Course Authored By:

Srinivasan Kannan
(also known as: Shrinivaasan Kannan, Shrinivas Kannan)
Ph: 9791499106, 9003082186
Krishna iResearch Open Source Products Profiles:
http://sourceforge.net/users/ka_shrinivaasan,
https://github.com/shrinivaasanka,
https://www.openhub.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

##
##

This is a non-linearly organized, continually updated set of course notes on Linux Kernel and Cloud
and supplements NeuronRain USBmd, VIRGO Linux and KingCobra Design Notes in:

NeuronRain Enterprise Version Design Documents:

USBmd USB and WiFi network analytics - https://github.com/shrinivaasanka/usb-md-github-code/blob/master/USBmd_notes.txt

VIRGO Linux - https://github.com/shrinivaasanka/virgo-linux-github-code/blob/master/virgo-docs/VirgoDesign.txt

KingCobra Kernelspace Messaging - https://github.com/shrinivaasanka/kingcobra-github-code/blob/master/KingCobraDesignNotes.txt
----
NeuronRain Research Version Design Documents:

USBmd USB and WiFi network analytics - https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt

VIRGO Linux - https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VirgoDesign.txt

KingCobra Kernelspace Messaging - https://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt
----
----
17 March 2017

VIRGO Linux Kernel Build Steps

VIRGO Linux kernel is an overlay of VIRGO codebase on kernel mainline (presently 4.1.5) source tree. Building a custom kernel for VIRGO is required for building new system calls and kernel modules in it. Shell script for builing kernel mainline is in:

https://github.com/shrinivaasanka/virgo-linux-github-code/blob/master/buildscript_4.1.5.sh

https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/buildscript_4.1.5.sh

VIRGO Linux kernel has following new system calls:

Cloud RPC system call:
    virgo_clone()
Cloud Kernel Memory Cache (kernelspace equivalent of memcache):
    virgo_malloc()
    virgo_get()
    virgo_set()
    virgo_free()
Cloud File System:
    virgo_open()
    virgo_close()
    virgo_read()
    virgo_write()

VIRGO Linux kernel has following new kernel modules (kernelsocket listeners) corresponding to previous system call clients:
----

1. cpupooling virtualization - VIRGO_clone() system call and VIRGO cpupooling driver by which a remote procedure can be invoked in kernelspace.(port: 10000)
2. memorypooling virtualization - VIRGO_malloc(), VIRGO_get(), VIRGO_set(), VIRGO_free() system calls and VIRGO memorypooling driver by which kernel memory can be allocated in remote node, written to, read and freed - A kernelspace memcache-ing.(port: 30000)
3. filesystem virtualization - VIRGO_open(), VIRGO_read(), VIRGO_write(), VIRGO_close() system calls and VIRGO cloud filesystem driver by which file IO in remote node can be done in kernelspace.(port: 50000)
4. config - VIRGO config driver for configuration symbols export.
5. queueing - VIRGO Queuing driver kernel service for queuing incoming requests, handle them with workqueue and invoke KingCobra service routines in kernelspace. (port: 60000)
6. cloudsync - kernel module for synchronization primitives (Bakery algorithm etc.,) with exported symbols that can be used in other VIRGO cloud modules for critical section lock() and unlock()
7. utils - utility driver that exports miscellaneous kernel functions that can be used across VIRGO Linux kernel
8. EventNet - eventnet kernel driver to vfs_read()/vfs_write() text files for EventNet vertex and edge messages (port: 20000)
9. Kernel_Analytics - kernel module that reads machine-learnt config key-value pairs set in /etc/virgo_kernel_analytics.conf. Any machine learning software can be used to get the key-value pairs for the config. This merges three facets - Machine Learning, Cloud Modules in VIRGO Linux-KingCobra-USBmd , Mainline Linux Kernel
10. SATURN program analysis wrapper driver.
and userspace test cases for the above.

Prerequisites for building VIRGO Linux kernel are similar to mainline kernel:
apt install libncurses5-dev gcc make git exuberant-ctags bc libssl-dev

Presently VIRGO kernel is 32-bit. Building 64 bit linux kernel requires long mode CPU flag (lm) in /proc/cpuinfo. Also booting a 64 bit kernel built on 32 bit kernel could cause init to fail as "init not found". This is because init is a symbolic link to systemd binary when kernel boots up which is 32-bit and not 64-bit. For this menuconfig in Kbuild provides IA32 Emulation config parameter.

