

Project OSv

Shang Wu
Bite Ye
Xinyu Han

2021-04-22
George Washington University

Content

- Overview of OSv
- Components of OSv
 - Memory & Thread
 - Filesystem
 - Network
- General answer to questions
 - System performance
 - Security
 - Optimizations
- Thoughts about the system
- Q&A

Overview of OSv

- It's a unikernel
 - Single address space
 - No system call overhead, simple function call
 - Small context switch overhead
- It's dedicated to virtual machine
 - No need to support kinds of hardware drivers
 - Isolation in hardware level
 - Small size
- It's for cloud
 - Easy to deploy
 - Small size + High performance == Cheap

Components of OSv Memory

- Implementation follows the POSIX-like APIs

```
struct page_allocator {  
    virtual bool map(uintptr_t offset, hw_ptep<0> ptep, pt_element<0> pte, bool write) = 0;  
    virtual bool map(uintptr_t offset, hw_ptep<1> ptep, pt_element<1> pte, bool write) = 0;  
    virtual bool unmap(void *addr, uintptr_t offset, hw_ptep<0> ptep) = 0;  
    virtual bool unmap(void *addr, uintptr_t offset, hw_ptep<1> ptep) = 0;  
    virtual ~page_allocator() {}  
};
```

Components of OSv Memory

- OSv uses single memory space
 - Both user and kernel shares the same memory space
 - Improve efficiency of the context switch
 - Does not need to switch the page table and flush TLB

Components of OSv Thread

- Six main characteristic for OSv
 - Lock-free
 - Preemptive
 - Tick-less
 - Fair
 - Scalable
 - Efficient

Components of OSv Filesystem

- “For its filesystem support, OSv follows a traditional UNIX-like VFS (virtual filesystem) design and adopts ZFS as its major filesystem.”
 - Also provides other options like ramfs and some other pseudo filesystems

```
int    sys_open(char *path, int flags, mode_t mode, struct file **fp);
int    sys_read(struct file *fp, const struct iovec *iov, size_t niov,
            off_t offset, size_t *count);
int    sys_write(struct file *fp, const struct iovec *iov, size_t niov,
            off_t offset, size_t *count);
int    sys_lseek(struct file *fp, off_t off, int type, off_t * cur_off);
```

Components of OSv Filesystem

- By utilizing function pointers, OSv adopts polymorphism to the filesystem part and make it extensible.

```
#define VOP_CREATE(DVP, N, M) ((DVP)->v_op->vop_create)(DVP, N, M)
#define VOP_MKDIR(DVP, N, M) ((DVP)->v_op->vop_mkdir)(DVP, N, M)
```


Components of OSv Filesystem

- Previous things mentioned are existed in any Unix-like operating system. In addition to those, OSv implemented a virtiofs based on virtio lib and FUSE (Filesystem in Userspace) to provide faster data exchange between host and guest.

✓ virtio-fs: initial implementation of read-only subset

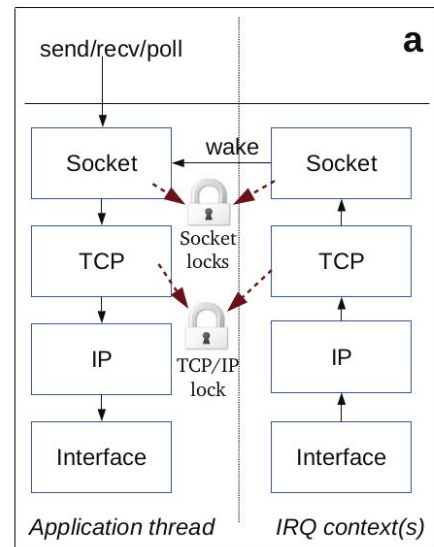
This patch provides initial implementation of read-only subset of new virtio-fs filesystem. One of the main two parts is new virtio driver - virtio-fs - that handles interactions with virtio device as specified in the new 1.2 virtio spec - <https://stefanha.github.io/virtio/virtio-fs.html#x1-41500011>. The second part implements VFS sublayer that provides virtio-fs filesystem in OSv and in essence serves as a passthrough between VFS abstract layer and virtio-fs device and host filesystem.

Components of OSv Network

- Problems of traditional network stack
 - Many layers in sending & receiving directions
 - Data copy from packet to skb to user
 - Lock contention
 - Inefficient data structure

Components of OSv Network

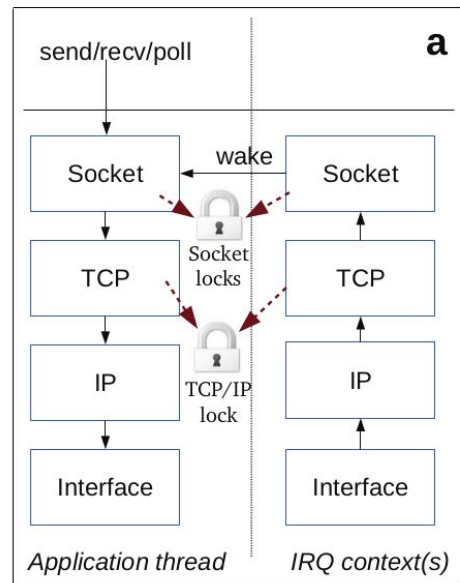
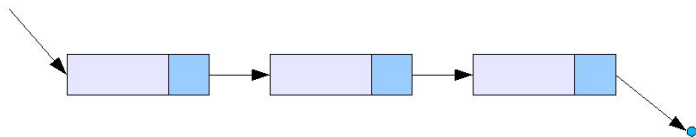
- Problems of traditional network stack
 - Many layers in sending & receiving directions
 - They are in different process/threads (context switches), or in different cores probably, thus bad CPU locality
 - System call overhead



Components of OSv Network

- Problems of traditional network stack

- Data copy from packet to skb to user
- Lock contention
 - Atomic operation freezes cache
- Inefficient data structure
 - Linked list is bad for cache



Components of OSv Network

- OSv solution - network channel by Van Jacobson in 2006

Components of OSv Network

- Context switches (different cores) problem is solved by the single address model, where all system calls are converted into simple function calls.
- OSv also removes layers in receiver side in kernel, instead replacing it with a simple classifier.

