

---

# **virgo64-linux-github-code**

## **Documentation**

*Release latest*

**Aug 16, 2023**



## CONTENTS

<b>1</b>	<b>888. CPU pooling or cloud ability in a system call:</b>	<b>3</b>
<b>2</b>	<b>Experimental Prototype</b>	<b>5</b>
<b>3</b>	<b>VIRGO - loadbalancer to get the host:ip of the least loaded node</b>	<b>7</b>
<b>4</b>	<b>889. Loadbalancer option 1 - Centralized loadbalancer registry that tracks load:</b>	<b>9</b>
<b>5</b>	<b>890. Loadbalancer option 2 - Linux Psuedorandom number generator based load balancer(experimental) instead of centralized registry that tracks load:</b>	<b>11</b>
5.1	891. Implemented VIRGO Linux components (as on 7 March 2016) . . . . .	11
5.2	VIRGO Features (list is quite dynamic and might be rewritten depending on feasibility - longterm with no deadline) . . . . .	12
<b>6</b>	<b>895.1 Schematic Diagram:</b>	<b>15</b>
<b>7</b>	<b>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.</b>	<b>17</b>
<b>8</b>	<b>Example scenario 898.2 without implementation:</b>	<b>19</b>
8.1	CODE COMMIT RELATED NOTES . . . . .	23
<b>9</b>	<b>927. VIRGO code commits as on 16/05/2013</b>	<b>25</b>
<b>10</b>	<b>928. VIRGO code commits as on 20/05/2013</b>	<b>27</b>
<b>11</b>	<b>929. VIRGO code commits as on 6/6/2013</b>	<b>29</b>
<b>12</b>	<b>930. VIRGO code commits as on 25/6/2013</b>	<b>31</b>
<b>13</b>	<b>931. VIRGO code commits as on 1/7/2013</b>	<b>33</b>
<b>14</b>	<b>932. commit as on 03/07/2013</b>	<b>35</b>
<b>15</b>	<b>933. commit as on 10/07/2013</b>	<b>37</b>
<b>16</b>	<b>934. commits as on 12/07/2013</b>	<b>39</b>
<b>17</b>	<b>935. commits as on 16/07/2013</b>	<b>41</b>
<b>18</b>	<b>936. commits as on 17/07/2013</b>	<b>43</b>

19	937. commits as on 23/07/2013	45
20	938. commits as on 24/07/2013	47
21	939. commits as on 29/07/2013	49
22	940. commits as on 30/07/2013	51
23	941. commits as on 01/08/2013 and 02/08/2013	53
24	942. commits as on 05/08/2013:	55
25	943. 11 August 2013:	57
26	944. commits as on 23 August 2013	59
27	945. commits as on 1 September 2013	61
28	946. commits as on 14 September 2013	63
29	949. Commits as on 17 September 2013	65
30	950. Commits as on 19 September 2013	67
31	951. Commits as on 23 September 2013	69
32	952. Commits as on 24 September 2013	71
33	953. Commits as on 25 September 2013	73
34	954. Commits as on 26 September 2013	75
35	955. Commits as on 27 September 2013	77
36	956. Commits as on 30 September 2013	79
37	957. Commits as on 1 October 2013	81
38	958. Commits as on 7 October 2013	83
39	959. Commits as on 9 October 2013 and 10 October 2013	85
40	960. Commits as on 11 October 2013	87
41	961. Commits as on 14 October 2013 and 15 October 2013	89
42	962. Commits as on 18 October 2013	91
43	963. Commits as on 21 October 2013	93
44	964. Commits as on 24 October 2013	95
45	965. Commits as on 25 October 2013	97
46	966. Commits as on 29 October 2013	99
47	967. Commits as on 2 November 2013	101
48	968. Commits as on 6 November 2013	103

49	969. Commits as on 7 November 2013	105
50	970. Commits as on 11 November 2013	107
51	971. Commits as on 22 November 2013	109
52	972. Commits as on 2 December 2013	111
53	973. Commits as on 5 December 2013	113
54	974. Commits as on 12 March 2014	115
55	975. Commits as on 20 March 2014	117
56	976. Commits as on 30 March 2014	119
57	977. Commits as on 6 April 2014	121
58	978. Commits as on 7 April 2014	123
59	979. Commits as on 25 April 2014	125
60	980. Commits as on 5 May 2014	127
61	981. Commits as on 7 May 2014	129
62	982. Commits as on 8 May 2014	131
63	983. Commits as on 6 June 2014	133
64	984. Commits as on 3 July 2014	135
65	985. 7 July 2014 - virgo_write() kernel panic notes:	137
66	986. Commits as on 29 July 2014	139
67	987. (FEATURE - DONE) VIRGO Kernel Modules and System Calls major rewrite for 3.15.5 kernel - 17 August 2014	141
68	Initial code commits for VIRGO EventNet kernel module service:	143
69	VIRGO Linux Kernel 4.1.5 port - related code changes - some important notes:	147
70	VIRGO Linux Kernel 4.1.5 - memory system calls:	149
71	VIRGO Linux Kernel 4.1.5 - filesystem calls- testcases and logs:	151
72	VIRGO Linux Kernel 4.1.5 filesystem syscalls:	153
73	VIRGO Linux 4.1.5 kernel memory syscalls:	155
74	VIRGO Linux Kernel 4.1.5 - Memory System Calls:	157
75	1016. (FEATURE - DONE) Python-C++-VIRGOKernel and Python-C-VIRGOKernel boost::python and cpython implementations:	159
76	1017. Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra	161
77	1022. (FEATURE-DONE) Socket Buffer Debug Utility Function - uses linux skbuff facility	163



```
/******  
#----- #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 #MER-  
CHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #GNU General Public License for more de-  
tails. #You should have received a copy of the GNU General Public License #along with this program. If not, see <http:  
//www.gnu.org/licenses/>. #-----  
#K.Srinivasan #NeuronRain Documentation and Licensing: http://neuronrain-documentation.  
readthedocs.io/en/latest/ #Personal website(research): https://sites.google.com/site/  
kuja27/ #-----  
*****/
```

885. 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.

886. 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 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.





## 888. 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**



## **889. 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 load monitoring 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>)





## 890. LOADBALANCER OPTION 2 - LINUX PSEUDORANDOM 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 =  $(\text{math.pow}(N, k+2) - k * \text{math.pow}(N, 2) + k * \text{math.pow}(N, 1) - 1) / (\text{math.pow}(N, k+3) - 2 * \text{math.pow}(N, k+2) + \text{math.pow}(N, k+1))$

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/`)

### 5.1 891. 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-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. 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-learnt config.

