Energy Management

1. Motivation
    a. Goal of an OS: resource management
        i. Decide which processes get which resource, how much
            1. Memory: page replacement, etc.
            2. CPU: scheduling
            3. I/O: network bandwidth, disk space and bandwidth
    b. What about energy?
        i. Energy: how much of your battery it is using; power x time used
        ii. Power: instantaneous draw of energy
        iii. Visibility
            1. Who is using energy, and how much
            2. Why hard?
                a. Need to look at services invoked by an app, devices used by an app
                b. Shared resources: radios
                    i. High power
                    ii. When turned on, stays on for a while
                    iii. Used by multiple applications
                    iv. Who pays how much?
                c. Systems don't have much to measure power use
                    i. CPUs now have sensors
                    ii. Measuring devices hard
                    iii. Need to model behavior and predict power/energy
                    iv. Example: energy of disk access (spin, seek, time), radio (transmission, receive based on signal strength)
                    v.
        iv. Control
            1. QUESTION: as a user, what do you want?
                a. Limit total energy usage?
                b. Ensure battery lifetime?
                c. Reserve life for high-priority apps (maps, phone, texting, camera)?
                d. WHAT IS ACTUAL USER GOAL?
            2. Control how much energy is being used by an application/service

        3. Want to control total (kill app at end) but also rate, so app lives
  c. Big ideas in energy management:
    i. Power proportional to ½ cv^2f
      1. C = capacitance, how much of chip is changing states
      2. F = frequency – reducing just F has a linear scale in power
      3. V = voltage – reduces power quadratically
      4. When F drops, can reduce V as well
        a. Get cubic effect in power drop
    ii. Energy:
      1. Simple model: just reducing frequency doesn't save energy
        a. Run ½ as fast, but run 2x long
        b. E = P*T; E=(p/2)*(t*2)
    iii. Hardware support for power management: (intel HW)
      1. G states – global states of whole system
        a. G0 = running
        b. G1 = sleeping (suspend, hibernate)
        c. G2 = soft off (power down but can be powered up by interrupts)
        d. G3 = mechanical off – no power
      2. P states: performance states
        a. voltage/frequency pairs. Lower state (P0) is higher performing
        b. P0 = high perf, high power
        c. P>0 = lower perf, lower power
        d. Processor automatically increases P state when CPU is underutilized
      3. C states – core states, idle power saving – subset of G0 (running)
        a. C0: core is running
        b. C1: idle (after "halt" or "mwait" instruction; but caches full
          i. Fast to enter, fast to leave
          ii. CPU clock stopped, but bus and interrupts still run full speed
          iii. Leave when interrupt arrives (e.g., timer, network packet)
        c. C>1: deeper sleep (slow to enter, slow to leave, but use less power)
          i. Turn off all clocks, including bus, APIC interrupts
          ii. Good to use if will sleep a while

4. S states = sleep states (subset of G1)
    a. S0 = not sleeping
    b. S3 = suspend
    c. S4 = hibernate

iv. Big idea 1: reduce frequency to what is required
1. Assume have a bottleneck resource: disk, memory, network
2. Run CPU at lowest frequency & voltage to provide service
    a. Seek 100% utilization of CPU
    b. Get V,F scaling benefit
    c. No point in running CPU faster as waiting on other resources
3. Implies perfect rate depends on the application

v. Big idea 2: idle as much as you can
1. If you finish a computation faster, you can turn off more of the system
2. Example: power off more CPUs, power down memory, I/O devices, display
3. Suppose CPU is 50% of system power
4. Run at full speed, uses energy X
5. If run CPU twice as fast and use 2x power, now have:
    a. Normally: E = X * t (CPU) + X * t (rest of system) = 2Xt
    b. Now: E = 2x*t/2 (cpu) + x * t/2 = 1.5Xt
6. Called "Computational sprinting" – finish quickly so you can turn more of the system off

2. Cinder
    a. Goals:
        i. Isolation: separate energy draw of each app; one app shouldn't be able to (without permission) draw down your whole battery
        ii. Delegation: give energy to a service/app doing work on your behalf
            1. E.g. networking stack
            2. Like ticket transfers
        iii. Subdivision: can partition energy, share some but keep some
            1. Don't give delegates full access to all energy, just to a part
            2. Example: browser plugins. Need some energy but not much, don't want abuse
    b. Abstractions:
        i. Reserves: right to use a certain amount of energy
            1. Like a virtual battery
            2. Amount in reserve goes down when energy consumed

