CSE134 Assignment 4 Design Doc

Group ABAJ – Alexander Sung, Benson Ho, Aidan Sojourner, James Quiaoit

June 3, 2021

1 Introduction

This is the design document for Assignment 4 (`ddfs`) for CSE 134, Spring 2021, at the University of California, Santa Cruz.

1.1 Goals

`ddfs` is a deduplicating in-kernel file system for FreeBSD that operates very similarly to Berkeley FFS via leverage of FreeBSD UFS and FFS source code.

- "Deduplication means that there's exactly one copy of any given data block in the file system."
- Each file in `ddfs` is uniquely represented as a 160-bit numeric key, shown to the user as 40 hexadecimal characters.
- Each "key" is associated with a single 4KiB "value", which is initialized to all 0s when the key-value pair is first created.
- Keys can be created, retrieved, renamed, updated, and removed using standard UNIX file operations.
- `ddfs` block pointers are 20 bytes (160 bits) instead of 8 bytes, and limiting the total number of block pointers to 6. (3 direct, 1 single indirect, 1 double indirect, 1 triple indirect). A `ddfs` file is limited to a maximum of 8531487 blocks, approximately 32 GiB per file.

2 Design

2.1 Disk Layout

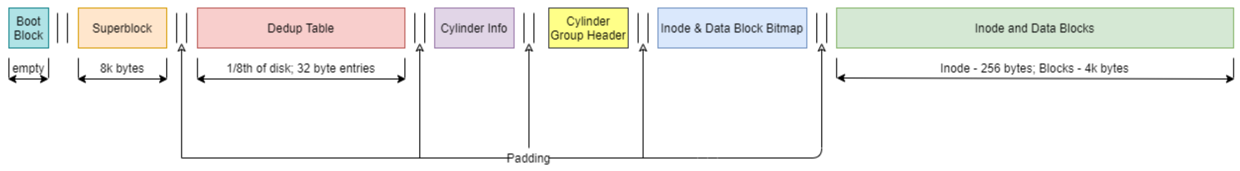

Figure 1: `ddfs` Disk Layout

The filesystem "superblock" is allocated at the very beginning of the disk. It contains metadata about the filesystem, and is described in detail in the Superblock section.

Each file in `ddfs` is represented by a pair of (inode, block), where the inode coontains metadata about the file, and the block contains the contents.

The contents of each inode are described in the Inode section.

2.1.1 Superblock

The `ddfs` superblock on-disk contains the following entries:

2.1.2 Deduplication Table

2.1.3 Cylinder Information and Group Header

2.1.4 Inode

The ddfs inode represents the state of each file on disk.

```
struct inode {
    TAILQ_ENTRY(inode) i_nextsnap; /* snapshot file list. */
    struct  vnode  *i_vnode;/* Vnode associated with this inode. */
    struct  ufsmount *i_ump;/* Ufsmount point associated with this inode. */
    struct   dquot *i_dquot[MAXQUOTAS]; /* Dquot structures. */
    union {
        struct dirhash *dirhash; /* Hashing for large directories. */
        daddr_t *snapblklist;    /* Collect expunged snapshot blocks. */
    } i_un;
    /*
     * The real copy of the on-disk inode.
     */
    union {
        struct ufs1_dinode *din1;   /* UFS1 on-disk dinode. */
        struct ufs2_dinode *din2;   /* UFS2 on-disk dinode. */
    } dinode_u;

    ino_t     i_number; /* The identity of the inode. */
    u_int32_t i_flag;   /* flags, see below */
    int    i_effnlink;   /* i_nlink when I/O completes */

    /*
     * Side effects; used during directory lookup.
     */
    int32_t   i_count;  /* Size of free slot in directory. */
    doff_t    i_endoff; /* End of useful stuff in directory. */
    doff_t    i_diroff; /* Offset in dir, where we found last entry. */
    doff_t    i_offset; /* Offset of free space in directory. */
#ifdef DIAGNOSTIC
    int         i_lock_gen;
    struct iown_tracker i_count_tracker;
    struct iown_tracker i_endoff_tracker;
    struct iown_tracker i_offset_tracker;
#endif

    int i_nextclustercg; /* last cg searched for cluster */

    /*
     * Data for extended attribute modification.
     */
    u_char    *i_ea_area;   /* Pointer to malloced copy of EA area */
    unsigned  i_ea_len; /* Length of i_ea_area */
    int    i_ea_error;   /* First errno in transaction */
    int    i_ea_refs;    /* Number of users of EA area */

    /*
     * Copies from the on-disk dinode itself.
     */
    u_int64_t i_size;   /* File byte count. */
    u_int64_t i_gen;    /* Generation number. */
    u_int32_t i_flags;  /* Status flags (chflags). */
    u_int32_t i_uid;    /* File owner. */
    u_int32_t i_gid;    /* File group. */
    u_int16_t i_mode;   /* IFMT, permissions; see below. */
    int16_t   i_nlink;  /* File link count. */
};
```

Each `inode` is 256 bytes in size, meaning 16 of them can fit in one 4KiB block.

2.1.5 In-memory

2.2 Additional Data Structures and Algorithms

2.2.1 Mount Structures

2.3 Initializing the Filesystem – `mkfs`

3 Testing

Basic functionality was tested with user-space tools like `cat`, `touch`, `rm`, `mv`, `stat`, and `ls`.

4 Limitations and Known Issues

References

The following files in the FreeBSD source tree were consulted. Unless otherwise specified in the source, code from these files was not copied. Instead, the files were used as a reference and studied to understand the various VFS and vnode operations.

Additionally, the following web resources were used:

The filesystem layout diagram was created using https://diagrams.net.