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

892. (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)
893. (FEATURE - Special case implementation DONE) 2. Object Marshalling and Unmarshalling (Serialization) Features - Feature 4 is a marshalling feature too as Python world PyObjects are serialized into VIRGO linux kernel and unmarshalled back bottom-up with CPython and Boost::Python C++ invocations - CPython and Boost internally take care of serialization.
894. (FEATURE - DONE) 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](http://sourceforge.net/p/virgo-linux/code-0/210/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf)

895. (FEATURE - DONE) Integrated testing of AsFer-VIRGO Linux Kernel request roundtrip - invocation of VIRGO linux kernel system calls from AsFer Python via C++ or C extensions - Commits for this have been done on 29 January 2016. This unifies userlevel applications and kernelspace modules so that AsFer Python makes VIRGO linux kernel an extension. Following is schematic diagram and More details in commit notes below.



## 895.1 SCHEMATIC DIAGRAM:

AsFer Python —> Boost::Python C++ Extension —> VIRGO memory system calls —> VIRGO  
Linux Kernel Memory Drivers / V

AsFer Python —> CPython Extensions —> VIRGO memory system calls —> VIRGO  
Linux Kernel Memory Drivers  
/ V ||

896. (FEATURE - DONE) Multithreading of VIRGO clouDEXEC kernel module (if not already done by kernel module subsystem internally)
897. (FEATURE - DONE) 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 memorypooling 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.
898. (FEATURE - DONE-Minimum Functionality - this section is an extended draft on respective topics in NeuronRain AstroInfer design - <https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt>) 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.

- 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 898.2 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.
- **Autonomous 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.

E.g. Autopilot in Tesla Cars processes Petabytes of information (Smooth-as-Silk algorithm) from sensors which are fed to neural networks computed on a cloud - <https://www.teslarati.com/tesla-firmware-v8-1-17-22-26-autopilot-2-0-smooth-silk-update-video/>.

898.4 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) 898.5 Self-learning, Predictive Systems - <https://icri-ci.technion.ac.il/projects/past-projects/machine-learning-for-architecture-self-learning-predictive-computer-systems/> 898.6 Linux Process Scheduling and Machine Learning - <http://www.cs.ucr.edu/~kishore/papers/tencon.pdf> 898.7 Network Latency and Machine Learning - [https://users.soe.ucsc.edu/~slukin/rtt\\_paper.pdf](https://users.soe.ucsc.edu/~slukin/rtt_paper.pdf) 898.8 Machine Learning based Meta-Scheduler for Multicore processors - [https://books.google.co.in/books?id=1GWcHmCrI0QC&pg=PA528&lpg=PA528&dq=linux+kernel+machine+learning&source=bl&ots=zfJsq\\_uu5q&sig=mMIUZ-oyJIwZXtYj4HntrQE8NSk&hl=en&sa=X&ved=0CCAQ6AEwATgKahUKEwjs9sqF9vPIAhVBFZQKHbNtA6A](https://books.google.co.in/books?id=1GWcHmCrI0QC&pg=PA528&lpg=PA528&dq=linux+kernel+machine+learning&source=bl&ots=zfJsq_uu5q&sig=mMIUZ-oyJIwZXtYj4HntrQE8NSk&hl=en&sa=X&ved=0CCAQ6AEwATgKahUKEwjs9sqF9vPIAhVBFZQKHbNtA6A)

899. 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)

900. (FEATURE - ONGOING) 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.
901. (FEATURE - DONE) 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.
902. (FEATURE - DONE) 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.
903. (FEATURE - DONE) VIRGO Cloud File System commands through syscall paths - virgo\_open(),virgo\_close(),virgo\_read() and virgo\_write(). All the syscalls have been implemented with test-cases and more bugs fixed. After fullbuild and testing, virgo\_open() and virgo\_read() work and copy\_to\_user() is working.
904. (FEATURE - DONE) 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.
905. 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.
906. (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.
907. (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.
908. (USERSPACE C++ usecase implemented in GRAFIT course material - <https://gitlab.com/shrinivaasanka/Grafit>) Implementations of prototypical Software Transactional Memory and LockFree Datastructures for VIRGO memory pooling.
909. Scalability features for Multicore machines - references: (<http://halobates.de/lk09-scalability.pdf>, <http://pdos.csail.mit.edu/papers/linux/osdi10.pdf>)
910. (USERSPACE C++ usecase implemented in GRAFIT course material - <https://gitlab.com/shrinivaasanka/Grafit>) Read-Copy-Update algorithm implementation for VIRGO memory pooling that supports multiple simultaneous

versions of memory for readers - widely used in redesigned Linux Kernel.

911. (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.
912. (FEATURE - DONE) Bakery Algorithm implementation - cloudsyntax kernel module
913. (FEATURE - minimal EventNet Logical Clock primitive implemented in AstroInfer and this section is an extended draft on respective topics in NeuronRain AstroInfer design - <https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt>) Implementation of Distributed Systems primitives for VIRGO cloud viz., Logical Clocks, Termination Detection, Snapshots, Cache Coherency subsystem etc.,(as part of cloudsyntax driver module). Already a simple timestamp generation feature has been implemented for KingCobra requests with <ipaddress>:<localmachinetimestamp> format
914. (FEATURE - minimum functionality DONE - this section is an extended draft on respective packing/filling/tiling topics in NeuronRain AstroInfer design specific to kernel SLAB allocator - <https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt>) 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>).

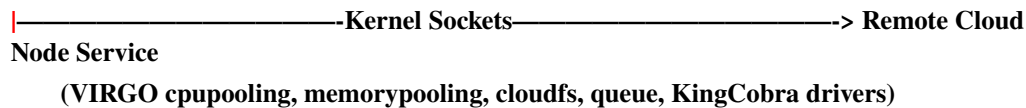
915. (FEATURE - ONGOING) Cleanup the code and remove unnecessary comments.
916. (FEATURE - DONE) Documentation - This design document is also a documentation for commit notes and other build and debugging technical details. Doxygen html cross-reference documentation for AsFer, USBmd, VIRGO, KingCobra and Acadpdrafts has been created along with summed-up design document and committed to GitHub Repository at [https://github.com/shrinivaasanka/Krishna\\_iResearch\\_DoxygenDocs](https://github.com/shrinivaasanka/Krishna_iResearch_DoxygenDocs)
917. (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 kernelspace (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.