Booting a custom kernel could also cause partition issues while grub is updated sometimes with error "no such partition" and could drop to grub rescue> prompt. This can be remedied by:

1. grub rescue> ls

which lists the partitions in format (hd0, msdos<#number>)

2. grub rescue> set root=(hd0, msdos<#number>) {for each such partition}
3. grub rescue> set prefix=(hd0, msdos<#number>)/boot/grub
4. grub rescue> insmod normal
5. grub rescue> normal

which boots grub normally. This has to be persisted with:

6. update-grub2
7. grub-install /dev/sda

on reboot

Some grub errors can be debugged by adding kernel boot parameters in /boot/grub/grub.cfg or at boot time edit of the entry with "...$vt_handoff". Appending options "debug init=<path-to-systemd>" to this line might help.

Sometimes repetitive builds and grub updates also cause a rare file system corruption as below which causes root shell drop:

```
ALERT! <diskid> does not exist
Boot args (cat /proc/cmdline)
- Check rootdelay
- Check root
Missing modules (cat /proc/modules, ls /dev)
Dropping to root shell
(initramfs)
```

This is remedied by:

```
(initramfs) modprobe dm-mod
(initramfs) lvm
lvm> vgchange -ay
lvm> exit
(initramfs) exit
```

followed by:

```
mount -o remount,rw /dev/sda<number> <mount_point_directory>
```

Complete documentation of VIRGO Linux with Design Documents is in:
https://github.com/shrinivaasanka/virgo-linux-github-code/tree/master/virgo-docs/
https://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/

----

20 March 2017

VIRGO Linux 64 bit build - continued:

*) After configuring IA32 simulation and a successful build, there could still be initramfs errors
because of i915 drivers mismatch as below:
"Possible missing firmware --------- for i915"
*) Previous error requires installation of latest linux i915 graphics drivers (though there may not be a matching hardware for it) listed in https://01.org/linuxgraphics/downloads/firmware: for Kabylake, Skylake, Broxton graphics processors
*) update-initramfs has to be done again for correct /boot initrd image to be created for 4.10.3 after updating i915 drivers previously listed
*) In some cases image may be hidden in grub boot menu. For this an already available boot-repair tool scans the /boot images and reinstalls grub menu.
*) On successful 64 bit build, uname -a should have x86_64 as below:
Linux kashrinivaasan-Inspiron-1545 4.10.3 #1 SMP Fri Mar 17 18:30:40 IST 2017 x86_64 x86_64 x86_64 GNU/Linux

----
24 September 2017

NeuronRain VIRGO64 presently has been ported to 4.13.3 Linux Kernel. Also the accompanying USBmd and KingCobra kernel modules have been
split into 32-bit and 64-bit versions and separate repositories have been created for them in SourceForge and GitHub. 4.13.3 kernel has
transport layer security enshrined in kernel which is essential for securing kernelspace cloud traffic. A detailed FAQ on technical aspects of VIRGO 64-bit version and KTLS is in http://neuronrain-documentation.readthedocs.io/en/latest/ .

Logging Servers and Clients - Some Implementation Examples - 28 September 2018

In large cloud installations, there often arises a need to log frequently occuring network event traces to log
files realtime minimizing network latency. Example 1 in references is the Trace utility implemented in an application server (Java) for logging Global and Local JTS Transaction (XA) events with facility for Trace Levels. Example 2 describes sequential and parallel implementations of a logging server in a network and logging clients connecting to this server. Reactor pattern is a Listener on pending socket descriptors (select/poll) and events to be serviced. Example 3 in the references: NeuronRain VIRGO32 and VIRGO64 Linux Kernels implement a kernel service module and logging client utility kernel function for writing EventNet log messages to EventNet Edges and Vertices files sent from remote cloud nodes by eventnet_log() - implements Acceptor-Worker Kernel Threads (Router-Dealer) pattern. Example 4 is the ZeroMQ Publish-Subscribe protocol for designing a log collector subscriber and log client publishers.

