

```

/*****
#-----
#NEURONRAIN VIRGO - Cloud, Machine Learning and Queue augmented Linux Kernel Fork-off
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#You should have received a copy of the GNU General Public License
#along with this program. If not, see <http://www.gnu.org/licenses/>.
#-----
#Copyleft (Copyright+):
#Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) 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
#-----
*****/

```

VIRGO is an operating system kernel forked off from Linux kernel mainline to add cloud functionalities (system calls, modules etc.,) within kernel itself with machine learning, analytics, debugging, queueing support in the deepest layer of OSI stack i.e AsFer, USBmd, KingCobra together with VIRGO constitute the previous functionalities. Presently there seems to be no cloud implementation with fine-grained cloud primitives (system calls, modules etc.,) included in kernel itself though there are coarse grained clustering and SunRPC implementations available. VIRGO complements other Clustering and application layer cloud OSes like cloudstack, openstack etc., in these aspects - CloudStack and OpenStack can be deployed on a VIRGO Linux Kernel Cloud - OpenStack nova compute, neutron network, cinder/swift storage subsystems can be augmented to have additional drivers that invoke lowlevel VIRGO syscall and kernel module primitives (assuming there are no coincidental replications of functionalities) thereby acting as a foundation to application layer cloud.

Remote Device Invocation , which is an old terminology for Internet-Of-Things has already been experimented in SunRPC and KOrbit CORBA-in-linux-kernel kernel modules in old linux kernels (<http://www.csn.ul.ie/~mark/fyp/fypfinal.html> - with Solaris MC and example Remote Device Client-Server Module implementation). VIRGO Linux with the larger encompassing NeuronRain suite is an effort to provide a unified end-to-end application-to-kernel machine-learning propelled cloud and internet-of-things framework.

Memory pooling:

Memory pooling is proposed to be implemented by a new `virgo_malloc()` system call that transparently allocates a block of virtual memory from memory pooled from virtual memory scattered across individual machines part of the cloud.

CPU pooling or cloud ability in a system call:

`Clone()` system call is linux specific and internally it invokes `sys_clone()`. All `fork()`, `vfork()` and `clone()` system calls internally invoke `do_fork()`. A new system call `virgo_clone()` is proposed to create a thread transparently on any of the available machines on the cloud. This creates a thread on a free or least-loaded machine on the cloud and returns the results.

`virgo_clone()` is a wrapper over `clone()` that looks up a map of machines-to-loadfactor and get the host with least load and invokes `clone()` on a function on that gets executed on the host. Usual cloud implementations provide userspace API that have something similar to this - `call(function, host)`. Loadfactor can be calculated through any of the prominent loadbalancing algorithm. Any example userspace code that uses `clone()` can be replaced with `virgo_clone()` and all such threads will be running in a cloud transparently. Presently Native POSIX threads library (NPTL) and older LinuxThreads thread libraries internally use `clone()`.

Kernel has support for kernel space sockets with `kernel_accept()`, `kernel_bind()`, `kernel_connect()`, `kernel_sendmsg()` and `kernel_recvmsg()` that can be used inside a kernel module. Virgo driver implements `virgo_clone()` system call that does a `kernel_connect()` to a remote kernel socket already `__sock_create()`-d, `kernel_bind()`-ed and `kernel_accept()`-ed and does `kernel_sendmsg()` of the function details and `kernel_recvmsg()` after function has been executed by `clone()` in remote machine. After `kernel_accept()` receives a connection it reads the function and parameter details. Using these `kthread_create()` is executed in the remote machine and results are written back to the originating machine. This is somewhat similar to SunRPC but adapted and made lightweight to suit `virgo_clone()` implementation without any external data representation.

Experimental Prototype

`virgo_clone()` system call and a kernel module `virgocloudexec` which implements Sun RPC interface have been implemented.

VIRGO - loadbalancer to get the host:ip of the least loaded node

Loadbalancer option 1 - Centralized loadbalancer registry that tracks load:

`Virgo_clone()` system call needs to lookup a registry or map of host-to-load and get the least loaded host:ip from it. This requires a load monitoring code to run periodically and update the map. If this registry is located on a single machine then simultaneous `virgo_clone()` calls from many machines on the cloud could choke the registry. Due to this, loadbalancer registry needs to run on a high-end machine. Alternatively, each machine can have its own view of the load and multiple copies of load-to-host registries can be stored in individual machines. Synchronization of the copies becomes a separate task in itself (Cache coherency). Either way gives a tradeoff between accuracy, latency and efficiency.

Many application level userspace load monitoring tools are available but as `virgo_clone()` is in kernel space, it needs to be investigated if kernel-to-kernel loadmonitoring can be done without userspace data transport. Most Cloud API explicitly invoke a function on a host. If this functionality is needed, `virgo_clone()` needs to take host:ip address as extra argument, but it reduces transparent execution.

(Design notes for LB option 1 handwritten by myself are at :<http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf>)

Loadbalancer option 2 - Linux Psuedorandom number generator based load balancer(experimental) instead of centralized registry that tracks load:

Each `virgo_clone()` client has a PRG which is queried (`/dev/random` or `/dev/urandom`) to get the id of the host to send the next `virgo_clone()` function to be executed
Expected number of requests per node is derived as:

expected number of requests per node = summation(each_value_for_the_random_variable_for_number_of_requests * probability_for_each_value) where random variable ranges from 1 to k where N is number of processors and k is the number of requests to be distributed on N nodes

=expected number of requests per node = $(\text{math.pow}(N, k+2) - k \cdot \text{math.pow}(N, 2) + k \cdot \text{math.pow}(N, 1) - 1) / (\text{math.pow}(N, k+3) - 2 \cdot \text{math.pow}(N, k+2) + \text{math.pow}(N, k+1))$

This loadbalancer is dependent on efficacy of the PRG and since each request is uniformly, identically, independently distributed use of PRG would distribute requests evenly. This obviates the need for loadtracking and coherency of the load-to-host table.

(Design notes for LB option 2 handwritten by myself at :<http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf>)

(python script in `virgo-python-src/`)

Implemented VIRGO Linux components (as on 7 March 2016)

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-learned 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. Testcases and kern.log testlogs for the above
11. SATURN program analysis wrapper driver.

Thus VIRGO Linux at present implements a minimum cloud OS (with cloud-wide cpu, memory and file system management) over Linux and potentially fills in a gap to integrate both software and hardware into cloud with machine learning and analytics abilities that is absent in application layer cloud implementations. Thus VIRGO cloud is an IoT operating system kernel too that enables any hardware to be remote controlled. Data analytics using AsFer can be done by just invoking requisite code from a kernelspace driver above and creating an updated driver binary (or) by kernel_analytics module which reads the userland machine-learned config.

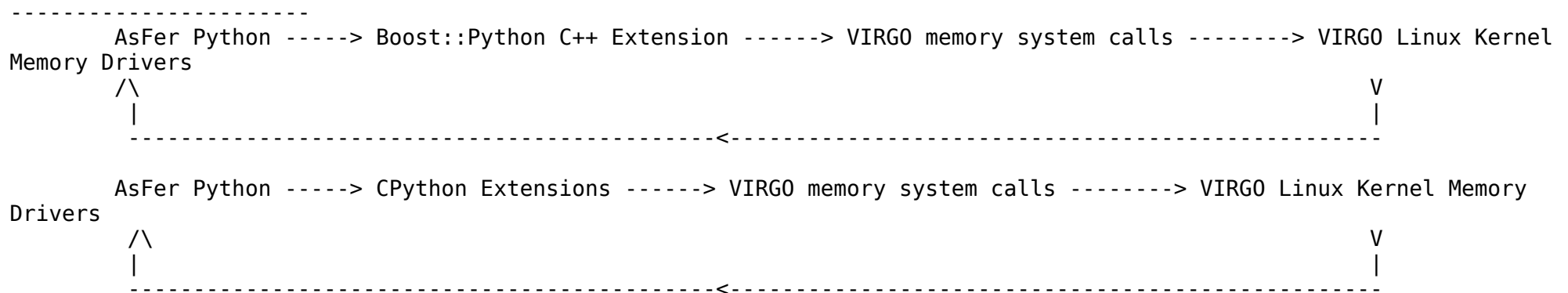
VIRGO ToDo and NiceToHave Features (list is quite dynamic and might be rewritten depending on feasibility - longterm with no deadline)

(FEATURE - DONE-minimum separate config file support in client and kernel service)1. More Sophisticated VIRGO config file and read_virgo_config() has to be invoked on syscall clients virgo_clone and virgo_malloc also. Earlier config was being read by kernel module only which would work only on a single machine. A separate config module kernel service has been added for future use while exporting kernel-wide configuration related symbols. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel. Each VIRGO module and system call now reads the config file independent of others - there is a read_virgo_config<module><client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.(- from commit comments done few months ago)

(FEATURE - DONE) 3. `Virgo_malloc()`, `virgo_set()`, `virgo_get()` and `virgo_free()` syscalls that virtualize the physical memory across all cloud nodes into a single logical memory behemoth (NUMA visavis UMA). (There are random crashes in `copy_to_user` and `copy_from_user` in syscall path for VIRGO memory pooling commands that were investigated but turned out to be mystery). These crashes have either been resolved or occur less in 3.15.5 and 4.1.5 kernels.