918. 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.
919. (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](http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourceproducts_archdiagram.pdf))
920. (FEATURE - DONE) KERNELSPACE EXECUTION ACROSS CLOUD NODES which geographically distribute userspace and kernelspace execution creating a logical abstraction for a cloudwide virtualized kernel:
 

Remote Cloud Node Client (cpupooling, eventnet, memorypooling, cloudfs, queueing - telnet and syscalls clients)

(Userspace) |



(Kernelspace execution)

V



(Userspace) |

921. (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.
922. (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
923. (FEATURE - DONE) EventNet kernel module service: VIRGO eventnet client (telnet) —> VIRGO EventNet kernel service —> EventNet graph text files
924. (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.

- 925. (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.
- 926. (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.

## 8.1 CODE COMMIT RELATED NOTES



## **927. 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), virgocloudexecsvc.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.





## **928. 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 cloudexec kernel module.
2. Makefile for test\_virgo\_clone.c and updated buildscript.sh for headers\_install for custom-built linux.



## **929. VIRGO CODE COMMITS AS ON 6/6/2013**

1. Message header related bug fixes



## **930. 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



## **931. 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.





## **932. COMMIT AS ON 03/07/2013**

PRG loadbalancer preliminary code implemented. More work to be done



## **933. 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



## **934. 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



## **935. 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.,





## **936. 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



## 937. 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>)



## 938. 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>)



## **939. 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





## **940. 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_clouddexec` kernel service to spawn a kernel thread that is executed in kernel addresspace. 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.



## 941. 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.



## **942. 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 `rn` or `”` “.



**943. 11 AUGUST 2013:**

Open Source Design and Academic Research Notes uploaded to [http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes\\_2013-08-11.pdf/download](http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download)





## **944. 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.



## 945. 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](http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf))



## 946. COMMITS AS ON 14 SEPTEMBER 2013

Updated virgo malloc design handwritten notes 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](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 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 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.



## **949. 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.





## **950. 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



## **951. 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](https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt))



## **952. 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.



## **953. 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.





## **954. 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.



## **955. COMMITS AS ON 27 SEPTEMBER 2013**

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



## **956. 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/`



## **957. COMMITS AS ON 1 OCTOBER 2013**

Removed strcpy in virgo\_malloc as ongoing bugfix for buffer truncation in syscall path.





## **958. 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.



## **959. 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.



## 960. 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.



## **961. 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





## 962. 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 where 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 reinterpret\_cast to void\*. Logs in test/



## **963. 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.



## 964. 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.



## 965. 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 clouddex 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.





## **966. COMMITS AS ON 29 OCTOBER 2013**

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



## **967. 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 kstrtol() that returns the wrong address in virgo\_cloud\_mempool\_kernelspace.ko that gives the address for data to set.



## **968. 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/kstrtox.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/`.



## 969. 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 "vir-godata:" 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.





## **970. 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.



## **971. 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 cloudexec mempool sendto code. Kernel panics do not occur after commenting kernel socket shutdown.



## **972. 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 erros, blocking errors etc., which are mostly Heisenbugs.



## **973. 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.





## **974. 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`.



## **975. 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



## **976. 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



## **977. 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





## **978. 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.



## **979. 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()



## **980. 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`



## **981. COMMITS AS ON 7 MAY 2014**

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





## **982. 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/`



## **983. 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.



## **984. COMMITS AS ON 3 JULY 2014**

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



## 985. 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://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://bugzilla.kernel.org/show_bug.cgi?id=54331) - <https://bbs.archlinux.org/viewtopic.php?id=156276>





## **986. 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



## **987. (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.

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.

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

- all kmallocs have been made into GFP\_ATOMIC instead of GFP\_KERNEL
- moved some kingcobra related header code before kernel\_recvmsg()
- 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\_recvmsg() crash was most probably due to Read-Only filesystem -errno printed is -30)

New kernel module cloudsyntax 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.

virgo\_bakery.h bakery\_lock() has been modified to take 2 parameters - thread\_id and number of for loops (1 or 2)

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.

## 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\_edgemsg#<id>#<from\_event>#<to\_event>”
  - Vertex message - “eventnet\_vertexmsg#<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.

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 vertice 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 edgemsg 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

- fixes for virgo eventnet vertex and edge message text file vfs\_write() errors
- kern.logs and screenshots
- 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](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
- 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.

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

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

VIRGO Kernel Modules: - memorypooling - 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.

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](http://sourceforge.net/users/ka_shrinivaasan) once SourceForge repos are restored. Time to move on to the manufacturing hub? GitHub ;-)

1002. 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 \_\_\_\_\_

1003. 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

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.

---

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)

- 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





## 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

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.

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



## 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.



## 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



## 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()

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.

- 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
- 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

NeuronRain VIRGO enterprise version 2016.1.10 released.



## 1016. (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



## 1017. 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\_clouddexec.ko (listens on port 10000)

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

- 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
- 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/

- 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
- 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



## 1022. (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

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

- 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/

- 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

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/

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.

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

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 1 \*(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.

Further analysis on direct VIRGO memory cache primitives telnet invocation - problems are similar to Boost::Python AsFer VIRGO system calls invocations.

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

\*) moved virgoeventnetclient\_driver\_build.sh to virgoutils\_driver\_build.sh in utils/ driver \*) Updated VIRGO Linux Build Steps for 4.10.3 \*) New repository has been created for 64-bit VIRGO Linux kernel based on 4.10.3 mainline kernel in GitHub and imported in SourceForge:

<https://github.com/shrinivaasanka/virgo64-linux-github-code> <https://sourceforge.net/p/virgo64-linux/>

\*) Though it could have been branched off from existing VIRGO repository (32-bit) which is based on 4.1.5 mainline kernel, creating a separate repository for 64-bit 4.10.3 VIRGO kernel code was simpler because:

- there have been directory path changes for syscall entries in 4.10.3 and some other KBuild entities
- Some script changes done for 4.1.5 in modpost and vmlinux phases are not required
- having two VIRGO branches one with 4.1.5 code and 32-bit driver .ko binaries and other with 4.10.3 code and 64-bit driver .ko

**binaries could be unmanageable and commits could go into wrong branch**

- 4.10.3 64-bit VIRGO kernel build is still in experimental phase and it is not known if 64-bit 4.10.3 build solves earlier panic

