**实验四报告要求**

Linux内核编程可用于扩展操作系统的功能，如TCP/IP协议栈中的拥塞控制算法可采用Linux内核编程的方式实现。在编译Linux内核时，需要安装必要的编译环境，具体命令如下：

$**sudo apt-get install build-essential linux-headers-$(uname -r)**

为了快速感受Linux内核编程的魅力，接下来给出一个学习编程语言时常用的Hello World程序，它在以内核模块的方式实现。具体的内核代码(**hello.c**)如下：

#include <linux/module.h> /\* Needed by all modules \*/

#include <linux/kernel.h> /\* Needed for KERN\_INFO \*/

#include <linux/init.h> /\* Needed for the macros \*/

///< The license type -- this affects runtime behavior

MODULE\_LICENSE("GPL");

///< The author -- visible when you use modinfo

MODULE\_AUTHOR("Norbert");

///< The description -- see modinfo

MODULE\_DESCRIPTION("A simple Hello world LKM!");

///< The version of the module

MODULE\_VERSION("0.1");

static int \_\_init hello\_start(void)

{

printk(KERN\_INFO "Loading hello module...\n");

printk(KERN\_INFO "Hello world\n");

return 0;

}

static void \_\_exit hello\_end(void)

{

printk(KERN\_INFO "Goodbye Mr.\n");

}

module\_init(hello\_start);

module\_exit(hello\_end);

为了编译源码方便，此处我们用make工具进行管理，需编写**Makefile**文件，具体Makefile内容如下：

obj-m = hello.o

all:

make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

将上述两个文件放置在同一个目录下，然后执行：$**make**。如果正确编译，可以得到**hello.ko**文件，利用命令$**sudo insmod hello.ko**将新模块插入内核中。如果想要移除插入的模块，可执行命令：$**sudo rmmod hello.ko**。为了查看内核模块的输出，可以执行命令：$**dmesg**。

上述是一个内核版本的Hello World程序完整实现和使用，接下来，大家需要自己完成以下内容：

1. 一个输出内核时间的内核模块；
2. 一个发送速率稳定的拥塞控制模块；

**实验提交内容**：实验提交两个模块，并完成实验报告。

[**每位同学独立完成，遇到疑问可询问助教和教师**]

第2个模块的参考代码：

#include <linux/module.h>

#include <net/tcp.h>

#include <linux/random.h>

#include <linux/mm.h>

#include <linux/inet\_diag.h>

#define CBR\_RATE\_MIN 1024u

struct cbr {

/\* rate control variables \*/

s64 rate; /\* current delivery rate \*/

};

static u32 cbr\_get\_rtt(struct tcp\_sock \*tp) //it's ok

{

if (tp->srtt\_us) {

return max(tp->srtt\_us >> 3, 1U);

} else {

return USEC\_PER\_MSEC;

}

}

/\* Initialize cwnd to support current pacing rate (more then 4 packets) \*/

static void cbr\_set\_cwnd(struct sock \*sk) //it's ok

{

struct tcp\_sock \*tp = tcp\_sk(sk);

u64 cwnd = sk->sk\_pacing\_rate;

cwnd \*= cbr\_get\_rtt(tcp\_sk(sk));

cwnd /= tp->mss\_cache;

cwnd /= USEC\_PER\_SEC;

cwnd \*= 2;

cwnd = max(4ULL, cwnd);

cwnd = min((u32)cwnd, tp->snd\_cwnd\_clamp); /\* apply cap \*/

tp->snd\_cwnd = cwnd;

}

/\* NetRate Main Function \*/

static void cbr\_main\_process(struct sock \*sk, const struct rate\_sample \*rs)

{

}

static void cbr\_init(struct sock \*sk)

{

struct cbr \*cbrt = inet\_csk\_ca(sk);

cbrt->rate = CBR\_RATE\_MIN\*512; //512KBps or 4Mbps

//cbrt->rate = CBR\_RATE\_MIN\*6\*1024;

tcp\_sk(sk)->snd\_ssthresh = TCP\_INFINITE\_SSTHRESH;

cmpxchg(&sk->sk\_pacing\_status, SK\_PACING\_NONE, SK\_PACING\_NEEDED);

sk->sk\_pacing\_rate = cbrt->rate;

cbr\_set\_cwnd(sk);

}

static void cbr\_cong\_avoid(struct sock \*sk, u32 ack, u32 acked) //it's ok

{

}

static void cbr\_pkts\_acked(struct sock \*sk, const struct ack\_sample \*acks) //it's ok

{

}

static void cbr\_ack\_event(struct sock \*sk, u32 flags) //it's ok

{

}

static void cbr\_cwnd\_event(struct sock \*sk, enum tcp\_ca\_event event) //it's ok

{

}

static void cbr\_release(struct sock \*sk) //it's ok

{

}

static u32 cbr\_undo\_cwnd(struct sock \*sk) //it's ok

{

return tcp\_sk(sk)->snd\_cwnd;

}

static u32 cbr\_ssthresh(struct sock \*sk) //it's ok

{

return TCP\_INFINITE\_SSTHRESH; /\* CBR does not use ssthresh \*/

}

static struct tcp\_congestion\_ops tcp\_cbr\_cong\_ops \_\_read\_mostly = {

.flags = TCP\_CONG\_NON\_RESTRICTED,

.name = "cbr",

.owner = THIS\_MODULE,

.init = cbr\_init,

.cong\_control = cbr\_main\_process,

//.set\_state = cbr\_set\_state,

/\* Keep the windows static \*/

.undo\_cwnd = cbr\_undo\_cwnd,

.release = cbr\_release,

/\* Slow start threshold will not exist \*/

.ssthresh = cbr\_ssthresh,

.cong\_avoid = cbr\_cong\_avoid,

.pkts\_acked = cbr\_pkts\_acked,

.in\_ack\_event = cbr\_ack\_event,

.cwnd\_event = cbr\_cwnd\_event,

};

/\* Kernel module section \*/

static int \_\_init cbr\_register(void)

{

BUILD\_BUG\_ON(sizeof(struct cbr) > ICSK\_CA\_PRIV\_SIZE);

printk(KERN\_INFO "cbr init reg\n");

return tcp\_register\_congestion\_control(&tcp\_cbr\_cong\_ops);

}

static void \_\_exit cbr\_unregister(void)

{

tcp\_unregister\_congestion\_control(&tcp\_cbr\_cong\_ops);

}

module\_init(cbr\_register);

module\_exit(cbr\_unregister);

MODULE\_AUTHOR("Xianliang Jiang <norbert.jiang@gmail.com>");

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_DESCRIPTION("TCP CBR");

ifneq ($(KERNELRELEASE),)

# kbuild part of makefile

obj-m := tcp\_cbr.o

else

# normal makefile

KDIR ?= /lib/modules/`uname -r`/build

default:

$(MAKE) -C $(KDIR) M=$$PWD

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

endif