Dune

What is Dune?
  Linux driver providing processes with access to privileged instructions
  Uses intel VMX to do so safely - run process in VMX non-root mode
    Dune driver in Linux kernel gets control back on VM exit events

What privileged instructions does it provide access to (Table 1)?
  Exceptions
    - LIDT: Load table saying where to jump on which exceptions/interrupts
    - LTR: Specifies stack pointers on switch to more privileged mode
    - IRET: Return from interrupt/exception (possibly lowering privilege)
    - STI, CLI: enable/disable interrupts
  Virtual memory
    - MOV %CRn: cr3 = page table base register (PML4 address, PCID),
                cr2 = page fault linear address,
                cr0, cr4 = various flags (e.g., enable/disable G bit, PCID)
    - INVLPG: knock single page out of TLB
    - INVPCID: knock all entries matching PCID out of TLB
  Privilege modes:
    - SYSRET, SYSEXIT: return from system call
    - IRET: return from trap/exception
  Segmentation: LGDT, LLDT

Who is writing this paper?
  Bunch of academics, no industry
What problem is Dune trying to solve?
  Academic problems? Get student a job at a prestigious school? (admittedly)
  Have some actual impact with kernel research!
    I've personally worked on 3 research OSes (Exokernel, AsbestOS, HiStar)
    Great results, successful students, widely read papers, ...
    But actual software artifacts completely unused a few years later
  Why is it so hard to have impact with a research OS?
    Usable OS requires a lot of code
      E.g., develop OS to explore new networking idea, need file system too
      Application compatibility is incredibly hard
      Just getting system you can manage via SSH is very hard
    Different research OS ideas don't compose easily
      E.g., say you like exokernel and sv6--can't run both
    Annoying to have new OS for one application (e.g., VM for faster GC)

What does a process need to do to get into Dune mode?
  Set up its page tables beforehand
  Open /dev/dune, do ioctl with page table base register value

How is guest physical memory set up?
  Ideally guest physical memory = host virtual memory, EPT = linux host PT
  What hardware limitations prevent this?
    1. EPT has different format from page tables
    2. Small host physical address space (36 bits compared to 48 for VA)
    3. No accessed and dirty bits in EPT
  How to address?
    1. Must manually maintain EPT; fill on demand
       Find out about invalidations through "MMU notifier chains"
       Basically KVM has to solve this, too, so piggyback on its mechanisms
    2. Address space compression maps (section 3.4)
       Low VM goes to low 4GiB ofguest physical
       Mmap regions goes to next 4 GiB
       Stack goes to third 4 GiB region, for 12 GiB total
       Guest page tables expand this to original layout
    3. Conservatively report always dirty and accessed (section 3.7)

What does the system call instruction do in a Dune process?
  Vectors to the process itself, to address in MSR_LSTAR (sec 3.5)

```
  If at CPL3 (user mode), will switch to CPL0 (supervisor mode)
  Does *not* cause a VM exit, so doesn't actually call into Linux kernel
How does a Dune process actually make a Linux system call?
  VMCALL instruction causes VM exit, Dune driver forwards to system call
  What about pointers in system call arguments?
    Must be translated from guest physical to host virtual
  How does binary compatibility work?
    Record whether at CPL0 or CPL3 in syscall handler
    From CPL3, forward to in-process handler (allows sandboxing)
    From CPL0, forward to kernel with VMCALL

How to handle signals?
  Many such as SIGSEGV are obviated by hardware exceptions
  Others delivered as interrupts--why?
    Can ensure sandbox (CPL3) code does not ignore interrupts
    Also potentially makes STI/CLI useful for delaying signals

What optimizations can Dune make compared to a VMM
  Save and restore less state (e.g., don't virtualize cycle counter)
  Dune process doesn't need its own network stack
  No need for fake framebuffer
  PCID (Linux avoids management complexity of 12-bit namespace, TLB shootdowns)

What evaluation questions should we ask?
  1. Is this useful?
  2. What's the complexity?  (Dune kernel module only 2509 LOC)
  3. What's the performance cost/gain?
Can answer each of the above question in the context of 3 applications
  A. sandbox
  B. Wedge
  C. Garbage collection

How does Dune sandbox work?
  Run untrusted code in CPL3, with sandbox runtime in CPL0
    Runtime uses hardware memory protection just like a kernel
    Intercepts system calls to apply policy
  Examples: firewall network access, or checkpoint and restore program
  How does sandbox runtime load a binary?
    Recall dynamic ELF executables loaded by ld-linux.so
    Dune library just adds dumb ELF loader good enough to load ld-linux.so
    Rest happens entirely within untrusted sandbox code
Are sandboxes useful?
  Google spent many engineer-years on Native Client
  OSes have added less powerful features to kernel (AppArmor, systrace/pledge)
How is complexity?
  450 SLOC for checkpointing, 200 SLOC for firewall
  Native client is 1 GiB of source code (including modified compiler)
How is performance?
  Figure 3: very little overhead except a few benchmarks (e.g., mcf)
    What's going on with mcf?
      Lots of TLB misses (#EPT levels)*(#PT levels) in worst case
        Table 2: 86.4 vs 35.8 cycles for TLB miss with/without EPT
      Similar slowdown happens when running under VMWare Player
      Solution?  Use large page translations (helps vanilla Linux, too)
  Figure 4: Lighttpd 2% slowdown
    Why is VMware Player so much slower than Dune?  requires 2 network stacks

What is Wedge and how does it work?
  Idea: reset web server for each request--erasing effects of any compromise
  Could use fork(), but too slow--instead, introduce faster *strheads*
  Previous paper introduced sthreads using kernel/user hack
  With Dune just use address spaces, including PCID for fast switching
Is wedge usefull?
  Remote exploits happen and are pretty serious, lots of work on hardening
  Not clear from here what attacks wedge blocks, that is previous paper
```

  At least seems like it would be harder to exploit a priori
How is complexity?
  Original wedge 700-like kernel hack + 1300-line user mode
  Dune implementation just 750 lines of user code
How is performance (Table 5)?
  20% improvement over previous sthread implementation for http request
  Anything missing from table? baseline non-wedge numbers?

Garbage collection
  Why might Dune help with garbage collection?
    Need to find all pointers to objects
    If page known not dirty, pointers will be same as during last scan
    Boehm GC has two techniques for finding dirty pages (sec 5.3):
      Use mprotect, catch SIGSEGV on write, then mark dirty and unprotect
      Query kernel for dirty bit?
  Dune measures several approaches
    Drop in replacement for mprotect is much faster (only 50 LOC change)
    Replace mprotect but use INVLPG instead of flushing whole TLB for 1 page
    Just look at the dirty bit in the page table (simple load instruction)
Is this useful?
  GC is widely used for many languages
  Azul systems already demonstrated benefit to extending kernel for GC
  Dune is kernel module, maybe easier to deploy than kernel with custom VM
How is complexity?
  Only changed 82 LOC for checking dirty bit version, similar for others
How is performance?
  26.4%-40.7% improvement for dirty bit version, all for 82-line code change!
  What about XML benchmark? Never runs GC, so seeing EPT overhead
    Force GC and get a 12.8% speedup