**problems in 4.1.5**

- If necessary one of these two repositories could be made branch of the other later

\*) Prima facie, 64 bit kernel is quite finicky and importunate compared to 32 bit and 64 bit specific idiosyncrasies are to the fore. \*) During the past 1 week, quite a few variants of kernel and drivers builds were tried with KASAN enabled and without KASAN (Google Kernel Address Sanitizer) \*) KASAN shows quite huge number of user memory accesses which later translate to panics. \*) Most nagging of these was kernel\_recvmmsg() panic. \*) Added and updated skbuff socket debug utility driver with a new debug function and to print more fields of skbuff \*) KASAN was complaining

about `_asan_load8` (loading 8 userspace bytes) \*) All erroneous return data types in VIRGO mempool ops structure have been corrected in VIRGO headers \*) all type casts have been sanitized \*) Changed all kernel stack allocations to kernel heap `kzallocs` \*) This later caused a crash in `inet_sendmsg` in `kernel_sendmsg()` \*) `gdb64` disassemble showed a trapping instruction: `testb $0x6,0x91(%14)` with corresponding source line: `sg = !(sk->sk_route_caps & NETIF_F_SG)` in `tcp_sendmsg()` (`net/ipv4/tcp.c`) \*) changed `kernel_sendmsg()` to `sock->ops->sendmsg()` \*) These commits are still ongoing analysis only. \*) Screenshots for these have been added to `debug-info/`

\*) Previous commit was crashing inside `tcp_sendmsg()` \*) GDB64 disassembly shows NULL values for register R12 which is added with an offset 91 and is an operand in `testb` \*) Protected all `kernel_sendmsg()` and `kernel_recvmsg()` in both system calls side and drivers side with

```
oldfs=get_fs(), set_ds(KERNEL_DS) and set_fs(oldfs)
```

blocks without which there are random `kernel_sendmsg` and `kernel_recvmsg` hangs \*) Removed `init_net` and `sock_create_kern` usage everywhere and replaced them with `sock_create` calls \*) Tried `MSG_FASTOPEN` flags but it does not help much in resolving `tcp_sendmsg()` NULL pointer dereference issue. `MSG_FASTOPEN` just speeds up the message delivery by piggybacking the message payload before complete handshake is established (SYN, SYN-ACK, SYN-ACK-ACK) in SYN-ACK itself. But eventually it has to be enabled as fast open is becoming a standard. \*) KASAN reports have been enabled. \*) Added more debug code in `skbuff` debug utility functions in `utils` driver to check if `sk->prot` is a problem. \*) Replaced `kernel_sendmsg` with a `sock->ops->sendmsg()` in mempool `sendto` function which otherwise crashes in `tcp_sendmsg()`. \*) With `sock->ops->sendmsg()` systemcalls <—> drivers two-way request-reply works but still there are random -32 (broken pipe) and -104 (Connection Reset by Peer) errors \*) Logs for working `sys_virgo_malloc()` call with correctly returned VIRGO Unique ID for memory allocated has been committed to `test_logs` in `virgocloudexecmempool` \*) `sock->ops->sendmsg()` in mempool driver `sendto` function requires a `MSG_NOSIGNAL` flag which prevents `SIGPIPE` signal though not fully \*) Reason for random broken pipe and connection reset by peer errors in mempool `sendto` is unknown. Both sides have connections open and there is no noticeable traffic. \*) While socket communications in 32 bit VIRGO kernel syscalls and drivers work with no issues, why 64-bit has so many hurdles is puzzling. Reasons could be 64 bit address alignment issues, 64 bit specific `#ifdefs` in kernel code flow, major changes from 4.1.5 to 4.10.3 kernels etc., \*) NULL values for register R12 indicate already freed `skbuff` data which are accessed/double-freed. Kernel TCP engine has a circular linked list of `skbuff` write queue which is iterated in `skbuff` `utils` driver debug functions. \*) TCP engine clones the head data of `skbuff` queue, transmits it and waits for an ACK or timeout. Data is freed only if ACK or timeout occurs. And head of the queue is advanced to next element in write queue and this continues till write queue is empty waiting for more messages. \*) If ACK is not received, head data is cloned again and retransmitted by sequence number flow control.

\*) `kernel_sendmsg()` has been replaced with `sock->ops->sendmsg()` because `kernel_sendmsg()` is quite erratic in 4.10.3 64 bit \*) There were connection reset errors in system calls side for `virgo_malloc/`. This was probably because `sock->ops->sendmsg()` requires `MSG_DONTWAIT` and `MSG_NOSIGNAL` flags and `sendmsg` does not block. \*) `sock` release happens and `virgo_malloc` syscalls receives -104 error \*) Temporarily `sock_release` has been commented. Rather socket timeout should be relied upon which should do automatic release of socket resources \*) Similar flags have been applied in `virgo_malloc` syscalls too. \*) Logs with above changes do not have reset errors as earlier. \*) `virgo set/get` still crashes because 64 bit id is truncated which would require data type changes for 64 bit \*) `test_virgo_malloc` test case has been rebuilt with `-m64` flag for invocation of 64 bit syscalls by numeric ids

\*) There is something seriously wrong with 4.10.3 kernel sockets in 64 bit build VIRGO send/rcv messages and even accept/listen too. \*) All kernel socket functionalities which work well in 4.1.5 32 bit VIRGO, have random hangs, panics in 4.10.3 VIRGO64 build mostly in `inet_recvmsg/sendmsg` code path \*) KASAN shows attempts to access user address which occurs despite `set_fs(KERNEL_DS)` \*) Crash stack is similar to previous crashes in `tcp_sendmsg()` \*) Tried different address and protocol families for kernel socket accept (TCP,UDP,RAW sockets) \*) With Datagram sockets, `kernel_listen()` mysteriously fails with -95 error in `kernel_bind(operation not supported)` \*) With RAW sockets, `kernel_listen()` fails with -93 error for `AF_PACKET` (protocol not supported) \*) `tcpdump` `pcap` sniffer doesn't show anything unruly. \*) This could either be a problem with kernel build (unlikely), `Kbuild.config` or could have extraneous reasons. But `.config` for 4.1.5 and 4.10.3 are similar. \*) Only major difference between 4.10.3 and 4.1.5 is `init_net` added in `sock_create_kern()` internally \*) datatype of VIRGO Unique ID has been changed to unsigned long long (`_u64`) \*) tried with `INADDR_LOOPBACK` in place of `INADDR_ANY` \*) also tried with disabled multi(homing) in `/etc/hosts.conf` \*) Above random kernel socket hangs occur across all VIRGO system calls and drivers transport. \*)

Utils kernel socket client to EventNet kernel service also has similar inet\_recvmmsg/inet\_sendmmsg panic problems.