Initial Design Handwritten notes committed at: http://sourceforge.net/p/virgo-linux/code-0/210/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf

4.1 Schematic Diagram:



(FEATURE - DONE) 6. Sophisticated queuing and persistence of CPU and Memory pooling requests in Kernel Side (by possibly improving already existing kernel workqueues). Either open source implementations like ZeroMQ/ActiveMQ can be used or Queuing implementation has to be written from scratch or both. ActiveMQ supports REST APIs and is JMS implementation. This feature has been marked completed because recently NeuronRain AsFer backend has been updated to support KingCobra REQUEST_REPLY.queue as a datasource for Streaming Abstract Generator. By enabling use_as_kingcobra_service=1 in cpupooling and mempooling VIRGO drivers, any incoming CPU and Memory related request can be routed to KingCobra by linux workqueue in VIRGO queue and disk persisted (/var/log/REQUEST_REPLY.queue) by KingCobra servicerequest recipient. Also Kafka Publisher/Subscriber have been implemented in NeuronRain AsFer which invoke Streaming Abstract Generator with KingCobra REQUEST_REPLY.queue as datasource to publish persisted already received CPU and Memory requests to Kafka Message Queue.

Thus queuing and persistence for VIRGO CPU and Memory is in place. ZeroMQ does not have persistence and is used for NeuronRain client side Router-Dealer concurrent request servicing pattern.

(FEATURE - DONE-Minimum Functionality) 7. Integration of Asfer(AstroInfer) algorithm codes into VIRGO which would add machine learning capabilities into VIRGO - That is, VIRGO cloud subsystem which is part of a linux kernel installation "learns" and "adapts" to the processes that are executed on VIRGO. This catapults the power of the Kernel and Operating System into an artificially (rather approximately naturally) intelligent computing platform (a software "brain"). For example VIRGO can "learn" about "execution times" of processes and suitably act for future processes. PAC Learning of functions could be theoretical basis for this. Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config have been done. This config file virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel is no longer a static predictable blob behemoth. With this, VIRGO is an Internet-of-Things kernel that does analytics and based on analytics variable values integrated hardware can be controlled across the cloud through remote kernel module function invocation. This facility has been made dynamic with Boost::Python C++ and CPython extensions that permit flow of objects from machine learnt AsFer kernel analytics variables to VIRGO Linux Kernel memory drivers via VIRGO system calls directly and back - Commits on 29 January 2016 - this should obviate re-reading /etc/virgo_kernel_analytics.conf and is an exemplary implementation which unifies C++/Python into C/Kernel.

Example scenario 1 without implementation:

- Philips Hue IoT mobile app controlled bulb - <http://www2.meethue.com/en-xx/>
- kernel_analytics module learns key-value pairs from the AsFer code and exports it VIRGO kernel wide
- A driver function with in bulb embedded device driver can be invoked through VIRGO cpupooling (invoked from remote virgo_clone() system_call)

based on if-else clause of the kernel_analytics variable i.e remote_client invokes virgo_clone() with function argument "lights on" which is routed to another cloud node. The recipient cloud node "learns" from AsFer kernel_analytics that Voltage is low or Battery is low from logs and decides to switch in high beam or low beam.

Example scenario 2 without implementation:

- A swivel security camera driver is remotely invoked via virgo_clone() in the VIRGO cloud.
- The camera driver uses a machine learnt variable exported by kernel_analytics-and-AsFer to pan the camera by how much degrees.

Example scenario 3 without implementation - probably one of the best applications of NeuronRain IoT OS:

- Automatic Driverless Automobiles - a VIRGO driver for a vehicle which learns kernel analytics variables (driving directions) set by kernel_analytics driver and AsFer Machine Learning. A naive algorithm for Driverless Car (with some added modifications over A-Star and Motion planning algorithms):
 - AsFer analytics receives obstacle distance data 360+360 degrees (vertical and horizontal) around the vehicle (e.g

ultrasound sensors) which is updated in a Spark DataFrame table with high frequency (100 times per second).

- VIRGO Linux kernel on vehicle has two special drivers for Gear-Clutch-Break-Accelerator-Fuel(GCBAF) and Steering listening on some ports.
- AsFer analytics with high frequency computes threshold variables for applying break, clutch, gear, velocity, direction, fuel changes which are written to kernel_analytics.conf realtime based on distance data from Spark table.
- These analytics variables are continuously read by GCBAF and Steering drivers which autopilot the vehicle.
- Above applies to Fly-by-wire aeronautics too with appropriate changes in analytics variables computed.
- The crucial parameter is the response time in variable computation and table updates which requires a huge computing power unless the vehicle is hooked onto a Spark cloud in motion by wireless which process the table and compute analytic variables.

----- References for Machine Learning + Linux Kernel -----

7.1 KernTune - [http://repository.uwc.ac.za/xmlui/bitstream/handle/10566/53/Yi_KernTune\(2007\).pdf?sequence=3](http://repository.uwc.ac.za/xmlui/bitstream/handle/10566/53/Yi_KernTune(2007).pdf?sequence=3)

7.2 Self-learning, Predictive Systems - <https://icri-ci.technion.ac.il/projects/past-projects/machine-learning-for-architecture-self-learning-predictive-computer-systems/>

7.3 Linux Process Scheduling and Machine Learning - <http://www.cs.ucr.edu/~kishore/papers/tencon.pdf>

7.4 Network Latency and Machine Learning - https://users.soe.ucsc.edu/~slukin/rtt_paper.pdf

7.5 Machine Learning based Meta-Scheduler for Multicore processors - https://books.google.co.in/books?id=1GWcHmCrl0QC&pg=PA528&lpg=PA528&dq=linux+kernel+machine+learning&source=bl&ots=zfJsq_uu5q&sig=mMIUZ-oyJIwZXtYj4HntrQE8NSk&hl=en&sa=X&ved=0CCAQ6AEwATgKahUKEwjs9sqF9vPIAhVBFZQKHbNtA6A

8. A Symmetric Multi Processing subsystem Scheduler that virtualizes all nodes in cloud (probably this would involve improving the loadbalancer into a scheduler with priority queues)

(FEATURE - ONGOING) 9. Virgo is an effort to virtualize the cloud as a single machine - Here cloud is not limited to servers and desktops but also mobile devices that run linux variants like Android, and other Mobile OSes. In the longterm, Virgo may have to be ported or optimized for handheld devices. Boost::Python AsFer-VIRGO system call invocations implemented in NeuronRain is framework for implementing python applications interfacing with kernel. If deployed on Mobile processors (e.g by overlaying Android Kernel with VIRGO layer) there are IDEs like QPython to develop python apps for Android.

(FEATURE - DONE) 10. Memory Pooling Subsystem Driver - Virgo_malloc(), Virgo_set(), Virgo_get() and Virgo_free() system calls and their Kernel Module Implementations. In addition to syscall path, telnet or userspace socket client interface is also provided for both VIRGO CPU pooling(virgo_clone()) and VIRGO Memory Pooling Drivers.

(FEATURE - DONE) 11. Virgo Cloud File System with virgo_cloud_open(), virgo_cloud_read() , virgo_cloud_write() and virgo_cloud_close() commands invoked through telnet path has been implemented that transcends disk storage in all nodes in the cloud. It is also fanciful feature addition that would make VIRGO a complete all-pervading cloud platform. The remote telnet clients send the file path and the buf to be read or data to be written. The Virgo File System kernel driver service creates a unique Virgo File Descriptor for each struct file* opened by filp_open() and is returned to client. Earlier design option to use a hashmap (linux/hashmap.h) looked less attractive as file descriptor is an obvious unique description for open file and also map becomes unscalable. The kernel upcall path has been implemented (paramIsExecutable=0) and may

not be necessary in most cases and all above cloudfs commands work in kernelspace using VFS calls.

(FEATURE - DONE) 12. VIRGO Cloud File System commands through syscall paths - `virgo_open()`, `virgo_close()`, `virgo_read()` and `virgo_write()`. All the syscalls have been implemented with testcases and more bugs fixed. After fullbuild and testing, `virgo_open()` and `virgo_read()` work and `copy_to_user()` is working.

(FEATURE - DONE) 13. VIRGO memory pooling feature is also a distributed key-value store similar to other prominent key-store software like BigTable implementations, Dynamo, memory caching tools etc., but with a difference that VIRGO mempool is implemented as part of Linux Kernel itself thus circumventing userspace latencies. Due to Kernel space VIRGO mempool has an added power to store and retrieve key-value pair in hardware devices directly which otherwise is difficult in userspace implementations.

14. VIRGO memory pooling can be improved with disk persistence for in-memory key-value store using `virgo_malloc()`, `virgo_set()`, `virgo_get()` and `virgo_free()` calls. Probably this might be just a set of invocations of read and write ops in disk driver or using sysfs. Probably this could be redundant as the VIRGO filesystem primitives have been implemented that write to a remote host's filesystem in kernelspace.

15. (FEATURE-DONE) Socket Debugging, Program Analysis and Verification features for user code that can find bugs statically. Socket skbuff debug utility and SATURN Program Analysis Software has been integrated into NEURONRAIN VIRGO Linux Kernel.

16(FEATURE - DONE-Minimum Functionality). Operating System Logfile analysis using Machine Learning code in AstroInfer for finding patterns of processes execution and learn rules from the log. Kernel_Analytics VIRGO module reads `/etc/virgo_kernel_analytics.conf` config key-value pairs which are set by AsFer or other Machine Learning Software. At present an Apache Spark usecase that mines Uncomplicated Fire Wall logs in `kern.log` for most prominent source IP has been implemented in AsFer codebase : <http://sourceforge.net/p/asfer/code/704/tree/python-src/SparkKernelLogMapReduceParser.py> . This is set as a key-value config in `/etc/virgo_kernel_analytics.conf` read and exported by `kernel_analytics` module.

