Support Dynamically Linked Executables via Linux ld.so and Implement ENA Driver

Expand Application of OSv
Agenda

- Support statically linked executables and dynamically linked executables via Linux ld.so
- ENA driver and AWS Nitro
- XConfig preview
- Upcoming 1.0 release and beyond
Most applications do NOT make system calls into Linux kernel directly
Instead, they call libc functions that delegate to SYSCALL or SVC instruction
The OSv built-into-kernel dynamic linker memory-maps ELF files and resolves the undefined symbols by pointing them to OSv implementations
Supported types
- Shared Libraries and Dynamically Linked Executables
- PIEs and non-PIC
Benefit
- Fast local function calls without SYSCALL/SVC overhead
Drawback
- Linux compatibility is a moving target
OSv built-in dynamic linker and libc
Statically linked executable

- Statically linked executables make direct system calls to Linux kernel
- OSv initially implemented ~70 syscalls to support Golang executables
- ~60 new syscalls implemented including the key ones like brk() and clone() in order to support statically linked executables
- Most challenging part was to support application thread-local storage (TLS)
- Expose vDSO as part of the kernel image
- Benefit
  - Better Linux compatibility
- Drawback
  - Overhead of system calls
Statically linked executable

Program ELF

PT_LOAD segment 1

PT_LOAD segment 2

PT_LOAD segment 3

MOVQ RAX, $1

SYSCALL

call puts

mmap

OSv Kernel

Dynamic Linker

SYSCALL handler

write()
Linux dynamic linker and glibc

- Run dynamically linked programs using the Linux dynamic linker (LD) instead of the OSv built-in one
  - `scripts/run.py -e '/lib64/ld-linux-x86-64.so.2 /hello'
- Needs to add `ld-linux-x86-64.so.2` or `ld-linux-aarch64.so.1` and other libc library files to the image
- Benefits
  - Better Linux compatibility
  - Ability to take advantage of glibc optimizations
- Drawbacks
  - Overhead of system calls
  - Inability to use the OSv libc optimizations
Linux dynamic linker and glibc

- Linux Dynamic Linker (ld.so)
- Program ELF
  - load in memory
  - call puts
  - MOVQ RAX, $1
  - SYSCALL

- OSv Kernel
  - mmap
  - Dynamic Linker
  - SYSCALL handler
    - mmap()
    - write()
Strace

[wkozaczuk@fedora-mbpro osv-master]$ ./scripts/run.py -e '---strace --trace=syscall /hello-static-pie'
OSv v0.57.0-142-gcb7d1803
eth0: 192.168.122.15
Booted up in 164.27 ms
Cmdline: /hello-static-pie
/hello-static-p 0 0.130731389 syscall_arch_prctl(0xffffffff <= 12289 0x200910)
syscall(): unimplemented system call 334
/hello-static-p 0 0.132313326 syscall_sys_brk(0x400000 <= 0x0)
/hello-static-p 0 0.132575378 syscall_sys_brk(0x400d00 <= 0x400d00)
Hello from C code
/hello-static-p 0 0.132576287 syscall_arch_prctl(0x0 <= 4098 0x400380)
/hello-static-p 0 0.132579222 syscall_sys_set_tid_address(45 <= 0x20000400650)
/hello-static-p 0 0.132579848 syscall_sys_set_robust_list(0 <= 0x20000040060 24)
/hello-static-p 0 0.134435177 syscall_prlimit64(0 <= 0 3 0 0x200000200830)
/hello-static-p 0 0.135128185 syscall_readlink(17 <= "/proc/self/exe" 0x1ff7a0 4096)
/hello-static-p 0 0.135463178 syscall_getrandom(18446744073709551615 <= 0xb9100 8 1)
/hello-static-p 0 0.135467276 syscall_clock_gettime(0 <= 1 0x2000001ff730)
/hello-static-p 0 0.135467663 syscall_clock_gettime(0 <= 1 0x2000001ff730)
/hello-static-p 0 0.135469839 syscall_sys_brk(0x400d00 <= 0x0)
/hello-static-p 0 0.135473147 syscall_sys_brk(0x421d00 <= 0x421d00)
/hello-static-p 0 0.135473490 syscall_sys_brk(0x422000 <= 0x422000)
/hello-static-p 0 0.136582837 syscall_mprotect(0 <= 0xae000 16384 1)
/hello-static-p 0 0.136594638 syscall_fstatat(0 <= 1 "" 0x2000000200630 010000)
/hello-static-p 0 0.136596784 syscall_sys_lchown(0 <= 1 21565 35184374187408)
/hello-static-p 0 0.137873814 syscall_write(0x12 <= 1 0x20000041610 0x12)
ENA Driver

