搭建Linux核心開發Debug環境

方案

- 我的測試系統在QEMU中運行, Host和Guest的架構都是x86_64,用Busybox生成的initrd做為根文件系統, KGDB做為調試器,
- Kernel : linux-3.15.5
- Busybox : busybox-1.22.1

產生核心

- 核心中需要打開的選項是

```
CONFIG_EXPERIMENTAL=y
CONFIG_DEBUG_INFO=y
CONFIG_KGD=y
CONFIG_KGDB_SERIAL_CONSOLE=y
CONFIG_DEBUG_RODATA=n
```

- time make -j8 2>&1 | tee build.log

產生根文件系統

- 打開 Busybox 選項

```
CONFIG_STATIC=y
CONFIG_INSTALL_NO_USR=y
```

- time make -j8 2>&1 | tee build.log
- make install

創建initrd根文件系統

- mkdir temp && cd temp
- 創建系統目錄

```
mkdir p dev etc/init.d mnt proc root sys tmp
chmod a+rwxt tmp
cp rf ../busybox/_install/* ./
```

- 掛載系統目錄

```
cat << EOF > etc/fstab
proc   /proc  proc  defaults  0  0
sysfs   /sys  sysfs defaults  0  0
tmpfs   /tmp  tmpfs defaults  0  0
EOF
```

```
cat << EOF > etc/inittab
::sysinit:/etc/init.d/rcS
::respawn:/bin/sh
tty2::askfirst:/bin/sh
::ctrlaltdel:/bin/umount a r
EOF
```

```
#! /bin/sh
MAC=08:90:90:59:62:21
IP=192.168.100.2
Mask=255.255.255.0
Gateway=192.168.100.1


/sbin/ifconfig lo 127.0.0.1
ifconfig eth0 down
ifconfig eth0 hw ether $MAC
ifconfig eth0 $IP netmask $Mask up
route add default gw $Gateway

/bin/mount -a
/bin/mount -t  sysfs sysfs /sys
/bin/mount -t tmpfs tmpfs /dev
/sbin/mdev -s

mount -o remount,rw,noatime -n /dev/root /
```

- chmod 755 etc/init.d/rcS
- 產生 rootfs, 可在temp 下新增修改檔案

```
find ./ | cpio o H newc | gzip > ../rootfs.img
```

安裝 tftp

- sudo apt-get install tftp-hpa tftpd-hpa
- /etc/default/tftpd-hpa

```
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="-l -c -s"
```

- 建立 tftpboot

```
sudo mkdir /tftpboot
sudo chmod 777 /tftpboot
```

- restart service

```
sudo service tftpd-hpa restart
```

- test tftp

```
tftp host—ip  // ex tftp localhost
get or put
```

啟動 tap

- sudo ./nettap.sh // 必須加上 sudo

```
tunctl -u shihyu -t tap0
ifconfig tap0 192.168.100.1 up
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -I FORWARD 1 -i tap0 -j ACCEPT
iptables -I FORWARD 1 -o tap0 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

- 如果成功下ifconfig 會出現下面 message

```
tap0      Link encap:Ethernet  HWaddr 0a:18:ee:c4:f0:d0
          inet addr:192.168.100.1  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

啟動QEMU

```
qemu-system-x86_64 -kernel bzImage -append "root=/dev/ram rdinit=/sbin/init" -
initrd rootfs.img  -k en-us -net nic,model=virtio -net tap,ifname=tap0,script=
no
```

```
qemu-system-x86_64 -kernel bzImage -append "root=/dev/ram rdinit=/sbin/init" -
initrd rootfs.img  -k en-us -net nic,model=virtio -net tap,ifname=tap0,script=
no  -gdb tcp::1234 -S
```

啟動 GDB

```
target remote localhost:1234
```

- 比如在sched_clock函數處設置斷點 break sched_clock , continue 繼續運行, 到達斷點後打印jiffies_64變量 print jiffies_64 等等
- 另外, 運行過程中可以在測試系統裡執行 echo g > /proc/sysrqtrigger 讓gdb重新得到控制權

Hello world

- gcc -static test.c -o test //記得加上 -static , 因為沒動態函數庫支援

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("\nHello world\n");
    return 0;
}
```

載入 Linux Module

- globalmem.c

