6.828 2011 Lecture 9: File System

topic: file systems
  what they are
  how the xv6 file system works
  intro to larger topics

FS goals?
  durable storage
  multiplexing
  sharing
  organization

why are file systems interesting?
  they are a critical (and fruitful) piece of many stories
  performance
  security
  fault tolerance
  O/S API design
  user interfaces
  and you will implement one for JOS

high-level choices made by UNIX (and xv6)
  granularity: files (vs virtual disk, DB)
  content: byte array (vs 80-byte records, BTree)
  naming: human-readable (vs object IDs)
  organization: name hierarchy
  synchronization: none (vs locking, versions)

what's the API?
  fd = open("x/y", O_CREATE);
  write(fd, "abc", 3);
  link("x/y", "x/z");
  unlink("x/y");

FS abstraction turns out to be useful
  pipes
  devices /dev/console
  Linux /proc
  /afs
  Plan 9
  point: apps don't have to know about each of these objects separately
    write app to general FS/FD interface, works for many underlying things

a few implications of the API:
  fd refers to something
    that is preserved even if file name changes
    or if file is deleted while open!
  a file can have multiple links
    i.e. occur in multiple directories
    no one of those occurences is special
    so cannot store info about file in the directory
  thus:

a file exists independent of its names
called an "inode"
inode must have link count (tells us when to free)
inode must have count of open FDs
inode deallocation deferred until last link and FD are gone

let's talk about xv6

FS software layers
system calls
name ops | file descriptors
inode ops
inode cache (really just active inodes)
transactions
buffer cache
disk driver

on-disk layout
we will view disk as linear array of 512-byte sectors
though, when thinking about performance, remember concentric tracks
0: unused
1: super block (size, ninodes)
2: array of inodes, packed into blocks
X: block in-used bitmap (0=free, 1=inuse)
Y: file/dir content blocks
Z: log for transactions
end of disk

on-disk inode
type (free, file, directory, device)
nlink
size
addrs[12+1]

direct and indirect blocks

each i-node has an i-number
easy to turn i-number into inode
location on disk: sector 2 + 64*inum

directory contents
directory much like a file
but user can't directly write
content is array of dirents
dirent:
inum
14-byte file name
dirent is free if inum is zero

you should view FS as an on-disk data structure
[tree: dirs, inodes, blocks]
two allocation pools: inodes and blocks

let's look at xv6 in action

focus on disk writes, as in homework
illustrate on-disk data structures via how updated

[hand out the following]

Q: how does xv6 create a file?

$ echo > a
log_write 4 ialloc (44, from create 54)
log_write 4 iupdate (44, from create 54)
log_write 29 writei (47, from dirlink 48, from create 54)
log_write 2 iupdate

Q: what is writei writing?

Q: what is the 2nd iupdate about?

Q: what if there are concurrent calls to ialloc?
will they get the same inode?
note bread / bwrite / brelse in ialloc
bread sheet 39
diagram of block cache
focus on B_BUSY and sleep()
Q: why goto loop?

Q: how does xv6 write data to a file?

$ echo x > a
log_write 28 balloc (43, from bmap 46, from writei 47)
log_write 417 bzero
log_write 417 writei
log_write 4 iupdate
log_write 417 writei
log_write 4 iupdate

Q: why the iupdate?

Q: why *two* iupdates?

Q: how does xv6 delete a file?

$ rm a
log_write 29 writei
log_write 4 iupdate
log_write 28 bfree
log_write 4 iupdate
log_write 4 iupdate

Q: what's in block 29?

Q: what are the iupdates doing?

Q: what is the block cache replacement policy?
bget and brelse, sheet 39

Q: is that the best replacement policy?

Q: when does xv6 write user data to disk?
  writei 47, bwrite 39, iderw 38 (which sleeps)

Q: is that a good write policy?
  performance?
  correctness? what's the danger?

Q: when does xv6 write meta-data to disk?
  same policy as user data
  is that a good meta-data write policy?
    performance
    correctness (order)
  tune in next lecture...

Q: what if lots of processes need to read the disk? who goes first?
  iderw 38 appends to idequeue
  idestart looks at head of list
  ideintr pops head, starts next
  so FIFO

Q: is FIFO a good disk scheduling policy?
  priority to interactive programs?
  elevator sort?

Q: how fast can an xv6 application read big files?
  contiguous blocks?
  blow a rotation -- no prefetch?

Q: why does it make sense to have a double copy of I/O?
  disk to buffer cache
  buffer cache to user space
  can we fix it to get better performance?

Q: how much RAM should we dedicate to disk buffers?

6.828 Operating System Engineering

Fall 2012