17. Implementations of prototypical Software Transactional Memory and LockFree Datastructures for VIRGO memory pooling.

18. Scalability features for Multicore machines - references:
(<http://halobates.de/lk09-scalability.pdf>, <http://pdos.csail.mit.edu/papers/linux/osdi10.pdf>)

19. Read-Copy-Update algorithm implementation for VIRGO memory pooling that supports multiple simultaneous versions of memory for readers - widely used in redesigned Linux Kernel.

20. (FEATURE - SATURN integration - minimum functionality DONE) Program Comprehension features as an add-on described in : <https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0>. SATURN program analysis has been integrated into VIRGO linux with a stub driver.

21. (FEATURE - DONE) Bakery Algorithm implementation - cloudsync kernel module

22. (FEATURE - ONGOING) Implementation of Distributed Systems primitives for VIRGO cloud viz., Logical Clocks, Termination Detection, Snapshots, Cache Coherency subsystem etc.,(as part of cloudsync driver module). Already a simple timestamp

generation feature has been implemented for KingCobra requests with <ipaddress>:<localmachinetimestamp> format

23. (FEATURE - minimum functionality DONE) Enhancements to kmem if it makes sense, because it is better to rely on virgo_malloc() for per machine memory management and wrap it around with a cloudwide VIRGO Unique ID based address lookup implementation of which is already in place.

Kernel Malloc syscall kmalloc() internally works as follows:

- kmem_cache_t object has pointers to 3 lists
- These 3 lists are full objects SLAB list, partial objects SLAB list and free objects SLAB list - all are lists of objects of same size
- and cache_cache is the global list of all caches created thus far.
- Any kmalloc() allocation searches partial objects SLAB list and allocates a memory block with kmem_cache_alloc() from the first SLAB available - returned to caller.
- Any kfree() returns an object to a free SLAB list
- Full SLABs are removed from partial SLAB list and appended to full SLAB list
- SLABs are virtual memory pages created with kmem_cache_create
- Each SLAB in SLABs list has blocks of similar sized objects (e.g. multiples of two). Closest matching block is returned and fragmentation is minimized (incidentally this is the knapsack and packing optimization LP problem and thus NP-complete).

KERNELSPACE:

VIRGO address translation table already implements a tree registry of vtables each of capacity 3000 that keep track of kmalloc() allocations across all cloud nodes. Implementation of SLAB allocator for kmalloc() creates a kmem_cache(s) of similar sized objects and kmem_cache_alloc() allocates from these caches. kmalloc() already does lot of per-machine optimizations. VIRGO vtable registry tree maintained in VIRGO memory syscall end combined with per-machine kmalloc() cache_cache already look sufficient. Instrumenting kmem_cache_create() with #ifdef SLAB_CLOUD_MALLOC flags to do RPC looks superfluous. Hence marking this action item as done. Any further optimization can be done on top of existing VIRGO address translation table struct - e.g bookkeeping flags, function pointers etc.,.

USERSPACE: sbrk() and brk() are no longer used internally in malloc() library routines. Instead mmap() has replaced it (<http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/lecture.html>, <http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/diff.html>).

24.(FEATURE - ONGOING) Cleanup the code and remove unnecessary comments.

25.(FEATURE - DONE) Documentation - This design document is also a documentation for commit notes and other build and debugging technical details. Doxygen html cross-reference documentation for AsFer, USBmd, VIRGO, KingCobra and Acadpdrfts has been created along with summed-up design document and committed to GitHub Repository at https://github.com/shrinivaasanka/Krishna_iResearch_DoxygenDocs

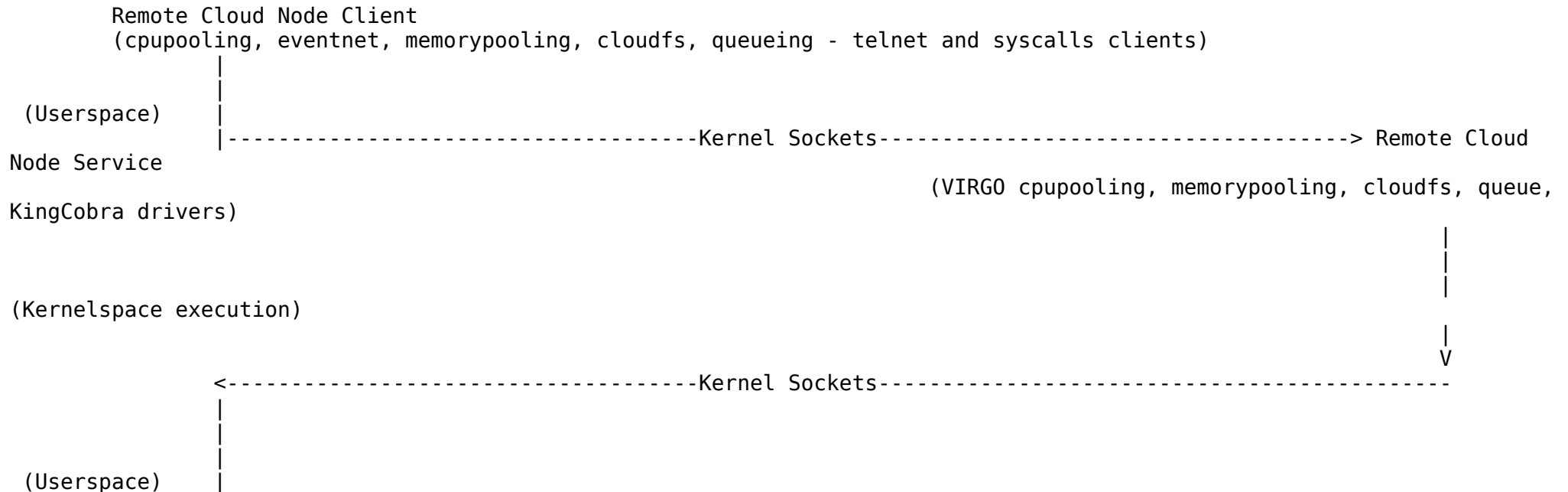
26. (FEATURE - DONE) Telnet path to virgo_cloud_malloc, virgo_cloud_set and virgo_cloud_get has been tested and working. This is similar to memcached but stores key-value in kernel space (and hence has the ability to write to and retrieve from any device driver memory viz., cards, handheld devices). An optional todo is to write a script or userspace socket client that connects to VIRGO mempool driver for these commands.

27. Augment the Linux kernel workqueue implementation (<http://lxr.free-electrons.com/source/kernel/workqueue.c>) with disk

persistence if feasible and doesn't break other subsystems - this might require additional persistence flags in work_struct and additional #ifdefs in most of the queue functions that write and read from the disk. Related to item 6 above.

28.(FEATURE - DONE) VIRGO queue driver with native userspace queue and kernel workqueue-handler framework that is optionally used for KingCobra and is invoked through VIRGO cpupooling and memorypooling drivers. (Schematic in <http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt> and http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourceproducts_archdiagram.pdf)

29.(FEATURE - DONE) KERNELSPACE EXECUTION ACROSS CLOUD NODES which geographically distribute userspace and kernelspace execution creating a logical abstraction for a cloudwide virtualized kernel:



30. (FEATURE - DONE) VIRGO platform as on 5 May 2014 implements a minimum set of features and kernelsocket commands required for a cloud OS kernel: CPU virtualization(virgo_clone), Memory virtualization(virgo_malloc,virgo_get,virgo_set,virgo_free) and a distributed cloud file system(virgo_open,virgo_close,virgo_read,virgo_write) on the cloud nodes and thus gives a logical view of one unified, distributed linux kernel across all cloud nodes that splits userspace and kernelspace execution across cloud as above.

31. (FEATURE - DONE) VIRGO Queue standalone kernel service has been implemented in addition to paths in schematics above. VIRGO Queue listens on hardcoded port 60000 and enqueues the incoming requests to VIRGO queue which is serviced by KingCobra:

VIRGO Queue client(e.g telnet) -----> VIRGO Queue kernel service ---> Linux Workqueue handler -----> KingCobra

32. (FEATURE - DONE) EventNet kernel module service:

VIRGO eventnet client (telnet) -----> VIRGO EventNet kernel service -----> EventNet graph text files

33. (FEATURE - DONE) Related to point 22 - Reuse EventNet cloudwide logical time infinite graph in AsFer in place of Logical clocks. At present the eventnet driver listens on port 20000 and writes the edges and vertices files in kernel using `vfs_read()/vfs_write()`. These text files can then be read by the AsFer code to generate DOT files and visualize the graph with graphviz.

34. (FEATURE - OPTIONAL) The kernel modules services listening on ports could return a JSON response when connected instead of plaintext, conforming to REST protocol. Additional options for protocol buffers which are becoming a standard data interchange format.

35. (FEATURE-Minimum Functionality DONE) Pointer Swizzling and Unswizzling of VIRGO addressspace pointers to/from VIRGO Unique ID (VUID). Presently VIRGO memory system calls implement a basic minimal pointer address translation to unique kmem location identifier.

