Layered Block-Structured File System

Robbert van Renesse

Intro

- Underneath any file system, database system, etc. there are one or more *block stores*
- A block store provides a disk-like interface:
  - a storage object is a sequence of blocks
    - typically, a few kilobytes
  - you can read or write a block at a time
- The block store abstraction doesn't deal with file naming, security, etc., just storage

Block Store Abstraction

- A block store consists of a collection of *i-nodes*
- Each i-node is a finite sequence of *blocks*
- Simple interface (struct bd):
  - `struct block`: block of size BLOCK_SIZE
  - `alloc()` → inode identifier
  - `size(inode)` → maximum size of inode
  - `read(inode, block offset)` → block
  - `write(inode, block offset, block)`
  - `free(inode)`

Example block stores

- "ramdisk": a simulated disk in memory
  - `void ramdisk_init(`
    `struct bd *iface,`
    `struct ramdisk_state *state,`
    `void *mem,`
    `int nblocks);`

Example code

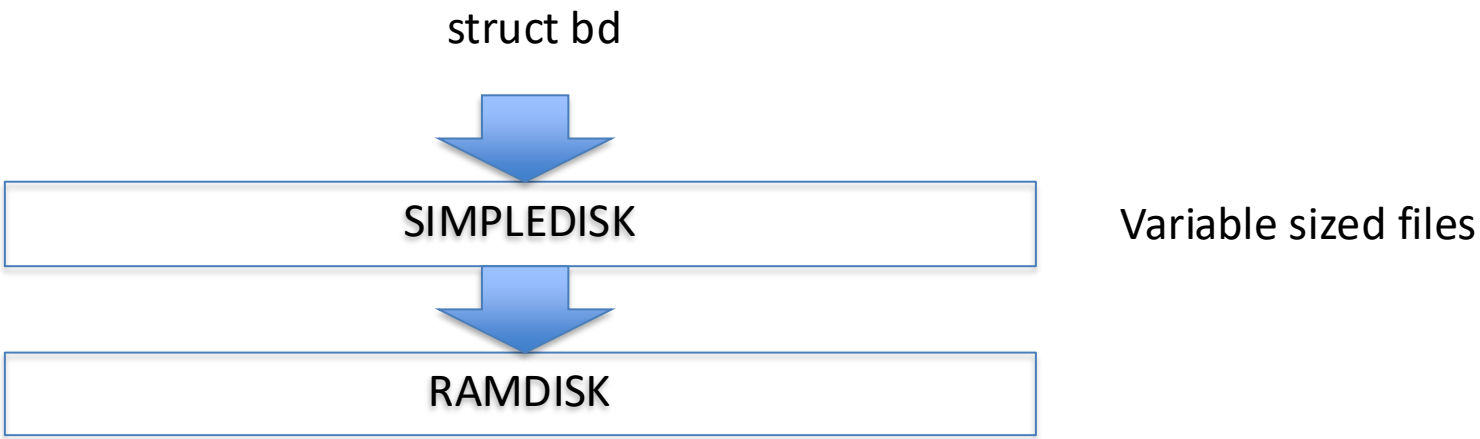
```
extern struct block ramdisk[], __ramdisk_end[];
struct bd ramdisk_iface;
struct ramdisk_state ramdisk_state;

void files_init(void) {
    bd_init();  // initialize block device module

    // Initialize the RAM disk block device layer
    ramdisk_init(&ramdisk_iface, &ramdisk_state,
                 ramdisk, __ramdisk_end - ramdisk);
     int ino = ramdisk_iface.alloc(&ramdisk_state);
    int size = ramdisk_iface.size(&ramdisk_state, ino);
}
```

Block Stores can be Layered!

Each layer presents a struct bd abstraction



Example code with layers

```
extern struct block ramdisk[], __ramdisk_end[];

struct bd ramdisk_iface;
struct ramdisk_state ramdisk_state;
struct bd simple_iface;
struct simple_state simple_state;

void simple_init(struct bd *iface, struct simple_state *s,
                 struct bd *lower, int inode_below, int format);

void files_init(void) {
    bd_init();  // initialize block device module

    // Initialize the RAM disk block device layer
    ramdisk_init(&ramdisk_iface, &ramdisk_state,
                 ramdisk, __ramdisk_end - ramdisk);

    // Add the "simple" block device layer
    simple_init(&simple_iface, &simple_state,
                            &ramdisk_iface, 0, 1);
}
```

P4: a “Unix-Like” File System

- Tree-based, multi-level index

UFS Superblock

Identifies file system's key parameters:

- inode array location and size
- location of free list