Systems Programming

Devices

H. Turgut Uyar Şima Uyar

2001-2009

Topics

I/O Subsystem
- Introduction
- Software
- Accessing Devices

Device Drivers
- Interface
- Implementation
- Advanced Operations

I/O Devices

- O/S controls all I/O devices
  - issues commands to devices
  - catches interrupts
  - handles errors
  - provides interface

Device Categories

- character devices
- block devices
- network interfaces
- clocks and timers

Character Devices

- stream of characters
- arbitrary-sized data transfer
- not addressable
  - no seek operation

Example

- console, terminals
- mice
- sound card
- serial/parallel port

Block Devices

- can host a filesystem
- data transfer in fixed-size blocks
- each block has its own address
  - read/write each block independently

Example

- disks

Device Controllers

- devices consist of:
  - mechanical components
  - electronic components: *device controller*
- O/S deals with controller
  - connected through a standard interface
  - SCSI, USB, Firewire, ...

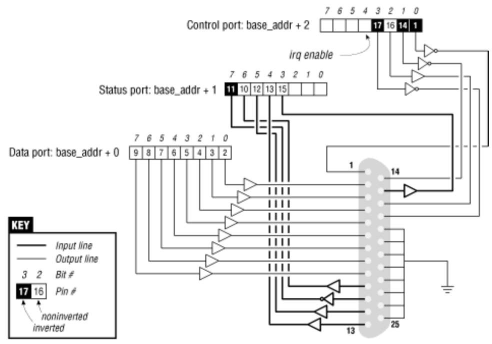
Controller Registers

- CPU communicates with controller through registers
- input/output register
- control register
  - sending commands to device
- status register
  - get/set state of device

Parallel Interface

- parallel interface base addresses on PC:
  - 0x378
  - 0x278
- ports:
  - +0: bidirectional data register
  - +1: status register (read-only): online, out-of-paper, busy
  - +2: control register (write-only): enable/disable interrupts

Parallel Interface



I/O Architecture

ports

- special address space for I/O
  - separate lines for I/O ports
  - special instructions

memory-mapped

- registers part of regular address space
- directly-mapped
  - part of address space reserved for I/O
  - virtual memory management disabled for that part
- software-mapped
  - I/O space part of virtual memory

I/O Software

- abstraction
  - standardized interface
  - uniform naming
- encapsulation
  - device drivers
- layering
  - lower layers hide the hardware specific operations
  - higher layers provide easy-to-use, regular interface to users

Device Issues

- blocking vs interrupt-driven
  - better for CPU to work interrupt-driven
  - better for user-space programs to work blocking
  - O/S makes interrupt-driven operations look blocking
- shared vs dedicated
  - shared between processes at the same time
  - dedicated to one process at a time

I/O Services

- copy semantics
  - transfer the snapshot of data at the time of the I/O request
- scheduling
  - issue order may not be the best execution order
- adapt between devices with different data-transfer sizes
- caching
- spooling, device reservation
- error handling

Spooling

- for dealing with dedicated devices in multi-tasking environments
- a daemon for controlling the device
- a spooling directory

Software Layers

- user-space applications
- device-independent software
- device drivers
- interrupt handlers

Interrupt Handlers

- interrupts hidden from rest of system
  - requesting process is blocked until I/O is completed
- when I/O is completed, interrupt occurs
  - process is made to unblock

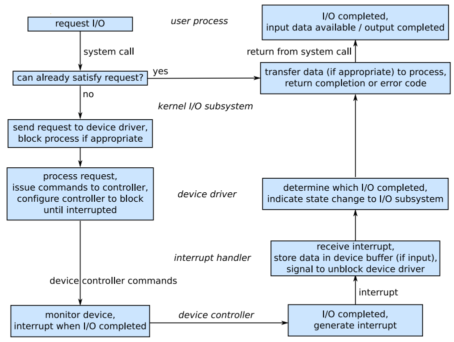
Device Drivers

- device-dependent code
- a driver for each device type
- accept request from device-independent software
- decide on sequence of controller operations

Device-Independent Software

- functions common to all devices
- uniform interface to user-level software
- device naming
- device protection
- provide device-independent block sizes
- buffering
- allocating and releasing dedicated devices
- error reporting