CODE COMMIT RELATED NOTES

VIRGO code commits as on 16/05/2013

1. VIRGO clouddriver with a listener kernel thread service has been implemented and it listens on port 10000 on system startup through `/etc/modules load-on-bootup` facility

2. VIRGO clouddriver `virgo_clone()` system call has been implemented that would `kernel_connect()` to the VIRGO clouddriver service listening at port 10000

3. VIRGO clouddriver has been split into `virgo.h` (VIRGO typedefs), `virgoclouddriver.h` (VIRGO clouddriver service that is invoked by `module_init()` of VIRGO clouddriver) and `virgo_clouddriver.c` (with module ops definitions)

4. VIRGO does not implement SUN RPC interface anymore and now has its own virgo ops.

5. Lot of Kbuild related commits with commented lines for future use have been done viz., to integrate VIRGO to Kbuild, `KBUILD_EXTRA_SYMBOLS` for cross-module symbol reference.

VIRGO code commits as on 20/05/2013

1. test_virgo_clone.c testcase for sys_virgo_clone() system call works and connections are established to VIRGO clouddex kernel module.

2. Makefile for test_virgo_clone.c and updated buildscript.sh for headers_install for custom-built linux.

VIRGO code commits as on 6/6/2013

1. Message header related bug fixes

VIRGO code commits as on 25/6/2013

1. telnet to kernel service was tested and found working
2. GFP_KERNEL changed to GFP_ATOMIC in VIRGO clouddex kernel service

VIRGO code commits as on 1/7/2013

1. Instead of printing iovec, printing buffer correctly prints the messages
2. wake_up_process() added and function received from virgo_clone() syscall is executed with kernel_thread and results returned to virgo_clone() syscall client.

commit as on 03/07/2013

PRG loadbalancer preliminary code implemented. More work to be done

commit as on 10/07/2013

Tested PRG loadbalancer read config code through telnet and virgo_clone. VFS code to read from virgo_cloud.conf commented for testing

commits as on 12/07/2013

PRG loadbalancer prototype has been completed and tested with test_virgo_clone and telnet and symbol export errors and PRG errors have been fixed

commits as on 16/07/2013

read_virgo_config() and read_virgo_clone_config()(replica of read_virgo_config()) have been implemented and tested to read the virgo_cloud.conf config parameters(at present the virgo_cloud.conf has comma separated list of ip addresses. Port is hardcoded to 10000 for uniformity across all nodes). Thus minimal cloud functionality with config file support is in place. Todo things include function pointer lookup in kernel service, more parameters to cloud config file if needed, individual configs for virgo_clone() and virgo kernel service, kernel-to-userspace upcall and execution instead of kernel space, performance tuning etc.,

commits as on 17/07/2013

 moved read_virgo_config() to VIRGOcloudexec's module_init so that config is read at boot time and exported symbols are set beforehand.

Also commented read_virgo_clone_config() as it is redundant

commits as on 23/07/2013

 Lack of reflection kind of facilities requires map of function_names to pointers_to_functions to be executed on cloud has to be lookedup in the map to get pointer to function. This map is not scalable if number of functions are in millions and size of the map increases linearly. Also having it in memory is both CPU and memory intensive. Moreover this map has to be synchronized in all nodes for coherency and consistency which is another intensive task. Thus name to pointer function table is at present not implemented. Suitable way to call a function by name of the function is yet to be found out and references in this topic are scarce.

If parameterIsExecutable is set to 1 the data received from virgo_clone() is not a function but name of executable This executable is then run on usermode using call_usermodehelper() which internally takes care of queueing the workstruct and executes the binary as child of keventd and reaps silently. Thus workqueue component of kernel is indirectly made use of. This is sometimes more flexible alternative that executes a binary itself on cloud and is preferable to clone()ing a function on cloud. Virgo_clone() syscall client or telnet needs to send the message with name of binary.

If parameterIsExecutable is set to 0 then data received from virgo_clone() is name of a function and is executed in else clause using dlsym() lookup and pthread_create() in user space. This unifies both call_usermodehelper() and creating a userspace thread with a fixed binary which is same for any function. The dlsym lookup requires mangled function names which need to be sent by virgo_clone or telnet. This is far more efficient than a function pointer table.

call_usermodehelper() Kernel upcall to usermode to exec a fixed binary that would inturn execute the cloneFunction in userspace by spawning a pthread. cloneFunction is name of the function and not binary. This clone function will be dlsym()ed and a pthread will be created by the fixed binary. Name of the fixed binary is hardcoded herein as "virgo_kernelupcall_plugin". This fixed binary takes clone function as argument. For testing libvirgo.so has been created from virgo_cloud_test.c and separate build script to build the cloud function binaries has been added.

- Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
 (<https://sites.google.com/site/kuja27>)

commits as on 24/07/2013

test_virgo_clone unit test case updated with mangled function name to be sent to remote cloud node. Tested with test_virgo_clone end-to-end and all features are working. But sometimes kernel_connect hangs randomly (this was observed only today and looks similar to blocking vs non-blocking problem. Origin unknown).

- Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
(<https://sites.google.com/site/kuja27>)

commits as on 29/07/2013

Added kernel mode execution in the clone_func and created a sample kernel_thread for a cloud function. Some File IO logging added to upcall binaries and parameterIsExecutable has been moved to virgo.h

commits as on 30/07/2013

New usecase virgo_cloud_test_kernelspace.ko kernel module has been added. This exports a function virgo_cloud_test_kernelspace() and is accessed by virgo_cloudeexec kernel service to spawn a kernel thread that is executed in kernel addressspace. This Kernel mode execution on cloud adds a unique ability to VIRGO cloud platform to seamlessly integrate hardware devices on to cloud and transparently send commands to them from a remote cloud node through virgo_clone().

Thus above feature adds power to VIRGO cloud to make it act as a single "logical device driver" though devices are in geographically in a remote server.

commits as on 01/08/2013 and 02/08/2013

Added Bash shell commandline with -c option for call_usermodehelper upcall clauses to pass in remote virgo_clone command message as arguments to it. Also tried output redirection but it works some times that too with a fatal kernel panic.

Ideal solutions are :

1. either to do a copy_from_user() for message buffer from user address space (or)
2. somehow rebuild the kernel with fd_install() pointing stdout to a VFS file* struct. In older kernels like 2.6.x, there is an fd_install code with in kmod.c (___call_usermodehelper()) which has been redesigned in kernel 3.x versions and fd_install has been removed in kmod.c .

3. Create a Netlink socket listener in userspace and send message up from kernel Netlink socket.

All the above are quite intensive and time consuming to implement. Moreover doing FileIO in usermode helper is strongly discouraged in kernel docs

Since Objective of VIRGO is to virtualize the cloud as single execution "machine", doing an upcall (which would run with root abilities) is redundant often and kernel mode execution is sufficient. Kernel mode execution with intermodule function invocation can literally take over the entire board in remote machine (since it can access PCI bus, RAM and all other device cards)

As a longterm design goal, VIRGO can be implemented as a separate protocol itself and sk_buff packet payload from remote machine can be parsed by kernel service and kernel_thread can be created for the message.

commits as on 05/08/2013:

Major commits done for kernel upcall usermode output logging with fd_install redirection to a VFS file. With this it has become easy for user space to communicate runtime data to Kernel space. Also a new strip_control_M() function has been added to strip \r\n or " ".

11 August 2013:

Open Source Design and Academic Research Notes uploaded to

http://sourceforge.net/projects/acadpdrfts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download

commits as on 23 August 2013

New Multithreading Feature added for VIRGO Kernel Service - action item 5 in ToDo list above (virgo_cloudexec driver module). All dependent headers changed for kernel threadlocalizing global data.

commits as on 1 September 2013

GNU Copyright license and Product Owner Profile (for identity of license issuer) have been committed. Also Virgo Memory Pooling - virgo_malloc() related initial design notes (handwritten scanned) have been committed(http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf)

commits as on 14 September 2013

Updated virgo malloc design handwritten nodes on kmalloc() and malloc() usage in kernelspace and userspace execution mode of virgo_cloudexec service (<http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo->

docs/VIRGO_Memory_Pooling_virgomalloc_design_notes_2_14September2013.pdf). As described in handwritten notes, virgo_malloc() and related system calls might be needed when a large scale allocation of kernel memory is needed when in kernel space execution mode and large scale userspace memory when in user modes (function and executable modes). Thus a cloud memory pool both in user and kernel space is possible.

 VIRGO virtual addressing

VIRGO virtual address is defined with the following datatype:

```
struct virgo_address
{
    int node_id;
    void* addr;
};
```

VIRGO address translation table is defined with following datatype:

```
struct virgo_addr_transtable
{
    int node_id;
    void* addr;
};
```

 VIRGO memory pooling prototypical implementation

VIRGO memory pooling implementation as per the design notes committed as above is to be implemented as a prototype under separate directory under drivers/virgo/memorypooling and \$LINUX_SRC_ROOT/virgo_malloc. But the underlying code is more or less similar to drivers/virgo/cpupooling and \$LINUX_SRC_ROOT/virgo_clone.

virgo_malloc() and related syscalls and virgo mempool driver connect to and listen on port different from cpupooling driver. Though all these code can be within cpupooling itself, mempooling is implemented as separate driver and co-exists with cpupooling on bootup (/etc/modules). This enables clear demarcation of functionalities for CPU and Memory virtualization.

Commits as on 17 September 2013

Initial untested prototype code - virgo_malloc and virgo mempool driver - for VIRGO Memory Pooling has been committed - copied and modified from virgo_clone client and kernel driver service.

