ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 4, June 2012

Device Driver Fault Simulation Using KEDR

Shakti D Shekar
VJTI, Mumbai
shaktishekar10@gmail.com

Prof. Dr. B B Meshram
VJTI, Mumbai
bbmeshram@vjti.org.in

Prof. Varshapriya
VJTI, Mumbai
varshapriyajn@vjti.org.in

Abstract- The failure resilience of a system is very important for availability and reliability. The systems like servers or systems where reliability and availability is highly demanding data loss is unacceptable during failures. The study on system crash stated that most of the failures in operating system environment are due to drivers failures. The device driver failure is very important in kernel environment. Once device driver failed detected system should cope with the failure and continue its operation for zero data loss. The system availability and reliability during various types of driver faults can be tested using fault injection or fault simulation tools. In this paper we discuss related work in resilience of drivers and we have simulated the faults using free tool available under KEDR framework.

Keywords- Availability, Reliability, Fault Simulation, KEDR

1 Introduction

The device driver is a kernel module which allows kernel to access hardware. Mostly kernel crash is due to failures of drivers. The reason behind driver failure is that new devices available increase the number of driver programs which are not come with operating system code. Thus device driver programmers are different form operating system programmers. The small bug into device driver can crash whole system or makes system unstable and unreliable. Sometimes rebooting of system can cope with problem of failure, but this is not acceptable in case of need of high availability like server system. So to make system highly available faults need to be isolate with continuing its operation and to make system highly reliable zero data loss need to be maintained in case of failure. This means that once device drivers failures occurs there should be mechanism that conceals or isolate the failures and with guarantee of no data loss. Thus system should include module that can bypass all operations of device driver in case of failure detection with minimum delay and as if there was no failure inside the kernel and while doing this data should not be lost. For example server is receiving or transmitting packets over network, if network device driver failed during operations then server system must have mechanism that should isolate or conceal this failure and can continue receiving or transmitting operation as if there were no failures that is there should be no data loss because of driver failure. Thus such server system must have fault detection mechanism, fault isolation and recovery mechanism to make feel to the system that there were no faults in the system. In this paper we have discussed fault simulation mechanism using free tool available in KEDR frame work on Linux platform. The paper is organized as follows: section 2 deals with related work, section 3 deals with required architecture, section 4 deals with KDER frame work, section 5 deals with fault simulation, section 6 deals with fault detection, section 7 deals with future work and finally we concluded in section 8.

2 Related Work

The research have been done on device driver failures and there recovery mechanism. For MINIX system a postmortem recovery procedure is given and continues normal operation in event of failures [1]. Here reincarnation server constantly monitors system and if problem detected reincarnation server takes action for recovery procedure. Thus it provides failure resilience.

The virtualized system like XEN adopts isolated I/O architecture which performs I/O device accesses. The switching mechanism provided to switch from faulted driver virtual machine to another one for fault tolerance [2]. In new XEN architecture driver domain is isolated from hypervisor and known as isolated driver domain (IDD). In [2], to detect faults of an IDD new mechanism introduced named Driver VM Monitor which periodically decides the fault state of IDDs

All Rights Reserved © 2012 IJARCET

and also it does not require kernel or application level modification.
Another approach is Nooks [3] which provides protected environment for driver execution. Nooks wrap calls from the operating system kernel into device drivers and from device drivers into the kernel. In Nooks all drivers execute in kernel address space but within different protection domains. In Nooks recovery is provided by restarting the driver but it does not conceal the failure.

Another approach is shadow driver [4] which is kernel agent which monitors device drivers and transparently recovers from driver failures. It conceals driver failure from its client while recovering from failure. The working of shadow driver is as follows: during normal operation shadow driver tracks the state of normal driver by monitoring all communication between the kernel and driver, when failure occurs, shadow insert itself temporarily in placed of failed driver servicing request on its behalf. The shadow driver includes Nooks and recovery system.

3 Architecture

The system to make highly available and reliable during failure, system must conceal failures and start recovery procedure. To test system for availability and reliability firstly we should have fault injection frame work or fault simulation frame work to inject or simulate faults into kernel modules manually and secondly we should have fault detection mechanism to detect faults into kernel in case of successful fault injection and thirdly very important component which conceals the faults and should take in charge of failed driver module to make feel to the system that there were no faults. While switching operation of failed driver to module who can take in charge of failed component there should be minimum switching delay and zero data loss.

4 KEDR Framework

KEDR stands for Kernel Drivers in Runtime [6]. KEDR framework provides the tools to perform runtime analysis of kernel modules like devices drivers, file system modules, etc. KEDR framework is free software and is distributed under the terms of GNU General Public License Version 2. The KEDR tools can work on kernel modules to detect memory leaks, perform fault simulation and more. It can operate on a kernel module chosen by the user as opposed to existing tools that usually operate on Linux kernel as a whole. We can work with target module and at the same time we can run KEDR tools for monitoring the operations of modules, checking if it works correctly, doing fault simulation, doing future analysis based on traced output. The KEDR cannot detect call to kmalloc because it is usually a micro or an inline function but it can detect functions which kmalloc expands. At a time we can analyze only one kernel module.