Implement the AWS ena driver by porting the FreeBSD version

- Adapt the FreeBSD code to make it work in OSv
- Minimize changes so that we can backport any potential bug fixes or enhancements in the future
- Reduce the code footprint by eliminating features that are either not relevant to OSv or not needed at this point (like ioctl(), sysctl(), etc)

- Resulting driver "costs" ~7k lines of mostly C code and ~56K larger kernel size
- Can only be tested on AWS Nitro EC2 instance
- Seems to be stable and yield decent performance based on the tests involving iperf3, netperf, and simple httpserver app
AWS Nitro

- ENA driver is enough to run OSv image with ramfs on Nitro EC2 instances
- New script `deploy_to_aws.sh` to streamline the process of uploading OSv image as a snapshot, creating AMI and finally instantiating EC2 instance
- NVMe driver is WIP
XConfig - WIP

- Continuation of the modularization / driver profiles effort
- Xconfig files
- Add #ifdef in relevant places
- Makefile acts on .config
  - Include/exclude relevant object files
  - Pass configuration options to relevant source files
- Let garbage collection remove remaining stuff
XConfig - menu example

Arrow keys navigate the menu. <Enter> selects submenus --- (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <??> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

device Drivers ---->
  Core Components ---->
  [*] Enable preemption
  [ ] Enable code tracing
  [ ] Enable memory debugging
  [ ] Enable debug logger
  [ ] Hide non-libc symbols
  [ ] Use lazy stack
  Select image filesystem (Zeta File Syst...
788K loader.elf uses 1.2M of memory

- Optimize kernel size to 788L to run on Firecracker with < 2MB of memory
- Reduce kernel size by:
  - Hiding most symbols
  - Excluding all drivers but virtio/mmio
  - Excluding tracepoints, dhcp and networking stack code
  - Excluding std::locale
  - Eventually enable LTO (Link Time Optimization)
- Lower memory usage by:
  - Reducing RCU defer queue
  - Reducing L1/L2 memory pool size
  - Disabling procfs and sysfs
  - Reducing kernel thread stack size to 16K
788K loader.elf uses 1.2M of memory

OSv runs on firecracker 1.6 with 3M

./scripts/firecracker.py -e '---norandom /hello' -m 3M -c 1
2024-01-19T12:46:02.341228985 [anonymous-instance:main] Running Firecracker v1.6.0
2024-01-19T12:46:02.358999267 [anonymous-instance:main] Successfully started microvm that was
configured from one single json
OSv v0.57.0-153-g2cadc9c1
failed to mount procfs, error = No such device
failed to mount sysfs, error = No such device
Booted up in 4.03 ms
Cmdline: /hello
Hello from C code
Page ranges allocated total: 1245184
2024-01-19T12:46:02.364956025 [anonymous-instance:fc_vcpu 0] Received KVM_EXIT_SHUTDOWN signal
2024-01-19T12:46:02.364991173 [anonymous-instance:main] Vmm is stopping.
2024-01-19T12:46:02.365073718 [anonymous-instance:main] Vmm is stopping.
2024-01-19T12:46:02.402077187 [anonymous-instance:main] Firecracker exiting successfully. exit_code=0
Upcoming 1.0 release

- Planned for 1st quarter of 2024
- Remaining work:
  - Finish KConfig work
  - Add support of Ext2/3/4 filesystem
  - Merge IPV6 branch
  - Potentially implement NVMe driver
    - There are 2 PRs as candidates
Beyond 1.0

- Capstan 2
  - Remove obsolete features and add new desired functionality
  - Support building images out of binaries or packages, running those locally, and provisioning to the cloud

- Performance and Security
  - Optimize futex
  - Add some spinning to lock-less mutex_lock
  - Optimize atomic operations on single CPU
  - Implement ALSR and make kernel relocatable

- Support AWS Graviton
  - Implement UEFI boot
  - Implement MSI/X and ACPIA on AArch64
Thanks

- Organizers
- ScyllaDB
  - Dor Laor
  - Nadav Har’El
- Other OSv contributors
- Please join us
OSv Resources and Q&A

- Original OSv paper -
- P99 presentation -
- Wiki pages:
  - Modularization - https://github.com/cloudius-systems/osv/wiki/Modularization
  - Filesystems - https://github.com/cloudius-systems/osv/wiki/Filesystems