drive (if they even have a drive). It would certainly be possible to use this model with floppys however it is not clear to us that this is as useful.

Another possibility being considered is the addition of a directive to mount the media "cached". This option would provide for some space on the local disk to be set aside as a cache for directories and files from the CD-ROM, so that the actual CD-ROM does not need to be accessed each time a file or directory is read.

6. Availability

6.1. Binary in Solaris

We are delivering the media manager into Solaris in phases, starting with minimal functionality required for the automatic mounting of CD-ROM's and floppies (diskovery). This will be delivered in a release in the near future. The next phase will include support for tape devices and will have all the interfaces documented.

6.2. Source

For support of new devices or labels on machines running Solaris, example code for devices and labels can be provided at no cost.

The product should be fairly portable to other UNIX implementations that support threads. Inquiries for full source to this product are welcome.

7. Author Information

Howard started his career at a startup, administering their machines. He then went to SRI International to do networking and kernel development. Next, he went to Sun to do kernel porting for a few years. After getting tired of the bay area, he returned to Austin to work for Tandem on various kernel development projects. Howard currently works for SunSoft's Rocky Mountain Technology Center in the Storage Management group. Howard can be reached at (719) 528-4614, or by mail at halt@central.sun.com.

8.

References

Alt93.H. Alt, ''Removable Media in Solaris,'' in *Proceedings of the USENIX Winter Conference*, USENIX Association, San Diego, California, January 1993.

Gin87.R. A. Gingell, M. Lee, X. T. Dang, and M. Weeks, ''Shared Libraries in SunOS,'' in *Proceedings of the USENIX Summer Conference*, USENIX Association, Phoenix, 1987.

McK84.
M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, ''A Fast File System for UNIX,'' in *Transactions on Computer Systems* , p. Volume 2, Number 3, August 1984.

VOLUME_ACTION= insert | notify | eject
Action that caused this program to be executed.
VOLUME_PATH=<pathname>
Path of the matched <regex> (/vol/dev/fd0/fred)
VOLUME_NAME=<volume_name>
Name of the volume in question (fred).
VOLUME_USER=<uid>
User id of the user causing the action to occur (3087).
VOLUME_SYMNAME=<symbolic_device_name>
Symbolic name of a device the volume is in (cdrom1, floppy0).
VOLUME_MEDIATYPE=<media_type>
Name of the type of media (cdrom, floppy, 8mm, mo)

Example 6: *List of environment variables*

The details of this list are beyond the scope of this paper, however for the purposes of debugging we use the following environment:

```
setenv VOLUME_PATH /vol/dev/dsk/c0t6/unnamed_cdrom
setenv VOLUME_MEDIATYPE cdrom
setenv VOLUME_NAME unnamed_cdrom
setenv VOLUME_ACTION insert
setenv VOLUME_SYMDEV cdrom0
alias ins 'setenv VOLUME_ACTION insert'
alias ej 'setenv VOLUME_ACTION eject'
```

Example 7: *Suggested debug environment*

This environment assumes you have a CD-ROM drive at c0t6, and a CD-ROM inserted named "unnamed_cdrom". As root then, /usr/sbin/diskovery can be run being told alternately that media is being "inserted" and "ejected", and actions can be easily debugged.

If one desires to use a source level debugger such as dbx, the action must be compiled with the "-g" flag. The problem with a debugger is that your shared object is not loaded into the "diskovery" image until it is actually being executed, so breakpoints cannot be set before it runs. One solution to this is setting a breakpoint in "dprintf", and having your action call this early on. Another option is to have it call an obscure system call or C library function. This requires some experimentation to get just right.

4. File Manager Support

An application that uses the action mechanism in diskovery is the file manager(1). The file manager allows the perusal of the UNIX name space in a graphical way, representing objects in the name space (files, directories, etc) as icons. These icons can be moved around and things happen in the expected way.

An extension to the file manager is to take advantage of the automatic recognition of new media. When a CD-ROM or floppy is inserted, the file manager pops up a new window positioned at the top of the newly mounted directory hierarchy. It also provides an easy to use interface for formatting and renaming media. In addition, media can be ejected through a button in the file manager window.

5. Futures

There are several possible extensions of this technology that we are examining, the first of which is automatic exporting of automatically mounted CD-ROM's. Through the automounter, a CD-ROM could be made instantly available on all machines on the network. This would allow users to peruse the /cdrom a hierarchy consisting of all the CD-ROM's on the net, rather than just the one they have in their

```
         * we return false here because audio might not be the only thing
         * on the disk, and we want actions to continue.
         */
        return (FALSE);
}
```

Example 5: *action_workman.c*

The *workman* program allows a user to play audio tracks on a CD-ROM. Workman is a public domain X program which pops up a window that allows control of the CD-ROM, as if it were a home CD player. This sample action figures out if there are audio tracks on the CD-ROM, and launches the workman program if there are.

