"A toolkit for user-level file systems," David Mazieres

We're reading this paper to
  1. See horrible details of making remote file systems work
  2. Understand the lab infrastructure

The paper describes a toolkit that helps you build user-level file systems
  Most file systems (disk FS, real NFS, &c) live in kernel
  Hard to modify, hard to debug, need privs
  The basic trick: user program looks like network NFS server
  Figure 1

What's going on in Figure 2?
  Why doesn't the user-level file call mount() itself?
  What does mount() do for an NFS server?
    kernel tables. vnode. file descriptor.
  Why is mount() privileged?
    setuid programs on FS.
    disk fs might be corrupted. nfs server hangs lock up client.
    misbehaving server can wedge client in other ways.
  What happens if a mounted NFS server stops responding?

What is the point of the automounter?

What happens when a user-level file system starts up?
  gives fd, dir name to automounter.

What happens when a user process first uses an automounted FS?
  lookup(d) -> automounter
    reply: fh1
  READLINK(fh1) -> automounter
    reply: "/sfs/.mnt/0"
  LOOKUP(".../0") -> am
    reply: fh2
  READLINK(fh) -> am
    no answer for now...
  automounter asks nfsmounter to mount FS's fd on /sfs/d
    nfsmounter: mount(fd, "/sfs/d")
    LOOKUP(d) -> am
      reply: fh3, not the same, real directory
    mount() then marks that vnode as mounted, with fd
  am replies to READLINK: "/sfs/d"
  LOOKUP(d) -> am
    reply: fh3
  GETATTR(fh3) now goes to FS server

What does nfsmounter do if, say, the automounter crashes?
  How does the nfsmounter learn of the crash?
  What's the goal of cleanup?

****

What's the big point?
  We might want to implement user-level file systems.
  Can do it by pretending to be an NFS server.
  Dave will help us get all the details right.

Picture. app, client NFS, udp socket, server process.

Examples?
  /proc. crypt fs. client side of network file system.
  automounter: /net/fs.mit.edu

What problems would we run into if we did this ourselves?
  Need to parse all those RPCs.
  How to mount the file system?
  What if our server crashes?
  Possibility of deadlock in buffer cache.
  Automounters have special mount-in-place problem.

What does the mount() system call do?
  For NFS, takes a socket file descriptor and a directory name.
  Looks up directory name's vnode.
  Tags vnode as a mount point, remembers socket.
  Subsequent vnode ops redirected to new type+socket.
  Note that we can mount on a directory in a mounted file system.
  Thus /net/fs.mit.edu/... can work.

What are the problems with crashed user-level servers?
  Won't respond, programs will hang forever.
  Want to unmount, to avoid future hanging processes,
    and maybe want to mount something else there.
  Unmount requires naming the mounted-on vnode.
  Special trouble if intermediate mounted file system doesn't work.

How does the paper help out?
  nfsmounter does all mounts.
    You give it your user-level server's socket, and mount point name.
    You give it *both ends* of the socket.
  nfsmounter keeps the socket file descriptor, as well as giving it to kernel.
  nfsmounter knows when your program crashes (eof on some other socket?).
  nfsmounter then answers NFS requests from the kernel on the socket.
  Returns errors for access to most files.
  But returns something reasonable along paths to mount points.
  Thus you can un-mount nested mounts.

What is the problem with deadlock?
  User-level file server needs a page of physical memory.
    To read from the disk, or to page in.
  Kernel chooses some dirty page, wants to clean by writing out.
    So user-level file server now waiting for write to finish.
  Dirty page might be from that user-level server's file system!
  It cannot respond to the WRITE rpc because it's blocked in the kernel.

How does the paper help with deadlock?
  Advice to avoid blocking disk I/O.
  Use Flash-like child processes to do disk I/O.
  Lock down all memory, to avoid page faults.

Why do you want an automounter?

```
  Without, you have to have lists of how to mount things.
    I.e. /home/to2 is toil:/disk/to2
  With, you have generic support for rule-based mounting
    I.e. /home/hostname/fsname is hostname:/disk/fsname
  Critical for large installations or wide-area sharing.

What does automounting in place mean?
  Before: /net is served by user-level automounter.
    It sees your LOOKUP for fs.mit.edu.
  After: fs.mit.edu NFS server mounted on /net/fs.mit.edu
    which is a vnode served by the automounter (tho not used for much).
    user-level automounter doesn't see any more LOOKUPs for it.

Why is automounting in place hard?
  It all starts with some random process doing a LOOKUP for
/net/fs.mit.edu
  automounter (or nfsmounter) wants to say mount(fd, "/net/fs.mit.edu")
  Kernel mount() will try to find vnode for /net/fs.mit.edu
  Trying to ask automounter (again) to LOOKUP /net/fs.mit.edu
  Worse, original process has locked /net directory vnode.

How does the paper help automounting in place?
  Must answer the original LOOKUP right away.
    So must have a vnode for the answer.
    Responds with a SYMLINK vnode pointing to /net/.mnt/0
    Client NFS code does READLINK, then a LOOKUP for /net/.mnt/0
      automounter does not answer yet
  Then automounter does whatever to get socket to real NFS server
  Asks nfsmounter to mount the socket on /net/fs.mit.edu
  mount() system call does a lookup for /net/fs.mit.edu
  How does automounter know not to treat it like a client?
    nfsmounter has some weird group ID, which is in the NFS RPC.
  This time, automounter returns an ordinary directory vnode.
  mount() succeeds.
  Now automounter answers original client's LOOKUP for /net/.mnt/0
    Returns a SYMLINK pointing to /net/fs.mit.edu
  And we're done.
```