JOS-Lab 5: File System

Disk Access

Exercise 1
Modify env_create() to give I/O privileges to the ring3 Env which type is ENV_TYPE_FS by setting the IOPL to 3 in the eflags.

```
    if (type == ENV_TYPE_FS) {
      e->env_tf.tf_eflags |= FL_IOPL_3;
    }
```

The Block Cache

Exercise 2
In bc_pgfault(), for a page fault within the block cache region, it first allocate a page for that page:

```
  void *pgaddr = ROUNDDOWN(addr, PGSIZE);
  r = sys_page_alloc(0, pgaddr, PTE_W|PTE_U|PTE_P);
  if (r < 0)
    panic("sys_page_alloc: %e", r);
```

Then load the contents of the block from the disk into that page:

```
  r = ide_read(blockno * BLKSECTS, pgaddr, BLKSECTS);
  if (r < 0)
    panic("ide_read failed");
```

Finally, it remaps the page to clear the dirty bit:

```
  r = sys_page_map(0, pgaddr, 0, pgaddr, PTE_W|PTE_U|PTE_P);
  if (r < 0)
    panic("sys_page_map: %e", r);
```

The check that the block was allocated must be done after reading the block in, otherwise if the block is just the block that contains the free block bitmap, the check will cause a page fault again and finally leads to a triple fault.

In flush_block(), if the page is mapped and dirty, it writes a block out to disk using ide_write() and remaps the page to clear the dirty bit.

```
 if (va_is_mapped(addr) && va_is_dirty(addr)) {
    void *pgaddr = ROUNDDOWN(addr, PGSIZE);
    r = ide_write(blockno * BLKSECTS, pgaddr, BLKSECTS);
    if (r < 0)
```

```
    panic("ide_write failed");
  r = sys_page_map(0, pgaddr, 0, pgaddr, PTE_W|PTE_U|PTE_P);
  if (r < 0)
    panic("sys_page_map: %e", r);
}
```

The Block Bitmap

Exercise 3
In alloc_block(), search the bitmap for a free block and mark it as in use. Immediately flush the changed bitmap block to disk.

```
int i;
 for (i = 0; i < super->s_nblocks; ++i) {
  if (block_is_free(i)) {
    bitmap[i/32] &= ~(1<<(i%32));
    flush_block(bitmap + i/32);
    return i;
  }
 }
return -E_NO_DISK;
```

File Operations

Exercise 4
In file_block_walk(), first check if filebno is out of range. Then if filebno is direct, the answer can be found in the f_direct array. Otherwise, we may need to allocate and initialize f_indirect block and find the answer there.

In file_get_block(), find the disk block number by file_block_walk(), allocate and initialize a block if it doesn't yet exist. Finally, set *blk to the address in memory where the filebno'th block of file 'f' would be mapped.

Client/Server File System Access

Exercise 5
In serve_read(), first look up the file id using openfile_lookup(). Then read the file content using file_read(). Finally, update the seek position.

In devfile_read(), first fill the fsipcbuf with the request arguments. Then make a request using fsipc(). Finally, move the result in fsipcbuf to the buffer.

Exercise 6
In serve_write(), first look up the file id using openfile_lookup(). Then write the content using file_write(). Finally, update the seek position.

In devfile_write(), first fill the fsipcbuf with the request arguments. Then make a request using fsipc().

Client-Side File Operations

Exercise 7
In open(), first find an unused file descriptor page using fd_alloc(). Then fill the fsipcbuf with the request arguments, send the FSREQ_OPEN request and map the returned file descriptor page at the appropriate fd address with fsipc(). Finally, return the file descriptor index using fd2num().

Spawning Processes

Exercise 8
In sys_env_set_trapframe(), first check that the tf pointer points to a good address. Then find the Env structure with envid2env() and copy the trap frame pointed by tf. Finally, modified the tf to make sure that user environments always run at CPL 3 with interrupts enabled.

For challenge, see the answers-lab5.txt.