            3. Cannot execute when reserve has insufficient energy
            4. Track energy consumption
            5. QUESTION: when plug in battery, what reserves get charged first, and how quickly?
        ii. Taps: connect reserve to another reserve using energy with rate limit
            1. Example: allow 1 mJ/sec (1 mw average)
            2. Alternate use: proportional tap
                a. Transfer a fraction of reserve instead of absolute energy amount
            3. Purpose: rate limit allows saying how long enegy will last
                a. Guarantee 5 hour battery life
            4. Use: can replenish a reserve threads use to execute at a fixed rate
                a. Prevents thread from running too much
                b. Implementation: periodically decrease one reserve, increase another
        iii. Resource consumption graph:
            1. Graph of reserves, taps, threads (or devices) connecting power sources to power users
    c. How meet goals:
        i. Isolation: separate reserves per app
        ii. Subdivision: app can create new reserves
        iii. Delegation: app can create tap from its reserve to child processes/threads/IPC targets
3. Implementation on a phone
    a. Power model: where get a model to calculate how much
    b. Online: use on-chip measurements to measure how much was used
        i. Problem: hard to account for I/O, devices
    c. Offline: run a bunch of workloads, measure behavior with performance counters, measure energy externally with tool
        i. Can calculate energy draw of different operations.
        ii. Limited to what can be measured (e.g. HTC dream can't count memory operations that use different energy than integer)
    d. Radio model
        i. Cost is basically independent of workload; doing anything to turn of radio is expensive
        ii. Radio stays on for a while at high power once activated
4. What not addressed:
    a. How manage use in kernel.
    b. How mange "wake locks" – keep phone at high power state when in use
5. Uses:

a. Sandboxing: give an app a fixed amount of energy or rate of energy via a reserve or a tap
b. Fine grained control – shared code handling multiple things, such as video plugin
    i. Tap per page to give it some energy for each activity it is doing
c. Reclaiming unused energy in a reserve:
    i. Send energy to a reserve
    ii. Use a proportional tap to send energy back to source reserve
        1. If not used, will eventually drain the reserve
        2. Proportional means if energy is low doesn't transfer much
        3. BUT: paper doesn't explain accumulating 10 seconds; why is that?
d. Hoarding
    i. Threads can create a reserve and store their energy in it; HOARDING
        1. Backwards taps to an app don't apply to new reserve: to system, looks like a use of energy
        2. If not create reserve, thread could move energy to reserve with slower backwards tap
    ii. Solution: long term decay on all reserves
        1. Every reserve has implicit backwards proportional tap
        2. Return 50% of reserves in 10 minutes
        3. Similar to "idle memory tax" in vmware
        4. Not apply to system reserves (e.g. network), only applications
e. Application use
    i. What does it mean to be energy aware?
        1. Applications adapt behavior according to available energy
        2. Reduce fidelity, reduce functionality under low energy
        3. Example: lower frame rate, reduce resolution of images
    ii. Background apps:
        1. Want to allow but ensure don't use much energy (matching user expectation)
        2. Solution: apps have two taps:
            a. Foreground: allow high use when in foreground
            b. Background: allow low rate use when in background (foreground set to zero)
        3. Task switcher turns off foreground tap when switch to new app
    iii. Shared power-consuming resources: GPS, network
        1. Give each resource a reserve

            2. Make apps using resource put a tap to reserve at a low rate
            3. Provides enough for periodic use, shares cost among all users
            4. QUESTION: How adjust rate as users come/go?
    iv. Network stack:
        1. How account execution in network stack back to application using network?
            a. Cinder uses protected procedure call; so thread migrates (like LRPC)
            b. Thread uses its own reserve as it execute
            c. Else:
                i. Would need to extend RPC/IPC to create/destroy a tap (like Lottery Scheduling)
                ii. PROBLEM Linux IPC mechanisms don't always identify source/destination, so hard to set up a tap.
        2. How handle expensive network start up?
            a. Create a reserve for all threads using network to put energy into for periodic use
            b. When enough energy available, network turns on and everyone uses it.
                i. BENEFIT: coordinates use across apps, so wait to turn on once rather than turning on at a different time for each app
            c. Charge based on eventual cost
                i. Radio turns off 20 seconds after last packet
                ii. If send packet 1 second after last packet, need to pay for 1 second of additional time (extension of turnoff time)
                iii. If send packet 15 seconds after last packet, need to pay for 15 seconds of additional time (was extension)
                iv. QUESTION: What if someone runs 1 second later?
                    1. It pays for 1 second of use, original app still pays for 15
                    2. Seems unfair
            d. Receiving packets:
                i. Charge receiving thread, even allow to go into debt (better than dropping packet)

        e. So:
            i. When receive packet, may delay response (send) until accumulate enough energy to run
            ii. Causes rate limiting
            iii.

6. Key questions:
    a. What happens to an app when its energy is limited?
        i. It doesn’t get scheduled until tap delivers energy
            1. What is the user experience?
        ii. Does it run often enough for interactivity?
            1. Example: could network connections timeout when run out of energy?
            2.