Components of OSv Network

```
class classifier {
public
    ...
    void add(ipv4_tcp_conn_id id, net_channel* channel);
    void remove(ipv4_tcp_conn_id id)
    ...
    bool post_packet(mbuf* m);
    ...
    ipv4_tcp_channels _ipv4_tcp_channels;
};

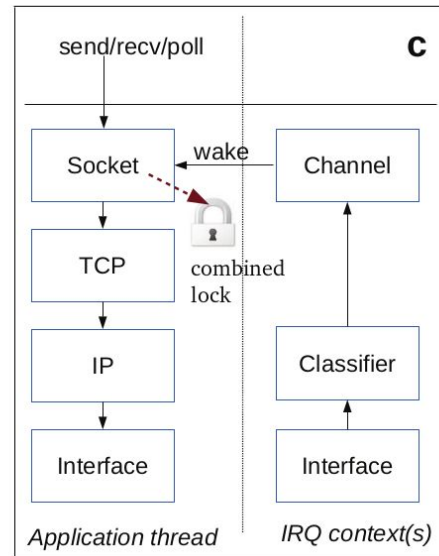
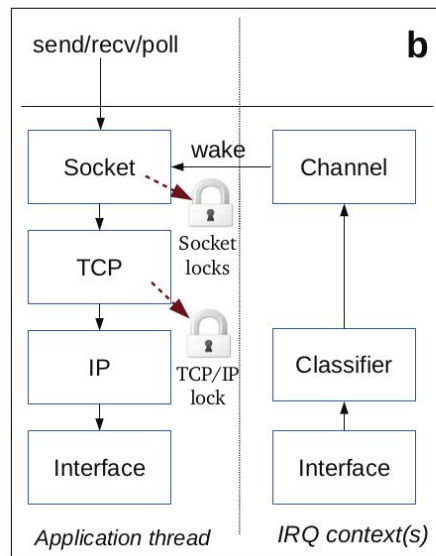
struct ipv4_tcp_conn_id {
    ...
    in_addr src_addr;
    in_addr dst_addr;
    in_port_t src_port;
    in_port_t dst_port;

    size_t hash() const {
        // FIXME: protection against hash attacks?
        return src_addr.s_addr ^ dst_addr.s_addr ^ src_port ^ dst_port;
    }
    bool operator==(const ipv4_tcp_conn_id& x) const {
        return src_addr == x.src_addr
            && dst_addr == x.dst_addr
            && src_port == x.src_port
            && dst_port == x.dst_port;
    }
};
```

- Hash different flows (different channels), thus sending different packets into different threads (improve CPU locality)
- One core per flow

Components of OSv Network

- Lock contention can be largely avoided because kernel now does not process heavy logic
- Only one thread is processing packets (socket sending & receiving buffers), thus locks can be merged.
- TCP layer lock can be combined since TCP processing is part of socket processing context



Components of OSv Network

- Linked list problem is replaced with channels.
- A channel is just a fixed-sized ring buffer queue, thus cache friendly

```
class net_channel {
private:
    std::function<void (mbuf*)> _process_packet;
    ring_spsc<mbuf*, 256> _queue;
    sched::thread_handle _waiting_thread CACHELINE_ALIGNED;
    // extra list of threads to wake
    osv::rcu_ptr<std::vector<pollreq*>> _pollers;
    osv::rcu_hashtable<epoll_ptr> _epollers;
    mutex _pollers_mutex;
    ...
    // producer: try to push a packet
    bool push(mbuf* m) { return _queue.push(m); }
    // consumer: wake the consumer (best used after multiple push(s))
    void wake() {
        _waiting_thread.wake();
        if (_pollers || !_epollers.empty()) {
            wake_pollers();
        }
    }
    // consumer: consume all available packets using process_packet()
    void process_queue();
    ...
}
```

General answer to questions

- Performance

- Good when dealing tasks that has frequent Linux syscalls & heavy network loading
 - Remove system call & single address model
 - Network channel strategy
- Memcached benchmark shows a 20% more requests per second capability compared with Linux

General answer to questions

- Performance
 - Not good when dealing with heavy disk I/O operations application
 - Coarse-grained locking in VFS could lock vnode for a long time
 - Not good for application like MySQL

General answer to questions

- Security
 - Single address space arises some security issues
 - Extra hardening has been provided by the developers
 - Musl macro hardening
 - Page guard
 - But who cares? Each application should have its own instance of operating system running under different protection domain provided by virtualization facilities running beneath (100% trust).
 - OSv relies on the security provided by the underlying facilities and mainly focus and the performance gain from single address space. This is a trade off.

General answer to questions

- Optimization
 - Single memory space
 - Give up spin-lock
 - New scheduler
 - JVM optimization

Thoughts about OSv

- Linux is everywhere, thus practical & useful
- Applications can be adopted quickly
- SaaS (software as a service) becoming much more popular than before. Along with the trend, thanks to hardware acceleration, virtualization and cloud based services became the dominant type of services provided on the internet.

Thank you