I/O Life Cycle



Accessing Devices

- directly
  - using ports or memory
- through device drivers
  - using the device driver interface

Direct Access

- special instructions
  - input: inb inw inl
  - output: outb outw outl
- get permission from O/S

ioperm system call

```
int ioperm(unsigned long from,
           unsigned long num,
           int turn_on);
```


Device Access Example

Example (Output to parallel interface)

```
ioperm(0x378, 1, 255);      mov  eax,101
outb(0xff, 0x378);          mov  ebx,378h
                            mov  ecx,1
                            mov  edx,255
                            int  80h

                            mov  dx,378h
                            mov  al,ffh
                            out  dx,al
```


Reading Material

- Silberschatz, 5/e
  - Chapter 12: I/O Systems

Device Drivers

- hiding details of the hardware's communication protocols
- providing a standard interface
  - Unix: same as the file interface:
    open, read, write, close

Device-Specific Operations

- some operations are neither read nor write
- ioctl : issue device-specific commands

Example (device-specific operations)

- eject CDROM
- make the speaker beep
- set communication parameters for modem

System Calls

open system call

```c
int open(const char *pathname,
         int flags,
         mode_t mode);
```

- return: file descriptor
- flags:
  - O_RDONLY O_WRONLY O_RDWR
  - O_CREAT O_APPEND
- mode: permissions

System Calls

close system call

```c
int close(int fd);
```

- return: success / failure

System Calls

read system call

```c
ssize_t read(int fd,
             void *buf,
             size_t count);
```

- return: how many bytes read (x)
  - $x = count$: successful completion
  - $0 < x < count$: partial transfer, retry remaining part
  - $x = 0$: end-of-file
  - $x < 0$: error

System Calls

write system call

```c
ssize_t write(int fd,
              const void *buf,
              size_t count);
```

- return: how many bytes written (x)
  - $x = count$: successful completion
  - $0 < x < count$: partial transfer, retry remaining part
  - $x = 0$: end-of-file
  - $x < 0$: error

System Calls

ioctl system call

```
int ioctl(int fd,
          int request,
          ...);
```

- return: depends on request
  - usually status code
- parameters of request are listed after request

Device Access Example

Example (Parallel port output)

```
...
fd = open("/dev/parport0", O_WRONLY);
if (fd == -1) {
    perror("cannot access device");
    exit(EXIT_FAILURE);
}
...
write(fd, buffer, len);
...
close(fd);
...
```


Device Specific Command Example

Example (Ejecting the CDROM)

```
int fd, status;

fd = open("/dev/cdrom", O_RDONLY);

status = ioctl(fd, CDROMEJECT);
if (status == -1) {
    perror("cannot eject CD-ROM");
    exit(EXIT_FAILURE);
}

close(fd);
```


Device Specific Command Example

Example (Making the speaker beep)

```
int fd, status, arg = 0x100011AA;

fd = open("/dev/console", O_RDWR);
status = ioctl(fd, KDMKTONE, arg);
if (status == -1) {
    perror("cannot generate beep");
    exit(EXIT_FAILURE);
}
close(fd);
```


Implementing Device Drivers

- implement system calls for device
  - use device-specific I/O instructions

Kernel System Call Interface

```
int foo_open(struct inode *inode,
             struct file *filp)
int foo_release(struct inode *inode,
                struct file *filp)

ssize_t foo_read(struct file *filp,
                 char __user *buf,
                 size_t count,
                 loff_t *f_pos)
ssize_t foo_write(struct file *filp,
                  const char __user *buf,
                  size_t count,
                  loff_t *f_pos)
```

Device Driver Example

Example (short)

- read/write I/O ports
- each device node accesses a different port:
  - `/dev/short0`: port at base
  - `/dev/short1`: port at base+1
- module parameters:
  - major number (default dynamic)
  - base address (default 0x378)

Device Driver Example

Example (global definitions)

```
#define SHORT_NR_PORTS 8

static int major = 0;
module_param(major, int, 0);

static unsigned long base = 0x378;
unsigned long short_base = 0;
module_param(base, long, 0);
```


Module Initialization

- allocate I/O region
  - base address
  - number of ports
- register the driver with the kernel
  - major-minor numbers
  - capabilities: file operations

File Operations

- mapping system calls to functions:
  struct file_operations
  - open
  - release
  - read
  - write
  - ...

