Linux Device Drivers – IOCTL

Jernej Vičič

March 15, 2018
ioctl system call.
short for: Input Output ConTroL,
shared interface for devices,
Description of ioctl

- devices are presented with files,
- input and output devices,
- we use read/write,
- this is not always enough,
- example: (old) modem,
Description of ioctl - modem

- connected through serial port,
- How to control the serial port:
  - set the speed of transmission (baudrate).
- use `ioctl` :).
Description of ioctl - examples

- control over devices other than the type of read/write,
- close the door,
- eject,
- error display,
- setting of the baud rate,
- self destruct :).
each device has its own set,
these are commands,
read commands (read ioctls),
write commands (write ioctls),
three parameters:
  file descriptor,
  ioctl command number
  a parameter of any type used for programmers purposes.
the usage of the third parameter depends on ioctl command,
the actual command is the second parameter,
possibilities:
  - no parameter,
  - integer,
  - pointer to data.
each device has its own set of commands,
the control of these commands is left to the programmers,
there is a tendency to use other means of communicating with
devices:
  - to include commands in a data stream,
  - use of virtual file systems (sysfs, proprietary),
  - ...
Still: ioctl is the easiest way to communicate with device
drivers.
int ioctl(int fd, unsigned long cmd, ...);

- userspace function signature,
- *fd* – file descriptor,
- *cmd* – number of the command (name),
- strange signature (dots),
- usually means a variable number of parameters,
- on system calls this is not possible,
- this is a trick that for the compiler, I
- n this case, the compiler does not check the parameter type,
- only 1 optional parameter,
- it is usually *char* * argp*. 
int (*ioctl) (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg);

- signature of the kernel space function,
- `inode` - the pointer to the inode,
- `filp` - the pointer to the file,
- `cmd` - the command number given by the user function,
- `arg` - an additional command parameter sent in `long` format even if it was specified as a pointer,
ioctl instructions

- command numbers should be unique for all devices,
- thus reducing the possibility of errors,
- selection of the number is described in `include/asm/ioctl.h` and `Documentation/ioctl-number.txt`.
symbols are defined in `linux/ioctl.h`,

*type* – magic number, for all `ioctl` of the device should be the same (8 bits),

*number* – number in series (8 bits),

*direction* – direction of data flow from the point of view of the application:
  - `_IOC_NONE` – no data flow,
  - `_IOC_READ` – read from device,
  - `_IOC_WRITE` – write to device,
  - `_IOC_READ—_IOC_WRITE` – read and write,

*size* – user data size (usually 13 or 14 bits),
ioctl, choose instruction number

helper macros:
- \texttt{IO(type,nr)} – no arguments,
- \texttt{_IOR(type,nr,datatype)} – read from device,
- \texttt{_IOW(type,nr,datatype)} – write to device,
- \texttt{_IOWR(type,nr,datatype)} – read and write,
/* Use ’k’ as magic number */
#define SCULL_IOC_MAGIC ’k’
/* Please use a different 8-bit number in your code */
#define SCULL_IOCRESET _IO(SCULL_IOC_MAGIC, 0)

/* 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 SCULL_IOCSQUANTUM _IOW(SCULL_IOC_MAGIC, 1, int)
#define SCULL_IOCSQSET _IOW(SCULL_IOC_MAGIC, 2, int)
#define SCULL_IOCTQUANTUM _IO(SCULL_IOC_MAGIC, 3)
#define SCULL_IOCTQSET _IO(SCULL_IOC_MAGIC, 4)
#define SCULL_IOCGQUANTUM _IOR(SCULL_IOC_MAGIC, 5, int)
#define SCULL_IOCGQSET _IOR(SCULL_IOC_MAGIC, 6, int)
#define SCULL_IOCQQUANTUM _IO(SCULL_IOC_MAGIC, 7)
#define SCULL_IOCQQSET _IO(SCULL_IOC_MAGIC, 8)
#define SCULL_IOCXQUANTUM _IOWR(SCULL_IOC_MAGIC, 9, int)
#define SCULL_IOCXQSET _IOWR(SCULL_IOC_MAGIC,10, int)
#define SCULL_IOCHQUANTUM _IO(SCULL_IOC_MAGIC, 11)
#define SCULL_IOCQHSET _IO(SCULL_IOC_MAGIC, 12)
#define SCULL_IOC_MAXNR 14
ioctl - return value

- ob napaki:
  - correct: -ENOTTY ("inappropriate ioctl for device"),
  - also possible: -EINVAL ("Invalid argument"),
recognized by the kernel,

