Library Operating System with Mainline Linux Network Stack

Hajime Tazaki, Ryo Nakamura, Yuji Sekiya
netdev0.1, Feb. 2015
Motivation

Why kernel space?
- Packets were expensive in 1970’s

Why not userspace?
- Well grown in decades, costs degrades
- Obtain network stack **personalization**
- **Controllable** by userspace utilities
Userspace network stacks

- A lot of userspace network stack
  - full scratch: mTCP, Mirage, lwIP
  - Porting: OSv, Sandstorm, libuinet (FreeBSD), Arrakis (lwIP), OpenOnload (lwIP?)

- Motivated by their own problems (specialized NIC, cloud, high-speed Apps)

- Writing a network stack is 1-week DIY,
  - but writing opera-table network stack is decades DIY (which is not DIY)
Questions

- How to benefit matured network stack in userspace?
- How to trivially introduce your idea on network stack?
- xxTCP, IPvX, etc..
- How to flexibly test your code with a complex scenario?
The answers

- Using Linux network stack *as-is*
- as a *userspace Library* (library operating system)
This talk is about

- an introduction of a library operating system for Linux
- and its implementation
- with a couple of useful use cases
Outlook (design)

- hardware-independent arch (arch/lib)
- 3 components
  - Host backend layer
  - Kernel layer
  - POSIX layer

https://github.com/libos-nuse/net-next-nuse
Outlook (cont’d)

4) applications magically runs
3) add POSIX glue code

1) Build Linux srctree w/ glues as a library

2) put backend (vNIC, clock source, scheduler) and bind
void schedule(void)
{
    lib_task_wait();
}

signed long schedule_timeout(signed long timeout)
{
    u64 ns;
    struct SimTask *self;

    if (timeout == MAX_SCHEDULE_TIMEOUT) {
        lib_task_wait();
        return MAX_SCHEDULE_TIMEOUT;
    }
    lib_assert(timeout >= 0);
    ns = ((_u64)timeout) * (10000000000 / HZ);
    self = lib_task_current();
    lib_event_schedule_ns(ns, &trampoline, self);
    lib_task_wait();
    /* we know that we are always perfectly on time. */
    return 0;
}
POSIX glue code

```c
int nuse_socket(int domain, int type, int protocol)
{
    lib_update_jiffies();
    struct socket *kernel_socket = malloc(sizeof(struct socket));
    int ret, real_fd;

    memset(kernel_socket, 0, sizeof(struct socket));
    ret = lib_sock_socket(domain, type, protocol, &kernel_socket);
    if (ret < 0)
        errno = -ret;
    (snip)
        lib_softirq_wakeup();
    return real_fd;
}
weak_alias(nuse_socket, socket);
```
Implementations (Instances)

- Direct Code Execution (DCE)
  - network simulator integration (ns-3)
  - for more testing

- Network Stack in Userspace (NUSE)
  - gives new platform of Linux network stack
  - for ad-hoc network stack
Direct Code Execution

- ns-3 integration
- deterministic scheduler
- single-process model virtualization
- dlmopen(3)-like virtualization
- full control over multiple network stacks
Execution (DCE)

- `main() => dlmopen(ping, liblinux.so)
  => main() => socket(2) => dce_socket()
  => (do whatever)
Network Stack in Userspace

- Userspace network stack running on Linux (POSIX) platform

- Network stack personalization

- Full features by design (full stack)

  - ARP/ND, UDP/TCP (all cc algorithm), SCTP, DCCP, QDISC, XFRM, netfilter, etc.
Execution (NUSE)

- LD_PRELOAD=libnuse-linux.so \ ping www.google.com

- ping(8) => socket(2) => nuse_socket() => raw(7) => (network)
When it’s useful?

- ad-hoc network stack (network stack personalization)
  - LD_PRELOAD=liblinux-mptcp.so firefox
- Bundle with kernel bypasses
  - Intel DPDK / netmap / PF_RING / etc.
- debugging/testing with ns-3
Testing workflow

1. Write/modify code (patches)

2. Write a test code (incl. packet exchanges)

3. if PASS; accept pull-request else; rejects
continuous integration (CI)

http://ns-3-dce.cloud.wide.ad.jp/jenkins/job/daily-net-next-sim/
T1) write a patch

Fixes: de3b7a06df1e1 ("xfrm6: Fix transport header offset in _decode_session6.")
Signed-off-by: Hajime Tazaki <tazaki@sfc.wide.ad.jp>

---
net/ipv6/xfrm6_policy.c | 1 +
1 file changed, 1 insertion(+)
diff --git a/net/ipv6/xfrm6_policy.c b/net/ipv6/xfrm6_policy.c
index 48bf5a0..8d2d01b 100644
--- a/net/ipv6/xfrm6_policy.c
+++ b/net/ipv6/xfrm6_policy.c
@@ -200,6 +200,7 @@ _decode_session6(struct sk_buff *skb, struct flowi *fl, int reverse)
 #if IS_ENABLED(CONFIG_IPV6_MIP6)
   case IPPROTO_MH:
     offset += ipv6_optlen(exthdr);
+    if (!onlyproto && pskb_may_pull(skb, nh + offset + 3 - skb->data)) {
       struct ip6_mh *mh;
   }

http://patchwork.ozlabs.org/patch/436351/
T2) write a test

- As ns-3 scenario
- C++ or python
- create a topology
- config nodes
- run/check results (e.g., ping6)

http://code.nsnam.org/thewajime/ns-3-dce-umip/file/tip/test/dce-umip-test.cc
#!/usr/bin/python

from ns.dce import *
from ns.core import *

nodes = NodeContainer()
nodes.Create (100)
dce = DceManagerHelper()
dce.SetNetworkStack ("liblinux.so")
dce.Install (nodes)

app = DceApplicationHelper()
app.SetBinary ("ping6")
app.Install (nodes)