File Operations Example

Example (file operations)

```
struct file_operations short_fops = {
    .owner   = THIS_MODULE,
    .open    = short_open,
    .release = short_release,
    .read    = short_read,
    .write   = short_write,
};
```


Region Allocation

Example (short_init)

```
if (!request_region(short_base,
                    SHORT_NR_PORTS,
                    "short")) {
    return -ENODEV;
}
```

Driver Registration

Example (short_init)

```
if (major) {      /* specified major number */
    dev = MKDEV(major, 0);
    result = register_chrdev_region(dev, 1,
                                    "short");
} else {          /* dynamic major number */
    result = alloc_chrdev_region(&dev, 0, 1,
                                 "short");
    major = MAJOR(dev);
}
if (result < 0) {
    /* release port region */
    return result;
}
```


Driver Registration (cont'd)

Example (short_init)

```
cdev_init(&cdev, &short_fops);
cdev.owner = THIS_MODULE;
cdev.ops = &short_fops;
cdev_add(&cdev, dev, 1);
```


Module Cleanup

Example (short_cleanup)

```
dev_t devno = MKDEV(major, 0);
cdev_del(&cdev);
unregister_chrdev_region(devno, 1);

release_region(short_base, SHORT_NR_PORTS);
```


Memory-Mapped I/O

Example

```
/* allocating region on init */
if (!request_mem_region(short_base,
                        SHORT_NR_PORTS,
                        "short")) {
    return -ENODEV;
}
short_base = (unsigned long) ioremap(short_base,
             SHORT_NR_PORTS);

/* releasing region on cleanup */
iounmap((void __iomem *) short_base);
release_mem_region(short_base, SHORT_NR_PORTS);
```


Kernel Data Structures

- a data structure for each open file:
  struct file
  - f_mode: readable, writable, both
  - f_pos: current reading/writing position
  - f_flags
  - f_op: operations associated with the file
  - private_data: pointer to allocated data
- a data structure for each device node:
  struct inode

Opening

- identify actual device
- check for device-specific errors
- initialize device
- allocate and initialize data structures

Reading

Example (short_read)

```
int retval = count;
int minor = iminor(filp->f_dentry->d_inode);
unsigned long port = short_base +
                     (minor & 0x0f);
unsigned char *kbuf, *ptr;

kbuf = kmalloc(count, GFP_KERNEL);
if (!kbuf)
    return -ENOMEM;
...
kfree(kbuf);
return retval;
```


Reading

Example (short_read)

```
ptr = kbuf;
while (count--) {
    *(ptr++) = inb(port);
    rmb();
}
if ((retval > 0) &&
        copy_to_user(buf, kbuf, retval))
    retval = -EFAULT;
```


Writing Example

Example (short_write)

```
if (copy_from_user(kbuf, buf, count))
    return -EFAULT;
ptr = kbuf;
while (count--) {
    outb(*(ptr++), port);
    wmb();
}
```


Advanced Driver Operations

- other system calls in the interface
  - ioctl, seek, ...

Kernel System Call Interface

```
loff_t foo_llseek(struct file *filp,
                  loff_t off,
                  int whence)

int foo_ioctl(struct inode *inode,
              struct file *filp,
              unsigned int cmd,
              unsigned long arg)
```

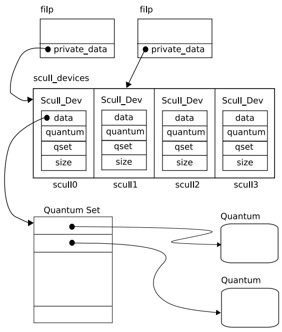

Advanced Driver Example

Example (simplified scull)

- use memory as device
  - `/dev/scull0`
  - `/dev/scull1`
- each device can hold data up to a limit
  - data persists during module's lifetime

Advanced Driver Example

Example (simplified scull)



- each device has a quantum set
- each quantum has the actual data
- memory is allocated as data is written

Global Definitions

Example (scull.h)

```
#define SCULL_MAJOR 0
#define SCULL_NR_DEVS 4
#define SCULL_QUANTUM 4000
#define SCULL_QSET 1000
```