```
/*======================================================================
    A globalmem driver as an example of char device drivers

    The initial developer of the original code is Baohua Song
    <author@linuxdriver.cn>. All Rights Reserved.
======================================================================*/
```

```c
#include <linux/module.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/mm.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <asm/io.h>
//#include <asm/system.h>
#include <asm/uaccess.h>
#include <linux/slab.h>

#define GLOBALMEM_SIZE  0x1000  /*全局内存最大4K字节*/
#define MEM_CLEAR 0x1  /*清0全局内存*/
#define GLOBALMEM_MAJOR 245    /*预设的globalmem的主设备号*/

static int globalmem_major = GLOBALMEM_MAJOR;
/*globalmem设备结构体*/
struct globalmem_dev {
    struct cdev cdev; /*cdev结构体*/
    unsigned char mem[GLOBALMEM_SIZE]; /*全局内存*/
};

struct globalmem_dev* globalmem_devp; /*设备结构体指针*/
/*文件打开函数*/
int globalmem_open(struct inode* inode, struct file* filp) {
    /*将设备结构体指针赋值给文件私有数据指针*/
    filp->private_data = globalmem_devp;
    return 0;
}
/*文件释放函数*/
int globalmem_release(struct inode* inode, struct file* filp) {
    return 0;
}

/* ioctl设备控制函数 */
static int globalmem_ioctl(struct inode* inodep, struct file* filp, unsigned
                           int cmd, unsigned long arg) {
    struct globalmem_dev* dev = filp->private_data;/*获得设备结构体指针*/

    switch (cmd) {
    case MEM_CLEAR:
        memset(dev->mem, 0, GLOBALMEM_SIZE);
        printk(KERN_INFO "globalmem is set to zero\n");
        break;

    default:
        return  - EINVAL;
    }

    return 0;
}
```

```c
/*读函数*/
static ssize_t globalmem_read(struct file* filp, char __user* buf, size_t size
,
                              loff_t* ppos) {
    unsigned long p =  *ppos;
    unsigned int count = size;
    int ret = 0;
    struct globalmem_dev* dev = filp->private_data; /*获得设备结构体指针*/

    /*分析和获取有效的写长度*/
    if (p >= GLOBALMEM_SIZE) {
        return count ?  - ENXIO : 0;
    }

    if (count > GLOBALMEM_SIZE - p) {
        count = GLOBALMEM_SIZE - p;
    }

    /*内核空间->用户空间*/
    if (copy_to_user(buf, (void*)(dev->mem + p), count)) {
        ret =  - EFAULT;
    } else {
        *ppos += count;
        ret = count;

        printk(KERN_INFO "read %d bytes(s) from %d\n", count, p);
    }

    return ret;
}

/*写函数*/
static ssize_t globalmem_write(struct file* filp, const char __user* buf,
                               size_t size, loff_t* ppos) {
    unsigned long p =  *ppos;
    unsigned int count = size;
    int ret = 0;
    struct globalmem_dev* dev = filp->private_data; /*获得设备结构体指针*/

    /*分析和获取有效的写长度*/
    if (p >= GLOBALMEM_SIZE) {
        return count ?  - ENXIO : 0;
    }

    if (count > GLOBALMEM_SIZE - p) {
        count = GLOBALMEM_SIZE - p;
    }

    /*用户空间->内核空间*/
    if (copy_from_user(dev->mem + p, buf, count)) {
        ret =  - EFAULT;
    } else {
        *ppos += count;
        ret = count;

        printk(KERN_INFO "written %d bytes(s) from %d\n", count, p);
    }

    return ret;
}

/* seek文件定位函数 */
static loff_t globalmem_llseek(struct file* filp, loff_t offset, int orig) {
    loff_t ret = 0;

    switch (orig) {
    case 0:   /*相对文件开始位置偏移*/
        if (offset < 0) {
            ret =  - EINVAL;
            break;
        }

        if ((unsigned int)offset > GLOBALMEM_SIZE) {
            ret =  - EINVAL;
            break;
        }

        filp->f_pos = (unsigned int)offset;
        ret = filp->f_pos;
        break;

    case 1:   /*相对文件当前位置偏移*/
        if ((filp->f_pos + offset) > GLOBALMEM_SIZE) {
            ret =  - EINVAL;
            break;
        }

        if ((filp->f_pos + offset) < 0) {
            ret =  - EINVAL;
            break;
        }

        filp->f_pos += offset;
        ret = filp->f_pos;
        break;

    default:
        ret =  - EINVAL;
        break;
    }

    return ret;
}

/*文件操作结构体*/
```

```c
static const struct file_operations globalmem_fops = {
    .owner = THIS_MODULE,
    .llseek = globalmem_llseek,
    .read = globalmem_read,
    .write = globalmem_write,
    .unlocked_ioctl = globalmem_ioctl,
    .open = globalmem_open,
    .release = globalmem_release,
};