4.1 System Requirements

KEDR system can work on Linux Kernel version 2.6.31 or newer and only x86 and x86-64 architectures are supported.

5 Fault Simulation [7]

From Linux kernel 2.6.20 fault injection framework is provided to injects failures in several layers in kernel like kmalloc(), alloc_pages and disk IO errors. By injecting faults on purpose we can test behavior of kernel module after fault injection because some errors do not occurs in all situations. In MINIX operating system SWIFI (software implemented fault injection) is available for fault injection [5]. In this paper we are using a free tool for fault simulation available under KEDR frame work [9].

In kernel module sometimes some code paths rarely executes and to see behavior of kernel module for such code path the fault simulation provided by KEDR is useful. The KEDR can see such rare condition's behavior of target module.
Here for target kernel module we have used driver module from examples provided in KEDR example directory. KEDR provides special configuration file for fault simulation fsim.conf. We have run KEDR on Ubuntu Linux 11.10 with kernel 2.6.39.4.

5.1 Simulation for kmalloc

Fig. 1 shows how KEDR can be used for fault simulation. To use KEDR for fault simulation we must have kernel module. First line shows make command to build .ko file, a kernel module. After

All Rights Reserved © 2012 IJARCET

this we need to start KEDR (line no.11) for fault simulation using target module name and using configuration file. Note that here we have used

no. 22. Next we need to set scenario common for any fault simulation point. After this we have set value 1 to expression meaning that fail always.

```
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# make
make -C /lib/modules/3.0.0-12-generic/build M=`pwd` modules
make[1]: Entering directory `/usr/src/linux-headers-3.0.0-12-generic'
  CC [M]  /home/sony/Desktop/sample_target/cfake.o
  LD [M]  /home/sony/Desktop/sample_target/kedr_sample_target.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/sony/Desktop/sample_target/kedr_sample_target.mod.o
  LD [M]  /home/sony/Desktop/sample_target/kedr_sample_target.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.0.0-12-generic'
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# /usr/local/bin/kedr start
kedr_sample_target -f fsim.conf
Starting KEDR...
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr.ko target_name=kedr_sample_target
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fault_simulation.ko
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fsim_cmm.ko
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fsim_user_space_access.ko
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fsim_capable.ko
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fsim_mem_util.ko
/sbin/insmod /usr/local/lib/modules/3.0.0-12-generic/misc/kedr_fsim_vmm.ko
KEDR started.
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# insmod /usr/local/lib/modules/`uname -
r`/misc/kedr_fsim_indicator_common.ko
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "common" >
/sys/kernel/debug/kedr_fault_simulation/points/kmalloc/current_indicator
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "1" >
/sys/kernel/debug/kedr_fault_simulation/points/kmalloc/expression
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# insmod kedr_sample_target.ko
insmod: error inserting 'kedr_sample_target.ko': -1 Cannot allocate memory
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "0" >
/sys/kernel/debug/kedr_fault_simulation/points/kmalloc/expression
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# insmod kedr_sample_target.ko
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target#
```

Fig. 1

sample_target as an example from the examples provided by KEDR and for configuration we have used fsim.conf, a configuration file provided by KEDR. Now KEDR is loaded and connected with our sample kernel module. At this point KEDR loads many fault simulation modules like kedr_fault_simulation.ko, kedr_fsim_cmm.ko and so on. Now to make KEDR aware of all types of simulation we need to load module kedr_fsim_indicator_common.ko as shown at line

By doing this we have set the scenario that module will not get memory for target module loading operation. To check this we have executed insmod for loading of target module and we got exactly what we want. At line no. 29 we got an error message that cannot allocate memory. At next line we have set value 0 to expression to make fault simulation fail. This value is by default meaning that never simulate the fault. At next line we have executed insmod and this time no error message

All Rights Reserved © 2012 IJARCET

that means successful loading of target module that is memory got allocated to target module.

