AMD cannot mount directly to the provided file system: "Magic" symlinks provide redirection evil.
    - A "hole" exists where accesses to the file systems can occur outside of the Automounter framework.
  - AMD's NFS emulation makes it incredibly portable, however.

AM-UTILS Replacements

- Sun AutoFS for Solaris
  - Implemented in-kernel, communicates with the AMD automounter (or others)
    - Serialized mounting behavior
- Linux AutoFS
  - Implemented in kernel, communicates with a Linux specific automounter
    - This automounter is very buggy, and crash prone
    - This automounter can hang the entire kernel, if it crashes
- KDE's HAL mechanisms
  - Not a true file system, but provides automount like facilities via the GUI backends.
  - Very user intuitive and friendly
    - Provides a Win32 like experience with respect to removable data storage.

Shortcomings of these Replacements

- Most kernel automount layers lack many useful features and capabilities
  - The automount daemon is not restartable
    - Crashing or stalling the automount daemon can cause parts of the system to wedge
  - The automount system can only handle one mount request at a time, in a serialized fashion
- KDE's HAL is not a proper file system interface: It requires the of parts of KDE
  - It does not interact well with shell scripts, and other tools.
  - The HAL is not useful for complex network file system automounter setups.

AutoFS for FreeBSD

- Google Summer of Code Project 2006
  - Benno Rice mentored
  - Erez Zadok advised
  - Alfred Perlstein has been very encouraging
- AutoFS for FreeBSD implements a protocol design for automounter file systems which solves many of the problems in previous automounter file systems.

New AutoFS Features

- Restartable protocol
  - Filesystem mount states are cached by the AutoFS kernel code, to be reported to user processes, when needed for restart.
  - Crashing AMD does not have to wedge the system anymore.
- Asynchronous mount request handling
  - Mounts can be done in parallel, and reported as completed in any order (regardless of the order of requests.)
  - Processes do not have to wait for slow mounts, just because they “asked” for a resource too late. (You’re not stuck in traffic behind the slow car.)

More Features

- Lightweight protocol
  - The protocol thus far only defines some 8 major commands
    - This makes it easy to implement for clients
  - The protocol command space is largely undefined leaving plenty of room for expansion

- Automounter session tracking lets the AutoFS permit direct mounting of filesystems.
  - No more magic symlink forest
  - No more backdoor to the mount path

Basic Protocol Commands (AutoFS)

- Commands AutoFS can send to AMD
  - Request Mount
    - Request Unmount (variant of mount request)
    - Other requests can be pooled here?
  - Greeting
    - Sends mount state to client daemon on connection
  - Mount modification response
    - Inform Automounter of the new state of mount settings, after a change request
  - Acknowledge
    - Sent to complete protocol transactions