References:

1.Oracle Glassfish Java EE - (Earlier Sun Microsystems iPlanet Application Server - IAS) - JTS XA Transaction Logging -

https://github.com/javaee/glassfish/blob/master/appserver/transaction/jts/src/main/java/com/sun/jts/trace and https://github.com/javaee/glassfish/tree/master/appserver/transaction/jts/src/main/java/com/sun/jts/utils - IAS_JTS_TRACE - authors: "mailto:k.venugopal@sun.comi,kannan.srinivasan@sun.com" - "...public static void setTraceWriter(PrintWriter traceWriter) { m_traceWriter = traceWriter; }..." - Wrapper Facade pattern - wraps a writer object
2.Beautiful Code - Chapter 26 - Labor-Saving Architecture - An Object Oriented Framework for Networked Architecture - Concurrent Logging Servers in C++ - Reactor Pattern for Logging Concurrent Events
3.EventNet Kernel Service Module and EventNet Logging Client function in NeuronRain VIRGO Linux Kernel - https://gitlab.com/shrinivaasanka/virgo64-linux-github-code/tree/master/linux-kernel-extensions/drivers/virgo/eventnet and https://gitlab.com/shrinivaasanka/virgo64-linux-github-code/blob/master/linux-kernel-extensions/drivers/virgo/utils/virgo_generic_kernelsock_client.c - eventnet_log()
4.ZeroMQ Pub-Sub for distributed logging - http://zguide.zeromq.org/php:chapter8#toc31

Linux Kernel Development (System calls and Drivers) - some low level architecture specific issues - 14 May 2019

Following are few architecture specific idiosyncracies and heisen bugs that could cause harrowing experience while writing new kernel system calls and drivers (32 bit versus 64 bit, dual core versus quad core versus octa core):

1.strcpy() buffer overflow (kernel has its own implementation of glic string library in include/linux) - there is a documented intel x86_64 buffer overrun error in some chips
2.char * to const char* cast requirement
3. (u8*) cast for sin_addr in kernel sockets
4. memcpy() versus copy_to_user() or copy_from_user() - in some architectures memcpy() works while in others copy_xxx() pairs work without crashes
5. Requirement for __user qualifier macro for some system call parameters passed to copy_from_user() and copy_to_user()
6. Correct location of kernel glibc string library headers in Makefile paths
7. strscpy() instead of strcpy() - works some times
8. Usage of BUF_SIZE (buffer size as macro) instead of strlen()
9. kstrdup() might crash sometimes in strlen() which can be replaced by strcpy() and strcat()
10. User memory access warning/error in KASAN because of copy_from_user() and copy_to_user() (access_ok() assertions crash)
11. memcpy() is less secure than copy_xxx() pairs
12. __put_user() and strncpy_from_user() might sometimes circumvent crashes
13. Setting the segment correctly - get_fs()/set_ds(KERNEL_DS) must correctly encapsulate kernel data access.
14. copy_user_generic() avoids crashes in some cases if previous do not work
15. cast char to unsigned long long which bypasses lot of buffer issues and __put_user() supports only unsigned long long
16. unsigned long long has the advantage of abstracting any datatype.

17. Being compatible for both 32 and 64 bits simultaneously could be a problem - u8 and const char* cast
18. memcpy() could fail in certain multicores
19. strncpy_from_user() in place of copy_from_user() creates stabler kernel syscalls and driver builds
20. __put_user() in place of copy_to_user() is stabler in some multicores
21. unsigned long long is sometimes stabler than char __user*
22. sizeof() might have to be replaced by BUF_SIZE macro for all copy functions
23. in4_pton() might fail in some cases while in_aton() might work

----

Linux Kernel Development (System calls and Drivers) - some low level architecture specific issues 2 - 10 June 2019

Following are some architecture specific problems while invoking kernel sockets (e.g simulating a telnet client and server within
kernel):

1. System calls (especially written new) often involve userspace strings passed on to kernel (and sometimes transported to a remote
kernel by kernel sockets) which are mandated to be marked as "const __user char*"
2. __user pointer marked as previous informs the kernel that a userspace data has to be handled by kernel by copying it to kernelspace
3. Accessing __user pointers directly within system calls and kernel modules (e.g in printk) creates a fault because a userspace
pointer is accessed in kernel space illegitimately.
4. Special functions like copy_from_user()/strncpy_from_user() are meant for copying userspace __user pointed data to kernelspace
pointers.
5. This userspace to kernelspace copy requires setting kernel data segments appropriately by set_ds(KERNEL_DS) followed by the invocation and
resetting the datasegment to status quo ante.
6. Buffers used within the kernel modules and system calls for copying user buffers must have sufficient size and strlen() may not
work for strings which are not null terminated causing overflow.
7. copy_to_user() causes weird faults and there is no equivalent strncpy_to_user() for out parameters in system calls which might necessitate __put_user() to a generic data type e.g unsigned long long and reinterpret-cast to a char literal. Sequential invocation
of these creates an array of chars or strings.
8. -32, -107, -101 errors are often witnessed in kernel_connect() which is probably caused by in4_pton()