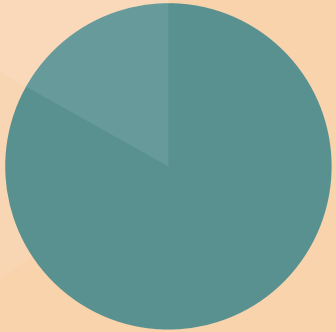
Systems Programming Presentation

By: Gino Coppola, Greg Leitkowski, and Ezequiel Salas



File System Introduction

- Simplified version of Unix V7
- Uses Inodes to keep track of metadata for both files and directories
- All Inodes and data blocks are kept track of by the file system
- File system has a linked list of memory for the data blocks and Inodes
- The upper gig of memory is blocked out so the linked lists can be allocated there without interruption from the kernel

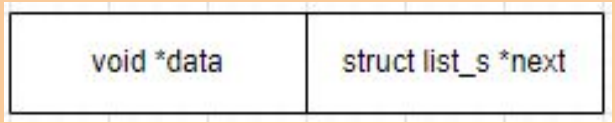
Structure of an Inode

- The Inode contains the size of the data, the number of pointers that are active, if the first pointer is direct, and the list of pointers
- Only the first pointer in the array of pointers can be direct. It can be indirect
- The rest of the pointers must be indirect

Structure of a File

- Each file contains a 16 byte array to represent the name, the index into the inode, and the 512 byte long data block

Structure of the Memory Linked List



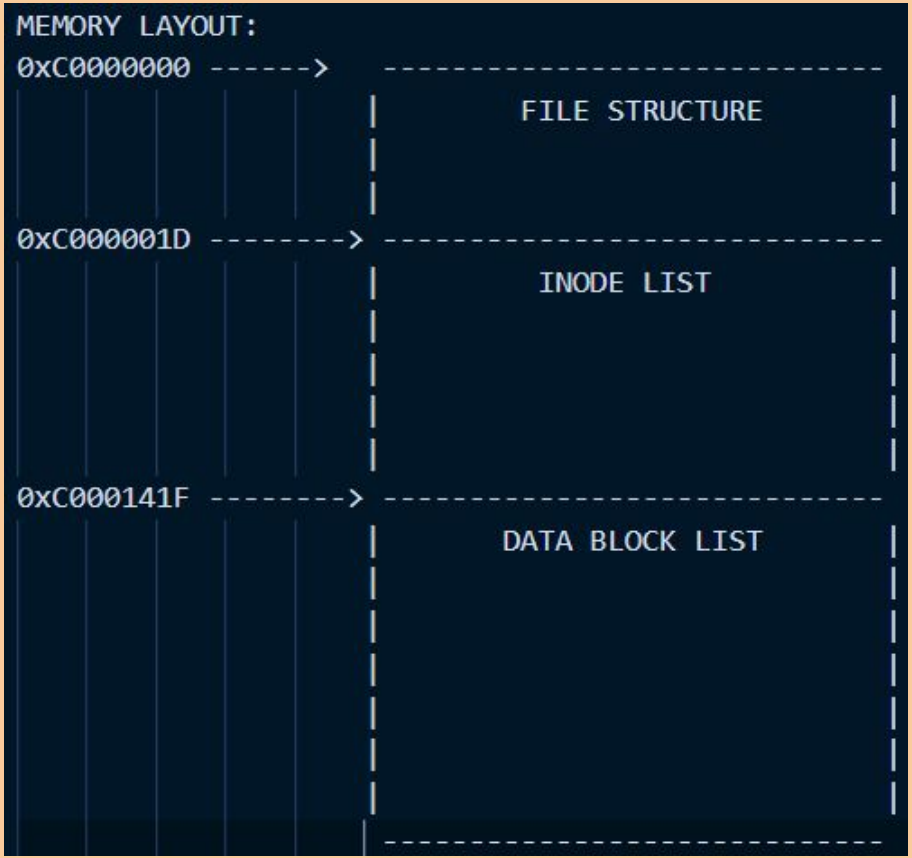
- The data pointer is used to pointer to either an Inode or a file
- The file system init process will allocate all of them before they are needed
- Each block will be allocated based on the predetermined size. This way, we don’t have to use malloc or the kernel allocator for our files