Module Parameters

Example

```
int scull_major = SCULL_MAJOR;
int scull_minor = 0;
int scull_nr_devs = SCULL_NR_DEVS;
int scull_quantum = SCULL_QUANTUM;
int scull_qset = SCULL_QSET;

module_param(scull_major, int, S_IRUGO);
module_param(scull_minor, int, S_IRUGO);
module_param(scull_nr_devs, int, S_IRUGO);
module_param(scull_quantum, int, S_IRUGO);
module_param(scull_qset, int, S_IRUGO);
```


Data Structures

Example

```
struct scull_dev {
    char **data;
    int quantum;
    int qset;
    unsigned long size;
    struct semaphore sem;
    struct cdev cdev;
};

struct scull_dev *scull_devices;
```


Device Operations

Example

```
struct file_operations scull_fops = {
    .owner   = THIS_MODULE,
    .llseek  = scull_llseek,
    .read    = scull_read,
    .write   = scull_write,
    .ioctl   = scull_ioctl,
    .open    = scull_open,
    .release = scull_release,
};
```


Module Initialization

Example (scull_init_module)

```
  /* get major and minor numbers */
  /* allocate and initialize devices */

  return 0; /* succeed */

fail:
  scull_cleanup_module();
  return result;
```

Module Initialization

Example (get major and minor numbers)

```
if (scull_major) {
    devno = MKDEV(scull_major, scull_minor);
    result = register_chrdev_region(devno,
                scull_nr_devs, "scull");
} else {
    result = alloc_chrdev_region(&devno,
        scull_minor, scull_nr_devs, "scull");
    scull_major = MAJOR(devno);
}
if (result < 0) {
    return result;
}
```


Module Initialization

Example (allocate data structures)

```
scull_devices = kmalloc(scull_nr_devs *
       sizeof(struct scull_dev), GFP_KERNEL);
if (!scull_devices) {
    result = -ENOMEM;
    goto fail;
}
memset(scull_devices, 0, scull_nr_devs *
          sizeof(struct scull_dev));
```


Module Initialization

Example (initialize devices)

```
for (i = 0; i < scull_nr_devs; i++) {
    dev = &scull_devices[i];
    dev->quantum = scull_quantum;
    dev->qset = scull_qset;
    init_MUTEX(&dev->sem);

    devno = MKDEV(scull_major, scull_minor + i);
    cdev_init(&dev->cdev, &scull_fops);
    dev->cdev.owner = THIS_MODULE;
    dev->cdev.ops = &scull_fops;
    cdev_add(&dev->cdev, devno, 1);
}
```


Module Cleanup

Example (scull_cleanup_module)

```
dev_t devno = MKDEV(scull_major, scull_minor);

if (scull_devices) {
    for (i = 0; i < scull_nr_devs; i++) {
        scull_trim(scull_devices + i);
        cdev_del(&scull_devices[i].cdev);
    }
    kfree(scull_devices);
}

unregister_chrdev_region(devno, scull_nr_devs);
```


Trimming the Device

Example (scull_trim)

```
if (dev->data) {
    for (i = 0; i < dev->qset; i++) {
        if (dev->data[i])
            kfree(dev->data[i]);
    }
    kfree(dev->data);
}
dev->data = NULL;
dev->quantum = scull_quantum;
dev->qset = scull_qset;
dev->size = 0;
```


Opening the Device

Example (scull_open)

```
struct scull_dev *dev;

dev = container_of(inode->i_cdev,
                   struct scull_dev, cdev);
filp->private_data = dev;

/* trim device if open was write-only */
...

return 0;
```

Opening the Device

Example

```
/* trim device if open was write-only */
if ((filp->f_flags & O_ACCMODE) == O_WRONLY) {
    if (down_interruptible(&dev->sem))
        return -ERESTARTSYS;
    scull_trim(dev);
    up(&dev->sem);
}
```


Writing to the Device

Example (scull_write)

```
  struct scull_dev *dev = filp->private_data;
  int quantum = dev->quantum, qset = dev->qset;
  int s_pos, q_pos;
  ssize_t retval = -ENOMEM;

  if (down_interruptible(&dev->sem))
      return -ERESTARTSYS;

  /* write */

out:
  up(&dev->sem);
  return retval;
```


Writing to the Device

Example (write)

```
if (*f_pos >= quantum * qset) {
    retval = 0;
    goto out;
}

s_pos = (long) *f_pos / quantum;
q_pos = (long) *f_pos % quantum;

/* allocate quantum if necessary */
/* adjust write amount */
/* copy from user space */
/* update size */
```