Commits as on 19 September 2013

3.7.8 Kernel full build done and compilation errors in VIRGO malloc and mempool driver code and more functions code added

Commits as on 23 September 2013

Updated virgo_malloc.c with two functions, int_to_str() and addr_to_str(), using kmalloc() with full kernel re-build. (Rather a re-re-build because some source file updates in previous build got deleted somehow mysteriously. This could be related to Cybercrime issues mentioned in https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt)

Commits as on 24 September 2013

Updated syscall*.tbl files, staging.sh, Makefiles for virgo_malloc(), virgo_set(), virgo_get() and virgo_free() memory pooling syscalls. New testcase test_virgo_malloc for virgo_malloc(), virgo_set(), virgo_get(), virgo_free() has been added to repository. This testcase might have to be updated if return type and args to virgo_malloc+ syscalls are to be changed.

Commits as on 25 September 2013

All build related errors fixed after kernel rebuild some changes made to function names to reflect their names specific to memory pooling. Updated /etc/modules also has been committed to repository.

Commits as on 26 September 2013

Circular dependency error in standalone build of cpu pooling and memory pooling drivers fixed and datatypes and declarations for CPU pooling and Memory Pooling drivers have been segregated into respective header files (virgo.h and virgo_mempool.h with corresponding service header files) to avoid any dependency error.

Commits as on 27 September 2013

Major commits for Memory Pooling Driver listen port change and parsing VIRGO memory pooling commands have been done.

Commits as on 30 September 2013

New parser functions added for parameter parsing and initial testing on virgo_malloc() works with telnet client with logs in test_logs/

Commits as on 1 October 2013

Removed strcpy in virgo_malloc as ongoing bugfix for buffer truncation in syscall path.

Commits as on 7 October 2013

Fixed the buffer truncation error from virgo_malloc syscall to mempool driver service which was caused by sizeof() for a char*. BUF_SIZE is now used for size in both syscall client and mempool kernel service.

Commits as on 9 October 2013 and 10 October 2013

Mempool driver kernelspace virgo mempool ops have been rewritten due to lack of facilities to return a value from kernel thread function. Since mempool service already spawns a kthread, this seems to be sufficient. Also the `iov.iov_len` in `virgo_malloc` has been changed from `BUF_SIZE` to `strlen(buf)` since `BUF_SIZE` causes the kernel socket to block as it waits for more data to be sent.

Commits as on 11 October 2013

`sscanf` format error for `virgo_cloud_malloc()` return pointer address and `sock_release()` null pointer exception has been rectified.

Added `str_to_addr()` utility function.

Commits as on 14 October 2013 and 15 October 2013

Updated todo list.

Rewritten `virgo_cloud_malloc()` syscall with:

- mutexed `virgo_cloud_malloc()` loop
- redefined virgo address translation table in `virgo_mempool.h`
- `str_to_addr()`: removed `(void**)` cast due to null `sscanf` though it should have worked

Commits as on 18 October 2013

Continued debugging of null `sscanf` - added `str_to_addr2()` which uses `simple_strtoll()` kernel function for scanning pointer as long long from string and casting it to `void*`. Also more `%p` qualifiers were added in `str_to_addr()` for debugging.

Based on latest `test_virgo_malloc` run, `simple_strtoll()` correctly parses the address string into a long long base 16 and then is reinterpreted_cast to `void*`. Logs in `test/`

Commits as on 21 October 2013

`Kern.log` for testing after `vtranstable` addr fix with `simple_strtoll()` added to repository and still the other `%p` qualifiers do not work and only `simple_strtoll()` parses the address correctly.

Commits as on 24 October 2013

Lot of bugfixes made to `virgo_malloc.c` for scanning address into `VIRGO` transtable and size computation. Testcase `test_virgo_malloc.c` has also been modified to do reinterpret cast of long long into `(struct virgo_address*)` and corresponding test logs have been added to repository under `virgo_malloc/test`.

Though the above `sys_virgo_malloc()` works, the return value is a kernel pointer if the `virgo_malloc` executes in the Kernel mode which is more likely than User mode (`call_usermodehelper` which is circuitous). Moreover `copy_from_user()` or

copy_to_user() may not be directly useful here as this is an address allocation routine. The long long reinterpret cast obfuscates the virgo_address(User or Kernel) as a large integer which is a unique id for the allocated memory on cloud. Initial testing of sys_virgo_set() causes a Kernel Panic as usual probably due to direct access of struct virgo_address*. Alternatives are to use only long long for allocation unique-id everywhere or do copy_to_user() or copy_from_user() of the address on a user supplied buffer. Also vtranstable can be made into a bucketed hash table that maps each alloc_id to a chained virgo malloc chunks than the present sequential addressing which is more similar to open addressing.

Commits as on 25 October 2013

virgo_malloc.c has been rewritten by adding a userspace __user pointer to virgo_get() and virgo_set() syscalls which are internally copied with copy_from_user() and copy_to_user() kernel function to get and set userspace from kernelspace. Header file syscalls.h has been updated with changed syscalls prototypes. Two functions have been added to map a VIRGO address to a unique virgo identifier and viceversa for abstracting hardware addresses from userspace as mentioned in previous commit notes. VIRGO cloud mempool kernelspace driver has been updated to use virgo_mempool_args* instead of void* and VIRGO cloudexec mempool driver has been updated accordingly during intermodule invocation. The virgo_malloc syscall client has been updated to modified signatures and return types for all mempool alloc, get, set, free syscalls.

Commits as on 29 October 2013

Miscellaneous ongoing bugfixes for virgo_set() syscall error in copy_from_user().

Commits as on 2 November 2013

Due to an issue which corrupts the kernel memory, presently telnet path to VIRGO mempool driver has been tested after commits on 31 October 2013 and 1 November 2013 and is working but again there is an issue in kstrtoul() that returns the wrong address in virgo_cloud_mempool_kernelspace.ko that gives the address for data to set.

Commits as on 6 November 2013

New parser function virgo_parse_integer() has been added to virgo_cloud_mempool_kernelspace driver module which is carried over from lib/kstrtoux.c and modified locally to add an if clause to discard quotes and unquotes. With this the telnet path commands for virgo_malloc() and virgo_set() are working. Today's kern.log has been added to repository in test_logs/.

Commits as on 7 November 2013

In addition to virgo_malloc and virgo_set, virgo_get is also working through telnet path after today's commit for "virgodata:" prefix in virgo_cloud_mempool_kernelspace.ko. This prefix is needed to differentiate data and address so that toAddressString() can be invoked to sprintf() the address in virgo_cloudexec_mempool.ko. Also mempool command parser has been updated to strcmp() virgo_cloud_get command also.

Commits as on 11 November 2013

More testing done on telnet path for virgo_malloc, virgo_set and virgo_get commands which work correctly. But there seem to be unrelated

kmem_cache_trace_alloc panics that follow each successful virgo command execution. kern.log for this has been added to repository.

Commits as on 22 November 2013

More testing done on telnet path for virgo_malloc, virgo_set and virgo_set after commenting kernel socket shutdown code in the VIRGO clouddexec

mempool sendto code. Kernel panics do not occur after commenting kernel socket shutdown.

Commits as on 2 December 2013

Lots of testing were done on telnet path and syscall path connection to VIRGO mempool driver and screenshots for working telnet path (virgo_malloc, virgo_set and virgo_get) have been committed to repository. Intriguingly, the syscall path is suddenly witnessing series of broken pipe errors, blocking errors etc., which are mostly Heisenbugs.

Commits as on 5 December 2013

More testing on system call path done for virgo_malloc(), virgo_set() and virgo_get() system calls with test_virgo_malloc.c. All three syscalls work in syscall path after lot of bugfixes. Kern.log that has logs for allocating memory in remote cloud node with virgo_malloc, sets data "test_virgo_malloc_data" with virgo_set and retrieves data with virgo_get.

VIRGO version 12.0 tagged.

Commits as on 12 March 2014

Initial VIRGO queueing driver implemented that flips between two internal queues: 1) a native queue implemented locally and 2) wrapper around linux kernel's workqueue facility 3) push_request() modified to pass on the request data to the workqueue handler using container_of on a wrapper structure virgo_workqueue_request.

Commits as on 20 March 2014

- VIRGO queue with additional boolean flags for its use as KingCobra queue
- KingCobra kernel space driver that is invoked by the VIRGO workqueue handler

Commits as on 30 March 2014

- VIRGO mempool driver has been augmented with use_as_kingcobra_service flags in CPU pooling and Memory pooling drivers

Commits as on 6 April 2014

- VIRGO mempool driver recvfrom() function's if clause for KingCobra has been updated for REQUEST header formatting mentioned in KingCobra design notes

Commits as on 7 April 2014

- generate_logical_timestamp() function has been implemented in VIRGO mempool driver that generates timestamps based on 3 boolean flags. At present machine_timestamp is generated and prepended to the request to be pushed to VIRGO queue driver and then serviced by KingCobra.

