Using Slackware for Linux Kernel development

So, hello from lockywolf.

At some point I needed to hack some network-related kernel code. Since it is not 1993, not 2003, and not even 2013, I, naturally, wanted to do “fast development using QEMU”, instead of painstakingly booting two machines with new kernels (it's network code, after all, so there needs to be a network of at least two machines).

So I have hacked a script for doing that, which I am quite pleased with. However, it is not very generic (as, I guess, most developer scripts are), I can't really publish it as a “ready to use” script on gitlab.

So I decided to put it here, not as a usable script, but as an example of something to be used as a basis for your own scripts.

Before you start, you should decide for yourself the following things:

1. where you keep the kernel tree (called /home/username/path/to/net-next-tree/net-next/ on this page)
2. where you keep iproute2 tools (called /home/username/path/to/iproute2-tree/iproute2/ on this page)
3. what userspace software you need to test your kernel (called /whatever/userspace/software/you/need here)
4. which module you want to test (called mymodule here (whithout .ko !))
5. which packages you want to have in your VMs

Below I am using the following packages from stock slackware 15.0 (kept in /var/cache/packages/):

1. openssh-9.2p1-x86_64-1_slack15.0.txz
2. openssl-solibs-1.1.1t-x86_64-1_slack15.0.txz
3. libevent-2.1.12-x86_64-3.txz
4. ncurses-6.3-x86_64-1.txz
5. tmux-3.2a-x86_64-1.txz
6. libmnl-1.0.4-x86_64-5.txz
7. iproute2-5.16.0-x86_64-1.txz
8. screen-4.9.0-x86_64-1.txz
9. libnl3-3.5.0-x86_64-3.txz
10. tcpdump-4.99.1-x86_64-1.txz

And the following packages built from SlackBuilds.org (kept in /var/sbopkg-built-packages):

1. dropbear-2022.83-x86_64-1_SBo.tgz
2. lksctp-tools-1.0.17-x86_64-3_SBo.tgz
3. iperf3-3.12-x86_64-2_SBo.tgz
4. wireshark-4.0.4-x86_64-1_SBo.tgz
5. libpcap-1.10.3-x86_64-1.txz
6. socat-noopenssl-1.7.4.4-x86_64-1.txz
7. netsniff-ng-0.6.8-x86_64-1_SBo.tgz
8. jq-1.6-x86_64-2_SBo.tgz

The VMs do not have any fake HDD, they are booting into their initrds, which are created using Slackware's standard mkinitrd.

The packages are installpkg'ed into those initrds, thanks to Slackware's excellent package system.

The init of that initrd is busybox, which is fine, as we do not need complicated process managent.

A sed line in that script is replacing the last action of the default init (chrooting onto a real FS) by launching tmux and initing the testing code.

Terminal control, process control, and such is done purely by tmux, without any getties, logins, and such. Key sequences mostly work. Network namespaces work.

The machines can be rebooted by abusing the sysrq trigger, give that you have it compiled in in your kernel config.

The ssh server is dropbear, the client is openssh; both machines can log into each other.