NS_TEST_ASSERT_MSG_EQ (m_pingStatus, true, "Ump test " << m_testname << " did not return successfully: " << g_testError)

Simulator.Stop (Seconds(1000.0))
Simulator.Run ()
Performance of NUSE

- 10G Ethernet back-to-back transmission
- IP forwarding
- native Linux, raw socket, tap, dpdk, netmap
Performance: setup

<table>
<thead>
<tr>
<th></th>
<th>NUSE node</th>
<th>Tx/Rx nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Xeon E5-2650v2 @ 2.60GHz</td>
<td>Xeon L3426 @ 1.87GHz</td>
</tr>
<tr>
<td></td>
<td>(16 core)</td>
<td>(8 core)</td>
</tr>
<tr>
<td>Memory</td>
<td>32GB</td>
<td>4GB</td>
</tr>
<tr>
<td>NIC</td>
<td>Intel X520</td>
<td>Intel X520</td>
</tr>
<tr>
<td>OS</td>
<td>host:3.13.0-32</td>
<td>host:3.13.0-32</td>
</tr>
<tr>
<td></td>
<td>nuse: 3.17.0-rc1</td>
<td></td>
</tr>
</tbody>
</table>

ping
flowgen

vnstat
(packet count)
Host Tx

ping (RTT)
native: ping A.B.C.D
others: ./nuse ping A.B.C.D
L3 Routing
Sender->NUSE->Receiver

ping (RTT)

Throughput (Mbps)

throughput (1024byte, UDP)
Alternatives

- UML/LKL (1proc/1vm, no POSIX i/f)
- Containers (can’t change kernel)
- scratch-based (mTCP,Mirage)
- rumpkernel (in NetBSD)
Limitations

- ad-hoc kernel glues required
  - when we changed a member of a struct, LibOS needs to follow it

- Performance drawbacks on NUSE
  - adapt known techniques (mTCP)
(not) Conclusions

- An abstraction for multiple benefits
- Conservative
  - Use past decades effort as much
  - with a small amount of effort
- Planing to RFC for upstreaming
- github: https://github.com/libos-nuse/net-next-nuse
- twitter: @thehajime
Backups
Bug reproducibility

(gdb) b mip6_mh_filter if dce_debug_nodeid()==0
Breakpoint 1 at 0x7ffff287c569: file net/ipv6/mip6.c, line 88.
<continue>
(gdb) bt 4
#0 mip6_mh_filter
   (sk=0x7ffff7f69e10, skb=0x7ffff7cde8b0)
at net/ipv6/mip6.c:109
#1 0x00007fffff2831418 in ipv6_raw_deliver
   (skb=0x7ffff7cde8b0, nexthdr=135)
at net/ipv6/raw.c:199
#2 0x00007fffff2831697 in raw6_local_deliver
   (skb=0x7ffff7cde8b0, nexthdr=135)
at net/ipv6/raw.c:232
#3 0x00007fffff27e6068 in ip6_input_finish
   (skb=0x7ffff7cde8b0)
at net/ipv6/ip6_input.c:197
Debugging

- Memory error detection
- among distributed nodes
- in a single process
- using valgrind
Fine-grained parameter coverage

Code coverage measurement with DCE
With fine-grained network, node, protocol parameters
1) kernel build

- build kernel source tree w/ the patch
  - make menuconfig ARCH=sim
  - make library ARCH=sim

→ libnuse-linux-3.17-rc1.so
Example: How timer works

- `add_timer()`
- `timer_list`
- `run_timer_softirq()`
- `timer handler`
- `TIMER_SOFTIRQ`
- `timer thread` (`timer_create(2)`)

```c
add_timer()
```

```c
run_timer_softirq()
```
Tx callgraph

sendmsg ()
lib_sock_sendmsg ()
sock_sendmsg ()
ip_send_skb ()
ip_finish_output2 ()
dst_neigh_output ()
eigh_resolve_output ()
arp_solicit ()
dev_queue_xmit ()
lib_dev_xmit ()
nuse_vif_raw_write ()

(socket API)
(NUSE)
(existing -kernel)
(NUSE)
Rx callgraph

vNIC rx

start_thread ()
nuse_netdev_rx_trampoline ()
  nuse_vif_raw_read ()
  lib_dev_rx ()
  netif_rx ()

softirq rx

start_thread ()
do_softirq ()
  net_rx_action ()
  process_backlog ()
  __netif_receive_skb_core ()
  ip_rcv ()

(pthread)
(NUSE)
(ex-kernel)