Commits as on 25 April 2014

- client ip address in VIRGO mempool recvfrom KingCobra if clause is converted to host byte order from network byte order with ntohs()

Commits as on 5 May 2014

- Telnet path commands for VIRGO cloud file system - virgo_cloud_open(), virgo_cloud_read(), virgo_cloud_write(), virgo_cloud_close() has been implemented and test logs have been added to repository (drivers/virgo/cloudfs/ and cloudfs/testlogs) and kernel upcall path for paramIsExecutable=0

Commits as on 7 May 2014

- Bugfixes to tokenization in kernel upcall plugin with strsep() for args passed on to the userspace

Commits as on 8 May 2014

- Bugfixes to virgo_cloud_fs.c for kernel upcall (parameterIsExecutable=0) and with these the kernel to userspace upcall and writing to a file in userspace (virgofstest.txt) works. Logs and screenshots for this are added to repository in test_logs/

Commits as on 6 June 2014

- VIRGO File System Calls Path implementation has been committed. Lots of Linux Full Build compilation errors fixed and new integer parsing functionality added (similar to driver modules). For the timebeing all syscalls invoke loadbalancer. This may be further optimized with a sticky flag to remember the first invocation which might be usually virgo_open syscall to get the VFS descriptor that is used in subsequent syscalls.

Commits as on 3 July 2014

- More testing and bugfixes for VIRGO File System syscalls have been done. virgo_write() causes kernel panic.

7 July 2014 - virgo_write() kernel panic notes:

 warning within <http://lxr.free-electrons.com/source/arch/x86/kernel/smp.c#L121>:

```
static void native_smp_send_reschedule(int cpu)
{
    if (unlikely(cpu_is_offline(cpu))) {
        WARN_ON(1);
        return;
    }
    apic->send_IPI_mask(cpumask_of(cpu), RESCHEDULE_VECTOR);
}
```

This is probably a fixed kernel bug in <3.7.8 but recurring in 3.7.8:

- <http://lkml.iu.edu/hypermail/linux/kernel/1205.3/00653.html>
- http://www.kernelhub.org/?p=3&msg=74473&body_id=72338
- <http://lists.openwall.net/linux-kernel/2012/09/07/22>
- https://bugzilla.kernel.org/show_bug.cgi?id=54331
- <https://bbs.archlinux.org/viewtopic.php?id=156276>

Commits as on 29 July 2014

 All VIRGO drivers(cloudfs, queuing, cpupooling and memorypooling) have been built on 3.15.5 kernel with some Makefile changes for ccflags and paths

 Commits as on 17 August 2014

 (FEATURE - DONE) VIRGO Kernel Modules and System Calls major rewrite for 3.15.5 kernel - 17 August 2014

 1. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service

config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel.

2. Each VIRGO module and system call now reads the config file independent of others - there is a read_virgo_config_<module>_<client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities.

3. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters.

4. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules.
5. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves.
6. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf.
7. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.
8. VIRGO File system driver and system calls have been tested with above changes and the virgo_open(), virgo_read() and virgo_write() calls work with much less crashes and freeze problems compared to 3.7.8 (some crashes in VIRGO FS syscalls in 3.7.8 where already reported kernel bugs which seem to have been fixed in 3.15.5). Today's kern.log test logs have been committed to repository.

Committed as on 23 August 2014

Commenting use_as_kingcobra_service if clauses temporarily as disabling also doesnot work and only commenting the block works for VIRGO syscall path. Quite weird as to how this relates to the problem. As this is a heisenbug further testing is difficult and sufficient testing has been done with logs committed to repository. Probably a runtime symbol lookup for kingcobra causes the freeze.
For forwarding messages to KingCobra and VIRGO queues, cpupooling driver is sufficient which also has the use_as_kingcobra_service clause.

Committed as on 23 August 2014 and 24 August 2014

As cpupooling driver has the same crash problem with kernel_accept() when KingCobra has benn enabled, KingCobra clauses have been commented in both cpupooling and memorypooling drivers. Instead queueing driver has been updated with a kernel service infrastructure to accept connections at port 60000. With this following paths are available for KingCobra requests:

VIRGO cpupooling or memorypooling ==> VIRGO Queue ==> KingCobra

(or)

VIRGO Queue kernel service ==> KingCobra

Committed as on 26 August 2014

- all kmallocs have been made into GFP_ATOMIC instead of GFP_KERNEL
- moved some kingcobra related header code before kernel_recvmmsg()
- some header file changes for set_fs()

This code has been tested with modified code for KingCobra and the standalone kernel service that accepts requests from telnet directly at port 60000, pushes to virgo_queue and is handled to invoke KingCobra servicerequest kernelspace function, works (the kernel_recvmmsg() crash was most probably due to Read-Only filesystem -errno printed is -30)

VIRGO version 14.9.9 has been release tagged on 9 September 2014

Committed as on 26 November 2014

New kernel module cloudsync has been added to repository under drivers/virgo that can be used for synchronization(lock() and unlock()) necessities in VIRGO cloud. Presently Bakery Algorithm has been implemented.

Committed as on 27 November 2014

virgo_bakery.h bakery_lock() has been modified to take 2 parameters - thread_id and number of for loops (1 or 2)

Committed as on 2 December 2014

VIRGO bakery algorithm implementation has been rewritten with some bugfixes. Sometimes there are soft lockup errors due to looping in kernel time durations for which are kernel build configurable.

Committed as on 17 December 2014

Initial code commits for VIRGO EventNet kernel module service:

1.EventNet Kernel Service listens on port 20000

2.It receives eventnet log messages from VIRGO cloud nodes and writes the log messages after parsing into two text files /var/log/eventnet/EventNetEdges.txt and /var/log/eventnet/EventNetVertices.txt by VFS calls

3.These text files can then be processed by the EventNet implementations in AsFer (python pygraph and C++ boost::graph based)

4. Two new directories `virgo/utils` and `virgo/eventnet` have been added.

5. `virgo/eventnet` has the new VIRGO EventNet kernel module service implementation that listens on port 20000.

6. `virgo/utils` is the new generic utilities driver that has a `virgo_eventnet_log()` exported function which connects to EventNet kernel service and sends the vertex and edge eventnet log messages which are parsed by kernel service and written to the two text files above.

7. EventNet log messages have two formats:

- Edge message - "eventnet_edgmsg#<id>#<from_event>#<to_event>"
- Vertex message - "eventnet_vertxtmsg#<id>-<partakers csv>-<partaker conversations csv>"

8. The utilities driver `Module.symvers` have to be copied to any driver which are then merged with the symbol files of the corresponding driver. Target clean has to be commented while building the unified `Module.symvers` because it erases `symvers` carried over earlier.

9. `virgo/utils` driver can be populated with all necessary utility exported functions that might be needed in other VIRGO drivers.

10. Calls to `virgo_eventnet_log()` have to be `#ifdef` guarded as this is quite network intensive.

Commits as on 18 December 2014

Miscellaneous bugfixes, logs and screenshot

- `virgo_clouddexec_eventnet.c` - eventnet messages parser errors and `eventnet_func` bugs fixed
- `virgo_cloud_eventnet_kernelspace.c` - `filp_open()` args updated due to `vfs_write()` kernel panics. The `vertexmessage` `vfs_write` is done after looping through the vertex textfile and appending the conversation to the existing vertex. Some more code has to be added.
- VIRGO EventNet build script updated for copying `Module.symvers` from `utils` driver for merging with eventnet `Module.symvers` during `Kbuild`
- Other build generated sources and kernel objects
- new `testlogs` directory with screenshot for `edgmsg` sent to EventNet kernel service and `kern.log` with previous history for `vfs_write()` panics due to permissions and the logs for working `filp_open()` fixed version
- vertex message update

Commits as on 2,3,4 January 2015

- fixes for `virgo` eventnet vertex and edge message text file `vfs_write()` errors
- `kern.logs` and screenshots

 VIRGO version 15.1.8 release tagged on 8 January 2015

 Commits as on 3 March 2015 - Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config (and) VIRGO memorypooling Key-Value Store Architecture Diagram

- Architecture of Key-Value Store in memorypooling (virgo_malloc,virgo_get,virgo_set,virgo_free) has been uploaded as a diagram at http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGOLinuxKernel_KeyValueStore_and_Modules_Interaction.jpg
- new kernel_analytics driver for AsFer <=> VIRGO+USBmd+KingCobra interface has been added.
- virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel dynamically changes behaviour.
- kernel_analytics Driver build script has been added

 Commits as on 6 March 2015

- code has been added in VIRGO config module to import EXPORTed kernel_analytics config key-pair array set by Apache Spark (mined from Uncomplicated Fire Wall logs) and manually and write to kern.log.

 NeuronRain version 15.6.15 release tagged

 Portability to linux kernel 4.0.5

The VIRGO kernel module drivers are based on kernel 3.15.5. With kernel 4.0.5 kernel which is the latest following compilation and LD errors occur - this is on cloudfs VIRGO File System driver :

- msghdr has to be user_msghdr for iov and iov_len as there is a segregation of msghdr
- modules_install throws an error in scripts/Makefile.modinst while overwriting already installed module

 Commits as on 9 July 2015

VIRGO cpupooling driver has been ported to linux kernel 4.0.5 with msghdr changes as mentioned previously with kern.log for VIRGO cpupooling driver invoked in parameterIsExecutable=2 (kernel module invocation)

added in testlogs

Commits as on 10,11 July 2015

VIRGO Kernel Modules:

- mempooling
- cloudfs
- utils
- config
- kernel_analytics
- cloudsync
- eventnet
- queuing

along with cpupooling have been ported to Linux Kernel 4.0.5 - Makefile and header files have been updated wherever required.