The code below is org-mode-babel code, usable from emacs. C-c C-v C-b evaluated both of them, but they can be tested independently. The first block creates the VM images, the second one creates the tap interface on the host, and boots two QEMUs.

```
* mkinitrd-and-init

#+begin_src shell :exports both :results output
export DEVKERNEL=$(grep UTS_RELEASE ./include/generated/utsrelease.h | awk
'{print $3;}' | tr -d '"')
cd /home/username/path/to/iproute2-tree/iproute2/
make -j4
make install DESTDIR=/tmp/iproute2/

cd /home/username/path/to/net-next-tree/net-next/
make -j4
sudo make modules_install

sudo -E bash -s <<"EOF"
set -x
/sbin/mkinitrd -c -k 5.15.94:$DEVKERNEL -m e1000:yourmodule || exit 1
mkdir -p /boot/initrd-tree/tmp
mkdir -p /boot/initrd-tree/tmp/tmux
chmod 700 /tmp/tmux
cp /usr/bin/ldd /boot/initrd-tree/bin/
mkdir -p /boot/initrd-tree/etc/dropbear
mkdir -p /boot/initrd-tree/root/.ssh/
dropbearkey -t ed25519 -f /boot/initrd-
tree/etc/dropbear/dropbear_ed25519_host_key
dropbearconvert dropbear openssh /boot/initrd-
tree/etc/dropbear/dropbear_ed25519_host_key /boot/initrd-
tree/etc/dropbear/dropbear_ed25519_host_key.openssh
printf '%s %s ' 127.0.0.1 ssh-ed25519 >> /boot/initrd-
tree/root/.ssh/known_hosts
ssh-keygen -e -f /boot/initrd-
```

```
tree/etc/dropbear/dropbear_ed25519_host_key.openssh -e  | head -n 3 | tail -
n 1  >> /boot/initrd-tree/root/.ssh/known_hosts
printf '%s %s ' 192.168.100.1 ssh-ed25519 >> /boot/initrd-
tree/root/.ssh/known_hosts
ssh-keygen -e -f /boot/initrd-
tree/etc/dropbear/dropbear_ed25519_host_key.openssh -e  | head -n 3 | tail -
n 1  >> /boot/initrd-tree/root/.ssh/known_hosts
printf '%s %s ' 192.168.100.2 ssh-ed25519 >> /boot/initrd-
tree/root/.ssh/known_hosts
ssh-keygen -e -f /boot/initrd-
tree/etc/dropbear/dropbear_ed25519_host_key.openssh -e  | head -n 3 | tail -
n 1  >> /boot/initrd-tree/root/.ssh/known_hosts

cp /boot/initrd-tree/etc/dropbear/dropbear_ed25519_host_key.openssh
/boot/initrd-tree/root/.ssh/id_ed25519

printf '%s ' ssh-ed25519 >> /boot/initrd-tree/root/.ssh/id_ed25519.pub
ssh-keygen -e -f /boot/initrd-tree/root/.ssh/id_ed25519 -e  | head -n 3 |
tail -n 1  >> /boot/initrd-tree/root/.ssh/id_ed25519.pub

printf '%s %s initrd-key' ssh-ed25519  "$(ssh-keygen -e -f /boot/initrd-
tree/root/.ssh/id_ed25519 -e  | head -n 3 | tail -n 1)">>/boot/initrd-
tree/root/.ssh/authorized_keys

/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/dropbear-2022.83-x86_64-1_SBo.tgz || printf "install dropbear
failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/patches/packages/openssh-9.2p1-x86_64-1_slack15.0.txz
|| printf "install openssh failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/patches/packages/openssl-solibs-1.1.1t-
x86_64-1_slack15.0.txz  || printf "install openssl-solibs failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/slackware64/l/libevent-2.1.12-x86_64-3.txz  || printf
"install libevent failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/slackware64/l/ncurses-6.3-x86_64-1.txz  || printf
"install ncurses failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/slackware64/ap/tmux-3.2a-x86_64-1.txz  || printf
"install tmux failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/slackware64/n/libmnl-1.0.4-x86_64-5.txz   || printf
"install libmnl failed\n"
#/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/./slackware64/n/iproute2-5.16.0-x86_64-1.txz || printf
"install iproute2 failed\n"
( cd /home/username/path/to/iproute2-tree/iproute2/ ; make install
DESTDIR=/boot/initrd-tree)
/sbin/installpkg --root /boot/initrd-tree/
```

```
/var/cache/packages/slackware64/ap/screen-4.9.0-x86_64-1.txz || printf
"install screen failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/lksctp-tools-1.0.17-x86_64-3_SBo.tgz || printf "install sctp
failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/iperf3-3.12-x86_64-2_SBo.tgz || printf "install iperf3 failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/wireshark-4.0.4-x86_64-1_SBo.tgz || printf "install wireshark
failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/./slackware64/l/libnl3-3.5.0-x86_64-3.txz || printf
"install nl3 failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/libpcap-1.10.3-x86_64-1.txz || printf "install libpcap failed\n"
/sbin/installpkg --root /boot/initrd-tree/
/var/cache/packages/./slackware64/n/tcpdump-4.99.1-x86_64-1.txz  || printf
"install tcpdump failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-packages/socat-