) EventNet driver works in 64 bit VIRGO Linux \*) An example eventnet logging with utils virgo\_eventnet\_log() works now without tcp\_sendmmsg() related stalls in previous builds \*) Return Datatypes for all EventNet operations have been sanitized (struct socket was returned as int in 32 build and reinterpret-cast to struct socket\*. This reinterpret cast does not work in 64 bit) in eventnet header. \*) utils eventnet log in init() has been updated with a meaningful edge update message \*) kern.log for this has been added to eventnet/testlogs

) telnet requests to VIRGO memory(kernelmemcache), cpu and filesystem modules work after resolving issues with return value types \*) commented le32\_to\_cpu() and print\_buffer() which was suppressing lot of log messages. \*) VIRGO <driver> ops structures have been updated with correct datatypes. \*) reinterpret cast of struct socket to int has been completely done away with which could have caused 64bit specific panics \*) lot of kern.log(s) and screen captures have been added for telnet requests in testlogs/ of respective <driver> directory \*) Prima facie 64bit telnet requests to VIRGO module listeners are relatively stabler than 32bit \*) Previous code changes should be relevant to 32 bit VIRGO kernel too. \*) tcp\_sendmmsg()/tcp\_recvmmsg() related hangs could be mostly related to corrupted skbuff queue within each socket. \*) This is because replacing kernel\_<send/recv>mmsg() with sock\_<send/recv>mmsg() causes return value to be 0 while socket release crashes within skbuff related kernel functions. \*) To make socket state immutable, in VIRGO memory driver header files, client socket has been declared as const type.

) Rebuilt KingCobra 64bit kernel module \*) telnet requests to VIRGO64 Queueing module listener driver are serviced by KingCobra servicerequest \*) Request\_Reply queue persisted for this VIRGO Queue + KingCobra routing has been committed to c-src/testlogs. \*) kern.log for this routing has been committed in VIRGO64 queueing directory \*) Similar to other drivers struct socket reinterpret cast to int has been removed and has been made const in queuesvc kernel thread

\*) All testcases have been rebuilt \*) VIRGO kernel memcache,cpu and filesystem system calls have been updated with set\_fs()/get\_fs() blocks for kernel\_sendmmsg() and kernel\_recvmmsg() \*) Of these virgo\_clone() system call testcase (test\_virgo\_clone) works flawlessly and there are no tcp\_sendmmsg()/tcp\_recvmmsg() related kernel panics. \*) VIRGO memcache and filesystem system call testcases have usual tcp\_sendmmsg()/tcp\_recvmmsg() despite the kernel socket code being similar to VIRGO clone system call \*) Logs for VIRGO clone system call to CPU kernel driver module have been committed to virgo\_clone/test/testlogs

\*) Changed iovec in virgo\_clone.c to kvec \*) test\_virgo\_filesystem.c and test\_virgo\_malloc.c VIRGO system calls testcases have been changed with some additional printf(s) in userspace \*) virgo\_malloc.c has been updated with BUF\_SIZE in iov\_len and memset to zero initialize the buffer. tcp\_sendmmsg()/tcp\_recvmmsg() pair were getting stuck in copy\_from\_iter\_full() memcpy with a NULL Dereference. memcpy() was reading past the buffer bound causing an overrun. strlen() didnt work for iov\_len. \*) virgo\_fs.c virgo\_write() memcpy has been changed back to copy\_from\_user() thereby restoring status quo ante (commented more than 3 years ago because of a kernel panic in older versions of 32 bit VIRGO kernel) \*) Logs for VIRGO kmemcache and filesystem system calls have been committed to respective system call directories. \*) With this all VIRGO64 functionalities work in both telnet and system calls requests routes end-to-end from clients to kernel modules with kernel sockets issues resolved fully. \*) Major findings are: - VIRGO 4.10.3 64 bit kernel is very much stable compared to 32 bit 4.1.5 kernel - there are no i915 related errors which happened in VIRGO 32 bit 4.1.5 kernel - Repetitive telnet and system calls requests to VIRGO modules are stable and there are no kernel panics like 4.1.5 32 bit kernel. - Google Kernel Address Sanitizer is quite helpful in finding stack overruns, null derefs, user memory accesses etc., - 64 bit kernel is visibly faster than 32 bit. - Virgo Unique ID is now extended to 2^64 with unsigned long long.

\*) Changed return value of virgo\_cloud\_free\_kernelspace() to a string literal "kernel memory freed" \*) Logs for VIRGO64 memory and filesystem calls to memory and filesystem drivers requests routing have been committed in test\_logs of both driver directories

Residual logs for VIRGO 64 bit 4.10.3 kernel committed.

\*) Changed LOOPBACK to INADDR\_ANY for VIRGO64 kernel memcache listen port \*) All VIRGO64 RPC, kernel memcache, cloud filesystem primitives have been retested \*) VIRGO64 mempool binaries have been rebuilt

(\*) VIRGO64 cloudfs driver has been rebuilt after changing virgofstest.txt file creation filp\_open() call (\*) Screenshots and logs for VIRGO64 Clone, Kernel MemCache and Cloud FS SystemCalls-Drivers interaction, socket transport debug messages and Kernel Address Sanitizer have been committed in virgo-docs/systemcalls\_drivers