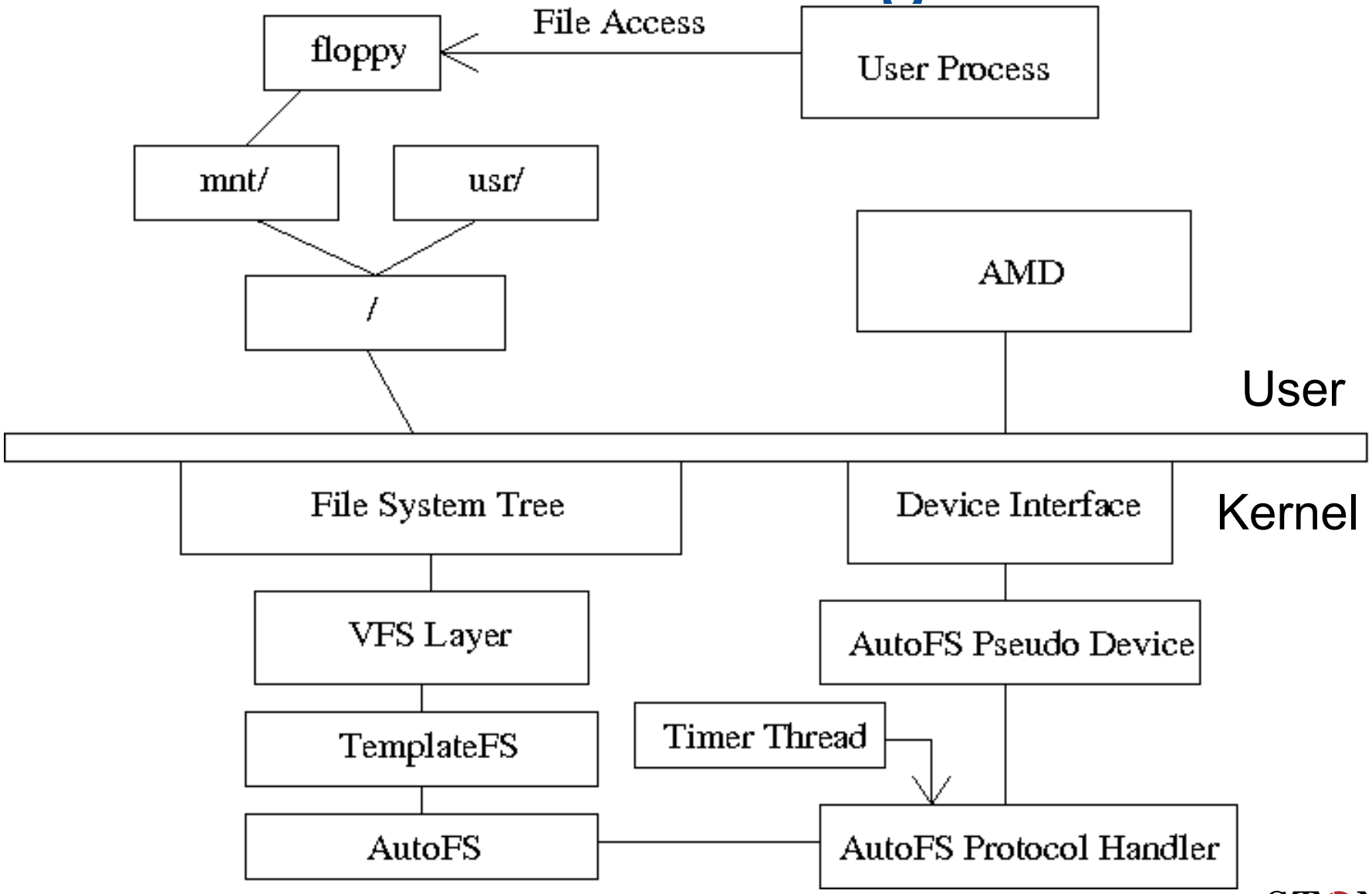
Basic Protocol Commands (AMD)

- Commands AMD can send to AutoFS
  - Mount Done
    - Also reports unmount done
    - Can be extended to handle reporting for other requests
  - Hello
    - Initializes an AutoFS session
  - Greeting Response
    - Passes to AutoFS any extra mount management information AMD would like to send
  - Modify Mounts
    - AMD can request that AutoFS change its mount management information at any time this way
  - Acknowledge
    - Used to note the end of an AMD initiated transaction

Putting It Together

- AutoFS Components
  - File system interface
  - Protocol channel interface
  - Protocol handler
  - Timer expiry mechanism
  - Shared data amongst all these components
- AutoFS's kernel module implements many things
  - Device Driver
  - File system
  - Kernel thread
  - Protocol endpoint

Overall Design



STONY BROOK
STATE UNIVERSITY OF NEW YORK

What is this “TemplateFS” I see?

- FreeBSD provides a very rich VFS mechanism for implementing all kinds of file systems.
  - Unfortunately this layer is not very friendly for “virtual” or “synthetic” file systems with no backing store
    - procfs
    - linprocfs
    - linsysfs
    - Many others are also in this category
  - Fortunately, procfs, and linprocfs share a common code library: pseudofs.

Pseudofs

- Written by DES
- Found in src/sys/fs/pseudofs
- Provides an interface for making a static synthetic file system
  - Provides hooks for handling many vnop calls on any “file” and some vfsop calls as well.
    - Procfs and linprocfs use this to implement things like /proc/1/cmdline functionality.
    - Linsysfs uses it for similar configuration reporting.
- Pseudofs does all the dirty work of managing memory and interfacing with the VFS
  - Client code works with simple pseudofs structures and can create or delete files in the tree with ease.

What Pseudofs Needed

- Pseudofs lacked the ability to trap the vfs_lookup chain of events
  - A file system paradigm which mounts on lookup needs this
- Pseudofs lacked the ability to create non-static client file systems
  - A single global AutoFS mount is insufficient for the goals of a proper automounter system
- Because pseudofs was not able to have multiple instances of a client file system, associating file system state to a pseudofs is difficult.

Enter TemplateFS

- TemplateFS is a code fork of pseudofs created originally for convenience
  - Repatching src/sys/fs/pseudofs with my changes after cvs updates to the main source tree was troublesome
    - The changes would break procfs and linprocfs making buildworld and buildkernel difficult
    - AutoFS required this hacked pseudofs to function
  - Setting pseudofs to compile as a module would help, but as I kept changing the hacks to pseudofs, I had to rebuild this module, in a privileged directory

Future of TemplateFS

- TemplateFS's capabilities diverge enough from pseudofs to warrant a fork, unless procfs and linprocfs are altered to suit these changes.
- Someone more familiar with the procfs and pseudofs code might want to take over the TemplateFS work.
- AutoFS only requires TemplateFS's added feature set -- it is strictly separated from TemplateFS.

