CSE221 Lecture 10

Aronya Baksy

November 2024

1 IBM VM/370 Time Sharing System

- Main components:
  - **Control Program** (CP): operating system that uses a computing machine to simulate multiple copies of the machine itself.
  - **Conversational Monitor System** (CMS): OS used within each VM, supports the interactive use of a machine by one person
  - **Remote Spooling and Communications Subsystem** (RSCS): Provide information transfer between linked machines
- Main goal: provide a bridge between (then) current paradigm of computing (batch processing, time sharing) and future paradigms of individual networked workstations (our present)

1.1 Control Program

- Uses **virtual machines** to **organize independent job streams**, uses a portion of total machine resources for itself, is **privileged**
- Each VM is identified by an entry in the CP directory. Directory entry contains: permanent and temporary virtual disks attached to the VM
- CP spool system virtualizes I/O devices (e.g. line printer)
- CP has to schedule h/w resources between VMs based on expected usage profiles
- Human operator uses terminal commands to interface with the CP which itself is highly privileged, these commands can add/delete hardware components
- All communication between VMs (via shared memory, disk files etc.) is handled by CP. This means all the VMs are independent

1.2 CMS

- a disk-file-oriented, single user operating system to support the personal use of a dedicated computer
- resides in low memory address, key protected to prevent deletion but can be modified by user programs
- Sharing info with other machines outside CMS is achieved using shared disks

1.3 RSCS

- interrupt-driven, multi-tasking system that uses a dedicated computer with attached communication h/w for the support of data file transfer
- Star topology with the CP being the center, all information flows through CP
- Direct transfer between card readers, hence works only between VMs on a single real machine
- Each VM and system in a VM/370 installation has its own unique ID

2 Xen and the art of Virtualization

2.1 Goals

- x86 VM monitor that allows safe and managed resource sharing without sacrificing performance or functionality
- Port existing OSes (Linux, BSD, WinXP) with minimal effort, minimal performance overhead
- 100 virtual machine instances simultaneously on a modern server (ca. 2003)

2.2 Challenges

- Isolation of VMs
- Support multiple different guest OS
- minimal perf. overhead
- Existing single OS servers do not provide isolation (one user/process's actions affect the others) and sys admin is vv complex
- Early container implementations have hidden interaction costs (e.g. paging/buffer access conflicts)
- Xen: multiplexes physical resources at the granularity of an entire OS, provide performance isolation between them
- Xen also has higher overhead of initialization and resource consumption, which authors are Ok with given their target of 100 instances

2.3 Approach: Paravirtualization

- Virtualizing x86 entirely is hard, especially MMU virtualization, because x86 was not designed for it
- VMWare ESX Approach: **binary rewriting**
  - The sensitive instructions in the guest OS are replaced by either Hypervisor calls which safely handle such sensitive instructions or by some undefined opcodes which result in a CPU trap handled by the hypervisor (this happens at runtime)
  - Maintain shadow data structures that mirror virtual data structures (e.g. page table) and trap on each modify attempt (expensive for memory intensive operations like creating a process, similar to L4Linux)
- Xen Approach: **paravirtualization**
  - **Paging**: Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.
  - **Privilege**: Guest OS must run at a lower privilege level than Xen.
  - **Exception Handling**: Guest OS must register a descriptor table for exception handlers with Xen. Only page fault handler has to be modified to avoid reading CR2 register and instead read faulting addr from stack frame
  - **Syscalls**: Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
  - **Interrupts**: Hardware interrupts are replaced with a lightweight event system.
- Domain0 hosts control software for scheduling and allocating h/w resources
- The control interface in domain0 exports Virtual network interfaces and block devices (VIF, VBD)
- DomU hosts the guest OS code

2.4 Memory Virtualization

- x86 does not support software TLBs or tagged TLBs, making virtualization hard
- Xen allows guest OSes to manage their TLBs, with minimal intervention for isolation
- Xen has reserved 64 MB at the top of every address space, hence TLB flushes not needed for every hypervsor call
- Guest OS page tables are read only for the guest OS, writes are trapped to Xen, translate from virtual address to physical address directly (Similar to Exokernel)
- Each time a guest OS creates a new page table, it provides Xen with a page of its memory which Xen is allowed to update
- Guest OSes can batch page table updates to amortize the cost of a hypervisor call

2.5 CPU Virtualization

- Guest OS modified to run in a lower privilege level than the hypervisor
- Makes use of x86 rings for protection: hypervisor in ring 0, guest OS in ring 1 (not 0), applications in ring 3
- All attempts by guest OS to run privileged instructions (PTE modify etc.) are failed or trapped
- Guest OS registers exception handlers which are identical to x86 handlers (only page fault handler is different as it can't read faulting address from the CR2 register anymore, hence Xen has to copy the stack frame to the guest OS and return control)
- Fast syscall handlers avoid the need to repeatedly enter the hypervisor
- All handlers are validated to ensure that they don't specify execution in ring 0

2.6 Detailed Design

- **Ring**: circular queue of descriptors created by a domain but accessible by Xen
- Two pairs of producer and consumer pointers around the ring: one for domains to **request** and for Xen to **consume** requests
- This abstraction is clean and works for a large number of devices
- Physical memory reservation for a new domain is specified before it is created
- Balloon driver adjusts a domain's phys mem usage by exchanging pages between Xen and the guest OS pager
- VIF implemented as 2 rings: transmit and receive buffer rings:
  - Transmit: guest enqueues a buffer descriptor, Xen copies the descriptor but NOT the actual packet (which stays pinned) and executes packet filter rules
  - Receive: Xen determines the destination VIF, enqueues descriptor onto the receive ring and exchanges a page from guest OS

3 Recent Hardware Advancements

- **Root ring 0** - an extra privilege layer more privileged than ring 0 that avoids the need to modify guest OS that are used to running in ring 0
- **Extended Page Table**: implements second-level address translation in hardware and avoids software overhead of maintaining shadow page tables
- **Single Root IO Virtualization** (SR-IOV) is a technology that allows a physical PCIe device to present itself multiple times through the PCIe bus (allows a single device to virtualize itself as multiple instances)