ECE 531 – Advanced Operating Systems
Lecture 33

Vince Weaver

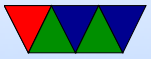
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 November 2025

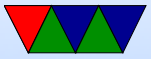
Announcements

- Project topics were due
- HW#8 will be posted
- Note there will be a second midterm, in-class, on Friday December 5th

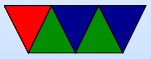


Brief Midterm Notes

- Next Friday (December 5th)
- Not cumulative, will be mostly topics since last midterm
- Virtual Memory
- Filesystems
- Graphics
- Multicore / Locking

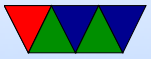


Project Update

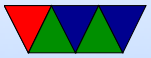


Linux VFS

- VFS interface - VFS / Virtual Filesystem / Virtual Filesystem Switch
- Makes all filesystems look like Linux filesystems. Might need hacks; i.e. for FAT have to fake a superblock, directory entries, and inodes (generate on the fly). Can be important having consistent inode numbers as filesystems like NFS use them even across reboots.
- Objects
  - superblock

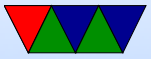


  - inode object (corresponds to file on disk)
  - file object – info on an open file (only exists in memory)
  - dentry object – directory entry.
- Can use default versions, such as default_llseek
- dentries are cached. As they get older they are freed.
- dentry operations table. hash. compare (how you handle case sensitive filesystems)



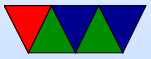
Linux Filesystem Interface

- linux/fs.h
- Module. Entry point init_romfs_fs(), exit_romfs_fs()
  - init_romfs_fs() – register_filesystem()
    name, romfs_mount, romfs_kill_sb
  - romfs_mount – mount_bdev(), romfs_fill_super
  - sb− >s_op=&romfs_super_ops();
  - romfs_iget() − > i_op struct, gets pointed to in each inode



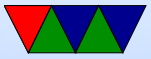
mounting

- Opens superblock
- Inserts into linked list of opened filesystems



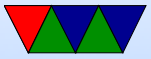
pathname lookup

- If begins with /, starts with current− >fs− >root
- otherwise, relative path, starts with current− >fs− >path
- looks up inode for starting directory, then traverses until it gets to the one wanted
- the dentry cache caches directory entries so the above can happen without having to do any disk reads if the directory was used recently before
- the access rights of intervening directories must be checked (execute, etc)



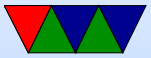
- symbolic links can be involved
- you might enter a different filesystem
- Should you cache invalid file lookups?

  Programs that try to open the same nonexistent config files on start, should we cache it doesn't exist to speed that up

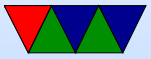


open syscall

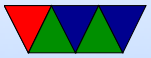
- `getname()` – safely copies name we want to open from userspace process
- get_unused_fd() to get the file descriptor
- calls filp_open()
  - creates new file structure
  - open_namei() – checks dentry cache first, otherwise hits disk and looks up dentry
  - lookup_dentry()
- validates and sets up the file



- returns a fd

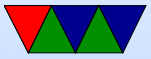


What about our 531 OS?

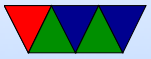


531 OS – open

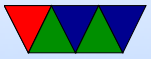
- open()
- sets up file descriptor, fill with info on file
  - uint64_t file_offset; where we are in reading
  - struct *file_ops;
  - struct inode();
  - count; (number of times opened)
  - flags
  - name
- file_ops pointer to the file_ops for the filesystem file is



on
  - read()
  - write()
  - llseek()
  - getdents()
  - ioctl()
  - open()
  - flush()
  - fcntl()
- inode struct
  - device

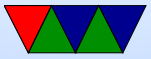


  - inode number
  - count
  - mode
  - hard links
  - uid/gid
  - rdev
  - size
  - superblock pointer
- file struct allocates open file slot in array, returns a number indicating which slot
- all I/O on this file descriptor



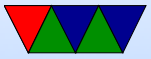
531 OS – doing a read

- read on file descriptor
- kernel looks up info using it as index
- get current offset to read from (file_offset)
- superblock pointer says which filesystem on
- inode number used in conjunction with that to find the file
  - on romfs, just maps directly to block
  - on fat32 will use it in conjunction with FAT
  - on ext2, do indirect



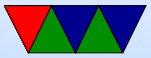
531 OS – doing a write

- much more complicated
- if creating new file, has to allocate new directory entry. allocate inode. allocate space for first block (all of these can fail, then what? be sure to error handle and deallocate properly)
- if write enough data to need new block, need to allocate new block on filesystem and hook it up (in FAT, indirect, etc)
- what if multiple programs writing to same file at once?



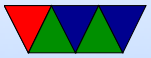
531 OS – deleting/truncating

- how do you delete a file?



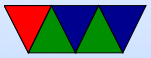
Multi-Processing

- In the old days your computer had a single CPU/core
- That was relatively simple to deal with
- Modern systems (even small embedded systems) have multiple cores



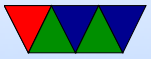
Hardware Concerns – Multi-Processing

- SMP/CMP (Symmetric or Chip Multi-processing): all cores are identical
- Asymmetric: cores can have different features (see ARM big.LITTLE or intel's efficiency cores)

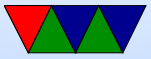


Hardware Concerns – Multi-Threading

- SMT (Simultaneous Multi-threading), Hyperthreading (Intel)
- Pipelined processor might not be able to fill all pipelines each cycle
- Add an extra instruction queue and have two programs issuing instructions to the pipelines
- Less transistors than extra core but usually not as much of a performance gain (can actually be worse!)
- OS often treats extra threads like extra processors for

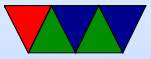


scheduling purposes



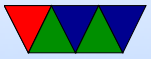
Hardware Concerns – Memory

- Shared memory vs Distributed
  Shared memory, a CPU can write a value to memory, read it back and it will be different (another CPU can write to it)
- How many copies of the OS? One per core or single image? One per core is more like a cluster.



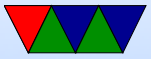
Hardware Concerns – NUMA

- In old days, single CPU with one single range of memory
- Modern CPUs, the memory controller (and DIMMs) might be run by separate packages
- Some RAM is more distant from a core than others
- This leads to NUMA (non-uniform memory access), some RAM takes longer to access
- OS should take this in account when starting processes / scheduling jobs
- UMA, NUMA, CC-NUMA (cache-coherent)



Multi-Processor Resource Sharing

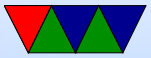
- How are resources shared in SMP system?
- Any core can access any of the devices. Need locking.



Multi-Processor Interrupts

- Have one core handle all interrupts?
  Might have better cache behavior
- Round-robin interrupts to each core?
  Reduces load on core0 but hurts others.

- Balance interrupt load across processors?



Helper Threads

- Linux has kernel threads (look in top for things starting with k or rcu).
- One of each type of thread per core
- Interrupt handlers have fast handler and worker threads.