3 groups,

- can be performed on each file (ordinary file, device, FIFO, plug),
- can be performed on ordinary files,
- specific to the file system.
Ukazi ioctl - predefined instructions

- **FIOCLEX** – set flag (File IOctl CLose on EXec) "set close-on-exec flag", at the start of a new program the file descriptor is closed,

- **FIONCLEX** – unset flag (File IOctl Not CLos on EXec) "Clear the close-on-exec flag", opposite of the upper command,

- **FIOASYNC** – not used, sets/unsets asynch messages for the file,

- **FIOQSIZE** – file size or directory size,

- **FIONBIO** – “File IOctl Non-Blocking I/O”. 
ioctl - use of the third argument

- if it is an integer, no problem,
- if it is a pointer, ...
<asm/uaccess.h>;

int access_ok(int type, const void *addr, unsigned long size);

- check if the address is valid
- type – choose VERIFY_READ or VERIFY_WRITE,
- addr – address in user space,
- size – byte count, for int: sizeof(int).
int err = 0, tmp;
int retval = 0;

/*
 * extract the type and number bitfields, and don’t decode
 * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok()
 */
if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC) return -ENOTTY;
if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;

/*
 * the direction is a bitmask, and VERIFY_WRITE catches R/W
 * transfers. ‘Type’ is user-oriented, while
 * access_ok is kernel-oriented, so the concept of "read" and
 * "write" is reversed
 */
if (_IOC_DIR(cmd) & _IOC_READ)
    err = !access_ok(VERIFY_WRITE, (void __user *)arg, _IOC_SIZE(cmd));
else if (_IOC_DIR(cmd) & _IOC_WRITE)
    err = !access_ok(VERIFY_READ, (void __user *)arg, _IOC_SIZE(cmd));
if (err) return -EFAULT;
- driver-internal variables,
- piece of memory, that communicates through ioctl,
- applications that communicates with this driver,
- 3 files:
  - query_ioctl.h,
  - query_ioctl.c,
  - query_app.c.
Example II – usage

3 properties:
- Status,
- Dignity,
- Ego.

Uporaba:
- ./query_app – to display the driver variables,
- ./query_app -c – to clear the driver variables,
- ./query_app -g – to display the driver variables,
- ./query_app -s – to set the driver variables (not mentioned above).
switch(cmd) {
    case SCULL_IOCRESET:
        scull_quantum = SCULL_QUANTUM;
        scull_qset = SCULL_QSET;
        break;
    case SCULL_IOCSQUANTUM: /* Set: arg points to the value */
        if (! capable(CAP_SYS_ADMIN))
            return -EPERM;
        retval = __get_user(scull_quantum, (int __user *)arg);
        break;
    case SCULL_IOCTQUANTUM: /* Tell: arg is the value */
        if (! capable(CAP_SYS_ADMIN))
            return -EPERM;
        scull_quantum = arg;
        break;
    case SCULL_IOCGQUANTUM: /* Get: arg is pointer to result */
        retval = __put_user(scull_quantum, (int __user *)arg);
        break;
    case SCULL_IOCQQUANTUM: /* Query: return it (it’s positive) */
        return scull_quantum;
}
case SCULL_IOCXQUANTUM: /* eXchange: use arg as pointer */
    if (! capable (CAP_SYS_ADMIN))
        return -EPERM;
    tmp = scull_quantum;
    retval = __get_user(scull_quantum, (int __user *)arg);
    if (retval == 0)
        retval = __put_user(tmp, (int __user *)arg);
    break;

case SCULL_IOCHQUANTUM: /* sHift: like Tell + Query */
    if (! capable (CAP_SYS_ADMIN))
        return -EPERM;
    tmp = scull_quantum;
    scull_quantum = arg;
    return tmp;

default: /* redundant, as cmd was checked against MAXNR */
    return -ENOTTY;
}

return retval;
int quantum;
ioctl(fd, SCULL_IOCSQUANTUM, &quantum); /* Set by pointer */
ioctl(fd, SCULL_IOCTQUANTUM, quantum); /* Set by value */
ioctl(fd, SCULL_IOCGQUANTUM, &quantum); /* Get by pointer */
quantum = ioctl(fd, SCULL_IOCQQUANTUM); /* Get by return value */
ioctl(fd, SCULL_IOCXQUANTUM, &quantum); /* Exchange by pointer */
quantum = ioctl(fd, SCULL_IOCHQUANTUM, quantum); /* Exchange by value */