```
/**
 * \file versfs.c
 * \date November 2020
 * \author Scott F. Kaplan <sfkaplan@amherst.edu>
 *
 * A user-level file system that maintains, within the storage directory, a
 * versioned history of each file in the mount point.
 *
 * FUSE: Filesystem in Userspace
 * Copyright (C) 2001-2007  Miklos Szeredi <miklos@szeredi.hu>
 * Copyright (C) 2011       Sebastian Pipping <sebastian@pipping.org>
 *
 * This program can be distributed under the terms of the GNU GPL.
 */

#define FUSE_USE_VERSION 26

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#ifdef linux
/* For pread()/pwrite()/utimensat() */
#define _XOPEN_SOURCE 700
#endif

#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <dirent.h>
#include <errno.h>
#include <sys/time.h>
#ifdef HAVE_SETXATTR
#include <sys/xattr.h>
#endif

static char* storage_dir = NULL;
static char  storage_path[256];


char* prepend_storage_dir (char* pre_path, const char* path) {
  strcpy(pre_path, storage_dir);
  strcat(pre_path, path);
  return pre_path;
}


static int vers_getattr(const char *path, struct stat *stbuf)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = lstat(path, stbuf);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_access(const char *path, int mask)
{
```

```
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = access(path, mask);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_readlink(const char *path, char *buf, size_t size)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = readlink(path, buf, size - 1);
        if (res == -1)
                return -errno;

        buf[res] = '\0';
        return 0;
}


static int vers_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                        off_t offset, struct fuse_file_info *fi)
{
        DIR *dp;
        struct dirent *de;

        (void) offset;
        (void) fi;

        path = prepend_storage_dir(storage_path, path);
        dp = opendir(path);
        if (dp == NULL)
                return -errno;

        while ((de = readdir(dp)) != NULL) {
                struct stat st;
                memset(&st, 0, sizeof(st));
                st.st_ino = de->d_ino;
                st.st_mode = de->d_type << 12;
                if (filler(buf, de->d_name, &st, 0))
                        break;
        }

        closedir(dp);
        return 0;
}

static int vers_mknod(const char *path, mode_t mode, dev_t rdev)
{
        int res;

        /* On Linux this could just be 'mknod(path, mode, rdev)' but this
           is more portable */
        path = prepend_storage_dir(storage_path, path);
        if (S_ISREG(mode)) {
                res = open(path, O_CREAT | O_EXCL | O_WRONLY, mode);
                if (res >= 0)
                        res = close(res);
        } else if (S_ISFIFO(mode))
                res = mkfifo(path, mode);
        else
```

```
                res = mknod(path, mode, rdev);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_mkdir(const char *path, mode_t mode)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = mkdir(path, mode);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_unlink(const char *path)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = unlink(path);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_rmdir(const char *path)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = rmdir(path);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_symlink(const char *from, const char *to)
{
        int res;
        char storage_from[256];
        char storage_to[256];

        prepend_storage_dir(storage_from, from);
        prepend_storage_dir(storage_to,   to  );
        res = symlink(storage_from, storage_to);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_rename(const char *from, const char *to)
{
        int res;
        char storage_from[256];
        char storage_to[256];

        prepend_storage_dir(storage_from, from);
```

```
        prepend_storage_dir(storage_to,   to  );
        res = rename(storage_from, storage_to);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_link(const char *from, const char *to)
{
        int res;
        char storage_from[256];
        char storage_to[256];

        prepend_storage_dir(storage_from, from);
        prepend_storage_dir(storage_to,   to  );
        res = link(storage_from, storage_to);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_chmod(const char *path, mode_t mode)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = chmod(path, mode);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_chown(const char *path, uid_t uid, gid_t gid)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = lchown(path, uid, gid);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_truncate(const char *path, off_t size)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = truncate(path, size);
        if (res == -1)
                return -errno;

        return 0;
}

#ifdef HAVE_UTIMENSAT
static int vers_utimens(const char *path, const struct timespec ts[2])
{
        int res;

        /* don't use utime/utimes since they follow symlinks */
```

```
        path = prepend_storage_dir(storage_path, path);
        res = utimensat(0, path, ts, AT_SYMLINK_NOFOLLOW);
        if (res == -1)
                return -errno;

        return 0;
}
#endif

static int vers_open(const char *path, struct fuse_file_info *fi)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = open(path, fi->flags);
        if (res == -1)
                return -errno;

        close(res);

        return 0;
}

static int vers_read(const char *path, char *buf, size_t size, off_t offset,
                     struct fuse_file_info *fi)
{
        int fd;
        int res;
        int i;
        char temp_buf[size];

        (void) fi;
        path = prepend_storage_dir(storage_path, path);
        fd = open(path, O_RDONLY);
        if (fd == -1)
                return -errno;

        res = pread(fd, temp_buf, size, offset);
        if (res == -1)
                res = -errno;

        // Move data from temporary buffer into provided one.