Structure of the File System

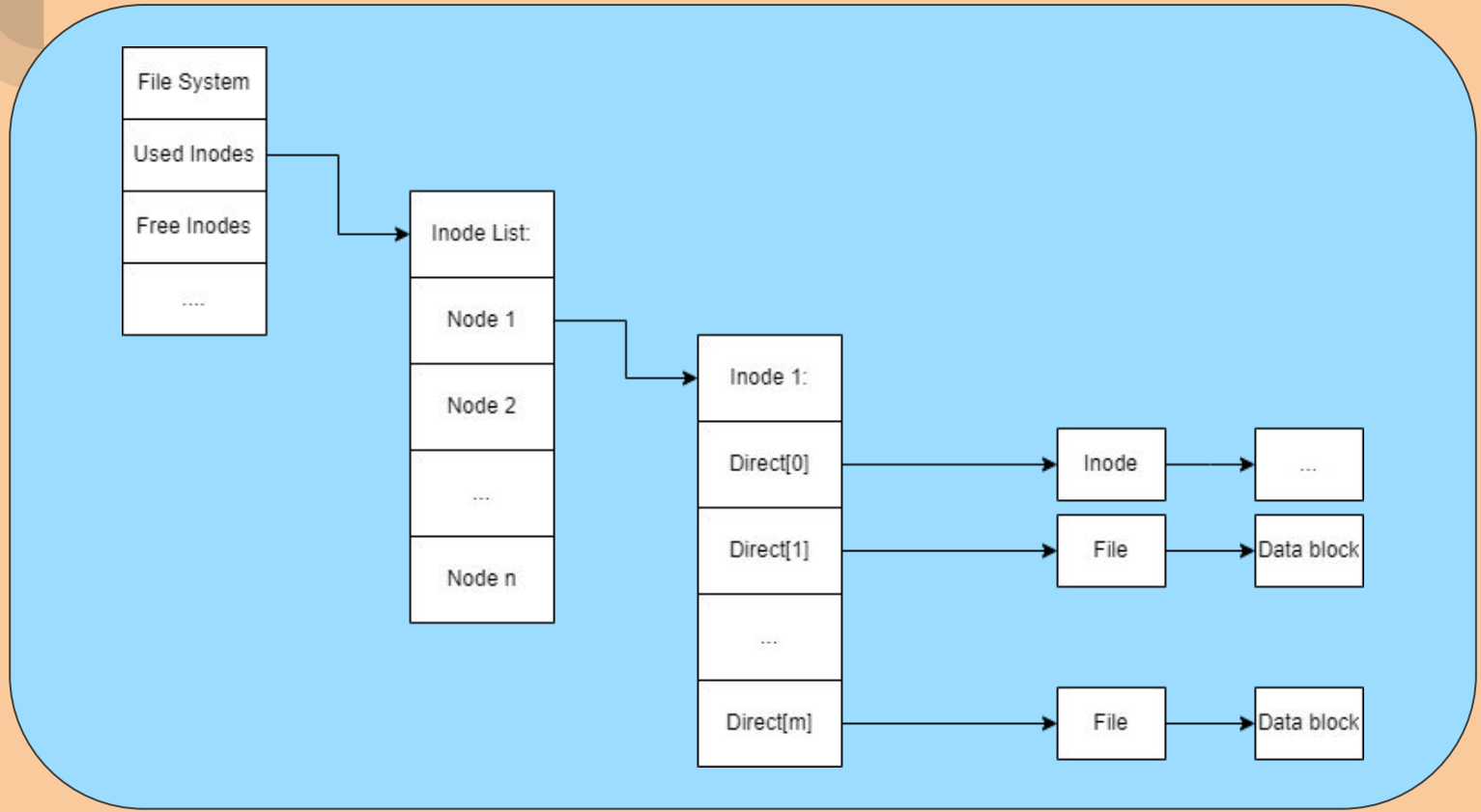
uint32_t num_free_pointers	uint32_t num_free_blocks	list_s *free_nodes	list_s *used_nodes	list_s *free_blocks	list_s *used_blocks	Inode_s *current_inode	Inode_s *previous_inode

- Contains the linked lists for the free/used data blocks and the free/used inodes
- Also contains the current number of free blocks/nodes and the current inode for the user and the previous inode

Memory Layout



How It Works


File System
Used Inodes
Free Inodes
....
Inode List:
Node 1
Node 2
...
Node n
Inode 1:
Direct[0]
Direct[1]
...
Direct[m]
Inode
...
File
Data block
File
Data block

What Can The File System Do?

- Create directories and files
- Read and write to files
- Add files into directories
- Delete the inodes and files
- Overwrites files so they are only spaces
- Move into a directory (but not out as of right now)
- Print information about the current directory

What Went Well?

- I learned a lot about the Unix V7 file system and how it operates
- I learned more in detail about coding with structures in C
- I learned more about memory mapping and memory layouts using non-provided C allocation functions

Problems Faced During My Portion

- Finding time to do this project outside of class
- Figuring out how my project can work with other group members
- Finding detailed and consistent documentation
- TESTING
- Deciding the overall structure and design of my system

What Would I Change?