(\*) Recently released mainline kernel version 4.13 integrates SSL/TLS into kernelspace- KTLS - for the first time.  
 (\*) KTLS is a standalone kernel module `af_ktls` (<https://github.com/ktls>) implemented by RedHat and Facebook for optimizing SSL handshake within kernelspace itself and reduce userspace-kernelspace switches. (\*) `sendfile()` system call in linux which is used for file transmission (combining read+write) from one fd to another uses this KTLS optimization in kernelspace in `af_ktls` codebase (`af_ktls` tool) (\*) VIRGO Linux kernel fork-off requires this kernelspace TLS functionality to fully secure traffic from system call client to remote

cloud node's kernel module listeners

(\*) Hence VIRGO64 linux kernel mainline base is urgently upgraded from 4.10.3 to 4.13.3 (\*) All system calls and kernel module code in VIRGO64 now have `#include(s)` for `tls.h` and invoke `kernel_setsockopt()` on the client-server kernelspace sockets for `SOL_TLS` and `TLS_TX` options and have been rebuilt. (\*) VIRGO64 RPC clone/kmemcache/cloudfs system calls to kernel module listeners have been tested with this new KTLS socket option, on rebuilt VIRGO64 kernel overlay-ed on 4.13.3 64-bit linux kernel (\*) 4.13 mainline kernel also has SMB CIFS bug fixes for recent malware attacks (WannaCry etc.,) which further ensures security of VIRGO64 linux fork-off kernelspace traffic. (\*) New buildscript for 4.13.3 linux kernel has been committed (\*) testlogs for VIRGO64 system calls and driver listeners KTLS transport have been committed in `virgo-docs/systemcalls_drivers/kern.log.VIRGO64_SystemCalls_Drivers.4.13.3_KTLS_kernelsockets.22September2017` (\*) After this upgrade, complete system calls to driver listener traffic is SSL enabled implicitly. (\*) Updated kernel object files for 4.13.3 build are part of this commit.

Updated `init.h` and `syscalls.h` headers for virgo system calls

(\*) VIRGO64 CPU/KMemCache/CloudFS system calls have been invoked by userspace testcases and all primitives work after 4.13.3 KTLS upgrade (\*) Some small modifications to system calls code have been made and rebuilt to remove redundant `iovbuf` variables in `printk(s)` (\*) `test_virgo_filesystem.c` testcase has been updated and rebuilt (\*) `kern.log(s)` for CPU/KMemCache/CloudFS systemcalls to driver listeners invocations have been committed to respective system call directories (\*) `virgofstest.txt` written to by `virgo_write()` has also been committed. But a weird behaviour is still observed similar to previous linux kernel versions (4.1.5 and 4.10.3): Repetitive invocations are required to flush the filesystem buffer to force write the file. (\*) No DRM GEM i915 panics are observed and stability of VIRGO64 + 4.13.3 linux kernel is more or equal to VIRGO64 + 4.10.3 linux kernel

(\*) New branch `VIRGO_KTLS` has been created after previous commit on 25 September 2017 and all 5 commits after 25 September 2017 till 28 September 2017 have been branched to `VIRGO_KTLS` (which has the `#ifdef` for `crypto_info`, reads from `/etc/virgo_ktls.conf` and a new `ktls` driver module) (\*) Following are the commit hashes and commandlines in GitHub and SourceForge:

```
git branch -b VIRGO_KTLS git branch master git rebase -i <SHA1_on_25September2017> git rebase
--continue git commit --amend git push --force
```

```
1960 ls 1961 git checkout VIRGO_KTLS 1962 git push origin VIRGO_KTLS 1963 git status 1964 git check-
out 1965 git checkout -b 1966 git branch 1967 git branch master 1968 git branch -h 1969 git branch
1970 git checkout master 1975 git checkout -b 1976 git checkout -b VIRGO_KTLS 1979 git push origin
VIRGO_KTLS 1990 git rebase -i bb661e908cba2a5357414e89166f29086a28bdf0 1991 git status 1992 git rebase -
i bb661e908cba2a5357414e89166f29086a28bdf0 1996 git rebase --continue 1997 git commit --amend 2019 git re-
base -i bb661e908cba2a5357414e89166f29086a28bdf0 2029 git rebase --continue 2037 git push --force 2091 git re-
base -i bb661e908cba2a5357414e89166f29086a28bdf0 2092 git rebase --continue 2093 git commit --amend 2094 git
push --force 2110 git branch 2111 git branch master 2112 git checkout master 2113 git branch 2114 git rebase -i
e76b4089633223f610fddc0e0eaff8c2cef8b9f1 2115 git commit --amend 2116 git rebase --continue 2117 git push --force
----- KTLS in 4.13.3 has support
```

for only private symmetric encryption. It does not support Public Key Encryption yet. Since this might take a while mainstream VIRGO64 code might change a lot for other features. Therefore, `VIRGO_KTLS` specific `crypto_info` code has been branched off and would parallelly evolve if PKI features are available on KTLS in next versions of Linux kernel.

`kern.log(s)` for VIRGO64 systemcalls-driver 4.13.3 64-bit upgrade tests on master branch after reversal and rebase of master HEAD yesterday for branching to `VIRGO_KTLS`. There is a weird General Protection Fault in intel atomic



commit work not seen thus far. Also -32 and -107 socket errors are frequent after reversal though code remains same. But all virgo clone/kmemcache/fs systemcalls functionalities work in invocations after GPF.

(\*) Utils Generic Socket Client function `virgo_eventnet_log()` for EventNet kernel module listener was repeatedly failing in `kernel_connect()` emitting -32 and -107 errors. (\*) `kernel_connect()` was guarded by `set_fs()` and `get_fs()` memory segment routines to prevent any memory corruption. After this stack out-of-bounds error was reported by kernel address sanitizer in `tcp_sendmsg()` and `copy_from_iter_full()` implying the message buffer was not properly read within kernel. (\*) After replacing `strlen(buf)` by `BUF_SIZE` in `msg` flags before `kernel_connect()` stack out-of-bounds error has been remedied and Utils to EventNet `virgo_eventnet_log()` doesn't crash in `tcp_sendmsg()` (\*) `kern.log` for this has been committed in `drivers/virgo/utils/testlogs/` (\*) Both eventnet and utils drivers have been rebuilt

(\*) `kernel_setsockopt()` for KTLS has been commented in all system calls and drivers because KTLS functionality has been branched to `VIRGO_KTLS` (\*) In `virgo_clone.c`, `iov.iov_len` has been set to `BUF_SIZE` (\*) `kernel_connect()` has been guarded by `set_fs()/get_ds()` in `VIRGO64` system call clients (\*) `test_virgo_malloc.c` testcase has been updated (\*) There was a weird problem in `in4_pton()`: `sin_addr.saddr` was not set correctly from string IP address and this was randomly occurring only in `virgo_set()` (\*) `in4_pton()` is implemented in `net/core/utils.c` and reads the string IP address digits and sums up the ASCII values to hex representation of the address. Bitwise operations within this functions were failing randomly. (\*) Repeated builds were done trying different possible fixes but didn't work e.g casting `saddr` to `(u8*)` (\*) There is an alternative `in_pton()` function which takes only String IP address and returns address as `__be32` (\*) After `in_pton()` in `virgo_set()` random faulty address conversion does not occur - `in_pton()` is differently implemented compared to `in4_pton()` (\*) `msg_hdr` has been initialized to `NULL` in `virgo_set()` (\*) Lot of debug `printk()`s have been added (\*) `kern.log` (.tar.gz) for RPC clone/KMemCache/Filesystem systemcalls-driver has been committed to `virgo-docs/systemcalls_drivers` (\*) `VIRGO` Linux build steps have been updated for example commandlines to overlay mainline kernel tree by `VIRGO64` source

commit 4e6681ade4ddbf1bed17f7c115b59a5ebf884256 Author: K.Srinivasan <[ka.shrinivaasan@gmail.com](mailto:ka.shrinivaasan@gmail.com)> Date: Fri Oct 6 11:36:15 2017 +0530