13. Now to make these function always failed we have set value 1 to expression file in both

```
...
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# insmod /usr/local/lib/modules/`uname -
r`/misc/kedr_fsim_indicator_common.ko
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "common" >
/sys/kernel/debug/kedr_fault_simulation/points/copy_from_user/current_indicator
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "common" >
/sys/kernel/debug/kedr_fault_simulation/points/copy_to_user/current_indicator
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# insmod kedr_sample_target.ko
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "This is fault simulation by KEDR" >
/dev/cfake0
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# cat /dev/cfake0
This is fault simulation by KEDR
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "1" >
/sys/kernel/debug/kedr_fault_simulation/points/copy_to_user/expression
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "1" >
/sys/kernel/debug/kedr_fault_simulation/points/copy_from_user/expression
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# echo "This is fault simulation by KEDR to
make operation failed" > /dev/cfake0
bash: echo: write error: Bad address
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target# cat /dev/cfake0
cat: /dev/cfake0: Bad address
root@sony-VPCEB14EN:/home/sony/Desktop/sample_target#
```

Fig. 2

5.2 Simulation for copy_from_user and copy_to_user

Here in next example we have simulated faults for copy_from_user and copy_to_user functions using same kernel module example provided by KEDR example directory. In fig. 2 we have skipped the output of make and starting KEDR using fsim.conf file because that will be same as before. After this again load the common fault simulation module of KEDR that is kedr_fsim_indicator.ko. Now to set fault simulation point set value "common" to current indicator in copy_from_user and copy_to_user directories as shown at line no. 3 and line no.6 of the fig. 2. After this we have loaded the target module. As here we have to simulate faults for the functions which copies data from and to user for device drivers, we need to read data from or write data into loaded target module that is device driver read-write operations. This is shown as successful operations at line no. 11 and line no. copy_to_user and copy_from_user as shown at line no. 15 and line no. 17. Now after doing read-write operations on target module we got an error message "Bad address". This means that fault got simulated.

6 Fault Detection

The fault detection can be done by looking at errors from standard error output or by looking at error log files or checking by return value from target module operation.

7 Future Work

The reliability and availability of system is very important and for this continuous research is going on. In case of failures due to kernel modules like device drivers we must have mechanism to cope with failures. To check system reliability and availability we have proposed very generic

All Rights Reserved © 2012 IJARCET

mechanism that will cope with various types of failures in kernel due to device drivers.

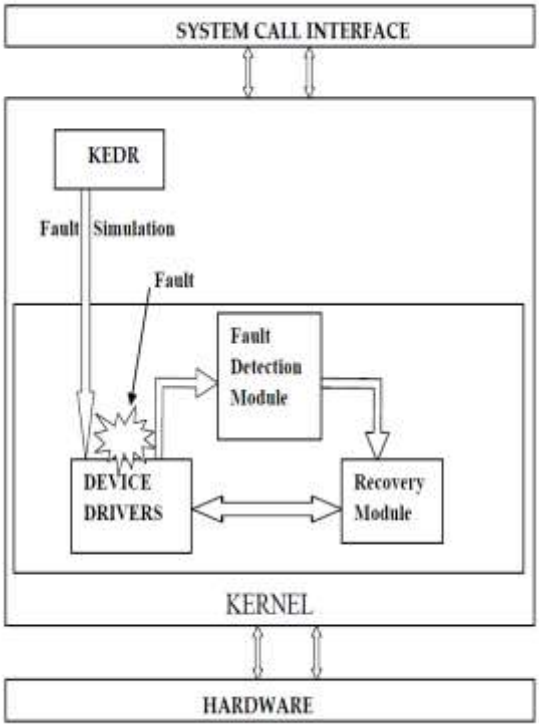

Fig. 3

Fig. 3 shows that there are four modules in the kernel, first the KEDR which is for fault simulation, second device drivers, third fault detection module which will continuously monitors device drivers operations using return value and if it success in failure detection then it will trigger forth module recovery module. The recovery module will have mechanism to cope with failures and while doing this work we need to maintain minimum delay and no data loss.

8 Conclusion

The operating system is reliable and available only when it can cope with failures and make its services continue without affecting. The device driver failures concealing and recovering mechanism with zero data loss can improve operating system reliability and availability. The KEDR frame work can be used to simulate faults into kernel modules. In this paper we have discussed and showed the KEDR fault simulation mechanism for important operations of device drivers that is memory allocation and device driver read-write operations. Thus to simulate the faults into dynamic kernel modules KEDR is very useful.

REFREENCES

[1] Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S. Tanenbaum. "Failure Resilience for Device Drivers" 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07).

[2] Heeseung Jo, Hwanju Kim, Jae-Wan Jang, Joonwon Lee, and Seungryoul Maeng. "Transparent Fault Tolerance of Device Drivers for Virtual Machines" IEEE TRANSACTIONS ON COMPUTERS, VOL. 59, NO. 11, NOVEMBER 2010

[3] Michael M. Swift, Steven Martin, Henry M. Levy, and Susan J. Eggers "Nooks: An Architecture for Reliable Device Drivers" Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France, Sept. 2002.

[4] Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy "Recovering Device Drivers" Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation, San Francisco, CA, Dec. 2004. .

[5]http://nooks.cs.washington.edu/

[6]http://kedr.berlios.de/

[7]http://kedr.berlios.de/kedrdoc/ kedr_manual_getting_stared.html

All Rights Reserved © 2012 IJARCET