        for (i = 0; i < size; i += 1) {
          buf[i] = temp_buf[i];
        }

        close(fd);
        return res;
}

static int vers_write(const char *path, const char *buf, size_t size,
                      off_t offset, struct fuse_file_info *fi)
{
        int fd;
        int res;
        int i;
        char temp_buf[size];

        (void) fi;
        path = prepend_storage_dir(storage_path, path);
        fd = open(path, O_WRONLY);
        if (fd == -1)
                return -errno;

        for (i = 0; i < size; i += 1) {
```

```
          temp_buf[i] = buf[i];
        }

        res = pwrite(fd, temp_buf, size, offset);
        if (res == -1)
                res = -errno;

        close(fd);
        return res;
}

static int vers_statfs(const char *path, struct statvfs *stbuf)
{
        int res;

        path = prepend_storage_dir(storage_path, path);
        res = statvfs(path, stbuf);
        if (res == -1)
                return -errno;

        return 0;
}

static int vers_release(const char *path, struct fuse_file_info *fi)
{
        /* Just a stub.  This method is optional and can safely be left
           unimplemented */

        (void) path;
        (void) fi;
        return 0;
}

static int vers_fsync(const char *path, int isdatasync,
                      struct fuse_file_info *fi)
{
        /* Just a stub.  This method is optional and can safely be left
           unimplemented */

        (void) path;
        (void) isdatasync;
        (void) fi;
        return 0;
}

#ifdef HAVE_POSIX_FALLOCATE
static int vers_fallocate(const char *path, int mode,
                          off_t offset, off_t length, struct fuse_file_info *fi)
{
        int fd;
        int res;

        (void) fi;

        if (mode)
                return -EOPNOTSUPP;

        path = prepend_storage_dir(storage_path, path);
        fd = open(path, O_WRONLY);
        if (fd == -1)
                return -errno;

        res = -posix_fallocate(fd, offset, length);

        close(fd);
```

```
        return res;
}
#endif

#ifdef HAVE_SETXATTR
/* xattr operations are optional and can safely be left unimplemented */
static int vers_setxattr(const char *path, const char *name, const char *value,
                         size_t size, int flags)
{
        path = prepend_storage_dir(storage_path, path);
        int res = lsetxattr(path, name, value, size, flags);
        if (res == -1)
                return -errno;
        return 0;
}

static int vers_getxattr(const char *path, const char *name, char *value,
                         size_t size)
{
        path = prepend_storage_dir(storage_path, path);
        int res = lgetxattr(path, name, value, size);
        if (res == -1)
                return -errno;
        return res;
}

static int vers_listxattr(const char *path, char *list, size_t size)
{
        path = prepend_storage_dir(storage_path, path);
        int res = llistxattr(path, list, size);
        if (res == -1)
                return -errno;
        return res;
}

static int vers_removexattr(const char *path, const char *name)
{
        path = prepend_storage_dir(storage_path, path);
        int res = lremovexattr(path, name);
        if (res == -1)
                return -errno;
        return 0;
}
#endif /* HAVE_SETXATTR */

static struct fuse_operations vers_oper = {
        .getattr        = vers_getattr,
        .access         = vers_access,
        .readlink       = vers_readlink,
        .readdir        = vers_readdir,
        .mknod          = vers_mknod,
        .mkdir          = vers_mkdir,
        .symlink        = vers_symlink,
        .unlink         = vers_unlink,
        .rmdir          = vers_rmdir,
        .rename         = vers_rename,
        .link           = vers_link,
        .chmod          = vers_chmod,
        .chown          = vers_chown,
        .truncate       = vers_truncate,
#ifdef HAVE_UTIMENSAT
        .utimens        = vers_utimens,
#endif
        .open           = vers_open,
        .read           = vers_read,
```

```
        .write          = vers_write,
        .statfs         = vers_statfs,
        .release        = vers_release,
        .fsync          = vers_fsync,
#ifdef HAVE_POSIX_FALLOCATE
        .fallocate      = vers_fallocate,
#endif
#ifdef HAVE_SETXATTR
        .setxattr       = vers_setxattr,
        .getxattr       = vers_getxattr,
        .listxattr      = vers_listxattr,
        .removexattr    = vers_removexattr,
#endif
};

int main(int argc, char *argv[])
{
        umask(0);
        if (argc < 3) {
          fprintf(stderr, "USAGE: %s <storage directory> <mount point> [ -d | -f | -s ]\n",
argv[0]);
          return 1;
        }
        storage_dir = argv[1];
        char* mount_dir = argv[2];
        if (storage_dir[0] != '/' || mount_dir[0] != '/') {
          fprintf(stderr, "ERROR: Directories must be absolute paths\n");
          return 1;
        }
        fprintf(stderr, "DEBUG: Mounting %s at %s\n", storage_dir, argv[2]);
        int short_argc = argc - 1;
        char* short_argv[short_argc];
        short_argv[0] = argv[0];
        for (int i = 2; i < argc; i += 1) {
          short_argv[i - 1] = argv[i];
        }
        return fuse_main(short_argc, short_argv, &vers_oper, NULL);
}
```