This workman action *is* strictly an example. It has a couple of flaws, however it still serves as an acceptable action programming example.

The arguments passed into the action function are an array of action_arg structures, and the canonical argc, argv. The argc, argv are the arguments from the line in the diskovery configuration file. In this case argv[1] is the string "/home/rmtc/halt/bin/workman" (see Example 4). The array of pointers to action_arg structures represents each valid partition. Although it was unnecessary to process the array for this action, the list of pointers can be examined until the aa_path entry is NULL. The aa_mountpoint entry can be examined for the path to the mounted partition.

The first thing this action does is retrieve the environment variable "VOLUME_ACTION". This is set by the media manager when the diskovery program is executed in response to "insert", "eject", or "notify" events. The notify case is not within the scope of this paper.

There is a debugging option to the diskovery program which turns on printing from the dprintf calls. The option, -D, can be added or removed from the /etc/vold.conf file as desired (see Example 1). This flag would not be enabled for normal operation.

Once the raw (or character) device is opened, ioctl(2)'s are issued to determine if this CD-ROM has audio tracks. This is done by reading the table of contents, then reading each track entry to determine if the flag indicates an audio track.

If there is audio on the CD-ROM, a standard fork/exec operation is performed. Standard file descriptors are redirected to the console, and the DISPLAY environment variable is set-up for the X application. **Note that diskovery runs as root, so the programmer must take care to reset the real and effective user-id to protect against security holes.** A complete program would also set the group-id to something innocuous.

One flaw of this example is in the interaction between the eject(1) command and workman. The trouble is when the media is ejected via the eject(1) command, the media manager blocks further accesses to that name and workman hangs. The workman program has a flag (-X) which means "exit on eject". This makes the eject button for workman operate properly (it ejects the CD-ROM, then goes away). A possible solution to this problem is to keep the pid of the executed workman hidden away somewhere, and then kill(2) the workman process on eject. Note that when the action *is* called for an eject event, it just returns instead of doing anything.

3.4.2. Debugging an Action

Actions are best debugged without having to insert and eject media all the time. The diskovery program takes its cues from a set of environment variables which are set by the media manager.

```
/* Look through the tracks to see if we have any audio */
te.cdte_format = CDROM_MSF;
for (i = (int)th.cdth_trk0; i < (int)th.cdth_trk1+1; i++) {
    te.cdte_track = i;
    if (ioctl(fd, CDROMREADTOCENTRY, &te) < 0)
        continue;
    if ((int)te.cdte_datamode == 255) {
        found_audio = TRUE;
        break;
    }
}

close(fd);

if (found_audio == FALSE) {
    dprintf("action_workman: no audio0);
    return (FALSE);
}

dprintf("action_workman: found audio (%d tracks)0,
    th.cdth_trk1 - th.cdth_trk0 + 1);
/* start workman: don't care about errors just fire and forget */
if (fork() == 0) {
    int fd;
    charhostname[MAXNAMELEN];
    chardispname[MAXNAMELEN];

    /* child */
    chdir(rmm_dsodir);
    /* stick his error messages out on the console */
    fd = open("/dev/console", O_RDWR);
    dup2(fd, 0);
    dup2(fd, 1);
    dup2(fd, 2);
    sysinfo(SI_HOSTNAME, hostname, MAXNAMELEN);
    sprintf(dispname, "DISPLAY=%s:0", hostname);
    putenv(dispname);
    /* just to keep workman happy */
    putenv("HOME=/tmp");
    /*
     * Don't want the workman user doing anything nasty.
     */
    setuid(1);   /* set user-id to daemon */
    seteuid(1);  /* set effective user-id to daemon */

    execl(argv[1], argv[1], "-c", aa[0]->aa_rawpath,
        "-o", "-X", NULL);
    fprintf(stderr, "exec of %s failed; %s", argv[1],
        strerror(errno));

    /* bummer, it failed -- EXIT, don't return!! */
    exit(0);
}
/*
```

```
#ifndef lint
#ident  "@(#)action_workman.c 1.5     92/09/25 SMI"
#endif  lint

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <diskovery.h>

#include <sys/types.h>
#include <sys/dkio.h>
#include <sys/cdio.h>
#include <sys/vtoc.h>
#include <sys/param.h>
#include <sys/systeminfo.h>

/*
 * If this cdrom has audio tracks, start up workman.
 */

int
action(struct action_arg **aa, int argc, char **argv)
{
    struct cdrom_tochdr  th;
    struct cdrom_tocentry    te;
    int          i;
    int          fd;
    int          found_audio = FALSE;
    extern char      *rmm_dsodir;
    char         *atype = getenv("VOLUME_ACTION");

    if (strcmp(atype, "insert") != 0)
        return (FALSE);

    if (aa[0]->aa_rawpath == NULL) {
        dprintf("action_workman: no rawpath0);
        return (FALSE);
    }

    /* open the raw device so we can issue ioctl(2)'s */
    dprintf("action_workman: %s0, aa[0]->aa_rawpath);
    if ((fd = open(aa[0]->aa_rawpath, O_RDONLY)) < 0) {
        dprintf("action_workman: open %m0);
        return (FALSE);
    }

    /* read the table of contents from the CD-ROM */
    if (ioctl(fd, CDROMREADTOCHDR, &th) < 0) {
        dprintf("action_workman: ioctl %m0);
        close(fd);
        return (FALSE);
    }
```