Writing to the Device

Example (allocate quantum)

```
if (!dev->data) {
    dev->data = kmalloc(qset * sizeof(char *),
                        GFP_KERNEL);
    if (!dev->data)
        goto out;
    memset(dev->data, 0, qset * sizeof(char *));
}
if (!dev->data[s_pos]) {
    dev->data[s_pos] = kmalloc(quantum,
                               GFP_KERNEL);
    if (!dev->data[s_pos])
        goto out;
}
```


Writing to the Device

Example (copy from user space)

```
/* adjust write amount */
if (count > quantum - q_pos)
    count = quantum - q_pos;

/* copy from user space */
if (copy_from_user(dev->data[s_pos] + q_pos,
                   buf, count)) {
    retval = -EFAULT;
    goto out;
}
```


Writing to the Device

Example (update size)

```
*f_pos += count;
retval = count;

/* update size */
if (dev->size < *f_pos)
    dev->size = *f_pos;
```

Reading from the Device

Example (scull_read)

```c
struct scull_dev *dev = filp->private_data;
int quantum = dev->quantum;
int s_pos, q_pos;
ssize_t retval = 0;

if (down_interruptible(&dev->sem))
    return -ERESTARTSYS;

/* read */

out:
up(&dev->sem);
return retval;
```


Reading from the Device

Example (read)

```c
if (*f_pos >= dev->size)
    goto out;
if (*f_pos + count > dev->size)
    count = dev->size - *f_pos;

s_pos = (long) *f_pos / quantum;
q_pos = (long) *f_pos % quantum;

if (dev->data == NULL || ! dev->data[s_pos])
    goto out;

/* adjust read amount */
/* copy to user space */
```


Reading from the Device

Example (copy to user space)

```c
/* adjust read amount */
if (count > quantum - q_pos)
    count = quantum - q_pos;

/* copy to user space */
if (copy_to_user(buf, dev->data[s_pos] + q_pos,
                 count)) {
    retval = -EFAULT;
    goto out;
}
*f_pos += count;
retval = count;
```


Seeking on the Device

Example

```c
switch(whence) {
    case 0: /* SEEK_SET */
        newpos = off;
        break;
    case 1: /* SEEK_CUR */
        newpos = filp->f_pos + off;
        break;
    case 2: /* SEEK_END */
        newpos = dev->size + off;
        break;
    default: /* can't happen */
        return -EINVAL;
}
```


Seeking on the Device

Example

```c
if (newpos < 0)
    return -EINVAL;
filp->f_pos = newpos;
return newpos;
```


Device-Specific Commands

Example

- SCULL_IOCRESET:
  assign default values to quantum set size and quantum size
- SCULL_IOCSQUANTUM: set quantum size from pointer
- SCULL_IOCTQUANTUM: (tell) set quantum size from value
- SCULL_IOCGQUANTUM: get quantum size to pointer
- SCULL_IOCQQUANTUM: (query) return quantum size
- SCULL_IOCXQUANTUM: (exchange) set + get
- SCULL_IOCHQUANTUM: (shift) tell + query
- similar operations for quantum set size

Device Operations

Example

```
switch (cmd) {
    case SCULL_IOCRESET:
        scull_quantum = SCULL_QUANTUM;
        scull_qset = SCULL_QSET;
        break;

    /* other cases */
}
```


Device Operations

Example (setting quantum size)

```
case SCULL_IOCSQUANTUM:
    if (! capable (CAP_SYS_ADMIN))
        return -EPERM;
    retval = __get_user(scull_quantum,
                        (int __user *) arg);
    break;

case SCULL_IOCTQUANTUM:
    if (! capable (CAP_SYS_ADMIN))
        return -EPERM;
    scull_quantum = arg;
    break;
```


Device Operations

Example (getting quantum size)

```
case SCULL_IOCGQUANTUM:
    retval = __put_user(scull_quantum,
                        (int __user *) arg);
    break;

case SCULL_IOCQQUANTUM:
    return scull_quantum;
```


Reading Material

- Corbet-Rubini-Hartman, 3/e
  - Chapter 3: Char Drivers
  - Chapter 9: Communicating with Hardware