(\*) telnet client connection to `VIRGO64` Queue and a subsequent workqueue routing (pub/sub) to `KingCobra64` has been tested on 4.13.3 (\*) `TX_TLS` socket option has not been disabled and is a no-op because it has no effect on the socket. (\*) `REQUEST_REPLY.queue` for this routing from `VIRGO64` queue and persisted by `KingCobra64` has been committed to `KingCobra64` repositories in GitHub and SourceForge

commit d4e95b58474838d65da9c69944c6287acbdfe72c Author: K.Srinivasan <[ka.shrinivaasan@gmail.com](mailto:ka.shrinivaasan@gmail.com)> Date: Fri Oct 6 11:05:21 2017 +0530

(\*) `VIRGO64` Telnet Clients to Driver listeners invocations have been tested by telnet connections (\*) Master branches in SourceForge and GitHub `VIRGO64` do not have KTLS provisions. Only `VIRGO_KTLS` branch has `crypto_info` and `setsockopt()`

for `TX_TLS` for kernel sockets.

(\*) It has been already mentioned in NeuronRain Documentation in <https://neuronrain-documentation.readthedocs.io/en/latest/> on securing `VIRGO` cloud nodes in the absence of KTLS - most obvious solution is to install VPN client-servers in all nodes which create Virtual IPs

on a secure tunnel (e.g OpenVPN).

**(\*) `VIRGO64` system call clients and driver listeners should read these Virtual IPs from `/etc/virgo_client.conf` and `/etc/virgo_cloud.conf` and cloud traffic is confined to the VPN tunnel.**

(\*) `VIRGO64` systemcalls have been invoked from unit test cases (`test_<system_call>`) in a loop of few hundred iterations (\*) No DRM GEM i915 panics or random crashes are observed and stability is good (\*) This is probably the first loop iterative testing of `VIRGO` system calls and drivers. (\*) Kernel logs for this have been committed to `virgo-docs/systemcalls_drivers` directory. (\*) Note on concurrency: Presently mutexing within system calls have been commented because in past linux versions mutexing within kernel was causing strange panic issues. As a design choice and feature-stability tradeoff (stability is more important than introducing additional code) mutexing has been lifted up to userspace. It is upto the user applications invoking the system calls to synchronize multiple user threads invoking

VIRGO64 system calls i.e VIRGO64 system calls are not re-entrant. This would allow just one kernel thread (mapped 1:1 to a user thread)

to execute in kernel space. Mostly this is relevant only to kmemcache system calls which have global in-kernel-memory address translation tables and next\_id variable. VIRGO clone/filesystem calls do not have global in-kernel-memory datastructures.

(\*) VIRGO64 systemcalls are invoked in a function which is called from 2 processes concurrently (\*) Mutexes between the processes are PTHREAD\_PROCESS\_SHARED attribute set. (\*) test\_virgo\_malloc.c unit testcase has been enhanced to fork() a process and invoke systemcalls in a function for 100 iterations each (\*) Logs for the Virgo Unique IDs malloc/set/get/free in the systemcalls side and kern.logs for the drivers have been committed to test/testlogs/ (\*) No DRM GEM i915 crashes were observed (\*) test\_virgo\_malloc.c testcase demonstrates the coarse grained critical section lock/unlock for kmemcache systemcalls and is a template that should be followed for any userspace application.

(#) Presently kernel analytics config have to be read from a file storage. This is a huge performance bottleneck when frequency of analytics variables written to is realtime. For example autonomous vehicles/drones write gigabytes of navigation data in few minutes. (#) Because of this /etc/virgo\_kernel\_analytics.conf grows tremendously. File I/O in linux kernel module is also fragile and not recommended. (#) Previous latency limitations necessitate an alternative high performance analytics config variable read algorithm (#) This commit introduces new streaming kernel analytics config reading function - It connects to a kernel analytics streaming IP address on hardcoded port 64000 and reads analytics key-value pairs in an infinite loop. (#) These read key-value pairs are stored in a kernel global ring buffer exported symbol (by modulus operator). Because of circular buffer, older kernel analytics variables are overwritten repetitively. (#) kernel socket message flags are set to MSG\_MORE | MSG\_FASTOPEN | MSG\_NOSIGNAL for high response time. MSG\_FASTOPEN works with no panic in 4.13.3 64-bit which was a problem in previous kernel versions. (#) kern.log for this has been committed to kernel\_analytics/testlogs/ (#) include/linux/virgo\_kernel\_analytics.h header file has been updated for declarations related to streaming analytics. (#) Webserver used for this is netcat started on port 64000 as:

```
nc -l 64000 >k1=v1 >k2=v2 ...
```

- (1) This is sequel to previous commit for Stream reading Kernel Analytics variables over a network socket
- (2) read\_streaming\_virgo\_kernel\_analytics\_conf() function is invoked in a separate kernel thread because module init is blocked

(#) VIRGO64 config module was loaded and exported kernel analytics variables read over socket by previous spin-off kernel thread are imported in VIRGO64 config init. (#) kern.log for this has been committed to testlogs/ (#) Pre-requisite: Webservice serving kernel\_analytics variables must be started before kernel\_analytics module is loaded in kernel. (#) By this a minimum facility for live reading analytics anywhere on cloud (it can be userspace or kernelspace) and exporting them to modules on a local cloud node kernel has been achieved - ideal for cloud-analytics-driven IoT

- (1) VIRGO64 System Calls to Drivers invocations on 4.13.3 kernel have been executed after enabling streaming kernel analytics
- (2) VIRGO64 RPC/KMemCache/CloudFS Drivers import, streamed variable-value pairs exported from kernel\_analytics read from netcat webservice
- (3) VIRGO64 KMemCache testcase has 2 concurrent processes invoking kmemcache systemcalls in a loop.
- (4) kern.log for this has been committed to virgo-docs/systemcalls\_drivers
- (5) virgofstest.txt written by CloudFS systemcalls-drivers invocation is also committed to virgo-docs/systemcalls\_drivers