Commits as on 20,21,22 July 2015

Due to SourceForge Storage Disaster(<http://sourceforge.net/blog/sourceforge-infrastructure-and-service-restoration/>), the github replica of VIRGO is urgently updated with some important changes for msg_iter, iovec etc., in 4.0.5 kernel port specifically for KingCobra and VIRGO Queueing. These have to be committed to SourceForge Krishna_iResearch repository at http://sourceforge.net/users/ka_shrinivaasan once SourceForge repos are restored. Time to move on to the manufacturing hub? GitHub ;-)

VIRGO Queueing Kernel Module Linux Kernel 4.0.5 port:

- msg_iter is used instead of user_msghdr
- kvec changed to iovec
- Miscellaneous BUF_SIZE related changes
- kern.logs for these have been added to testlogs
- Module.symvers has been recreated with KingCobra Module.symvers from 4.0.5 KingCobra build
- clean target commented in build script as it wipes out Module.symvers
- updated .ko and .mod.c

KingCobra Module Linux Kernel 4.0.5 port

- vfs_write() has a problem in 4.0.5
- the filp_open() args and flags which were working in 3.15.5 cause a kernel panic implicitly and nothing was written to logs
- It took a very long time to figure out the reason to be vfs_write and filp_open
- O_CREAT, O_RDWR and O_LARGEFILE cause the panic and only O_APPEND is working, but

does not do `vfs_write()`. All other VIRGO Queue + KingCobra functionalities work viz., enqueueing, workqueue handler invocation, dequeueing, invoking kingcobra kernelspace service request function from VIRGO queue handler, timestamp, timestamp and IP parser, `reply_to_publisher` etc.,

- As mentioned in Greg Kroah Hartman's "Driving me nuts", persistence in Kernel space is a bad idea but still seems to be a necessary stuff - yet only `vfs` calls are used which have to be safe
- Thus KingCobra has to be in-memory only in 4.0.5 if `vfs_write()` doesn't work
- Intriguingly cloudfs filesystems primitives - `virgo_cloud_open`, `virgo_cloud_read`, `virgo_cloud_write` etc., work perfectly and append to a file.
- `kern.logs` for these have been added to testlogs
- `Module.symvers` has been recreated for 4.0.5
- updated `.ko` and `.mod.c`

Due to SourceForge outage and for a future code diversification NeuronRain codebases (AsFer, USBmd, VIRGO, KingCobra) in <http://sourceforge.net/u/userid-769929/profile/> have been replicated in GitHub also - <https://github.com/shrinivaasanka> excluding some huge logs due to Large File Errors in GitHub.

Commits as on 30 July 2015

VIRGO system calls have been ported to Linux Kernel 4.0.5 with commented gcc option `-Wimplicit-function-declaration`, `msghdr` and `iovec` changes similar to drivers mentioned in previous commit notes above. But Kernel 4.1.3 has some Makefile and build issues.

The NeuronRain codebases in SourceForge and GitHub would henceforth be mostly and always out-of-sync and not guaranteed to be replicas - might get diversified into different research and development directions (e.g one codebase might be more focussed on IoT while the other towards enterprise bigdata analytics integration with kernel and training which is yet to be designed- depend on lot of constraints)

Commits as on 2,3 August 2015

- new `.config` file added which is created from `menuconfig`
- `drivers/Kconfig` has been updated with 4.0.5 `drivers/Kconfig` for trace event linker errors

Linux Kernel 4.0.5 - `KConfig` is `drivers/` has been updated to resolve RAS driver trace event linker error. RAS was not included in `KConfig` earlier.

- `link-vmLinux.sh` has been replaced with 4.0.5 kernel version

Commits as on 12 August 2015

VIRGO Linux Kernel 4.1.5 port - related code changes - some important notes:

-
- Linux Kernel 4.0.5 build suddenly had a serious root shell drop error in initramfs which was not resolved by:
 - adding rootdelay in grub
 - disabling uuid for block devices in grub config
 - mounting in read/write mode in recovery mode
 - no /dev/mapper related errors
 - repeated exits in root shell
 - delay before mount of root device in initrd scripts
 - mysteriously there were some firmware microcode bundle executions in ieucodetool
 - Above showed a serious grub corruption or /boot MBR bug or 4.0.5 VIRGO kernel build problem
 - Linux 4.0.x kernels are EOL-ed
 - Hence VIRGO is ported to 4.1.5 kernel released few days ago
 - Only minimum files have been changed as in commit log for Makefiles and syscall table and headers and a build script has been added
- for 4.1.5:
- Changed paths:
 - A buildscript_4.1.5.sh
 - M linux-kernel-extensions/Makefile
 - M linux-kernel-extensions/arch/x86/syscalls/Makefile
 - M linux-kernel-extensions/arch/x86/syscalls/syscall_32.tbl
 - M linux-kernel-extensions/drivers/Makefile
 - M linux-kernel-extensions/include/linux/syscalls.h
- Above minimum changes were enough to build an overlay-ed Linux Kernel with VIRGO codebase

Commits as on 14,15,16 August 2015

Executed the minimum end-end telnet path primitives in Linux kernel 4.1.5 VIRGO code:

- cpu virtualization
 - memory virtualization
 - filesystem virtualization (updated filp_open flags)
- and committed logs and screenshots for the above.

Commits as on 17 August 2015

VIRGO queue driver:

- Rebuilt Module.symvers
- kern.log for telnet request to VIRGO Queue + KingCobra queueing system in kernelspace

Commits as on 25,26 September 2015

VIRGO Linux Kernel 4.1.5 - memory system calls:

- updated testcases and added logs for syscalls invoked separately(malloc,set,get,free)
- The often observed unpredictable heisen kernel panics occur with 4.1.5 kernel too. The logs are 2.3G and only grepped output is committed to repository.
- virgo_malloc.c has been updated with kstrdup() to copy the buf to iov.iov_base which was earlier crashing in copy_from_iter() within tcp code. This problem did not happen in 3.15.5 kernel.
- But virgo_clone syscall code works without any changes to iov_base as above which does a strcpy() which is an internal memcpy() though. So what causes this crash in memory system calls alone is a mystery.
- new insmod script has been added to load the VIRGO memory modules as necessary instead of at boot time.
- test_virgo_malloc.c and its Makefile has been updated.

VIRGO Linux Kernel 4.1.5 - filesystem calls- testcases and logs:

- added insmod script for VIRGO filesystem drivers
 - test_virgo_filesystem.c has been updated for syscall numbers in 4.1.5 VIRGO kernel
 - virgo_fs.c syscalls code has been updated for iov.iov_base kstrdup() - without this there are kernel panics in copy_from_iter(). kern.log
- testlogs have been added, but there are heisen kernel panics. The virgo syscalls are executed but not written to kern.log due to these crashes.
- Thus execution logs are missing for VIRGO filesystem syscalls.

Commits as on 28,29 September 2015

VIRGO Linux Kernel 4.1.5 filesystem syscalls:

- Rewrote iov_base code with a separate iovbuf set to iov_base and strcpy()-ing the syscall command to iov_base similar to VIRGO memory syscalls
- Pleasantly the same iovbuf code that crashes in memory syscalls works for VIRGO FS without crash. Thus both virgo_clone and virgo_filesystem syscalls work without issues in 4.1.5 and virgo_malloc() works erratically in 4.1.5 which remains as issue.
- kern.log for VIRGO FS syscalls and virgofstest text file written by virgo_write() have been added to repository

VIRGO Linux 4.1.5 kernel memory syscalls:

- rewrote the iov_base buffer code for all VIRGO memory syscalls by allocating separate iovbuf and copying the message to it - this just replicates the virgo_clone syscall behaviour which works without any crashes mysteriously.
- did extensive repetitive tests that were frequented by numerous kernel panics and crashes
- The stability of syscalls code with 3.15.5 kernel appears to be completely absent in 4.1.5

- The telnet path works relatively better though
- Difference between virgo_clone and virgo_malloc syscalls despite having same kernel sockets code looks like a non-trivial bug and a kernel stability issue.
- kernel OOPS traces are quite erratic.
- Makefile path in testcase has been updated

Commits as on 4 October 2015

VIRGO Linux Kernel 4.1.5 - Memory System Calls:

- replaced copy_to_user() with a memcpy()
- updated the testcase with an example VUID hardcoded.
- str_to_addr2() is done on iov_base instead of buf which was causing NULL parsing
- kern.log with above resolutions and multiple VIRGO memory syscalls tests - malloc,get,set
- With above VIRGO malloc and set syscalls work relatively causing less number of random kernel panics
- return values of memory calls set to 0
- in virgo_get() syscall, memcpy() of iov_base is done to data_out userspace pointer
- kern.log with working logs for syscalls - virgo_malloc(), virgo_set(), virgo_get() but still there are random kernel panics
- Abridged kern.log for VIRGO Memory System Calls with 4.1.5 Kernel - shows example logs for virgo_malloc(), virgo_set() and virgo_get()

Commits as on 14 October 2015

VIRGO Queue Workqueue handler usermode clause has been updated with 4.1.5 kernel paths and kingcobra in user mode is enabled for invoking KingCobra Cloud Perfect Forwarding.

Commits as on 15 October 2015

- Updated VIRGO Queue kernel binaries and build generated sources
- virgo_queue.h has been modified for call_usermodehelper() - set_ds() and fd_install() have been uncommented for output redirection. Output redirection works but there are "alloc_fd: slot 1 not NULL!" errors at random (kern.log in kingcobra testlogs) which seems to be a new feature in 4.1.5 kernel. This did not happen in 3.7.8-3.15.5 kernels

Commits as on 3 November 2015

- kern.log for VIRGO kernel_analytics+config drivers which export the analytics variables from /etc/virgo_kernel_analytics.conf kernel-wide and print them in config driver has been added to config/testlogs
-

Commits as on 10 January 2016

NeuronRain VIRGO enterprise version 2016.1.10 released.