- Find a way to separate directories from Inodes since Inodes should only contain metadata about a structure, not the data itself
- Add functionality in the kernel process, so it can have different operations queued, and not be the only application running
- Change how I would have approached the beginning of the project

AHCI Driver Introduction

- AHCI (Advanced Host Control Interface) is memory interface between system memory and SATA devices
- Implementing driver to handle communication with SATA hard drive
- Planned integration with file system through reading and writing files onto the hard drive

Find AHCI Controller

- Enumerate PCI Bus
- AHCI Controller in systems is manufactured by Intel
    - vendor 0x8086 and device 0xA282
- AHCI Controllers are class 0x01 (Mass Storage Controller) and subclass 0x06 (Serial ATA Controller)
- Iterate through each bus and slot of PCI configuration space until device with those values is found.

Register	Offset	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x0	0x0	Device ID		Vendor ID	
0x1	0x4	Status		Command	
0x2	0x8	Class code	Subclass	Prog IF	Revision ID
0x3	0xC	BIST	Header type	Latency Timer	Cache Line Size
0x4	0x10	Base address #0 (BAR0)			
0x5	0x14	Base address #1 (BAR1)			
0x6	0x18	Base address #2 (BAR2)			
0x7	0x1C	Base address #3 (BAR3)			
0x8	0x20	Base address #4 (BAR4)			
0x9	0x24	Base address #5 (BAR5)			
0xA	0x28	Cardbus CIS Pointer			
0xB	0x2C	Subsystem ID		Subsystem Vendor ID	
0xC	0x30	Expansion ROM base address			
0xD	0x34	Reserved			Capabilities Pointer
0xE	0x38	Reserved			
0xF	0x3C	Max latency	Min Grant	Interrupt PIN	Interrupt Line

Find SATA Devices

- Devices can be found through HBA (Host Bus Adapter) registers
- Location of HBA registers is given by PCI BAR5 register, also known as AHCI Base Memory Register (ABAR)
- HBA registers have a ports implemented field
- Check each implemented port for a device
- If a port has a device connected, read signature to see if it’s SATA

Identify Correct Device

- Need to send IDENTIFY_DEVICE command to each device found
- Commands are sent and results are received through Frame Information Structure (FIS) in memory
- Give the current port the addresses of different FIS data structures
    - Host to Device
    - Device to Host
    - Command Table
- Input command information into command table
- Send command by setting command bit
- Receive data in Physical Region Descriptor Table (PRDT) list of Command Table
- Verify that data received matches expected data of Western Digital HDD rather than Biwin SSD

Figure 4: HBA Memory Space Usage

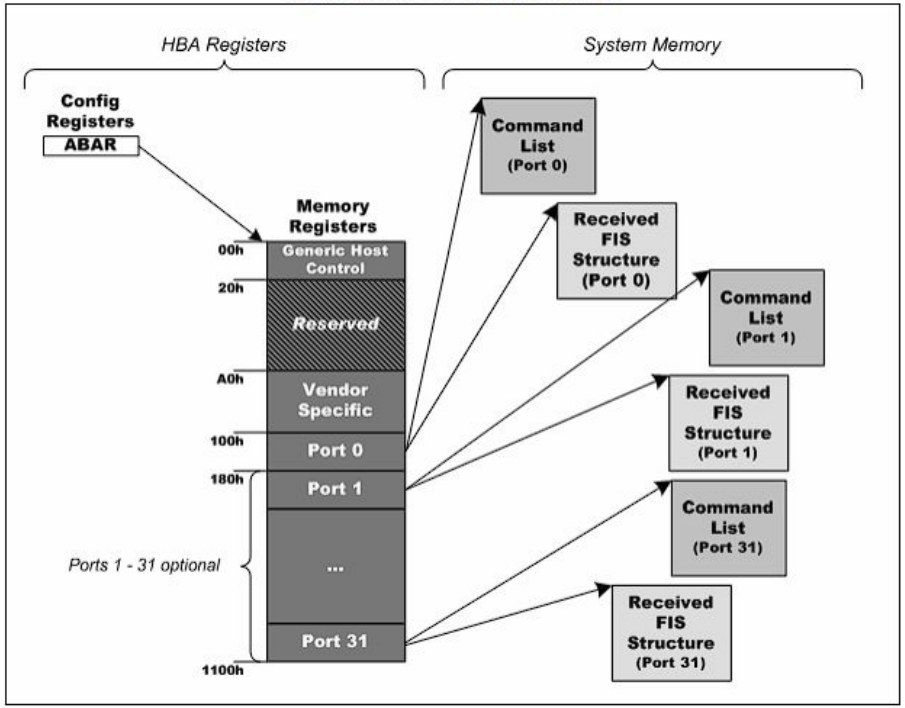
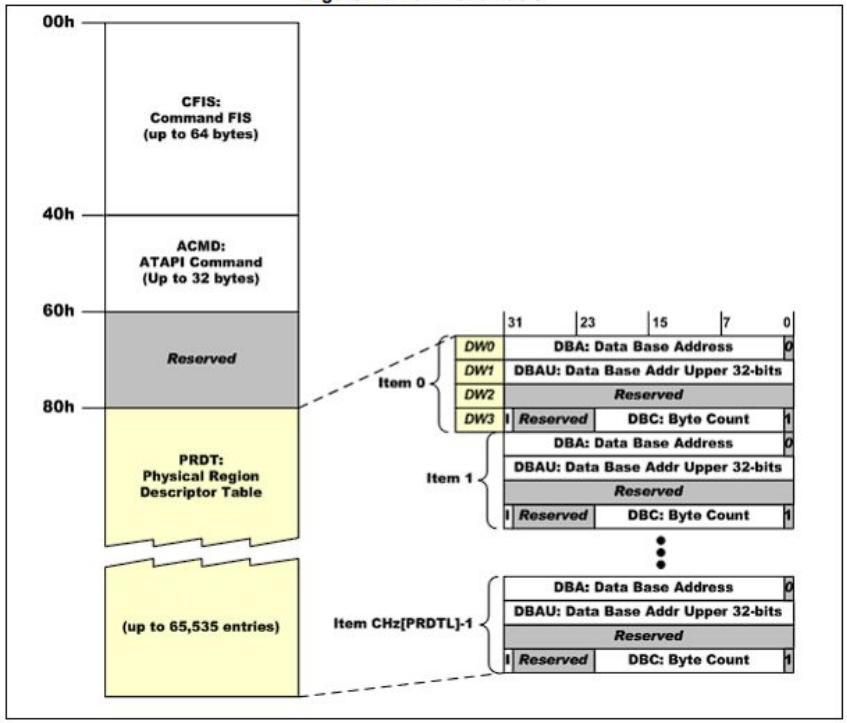


