Section 8: Lab 4C & Lab 5

CSE 451 18SP

Announcements

- 4 Additional Late days for Labs 1-5
  - This gives you a total of 8 late days for the labs
- Lab 4C deadline is May 22nd
- Lab X proposal Due May 18th
  - That's tomorrow!
- Lab 5 deadline is May 29th
  - No checkpoints this time, just the whole lab
- Lab X deadline is June 4th
  - Lab X demos are June 1st. You can show it off if you get it done by then!
  - Cannot use late days on Lab X

Hardware interrupts

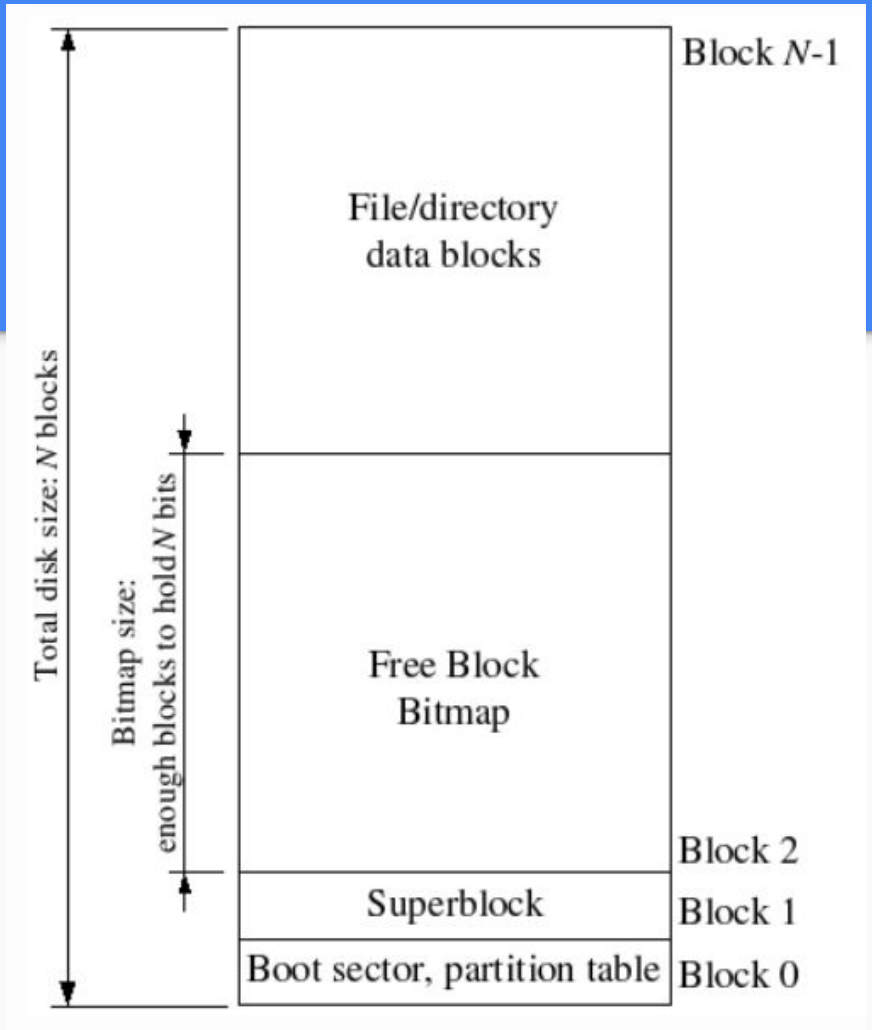
- Clock interrupts allow you to protect the system against bugs and malicious code in a user environment
- External Interrupts (ie, device interrupts) are referred to as IRQ's. There are 16 standard IRQ's that are used in JOS, numbered 0-15
  - IRQ's are mapped to the IDT as IRQ_OFFSET + IRQ#
  - You will need to set these up in kern/trapentry.S and kern/trap.c, very similar to what you have done in Lab 3
  - Also need to make sure that user environments run with interrupts enabled (wouldn't really be much point to adding them if they didn't do anything)
- Using IRQ's you can set up preemption and keeping track of time with clock interrupts

IPC (Inter-Process Communication)

- Up to this point, processes (environments) have been completely isolated from each other when it comes to memory and data. IPC allows processes to communicate and share data.
- Implemented using System calls, ipc_send and ipc_recieve
  - Allows a process to share a 32 bit value, and a page mapping. This is good way to transfer more data than 32 bits, an allows processes to set up shared memory