NeuronRain - AsFer commits for VIRGO - C++ and C Python extensions

- Commits as on 29 January 2016

(FEATURE - DONE) Python-C++-VIRGOKernel and Python-C-VIRGOKernel boost::python and cpython implementations:

-
- It is a known idiom that Linux Kernel and C++ are not compatible.
 - In this commit an important feature to invoke VIRGO Linux Kernel from userspace python libraries via two alternatives have been added.
 - In one alternative, C++ boost::python extensions have been added to encapsulate access to VIRGO memory system calls - virgo_malloc(), virgo_set(), virgo_get(), virgo_free(). Initial testing reveals that C++ and Kernel are not too incompatible and all the VIRGO memory system calls work well though initially there were some errors because of config issues.
 - In the other alternative, C Python extensions have been added that replicate boost::python extensions above in C - C Python with Linux kernel works exceedingly well compared to boost::python.
 - This functionality is required when there is a need to set kernel analytics configuration variables learnt by AsFer Machine Learning Code dynamically without re-reading /etc/virgo_kernel_analytics.conf.
 - This completes a major integration step of NeuronRain suite - request travel roundtrip to-and-fro top level machine-learning C++/python code and rock-bottom C linux kernel - bull tamed ;-).
 - This kind of python access to device drivers is available for Graphics Drivers already on linux (GPIO - for accessing device states)
 - logs for both C++ and C paths have been added in cpp_boost_python_extensions/ and cpython_extensions.
 - top level python scripts to access VIRGO kernel system calls have been added in both directories:
 - CPython - python cpython_extensions/asferpythonextensions.py
 - C++ Boost::Python - python cpp_boost_python_extensions/asferpythonextensions.py
 - .so, .o files with build commandlines(asferpythonextensions.build.out) for "python setup.py build" have been added in build lib and temp directories.
 - main implementations for C++ and C are in cpp_boost_python_extensions/asferpythonextensions.cpp and cpython_extensions/asferpythonextensions.c

Commits as on 12 February 2016

Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra

*) This was commented earlier for the past few years due to a serious kernel panic in previous kernel versions - <= 3.15.5

*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was causing following error in "use_as_kingcobra_service" clause :

- "gave up waiting for virgo_queue init, unknown symbol push_request()"

*) To address this a new boolean flag to selectively enable and disable VIRGO Queue kernel service mode "virgo_queue_reactor_service_mode" has been added.

*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request

*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as generic string and handed over to VIRGO queue and KingCobra which publishes it.

*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.

*) This makes virgo_clone() syscall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service" clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped by KingCobra.

*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace.

Prerequisites:

- insmod kingcobra_main_kernelspace.ko
- insmod virgo_queue.ko compiled with flag virgo_queue_reactor_service_mode=1
(when virgo_queue_reactor_service_mode=0, listens on port 60000 for direct telnet requests)
- insmod virgo_cloud_test_kernelspace.ko
- insmod virgo_cloudexec.ko (listens on port 10000)

Schematic Diagram

VIRGO clone system call/telnet client ---> VIRGO cpupooling(compiled with use_as_kingcobra_service=1) -----> VIRGO Queue kernel service (compiled with virgo_queue_reactor_service_mode=1) ---> Linux Workqueue handler -----> KingCobra

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

- Imported Kernel Analytics variables into CloudFS kernel module - printed in driver init()
- Module.symvers from kernel_analytics has been merged with CloudFS Module.symvers
- Logs for above has been added in cloudfs/test_logs/
- Makefile updated with correct fs path
- Copyleft notices updated

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

-
- Kernel Analytics driver exported variables have been imported in CPU virtualization driver
 - Module.symvers from kernel_analytics has been merged with Module.symvers in cpupooling
 - kern.log for this import added to cpupooling/virgocloudexec/test_logs/
-

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

- Imported kernel analytics variables into memory virtualization driver init() , exported from kernel_analytics driver
 - build shell script updated
 - logs added to test_logs/
 - Module.symvers from kernel_analytics has been merged with memory driver Module.symvers
 - Makefile updated
-

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

- Imported kernel analytics variables into VIRGO Queueing Driver
 - logs for this added in test_logs/
 - Makefile updated
 - Module.symvers from kernel_analytics has been merged with Queueing driver's Module.symvers
 - .ko, .o and build generated sources
-

Commits as on 16,17 February 2016

(FEATURE-DONE) Socket Buffer Debug Utility Function - uses linux skbuff facility

- In this commit a multipurpose socket buffer debug utility function has been added in utils driver and exported kernelwide.
 - It takes a socket as function argument does the following:
 - dereference the socket buffer head of skbuff per-socket transmit data queue
 - allocate skbuff with alloc_skb()
 - reserve head room with skb_reserve()
 - get a pointer to data payload with skb_put()
 - memcpy() an example const char* to skbuff data
 - Iterate through the linked list of skbuff queue in socket and print headroom and data pointers
 - This can be used as a packet sniffer anywhere within VIRGO linux network stack
 - Any skb_*() functions can be plugged-in here as deemed necessary.
 - kern.log(s) which print the socket internal skbuff data have been added to a new testlogs/ directory
 - .cmd files generated by kbuild
-

(FEATURE-DONE) Commits as on 24 February 2016

skbuff debug function in utils/ driver:
(*) Added an if clause to check NULLity of skbuff headroom before doing skb_alloc()
(*) kern.log for this commit has been added testlogs/
(*) Rebuilt kernel objects and sources

Commits as on 29 February 2016

(FEATURE-DONE) Software Analytics - SATURN Program Analysis added to VIRGO Linux kernel drivers

- SATURN (saturn.stanford.edu) Program Analysis and Verification software has been integrated into VIRGO Kernel as a Verification+SoftwareAnalytics subsystem
- A sample driver that can invoke an exported function has been added in drivers - saturn_program_analysis
- Detailed document for an example null pointer analysis usecase has been created in virgo-docs/VIRGO_SATURN_Program_Analysis_Integration.txt
- linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_analysis_trees/error.txt is the error report from SATURN
- SATURN generated preproc and trees are in linux-kernel-extensions/drivers/virgo/saturn_program_analysis/preproc and linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_analysis_trees/

Commits as on 10 March 2016

- SATURN analysis databases (.db) for locking, memory and CFG analysis.
- DOT and PNG files for locking, memory and CFG analysis.
- new folder saturn_calypso_files/ has been added in saturn_program_analysis/ with new .clp files virgosaturncfg.clp and virgosaturnmemory.clp
- SATURN alias analysis .db files

(FEATURE-DONE) NEURONRAIN - ASFER Commits for VIRGO - CloudFS systems calls integrated into Boost::Python C++ and Python CAPI invocations

AsFer Commits as on 30 May 2016

VIRGO CloudFS system calls have been added (invoked by unique number from syscall_32.tbl) for C++ Boost::Python interface to VIRGO Linux System Calls. Switch clause with a boolean flag has been introduced to select either VIRGO memory or filesystem calls.

kern.log and CloudFS textfile Logs for VIRGO memory and filesystem invocations from AsFer python have been committed to testlogs/

AsFer Commits as on 31 May 2016

Python CAPI interface to NEURONRAIN VIRGO Linux System Calls has been updated to include File System open, read, write primitives also.

Rebuilt extension binaries, kern.logs and example appended text file have been committed to testlogs/. This is exactly similar to

commits done for Boost::Python C++ interface. Switch clause has been added to select memory or filesystem VIRGO syscalls.

(BUG - STABILITY ISSUES) Commits - 25 July 2016 - Static Analysis of VIRGO Linux kernel for investigating heisencrashes

Initial Documentation for Smatch and Coccinelle kernel static analyzers executed on VIRGO Linux kernel - to be updated periodically with further analysis.

--
(BUG - STABILITY ISSUES) Commits - 1 August 2016 - VIRGO Linux Stability Issues - Ongoing Random oops and panics investigation

--
1. GFP_KERNEL has been replaced with GFP_ATOMIC flags in kmem allocations.

2. NULL checks have been introduced in lot of places involving strcpy, strcat, strcmp etc., to circumvent buffer overflows.

3. Though this has stabilized the driver to some extent, still there are OOPS in unrelated places deep with in kernel where paging datastructures are accessed - kmalloc somehow corrupts paging

4. OOPS are debugged via gdb as:

4.1 gdb ./vmlinux /proc/kcore

or

4.2 gdb <loadable_kernel_module>.o

followed by

4.3 l *(address+offset in OOPS dump)

5. kern.log(s) for the above have been committed in tar.gz format and have numerous OOPS occurred during repetitive telnet and syscall

invocation(boost::python C++) invocations of virgo memory system calls.

6. Paging related OOPS look like an offshoot of set_fs() encompassing the filp_open VFS calls.

(BUG-STABILITY ISSUES) Commits - 26 September 2016 - Ongoing Random Panic investigation

Further analysis on direct VIRGO memory cache primitives telnet invocation - problems are similar

to Boost::Python AsFer VIRGO system calls invocations.

(BUG-STABILITY ISSUES) Commits - 27 September 2016 - Ongoing Random Panic investigation

Analysis of VIRGO memory cache primitives reveal more inconsistencies in cacheline flushes between CPU and GPU.

Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
<http://sites.google.com/site/kuja27>