A UFS file system is identified is by reading the superblock and checking to see if the magic number is as expected. The best way to accomplish this is to lseek(2) to the offset of the superblock, then read(2) the data, as in the example.

To compile your ident function (or action, below) use the -G flag on the SunPro C compilers.

```
cc -O -G -o ident_frob.so.1 ident_frob.c
```

3.3. Mounting

Each partition on a piece of media is examined for a known file system. If a known valid file system exists, the file system is mounted at a known location.

Floppies are mounted on /floppy/<media_name>, and CD-ROM's are mounted as /cdrom/<media_name>. If a floppy or CD-ROM has partitions, each partition is mounted (e.g. /cdrom/solaris_2_0/s0, /cdrom/solaris_2_0/s2, etc).

Dirty file systems are cleaned before mounting, or mounted read-only if they are not cleanable.

Mounts are performed with the "nosuid" option, which keeps users from carrying around floppies or CD-ROM's with set-uid programs on them. The nosuid semantic has been extended to mean that access to block and character special devices is disallowed. This keeps a user from building a file system with devices like memory or disks to which they have privileged access.

Diskovery implements a policy which requires a file system to be unmounted before it can be ejected. The media manager does not require this to be the case, however doing this greatly simplifies the most common user model.

3.4. Actions

After mounting (or unmounting) the file system(s), a list of "actions" are run (Example 4). These actions can do things like notify the file manager program that new media has arrived. In this example, for CD-ROM's, an action is provided to check the media for audio tracks and execute the "workman" program.

```
# Actions
action cdrom action_filemgr.so
action floppy action_filemgr.so
action cdrom action_workman.so /home/rmtc/halt/bin/workman
```

Example 4: *excerpt from /etc/diskovery.conf*

Actions are executed in the order specified in the configuration file. If one action returns TRUE, no other actions are executed for that event (i.e. insert or eject).

3.4.1. A Sample Action

Action functions are simple to implement. The shared object is looked for in /usr/lib/diskovery. For example, the "workman" shared object would be /usr/lib/diskovery/action_workman.so.1. The diskovery program links with the shared object, when appropriate, looks up the function name "action" and calls it with the arguments as defined in /usr/include/diskovery.h.

```
#ifndef lint
#ident  "@(#)ident_ufs.c 1.3     92/09/23 SMI"
#endif

#include <stdio.h>
#include <fcntl.h>
#include <rpc/types.h>
#include <sys/types.h>
#include <sys/fs/ufs_fs.h>

#include <diskovery.h>

/*
 * We call it a ufs file system iff:
 *  The magic number for the superblock is correct.
 */
int
ident_fs(int fd, char *rawpath, int *clean, int verbose)
{
    struct fs fs;

    if (lseek(fd, SBOFF, SEEK_SET) < 0) {
        perror("ufs seek");
        return (FALSE);
    }

    if (read(fd, &fs, sizeof (fs)) < 0) {
        perror("ufs read");
        return (FALSE);
    }

    if (fs.fs_magic != FS_MAGIC)
        return (FALSE);

    if (fs.fs_clean == FSCLEAN || fs.fs_clean == FSSTABLE)
        *clean = TRUE;
    else
        *clean = FALSE;

    return (TRUE);
}
```

Example 3: *UFS identification code*

The function is always named ident_fs. The include file <*diskovery.h*> has all the appropriate definitions for building an "ident" function. The block device is opened for the ident function, and passed in as a file descriptor *fd.* The file descriptor is held open during the identification process to gain as much buffering as possible. The *rawpath* argument is a path to the character device. This is to allow the ident function to perform ioctls if required. The **clean* argument should be set to FALSE if the fsck(1m) program needs to be run on the file system before mounting, otherwise it should be set to TRUE. *Verbose* will be used in the future to request the ident function to print out "interesting" information about the file system. This is for information purposes only and does not have any direct relation to the process of identifying a file system.

eject(1)), the file system is unmounted and the media is ejected.

