CEng 536 Advanced Unix Kernel Project

Fall 2011

Due: January 21^{th}, 2011

In this project you will implement a pseudo character device driver for group communication. Each message will be delivered to subscribers only and subscription management and blocking can be possible. The devices can only start working through a simple authentication process. We will call this device as `ktwit`.
When the device is opened, reading and writing is not possible. The opening process should issue a `ioctl(fd,KTWAUTH, ...)` request to authenticate an id, which is an unsigned char value. In order to authenticate, a correct passid, a short integer should be given. After authentication, all writes to device will be sent to queues of the subscriber ids. An id can subscribe to more than one ids including itself. When device is read, the first message from a subscribed id is read.
In order to subscribe an id, an authenticated id sends a `ioctl(fd, KTWSUB,id)` request. In order to unsbuscribe, `ioctl(fd, KTWUNS,id)` request is sent. All of the messages coming from an id before subscription are lost. In other words, when an id writes a message, the message is delivered to current subscribers only. The message is not accessible for the others.
The authentication information is a map of **`unsigned char`** to **`short`** values, the id and the passid. When the device module is first loaded, this map is empty and there exists no id in the system. An unauthenticated process can issue a `ioctl(fd,KTWNEW, ...)` request to create a new id. If id is in use the request fails, otherwise the id is created with given passid and process is authenticated. An authenticated process can issue the same request to change its passid if the authenticated id matches the request. The authentication information is global to the device driver (or module), all minor devices use the same authentication map.
On the other hand the messaging is specific per minor device. An id can have different subscribers for different minor devices. Also the written messages are delivered to only authenticated subscribers of the same minor devices. A process has to authenticate for each minor device it wants to read. In other words each minor device is an instance of a different messaging system where authentication information is shared.
The module has the following parameters that can be set at module load time: ioctl() calls:

1. Prefered major number (ktw_major)
2. Number of minor devices (ktw_minor)
3. Maximum message size (ktw_maxsize)

If ktw_major is not specified, the system will automatically determine a major number. If it is specified, the provided number will be used. On module load, ktw_minor minor devices are created. Default is 4. ktw_maxsize is used to limit the size of each message written on the device. If a longer `write()` call is issued, the message is truncated to this size. Default is 256.
You need a group of `ioctl()` commands to configure each queue, They are defined within the `ktwit.h` as follows:

```
/* macros to conver a id,passid pair to single integer and vice versa  */
```

```
#define KTW_MKAUTH(id,passid)    ((passid << 16) | id)
#define KTW_ID(comb)    (unsigned char)(comb & 0xff)
#define KTW_PASSID(comb)    (short)(comb >> 16)

/* used to get the follower/subscribed sets each
   bit corressponds to an id. if set, id is in the set */

#define KTW_LBITS       (sizeof(long)*8)

struct ktw_set {
        long vals[256/KTW_LBITS];
};

#define KTW_ISSET(id,s) ( (s.vals[id / KTW_LBITS] &  \
                                (1 << (id % KTW_LBITS))) > 0)

#define KTW_IOC_MAGIC  0xfa

/*
 * S means "Set" through a ptr,
 * T means "Tell" directly with the argument value
 * G means "Get": reply by setting through a pointer
 * Q means "Query": response is on the return value
 * X means "eXchange": switch G and S atomically
 * H means "sHift": switch T and Q atomically
 */
#define KTW_IOCTAUTH    _IO(KTW_IOC_MAGIC,0)
#define KTW_IOCTNEW     _IO(KTW_IOC_MAGIC,1)
#define KTW_IOCTSUB     _IO(KTW_IOC_MAGIC,2)
#define KTW_IOCTUNSUB   _IO(KTW_IOC_MAGIC,3)
#define KTW_IOCCLR      _IO(KTW_IOC_MAGIC,4)
#define KTW_IOCTBLOCK   _IO(KTW_IOC_MAGIC,5)
#define KTW_IOCGSUBS    _IOR(KTW_IOC_MAGIC,6,int)
#define KTW_IOCGFOLL    _IOR(KTW_IOC_MAGIC,7,int)
```

These calls are explained as:

`KTW_IOCTAUTH`

This operation is used to authenticate to an id. The parameter of `ioctl()` is the id and passid information by macro KTW_MKAUTH. For example:
`ioctl(fd, KTW_IOCTAUTH, KTW_MKAUTH('A',12345))` will authenticate the id 'A' with passid `12345`.