1. VIRGO64 system calls to drivers interactions so far have been tested only on dual core 64 bit architecture.
2. In quad core 64 bit there have been random -32,-107, -101 errors in kernel\_connect() from system call clients to driver services in almost all three types of VIRGO64 system calls - clone/kmemcache/filesystem - to respective drivers
3. These errors do not occur if following in4\_pton() invocation is changed to in\_aton() before kernel\_connect() in system call clients and kernel is rebuilt with following change before kernel\_connect():

```
lin4_pton(vaddr->hstprt->hostip, strlen(vaddr->hstprt->hostip), (u8)&sin.sin_addr.s_addr,  
'0',NULL);*/
```

```
sin.sin_addr.s_addr=in_aton(vaddr->hstprt->hostip);
```

4. Since this problem occurs erratically and only on quadcore 64-bit and reasons for these random -32,-101,-107 errors are still unknown, no commit for this code change has been made and this issue is left as documented known issue.
5. Most likely the u8\* cast causes client socket address corruption.
6. Because of random -32,-101,-107 errors, in quadcore, system calls sometimes do not transmit client side commandline primitive strings to driver services.
1. Two analytics usecases mentioned in NeuronRainUsecases.txt in NeuronRain AsFer asfer-docs/ have been illustrated in this commit by 2 example drivers for PXRC Drone controller Driver and UVC Video WebCam Driver. 2. PXRC Phoenix RC flight controller is part of linux kernel from 4.17 and kernel major version has been bumped to 5.x.x recently. 3. Linux kernel 5.1.4 has been built by a new build script - buildscript\_5.1.4.sh. 4. Linux kernel 5.1.4 has recent versions of PXRC drone controller driver and a UVC video webcam driver (<http://www.ideasonboard.org/uvic/faq/>) 5. New directory linux-kernel-5.1.4-extensions/ has been created for VIRGO64 code built on kernel mainline version 5.1.4. No branch is created for version 5.1.4 because pxrc driver is part of kernel only from 4.17 while code in linux-kernel-extensions/ is based on kernel 4.13.3 for dual core 64-bit architecture. 6. New VIRGO64 build on 5.1.4 kernel is necessary only for PXRC, UVC and kernel\_analytics drivers while other VIRGO64 drivers have not been ported to 5.1.4 and are still on 4.13.3 kernel. 7. Two drivers for PXRC and UVC Webcam in 5.1.4 have been committed under linux-kernel-5.1.4-extensions/drivers/media/usb/uvic and linux-kernel-5.1.4-extensions/drivers/input/joystick 8. VIRGO64 kernel\_analytics driver for 5.1.4 has been committed under linux-kernel-5.1.4-extensions/drivers/virgo/kernel\_analytics. 9. Porting VIRGO64 kernel\_analytics driver from 4.13.3 to 5.1.4 required changing vfs\_read() of /etc/virgo\_kernel\_analytics.conf to kernel\_read() in config file read. 10. Drivers code for PXRC is in drivers/input/joystick/pxrc.c has been instrumented with few printk() statements that print the virgo\_kernel\_analytics\_conf array variable-value pairs exported by VIRGO64 kernel\_analytics driver. Kernel analytics variables are imported by #include of virgo\_config.h 11. Drivers code for UVC Video WebCam is in drivers/media/usb/uvic. File drivers/media/usb/uvic/uvic\_video.c has been instrumented with lot of uvc\_trace() statements which print kernel\_analytics driver exported analytics variable "match" and its boolean value. Kernel analytics variables are imported by #include of virgo\_config.h. Variable "match" has been set to "True" assuming a face recognition or retinal scan match by userspace analytics and /etc/virgo\_kernel\_analytics.conf has been written to. 12. UVC Video WebCam traces are enabled by:

```
echo 0xffff > /sys/module/uvicvideo/parameters/trace
```
13. Example kern.log(s) for PXRC and UVC drivers are committed under drivers/input/joystick/testlogs/kern.log.pxrc.28May2019 and drivers/media/usb/uvic/kern.log.uvicvideo.28May2019 which show UVC traces printing the imported kernel analytics variable "match=True" and PXRC driver registration. No other PXRC traces are printed because of lack of drones and drone licensing dependency
14. Both UVC and PXRC drivers, VIRGO64 kernel\_analytics driver and linux kernel 5.1.4 have been built on quad-core 64-bit architecture.
15. This example import of VIRGO64 kernel analytics variables into PXRC drone and UVC webcam drivers demonstrate passing of userspace analytics information to kernel for suitable action.
16. Driver build shell scripts have been committed to UVC and PXRC driver directories.
1. Existing workqueue underneath VIRGO64 queueing and requests routed by it to KingCobra64 messaging are old legacy workqueues which have been revamped to Concurrent Managed Workqueue which supports concurrent messaging and lot of other options in queue creation. 2. create\_workqueue() in VIRGO64 Queueing has been changed to alloc\_workqueue() of Concurrent Managed Workqueue. 3. VIRGO64 Queueing request routing to KingCobra64 messaging has been tested with CMWQ and queueing log and kingcobra64 Request-Reply Queue have been committed to respective testlogs of the drivers 4. reading from stream has been disabled in virgo\_kernel\_analytics.h 5. Reference



- CMWQ documentation - <https://www.kernel.org/doc/html/v4.11/core-api/workqueue.html> 6. Byzantine Fault Tolerance in KingCobra64 persisted queue can be made available by performant CMWQ and routing to the Replicas of REQUEST\_REPLY.queue by any of the practical BFT protocols available. 7. Most important application of CMWQ based VIRGO64-KingCobra64 is in the context of kernelspace hardware messaging in IoT,Drones and other analytics driven embedded systems. 8. An example usecase which is a mix of sync and async I/O in kernelspace:

(\*) Analytics Variables computed by userspace machine learning are read over socket stream by kernel\_analytics driver and

**exported kernelwide**

(\*) Some interested Drone driver in kernel (example PXRC) reads the analytics variables synchronously and sends reply messages asynchronously to VIRGO64 Queuing driver over kernel sockets. (\*) VIRGO Queuing routes the queued messages to KingCobra64 driver

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