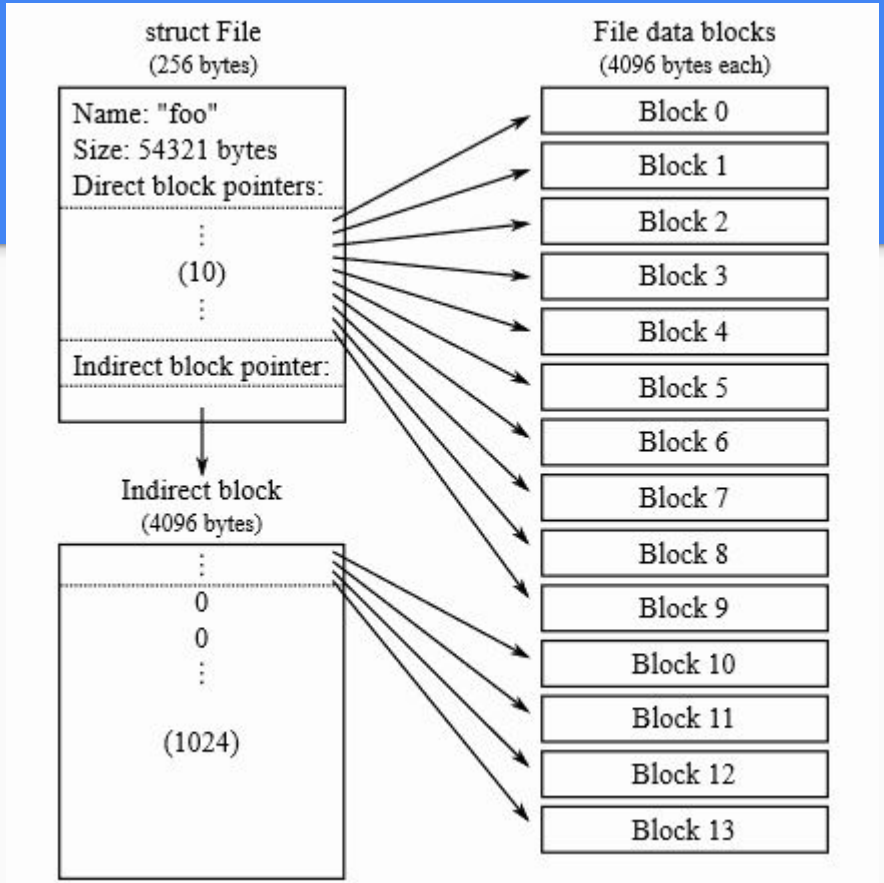
JOS File System

- In lecture yesterday we talked about the xv6 file system. The JOS file system has a lot of similarities, it is laid out like this:
  - Boot Block
    - The first block, contains the boot sector and partition table
  - Superblock
    - The second block, contains metadata about the disk layout
  - Bitmap
    - A number of blocks used to determine which disk blocks are allocated
  - File/Directory Data Blocks
    - Contains the file/directory metadata (called inodes in most cases)
    - Disk blocks containing the contents of files


Block N-1
File/directory
data blocks
Free Block
Bitmap
Block 2
Superblock
Block 1
Boot sector, partition table
Block 0
Total disk size: N blocks
Bitmap size:
enough blocks to hold N bits

File Struct

- f_name
  - The name of the file
- f_size
  - The size of the file
- f_type
  - The type of the File struct. It is either a file or a directory. If it is a directory, then that means that the data it points to is other File structs
- f_direct
  - An array of references to file data blocks (stored on disk)
- f_indirect
  - For larger files, this is a reference to a blocks that has more pointers to data blocks


struct File
(256 bytes)
Name: "foo"
Size: 54321 bytes
Direct block pointers:
(10)
Indirect block pointer:
Indirect block
(4096 bytes)
0
0
(1024)
File data blocks
(4096 bytes each)
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9
Block 10
Block 11
Block 12
Block 13

Disk Access

- It is nice you are working on a simulated OS, because you could potentially corrupt the disk when working on parts of this lab.
- If you accidentally corrupt the simulated disk and the OS breaks, you can reset it back to normal with:
  - $ rm obj/kern/kernel.img obj/fs/fs.img
    $ make
  - Or
  - $ make clean
    $ make

Implementing the File System

- You won't be writing the whole file system yourself from scratch, so make sure you read through all the new Lab 5 code carefully
  - inc/fs.c will be your friend
- Will need to read blocks into the block cache and flush them back to disk
  - The block cache is a simple buffer that stores disk blocks you are using, to reduce the number of times we have to go to and from disk (in general a slow operation)
  - fs/bc.c
- Will need to allocating disk blocks; mapping file offsets to disk blocks; and implementing read, write, and open in the IPC interface.
- When all is said and done, you can run a shell!