KSTAT: Linux Kernel Module to collect Network Statistics

Arpit Gupta

In this assignment, we build a kernel-level Network Statistics collection module. You are assigned to run a kernel module which collects statistics for outgoing packets. The statistical information that needs to be collected is **Total number of outgoing TCP packets.**

Guidelines

1. To get started, refer to [1] and previous HW to learn how to write a simple “Hello World” kernel module, how to compile it and how to load and unload a kernel module.
2. To implement your KSTAT, you need to first intercept the outgoing packets. The netfilter [2] mechanism in Linux provides you a convenient way to intercept packets. You can intercept packets at 5 hooks: PRE_ROUTING, POST_ROUTING, LOCAL_IN, LOCAL_OUT and FORWARD. You need to figure which hook(s) you should use to intercept packets for NAT. To use netfilter, you need to first register a hook function when the module is initialized. Then, whenever a packet traverse to the hook you registered, your hook function will be called to process the packet. In this problem your hook function will update/ create the STAT Structure. The packet is represented in a sk_buff structure, which is a very complex and huge structure. Refer to [2 3 4] for details about it. When you finish processing, you may return NF_ACCEPT to let the packet continue traversal as normal or NF_DROP to drop the packet. Finally, unregister the hook when exiting the module. As sample code for understanding is provided below.
3. You are required to report the outgoing packets as a proc file. The data should be stored in the proc directory “/proc/kstat/nout”. Please refer to [6,7,9] for an example of using proc file system.
4. Maintain a proc entry by the name “counter”(“/proc/kstat/counter”), which determines after how many packets a proc entry should be updated. For example if counter=10, then you will update proc entry after 10 packets have arrived at the netfilter. Provide counter=10 as the default value. Note that this entry can be changed while the module is running and it impacts the granularity at which proc file entry is updated.
5. You are also required to implement sysfs to start stop the STAT service. Refer [8] for details. Note that loading the module is different from starting the STAT service. When you load the module, the module is dynamically linked to the kernel. However, STAT service is not started until a sysfs parameters is set appropriately. Eg.

```
#> echo “1” > /sys/module/kstat/parameters/start
stop KSTAT:
#> echo “0” > /sys/module/kstat/parameters/start
```

Linux Distributions and Kernel Versions

Everyone has his/her own preference for different Linux distributions. Although I don't want to change your habit, you MUST implement this project with kernel version 2.6 or above. Even within 2.6, there are numerous subversions which contain considerable differences. To make everyone's life easier, I suggest you use Ubuntu 10.04 with the latest update . That's where I will test your codes. If you use other distributions/kernel versions, it's your responsibility to make sure that your code works under my test platform. Or you may demonstrate you code to me on your laptop

Grading

You need to turn in one your_unity_id_project2.tar.gz file after you tar-gzip all your source codes, Makefile and README. The points will be given by the following distribution.
Basic Functionality: 60
Procfs: 20
Sysfs:10
Comments and README: 10

Sample Code:

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/in.h>
static struct nf_hook_ops netfilter_ops_in;
unsigned int main_hook(unsigned int hooknum,
struct sk_buff *skb,
const struct net_device *in,
const struct net_device *out,
int (*okfn)(struct sk_buff*))
```

```
{
struct iphdr *iph;
if (!skb)return NF_ACCEPT;
iph = ip_hdr(skb);
if (!iph)return NF_ACCEPT;
if (iph->protocol==IPPROTO_TCP)
{
printk("got a TCP packet\n");
// your code here
}
return NF_ACCEPT;
}
static int __init init(void)
{
netfilter_ops_in.hook = main_hook;
netfilter_ops_in.pf = PF_INET;
netfilter_ops_in.hooknum = NF_INET_PRE_ROUTING;//choose your own hook
netfilter_ops_in.priority = NF_IP_PRI_FIRST;
nf_register_hook(&netfilter_ops_in);
return 0;
}
static void __exit cleanup(void)
{
nf_unregister_hook(&netfilter_ops_in);
}
module_init(init);
module_exit(cleanup);
MODULE_LICENSE("GPL");
```

References:

1. The Linux Kernel Module Programming Guide http://www.linuxtopia.org/online_books/linux_kernel/linux_kernel_module_programming_2.6/x121.html
2. Linux netfilter Hacking HOWTO http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.html & http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-3.html
3. How SKBs work http://vger.kernel.org/~davem/skb.html[
4. skb - Linux network buffers http://ftp.gnumonks.org/pub/doc/skb-doc.html
5. skbuff.net http://www.skbuff.net/skbuff.html
6. Access the Linux kernel using the /proc filesystem, available at http://www.ibm.com/developerworks/linux/library/l-proc.html

7. http://www.linux.com/learn/linux-training/37985-the-kernel-newbie-corner-kernel-debugging-using-proc-qsequenceq-files-part-1
8. Linux Kernel Module Programming Guide – Passing Command Line Arguments to a Module, available at http://www.linuxtopia.org/online_books/Linux_Kernel_Module_Programming_Guide/x323.html & http://www.makelinux.net/books/lkd2/ch16lev1sec6
9. Procfs Tutorial: http://www.stillhq.com/pdfdb/000445/data.pdf