ECE 598 – Advanced Operating Systems
Lecture 22

Vince Weaver

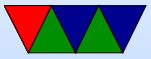
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

19 April 2016

Announcements

- Project update
- HW#9 posted, a bit late
- Midterm next Thursday

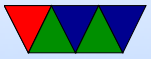


Homework #8

1. Filename in inode? UNIX no, mostly due to hard links. Can be other reasons too... what happens when you change filename, you'd have to somehow grow the inode; inodes were traditionally a fixed size.

2. Ext2 overhead, 1k blocksize. Note power of 2 vs marketing bytes

(a) with no indirect, 12 entries*1k = 12k
(b) with single, 12k+((1k/4)*1k) = 268k



(c) with double, 12k+256k+(256*256*1k) = 65804k (64MB)

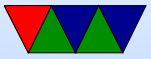
(d) with triple, 12k+256k+65804k+(256*256*256*1k) = 16,843,288k = 16GB

(e) overhead = inode + 1k (single) + 1k+256k (double) + 1k+256k+64M(triple) = roughly 66,051k (64MB) fairly small

(f) (not a question) 4kb blocksize, triple = 48k+1024k+4GB+16TB (with roughly 4GB overhead)

3. fat vs ext2

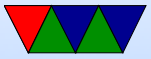
   speed? speed doing what?



maximum filesize permissions simplicity journaling is technically an ext3 feature portability

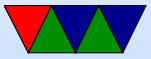
4. ls vs du
   shows holes in file
   man du should help
   also shows blocks used? but not things like indirect blocks, don't think there's a Linux syscall to report that kind of thing.

5. proc info generated on the fly (doesn't necessarily even live in ram, some of the /proc files are huge)



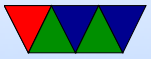
6. filesystems

- reiserfs
- hostfs (UML)
- jffs (journaling flash filesystem)
- efs, an SGI filesystem, not the windows encrypting fs
- NTFS
- coda
- overlayfs



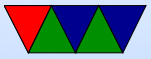
Processes – a Review

- Multiprogramming – multiple processes run at once
- Context switch – each process has own program counter saved and restored as well as other state (registers)
- OSes often have many things running, often in background.
  On Linux/UNIX sometimes called daemons
  Can use `top` or `ps` to view them.
- Creating new: on Unix its fork/exec, windows



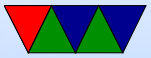
CreateProcess

- Children live in different address space, even though it is a copy of parent
- Process termination: what happens?
  Resources cleaned up. atexit routines run.
  How does it happen?
  `exit()` syscall (or return from main).
  Killed by a signal.
  Error
- Unix process hierarchy.



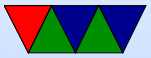
Parent and children, etc. not strictly possible to give your children away, although init inherits orphans

- Process control block.



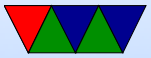
Process States

- Running – on CPU
- Ready – ready but no CPU available
- Blocked – waiting on I/O or resource
- Terminated

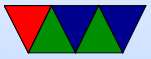


Scheduling

- Picks which jobs to run when
- Complex problem
- Simple: batch scheduling. Each run to completion.
- Multi-tasking.
- Computation often mixed with slow I/O
- Avoid context switching if possible

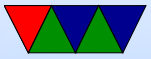


- Can switch when task voluntarily yields, if kernel blocks on I/O, or if timeslice runs out
- Simple round-robin scheduling
- Different type of processes. Long-running CPU bound where extra latency doesn't matter? Interactive things like GUI interfaces, video games, music playing where too much delay is bad? Real time constraints?



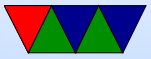
Scheduling Goals

- All: fairness, balance
- Batch: throughput (max jobs/hour), turnaround (time from submission to completion), CPU utilization (want it busy)
- Interactive: fast response, doesn't annoy users
- Real-time: meet deadlines, determinism



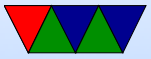
Batch Scheduling

- First-come-first-served (what if 2-day long job submitted first)
- Shortest job first
- Many others



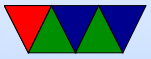
Interactive Scheduling

- Round-robin
- Priority – “nice” on UNIX
- Multiple Queues
- Others (shortest process, guaranteed, lottery)
- Fair scheduling – per user rather than per process



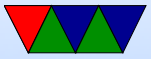
Real-time Scheduling

- Complex, more examples in 471 or real time OS course



The Linux Scheduler

- People often propose modifying the scheduler. That is tricky.
- Scheduler picks which jobs to run when.
- Optimal scheduler hard. What makes sense for a long-running HPC job doesn't necessarily make sense for an interactive GUI session. Also things like I/O (disk) get involved.
- You don't want it to have high latency

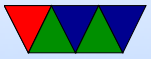


- Linux originally had a simple circular scheduler. Then for 2.4 through 2.6 had an O(N) scheduler

- Then in 2.6 until 2.6.23 had an O(1) scheduler (constant time, no many how many processes).

- Currently the "Completely Fair Scheduler" (with lots of drama). Is O(log N). Implementation of "weighted fair queuing"

- How do you schedule? Power? Per-task (5 jobs, each get 20%). Per user? (5 users, each get 20%).



Per-process? Per-thread? Multi-processors? Hyper-threading? Heterogeneous cores? Thermal issues?