/*初始化并注册cdev*/
static void globalmem_setup_cdev(struct globalmem_dev* dev, int index) {
    int err, devno = MKDEV(globalmem_major, index);

    cdev_init(&dev->cdev, &globalmem_fops);
    dev->cdev.owner = THIS_MODULE;
    dev->cdev.ops = &globalmem_fops;
    err = cdev_add(&dev->cdev, devno, 1);

    if (err) {
        printk(KERN_NOTICE "Error %d adding LED%d", err, index);
    }
}

/*设备驱动模块加载函数*/
int globalmem_init(void) {
    int result;
    dev_t devno = MKDEV(globalmem_major, 0);

    /* 申请设备号*/
    if (globalmem_major) {
        result = register_chrdev_region(devno, 1, "globalmem");
    } else { /* 动态申请设备号 */
        result = alloc_chrdev_region(&devno, 0, 1, "globalmem");
        globalmem_major = MAJOR(devno);
    }

    if (result < 0) {
        return result;
    }

    /* 动态申请设备结构体的内存*/
    globalmem_devp = kmalloc(sizeof(struct globalmem_dev), GFP_KERNEL);

    if (!globalmem_devp) {  /*申请失败*/
        result =  - ENOMEM;
        goto fail_malloc;
    }

    memset(globalmem_devp, 0, sizeof(struct globalmem_dev));

    globalmem_setup_cdev(globalmem_devp, 0);
    return 0;
fail_malloc:
    unregister_chrdev_region(devno, 1);
    return result;
}

/*模块卸载函数*/
void globalmem_exit(void) {
    cdev_del(&globalmem_devp->cdev);   /*注销cdev*/
    kfree(globalmem_devp);     /*释放设备结构体内存*/
    unregister_chrdev_region(MKDEV(globalmem_major, 0), 1); /*释放设备号*/
}

MODULE_AUTHOR("Song Baohua");
MODULE_LICENSE("Dual BSD/GPL");

module_param(globalmem_major, int, S_IRUGO);

module_init(globalmem_init);
module_exit(globalmem_exit);
```

- Makefile

```
obj-m   += globalmem.o
KDIR    = /home/shihyu/data/work/linux-3.15.5

EXTRA_CFLAGS=-g

build:kernel_modules

kernel_modules:
    make -C $(KDIR) M=$(CURDIR) modules

clean:
    make -C $(KDIR) M=$(CURDIR) clean
```

```
insmod globalmem.ko
mknod /dev/globalmem c 245 0
# section.sh globalmem > gdb
# put.sh gdb
```

- gdb 加入 ./globalmem.ko ; 0xffffffffa0000000 是 .text address

```
add-symbol-file ./globalmem.ko 0xffffffffa0000000 \
   -s .bss 0xffffffffa0000900 \
   -s .data 0xffffffffa0000698 \
   -s .gnu.linkonce.this_module 0xffffffffa00006a0 \
   -s .note.gnu.build-id 0xffffffffa0000488 \
   -s .rodata 0xffffffffa0000560 \
   -s .rodata.str1.1 0xffffffffa00004ac \
   -s .rodata.str1.8 0xffffffffa0000508 \
   -s .strtab 0xffffffffa00025e8 \
   -s .symtab 0xffffffffa0002000 \
   -s __mcount_loc 0xffffffffa0000658 \
   -s __param 0xffffffffa0000528
```

- (gdb) source MyGdbInit_first

```
# MyGdbInit_first
target remote localhost:1234
source /tftpboot/gdb
c
```

- (gdb) source MyGdbInit_second

```
# MyGdbInit_second
b globalmem_write
b globalmem_read
c
```