AutoFS Protocol Transactions

- All transactions are tagged with a "Transaction ID" to facilitate asynchronous responses
  - "TID"s can be as simple as time stamps
  - Nearly all transactions begin with the AutoFS
    - TID spoofing or collisons aren't serious issues.
    - Userspace TIDs are allocated a separate portion of the TID space (High bit set)
- Transactions can be "interlaced"

Sample Initial Transaction

- AMD says “Hello” to AutoFS
  - AMD requests to communicate a protocol version
- AutoFS responds with “Greeting”
  - Tells AMD what version of the protocol it uses
    - Version N+1 is a strict superset of N.
    - Both parties use the lower protocol version requested.
  - Tells AMD what the current AutoFS state is
    - Mounted file systems
    - Managed paths
- AMD responds with “Greeting Response”
  - Tells AutoFS what file system changes it would like to make
- AutoFS responds with “Modify Mounts Acknowledge”
  - Tells AMD what file system changes were made

Sample Mount Transaction

- User accesses a resource in an unmounted file system
  - “wine /mnt/floppy/virus.exe” for example
- AutoFS receives the vfs_lookup() request for “floppy/virus.exe”
  - Because “virus.exe” lives in “floppy/” AutoFS signals AMD to mount “/mnt/floppy”
  - AutoFS blocks the “wine” process, but lets all AMD’s children through.

Sample Mount Transaction (cont.)

- AMD runs “mount /dev/fd0 /mnt/floppy”, with AMD’s session ID, to pass through the block.
  - AMD Notifies AutoFS when the mount is done (Mount completed command.)
- AutoFS unblocks processes on “/mnt/floppy”
  - AutoFS also unblocks all stopped processes on “/mnt/floppy” (Including “wine”)
  - AutoFS also adds an expiry for “/mnt/floppy”
- AutoFS sends an “Acknowledge” to AMD, ending the transaction
  - When the expiry occurs, AutoFS sends an unmount request.
  - Unmount occurs in roughly the same set of steps.

Mount/Unmount Failures

- Mount
  - All currently blocked processes are unblocked
  - They eventually hit an ENOENT condition
  - AutoFS keeps the file system marked unmounted
    - Facilitates future mount attempts
- Unmount
  - Usually unmount failures are EBUSY conditions
  - AutoFS will just acknowledge
  - AutoFS keeps the file system mounted
    - This keeps things functioning correctly
  - AutoFS also resets the expiry timer
    - Facilitates future unmount attempts

Other Components

- TemplateFS
  - In perforce alongside the AutoFS code
- Mount_autofs
  - In perforce alongside the AutoFS code
- Afsconfig
  - In perforce alongside the AutoFS code
  - Used to configure the AutoFS system
    - Create multiple AutoFS sessions:
      - /mnt
      - /net
      - /homes
      - /servers
    - These sessions maintain their own state.

AMD Changes to Make

- Adding a protocol handler for FreeBSD AutoFS
- Making AutoFS talk to /dev/autofs%d instead of over RPCs
- These changes will help other OSes if my AutoFS is ported or rewritten for them.
- Changes to AM-Utils need to get to FreeBSD somehow

AMD Modifications

- AM-Utils is in ports -- Modifying the main tree will propogate these changes to FreeBSD
  - AM-Utils is maintained by my advisor, Erez Zadok
- AM-Utils could be brought into src/contrib
  - A simpler automounter could be written for stock FreeBSD
  - AM-Utils is very complex, but very powerful

Closing Thoughts

- AutoFS for FreeBSD can help it to get a competitive edge among other FOSS operating systems.
- AutoFS protocol is very simple, making future changes and re-implementations simple
  - The protocol helps to solve many of the issues with earlier automounter systems
- AutoFS is in perforce -- It should be in future FreeBSDs

Special Thanks

- Erez Zadok - FSL Advisor and Lead Researcher
- Benno Rice - GSoC Mentor
- Alfred Perlstein - Original FreeBSD AutoFS work, and encouragement
- Robert Watson - Advice and assistance in the early stages and encouragement
- Murray Stokely - For introducing me to everyone important at NYCBSDCon
- The entire FreeBSD community - For making me feel welcome, and anyone's assistance that I may have overlooked

Questions

- Thank you for your interest in AutoFS.