noopenssl-1.7.4.4-x86_64-1.txz || printf "install socat failed\n"
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/netsniff-ng-0.6.8-x86_64-1_SBo.tgz
/sbin/installpkg --root /boot/initrd-tree/ /var/sbopkg-built-
packages/jq-1.6-x86_64-2_SBo.tgz
mkdir -p /boot/initrd-tree/usr/lib64/locale
cp -r /usr/lib64/locale/C.utf8 /usr/lib64/locale/en_US* /boot/initrd-
tree/usr/lib64/locale/

mkdir -p /boot/initrd-tree/usr/lib/locale
cp -r /usr/lib/locale/C.utf8 /usr/lib64/locale/en_US* /boot/initrd-
tree/usr/lib/locale/
cp /usr/bin/locale /usr/bin/localedef /boot/initrd-tree/usr/bin/

mkdir -p /boot/initrd-tree/usr/share/i18n/
cp -r /usr/share/i18n/locales /boot/initrd-tree/usr/share/i18n/
mkdir -p /boot/initrd-tree/root/DevLinux

rm /boot/initrd-tree/sbin/poweroff
echo 'echo o > /proc/sysrq-trigger' > /boot/initrd-tree/sbin/poweroff
chmod +x /boot/initrd-tree/sbin/poweroff

rm /boot/initrd-tree/sbin/reboot
echo 'echo b > /proc/sysrq-trigger' > /boot/initrd-tree/sbin/reboot
chmod +x /boot/initrd-tree/sbin/reboot

ln -s '../../bin/sh' '/boot/initrd-tree/usr/bin/bash'

echo '/bin/bash' > /boot/initrd-tree/etc/shells
mkdir -p /boot/initrd-tree/root/DevLinux/
cp -r /whatever/userspace/software/you/need /boot/initrd-tree/root/DevLinux/
touch /boot/initrd-tree/root/my_setup.sh
```

```
EOF
cat  > /tmp/qemu_script.sh <<"EOF"
set -x
ip link set up lo
ip addr add 127.0.0.1 dev lo
IFNO=$(ip a show dev eth0 | grep ether | awk '{print $2;}' | awk -F: '{print
$6;}' | cut -c 2)
ip link set up eth0
ip addr add 192.168.100.$IFNO/24 dev eth0
dropbear
cd /root/
mkdir -p /var/run/netns/
sleep 1
tmux new-session -s test_session -d
tmux new-window -t test_session:1
tmux send-keys -t test_session:0 'export
PATH="/bin/:/sbin/:/usr/bin/:/usr/sbin/:$PATH"' Enter
tmux send-keys -t test_session:1 'export
PATH="/bin/:/sbin/:/usr/bin/:/usr/sbin/:$PATH"' Enter

tmux new-window -t test_session:2
tmux send-keys -t test_session:2 'export
PATH="/bin/:/sbin/:/usr/bin/:/usr/sbin/:$PATH"' Enter 'cd /root' Enter
./my_testscript.sh Enter

if   [[ "$IFNO" == 1 ]] ; then
printf 'client\n'
tmux send-keys -t test_session:1 C-c C-c Enter 'cd /root/DevLinux/' Enter
'./run-my-userspace-code-for-machine1.sh' Enter
tmux send-keys -t test_session:0 C-c C-c Enter 'my iproute commands, such as
ip on machine 1' Enter
tmux attach
elif [[ "$IFNO" == 2 ]] ; then
printf 'server\n'
tmux send-keys -t test_session:1 C-c C-c Enter 'cd /root/DevLinux/' Enter
'./run-my-userspace-code-for-machine1.sh' Enter
tmux send-keys -t test_session:0 C-c C-c Enter 'my iproute commands, such as
ip on machine 2' Enter
tmux attach
else
printf 'Error\n'
fi
EOF
sudo cp /tmp/qemu_script.sh /boot/initrd-tree/root/my_setup.sh
sudo -E bash -s <<"EOF"
chmod +x /boot/initrd-tree/root/my_setup.sh
echo 'modprobe -v e1000' > /boot/initrd-tree/load_kernel_modules
sed -i '/Switch to real root partition/a  mkdir -p /dev/pts ; mount -t
devpts devpts /dev/pts/' /boot/initrd-tree/init
sed -i '/mount -t devpts devpts/a  /root/my_setup.sh' /boot/initrd-tree/init
/sbin/mkinitrd -k 5.15.94:$DEVKERNEL -m e1000:mymodule
```

```
EOF

#+end_src


* boot

** tuntap

#+begin_src shell :exports both :results output
export DEVKERNEL=$(grep UTS_RELEASE ./include/generated/utsrelease.h | awk
'{print $3;}' | tr -d '"')
export DEVKERNELPATH="/home/username/path/to/net-next-tree/net-
next/arch/x86/boot/bzImage"
export WMI="$(whoami)"
sudo -E bash -s <<EOF
/sbin/ip link add bridge_for_qemu type bridge
/sbin/ip tuntap add dev tap_for_qemu1 mode tap user $WMI
/sbin/ip link set tap_for_qemu1 master bridge_for_qemu
/sbin/ip tuntap add dev tap_for_qemu2 mode tap user $WMI
/sbin/ip link set tap_for_qemu2 master bridge_for_qemu
EOF

qemu-kvm -kernel "$DEVKERNELPATH" -initrd /boot/initrd.gz -device
e1000,netdev=network0,mac=52:55:00:d1:55:01 -netdev
tap,id=network0,ifname=tap_for_qemu1,script=no,downscript=no -m 512M -append
'ip=eth0:dhcp' &
qemu-kvm -kernel "$DEVKERNELPATH" -initrd /boot/initrd.gz -device
e1000,netdev=network0,mac=52:55:00:d1:55:02 -netdev
tap,id=network0,ifname=tap_for_qemu2,script=no,downscript=no -m 512M -append
'ip=eth0:dhcp' &
disown
sleep 1
sudo -E bash -s <<EOF
/sbin/ip link set up tap_for_qemu1
/sbin/ip link set up tap_for_qemu2
/sbin/ip link set up bridge_for_qemu
EOF

#+end_src

#+RESULTS:
```

Sources

1. Kernel development with qemu
   https://blog.elastocloud.org/2015/06/rapid-linux-kernel-devtest-with-qemu.html
2. Using tap devices for communicating with virtual machines

https://blog.elastocloud.org/2015/07/qemukvm-bridged-network-with-tap.html