3.1. Connection to the Media Manager

```
# Events
insert /vol*/dev/fd[0-9]/* user=root /usr/sbin/diskovery -D
insert /vol*/dev/dsk/* user=root /usr/sbin/diskovery -D
eject /vol*/dev/fd[0-9]/* user=root /usr/sbin/diskovery -D
eject /vol*/dev/dsk/* user=root /usr/sbin/diskovery -D
```

Example 1: *excerpt from /etc/vold.conf*

The media manager is configured to use the diskovery program by the above lines in /etc/vold.conf. These specify that diskovery is to be called (in debug mode "-D") when media is inserted into the floppy or the CD-ROM, and when media is ejected from the floppy or CD-ROM.

3.2. File System Identification

Determination of the file system type is performed by a function that is kept in a dynamic shared object [Gin87]. This "ident" function decides if the type is right, and also lets the upper level know if it is "clean". The diskovery configuration file (Example 2) specifies which file systems are appropriate to which media, so that long searches can be avoided.

```
# File system identification
ident hsfs ident_hsfs.so cdrom
ident ufs ident_ufs.so cdrom floppy
ident pcfs ident_pcfs.so floppy
```

Example 2: *excerpt from /etc/diskovery.conf*

The shared object which identifies a file system is packaged along with other per-file system components. For example, ident_hsfs.so is really /usr/lib/fs/hsfs/ident_hsfs.so.1. The "hsfs" component of the name is derived from the argument on the line after the "ident" keyword. The ".1" part of the name specifies that the particular shared object implements version 1 of the interface. The ".1" is not specified in the configuration file, because diskovery knows to load only compatible versions of the shared object.

The last part of the line specifies what types of media this file system can be expected on. For example, there is little point in checking a floppy for a High Sierra file system. However, there might be a case where someone has placed an MS-DOS file system on a CD-ROM. If this were the case, the "ident pcfs ..." line would be edited to include "cdrom" on the end.

3.2.1. How to add a new file system type

A new file system type can be added with a minimum of programming effort. Example 3 shows the ident_fs function for UFS.[McK84]

CD-ROM's and Floppies in Solaris

Howard Alt

Sunsoft, Inc.

ABSTRACT

Since Sun has been supplying CD-ROM and floppy drives with our hardware, customers have been commenting on how difficult it is to use them. In particular, mounting a CD-ROM or floppy requires super-user privilege, as well as knowledge about what file system type is on the medium. This paper describes the mechanism that makes the user's life easier, and how users or third parties can extend its functionality.

1. Introduction

The UNIX interface to removable media (floppys, tapes, etc) has traditionally been a minimalist one. The user specifies the physical device in order to gain access to their media. There is no assurance that the expected media is in the device, and the only security is per-device. In addition, there is no general interface for applications to recognize the insertion of media.

Under UNIX, in order to gain access to a file system on a floppy, for example, a user must become root, know what file system type is on the floppy, and execute the mount command (e.g. mount -F pcfs /dev/fd0c /mnt). This is quite a lot for a user to know, especially the users that most work station companies are trying to attract these days.

In MS-DOS†, when a floppy is inserted it is instantly accessible as A: (or B:). The user is not required to know anything about mounting or file systems. The only way to reference a disk, however, is by drive name.

On the Macintosh††, users are not forced to deal with drive names, and they do not have to know about file system type. Macintosh users deal with media names, rather than devices.

The UNIX model makes it very difficult to layer applications that use the various forms of removable media. Users of removable media find UNIX systems very difficult to use.

2. Media Manager

We have chosen to take a generalized approach to solving the removable media problem, rather than one that is specific to CD-ROM and floppy. The media manager [Alt93] provides a layer of abstraction between a drive and the actual medium. Users (or applications) can refer to individual pieces of media without regard for what drive the reside in. In addition, new interfaces are introduced to allow applications to become aware of media being inserted or ejected. This media manager can be extended to deal with tapes (drives and autochangers) and magneto-optical disks in addition to CD-ROM's and floppies.

3. Diskovery

Diskovery is a program executed by the media manager in response to CD-ROM and floppy insertion events, and ejection requests. Diskovery looks at new media and attempts to divine what file system types are present, and to mount them in some well known location. When ejection is requested (via

†MS-DOS is a trademark of Microsoft Corp.

††Macintosh is a trademark of Apple Computer Corp.