Figure 12: Command Table



What Went Well

- Learned a lot about how device drivers work and are implemented
- More practice and understanding of C

Problems Faced

- Failing to properly initialize various registers so commands can be sent/received
- Fear of hurting the boot drive
- Being able to internalize documentation

What Would I Change?

- Make a more specific plan of steps I needed to take
- Follow better coding conventions so debugging is easier

Parallel Port/Dot Matrix Printer

- General driver meant to receive and send data via parallel port.
  - Done via a PCI expansion card
- Communicate specifically to a dot matrix printer
- Done through simple port communication to both send and receive bytes

Process

- Order a parallel port expansion card
- Identify the ports needed to talk through
  - Base off of linux addresses
  - Scan through PCI addresses
- Research status codes about the KX-P1180 Dot Matrix Printer
- Establish a “handshake” and begin communicating

Parallel Port/PCI Card

- Like many others I had to go through PCI cards
- Parallel port does set different rules to communicate with
  - Any data needs to be sent in complete bytes
  - Once all data is sent, an additional byte needs to be sent to have the printer read in everything

Dot Matrix Printer

- Establish a “handshake” and check on different possible status codes
  - First send a PRIME signal
  - BUSY, ERROR, PAPER OUT, OFF LINE
- If all good, wait for an ACK signal and start sending data
- Once all data is sent, send a STB signal to make the printer read in

	BUSY	SLCT	PO	$\overline{\text{ERROR}}$
ON LINE	LOW	HIGH	LOW	HIGH
OFF LINE	HIGH	LOW	LOW	LOW
PAPER OUT	HIGH	LOW	HIGH	LOW

Signal pin	Return side pin	Signal	Directin
1	19	STB	Input
2	20	DATA 1	Input
3	21	DATA 2	
4	22	DATA 3	
5	23	DATA 4	
6	24	DATA 5	
7	25	DATA 6	
8	26	DATA 7	
9	27	DATA 8	
10	28	ACK	Output
11	29	BUSY	Output
12		PO	Output
13		SLCT	Output
14		AUTO FEED XT	Input
15			
16		SG	
17		FG	
18		+5 V	Output
31	30	PRIME	Input
32		ERROR	Output
33		SG	
34			
35			
36			

Pin Configuration

What worked

- Accessing the PCI card
- Parallel port interfacing
- Receiving a status code from the printer

What didn’t work :(

- Sending data
- Changing printer states

What problems did I face

- Port not being on the motherboard
- Time management with every other project I have
- Further web exploration instead of sticking to the same sources
- Obtain paper or try any paper in it

What would I change

- Separate the functionality
  - Proper parallel port system driver
  - Standalone printer program
- Conforming to our established coding standards

Questions