If the given pair is in the authentication map, it succeeds (ioctl returns 0), otherwise -1 is returned and error no is set to EINVAL.

`KTW_IOCTNEW`

Similar to KTW_IOCTAUTH but creates the id and authenticates the process.
`ioctl(fd, KTW_IOCTNEW, KTW_MKAUTH('A',12345))`

If process is not authenticated yet, it tries to insert the id,passid pair into authentication map. If id exists in the map, it returns -1 and error no is set to EINVAL. Otherwise the id,passid pair is inserted and authentication succeeds.

If process is already authenticated and the authentication id is same with the id in the parameter, the given passid is updated on the authentication map (it works like password change). If ids are different it fails and error number is set to EINVAL.

All ioctl requests but KTW_IOCTNEW fails with error number EBADF if the process is not authenticated yet.

`KTW_IOCTSUB`
It is used to subscribe an id.
`ioctl(fd, KTW_IOCTSUB, 'A')`
After this call the currently authenticated id will get messages sent by id 'A'.

Note that the subscription information is persistent across multiple open/close sessions on the same minor device. It is kept per minor device until the module is unloaded.

If the id is already subscribed, it is silently ignored and operation succeeds.

`KTW_IOCTUNSUB`
It is used to unsubscribe an id.
`ioctl(fd, KTW_IOCTUNSUB, 'A')`
After this call the currently authenticated id will not get further messages sent by id 'A'.

If the id is not subscribed, it is silently ignored and operation succeeds.

`KTW_IOCCLR`
It is used to clear currently unread messages by the current id. `ioctl(fd, KTW_IOCCLR)`
After this call, all messages waiting to be read by the current id are deleted from the queue.

`KTW_IOCTBLOCK`
It is used for id to stop another id from getting messages. `ioctl(fd, KTW_IOCTBLOCK, 'A')`
After this call, the subscription of id 'A' to current id is broken and further subscription requests from 'A' fails. There is no way of unblocking a blocked id.

`KTW_IOCGSUBS`
It is used for getting a list of subscribed ids of current id. `struct ktw_set sset`
`ioctl(fd, KTW_IOCGSUBS, &sset)`
call will put the list of all subscribed ids into sset structure.

`KTW_IOCGFOLL`
It is used for getting a list of subscriber ids of current id. `struct ktw_set sset`
`ioctl(fd, KTW_IOCGFOLL, &sset)`
call will put the list of all followers into sset structure.

Implement this device driver with open, read, write, close, ioctl , and poll functions in Linux. You can use the `scull` sample device driver code given in the sources section of the home page:
`http://www.ceng.metu.edu.tr/courses/ceng536/sources/ldd3-samples-1.0.1`
Also you need to use the guidelines in the online book:
`http://www.ceng.metu.edu.tr/courses/ceng536/documents/ldd3_pdf/`
Please look at *Qemu* utility in all linux distributions. It is a processor simulator that can boot the kernel of your machine. It will help building and testing without crashing a working system.

```
sudo apt-get install qemu     # install qemu if it does not exist
sudo apt-get install debootstrap     # create a minimal debian

# now create a debian image
dd if=/dev/zero of=debian-img bs=1024 count=400000
mkfs.ext3 -L debian debian-img          # approve the next question
```

```
#mount it
sudo mount debian-img /mnt

# create a minimal debian system on image
sudo debootstrap sid /mnt ftp://ftp.linux.org.tr/debian

# edit /mnt/etc/shadow, delete the * in root entry like root::....
sudo umount /mnt

# start qemu if 32 bit system, use qemu
# replace XXX with the appropriate files under boot
qemu-system-x86_64  -kernel /boot/vmlinuzXXX \
        -initrd /boot/initrd.imgXXX -hda debian-sid -append root=/dev/sda

#login to emulated machine by root/no password
# then:
dhclient eth0
apt-get install ssh

# now you can use sftp,ssh to 10.0.2.2 to transfer files from host
```