Device Management

(w

axgopala@imperial.ac.uk

Device Management

Objectives

Fair access to shared devices

- Allocation of dedicated devices

Expl

Provide uniform simple view of I/O

- Hide complexity of device handling
- Give uniform naming and error handling

Device Variations

Character vs. Block

Block Devices

- Stores information in fixed-size blocks
- Transfers are in units of entire

Ch

- Delivers or accepts stream of characters, without regard to block structure
- Not addressable, does not have any seek operation

How does the OS actually communicate with the hardware?

CPU and Devices Communication

Each hardware controller has a few registers used for communication with the CPU

OS can write to these registers to command the device

- Deliver data
- [illegible]
- [illegible]
- Perform some action

OS can read from these registers to learn about

- State of the device
- Whether it is ready to accept commands
- . . .

I/O Software

Goals

Device independence from

- Device type (e.g. terminal, disk or DVD drive)
- Device instance (e.g. which disk)

Uniform naming name of a file should be a string or integer and n

Devi

- Unit of data transfer: character or block
- Supported operations: e.g. read, write, see
- Synchronous or asynchronous operatio
- Speed differences
- Sharable (e.g. disks) or single user (e.g. printer, DVD-RW)
- Error handling
- Buffering

Programmed I/O

Inte

Dire

Example used: Printing a string

Programmed I/O

```
copy_from_user (buffer, p, count);      // p = kernel buffe
for (i = 0; i < count; i++) {           // loop on every char
  while (*printer_status_reg !=READY);  // loop until ready
  *printer_data_register = p[i];        // output one char
}
return_to_user ();
```

Interrupts

The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires

Interrupt Driven I/O

```
copy_from_user (buffer, p, count);
enable_interrupts ();
while (*printer_status_reg != READY);
*printer_data_register = p[0];
scheduler ();
```

```
[illegible]
} else {
  *printer_data_register = p[i];
  count = count - 1;
  i++;
}
acknowledge_interrupt ();
return_from_interrupt ();
```

Interrupt service procedure for the printer

Direct Memory Access (DMA)

Operation of a DMA transfer

DMA I/O

```
copy_from_user (buffer, p, count);
set_up_DMA_controller ();
scheduler ();
```

```
acknowledge_interrupt ();
unblock_user ();
return_from_interrupt ();
```

Interrupt service procedure

I/O Layering

I/O Layers

Overview

I/O Layers

Overview

Interrupt Handlers
Steps

1. Driver starts an I/O operation block and blocks until the I/O has completed
2.

3. When interrupts happens, the interrupt p whatever it has to in order to handle the interru

4. Then it will unblock the driver that started it

Device Drivers

Device-specific code for controlling an I/O device

A driver for a mouse differs from a driver for a HDD

Handles one device type, or at most, one class of closely related devices

Part o

Posi

cont

Most OSs define a standard interface (between O

for block devices and character devices

Must be flexible and be able to handle errors, several i

etc.

Allowed to call only a handful of system calls, e.g. to allocate memory for a buffer

Device-Independent I/O Software

Some parts are device-specific but others are device independent

There is no strict boundary between device-specific and device independent software and varies between OSs

Mos

-
- Buffering
- Error reporting
- Allocating and releasing dedicated devic
- Providing a device-independent block size

Imperial College London

Uniform Interfacing for Device Drivers I

(a) Without a standard driver interface

(b) With a standard driver interface

Uniform Interfacing for Device Drivers II

Interface between the driver and OS is defined

OS can install new driver easily and the writer of the driver knows what it can expect from the OS

In practice, not all devices are absolutely identical, but there are only a small number of device types

For ea
funct

Ofte
functions

OS records the address of the table when the driver is load
indirect calls via this table

Another aspect of having a uniform interface is how I/O devices are named: each device has a major device number and minor device number

Closely related to naming is protection → devices appear as files in the file system, so usual protection rules could be used

Buffering I

(a) Unbuffered input

(b) Buffering in user space

(c) Buffering in the kernel followed by copying to user space

(d) Double buffering in the kernel

Buffering II

Ways to handle data streams from I/O devices

- Each interrupt may wake-up user's process
- OS writes into user space's buffer and wakes up user's process once the buffer is full. What happens if the buffer is paged
- [illegible] arrives at the time when the buffer is being copied to user's space?
    - Double buffering
    - Circular buffering
- Buffering is also important for output → e.g. when sending data over a slow telephone line

Buffering III

Networking may involve multiple copies of a packet

Error Reporting

Errors are far more common in the context of I/O than in other contexts

Many errors are device-specific and must be handled by appropriate driver, but the framework for error handling is device independent

Clas

-

  there's only two
  - Solution → just report back an error co

- Actual I/O errors → write to a disk blo damaged or read from a camera that is turned off
  - Solution → it is up to the driver to decide what to do, whether to try to solve the problem or report back the error code

Imperial College London

Device-Independent Block Size

Different disks may have different sector sizes

It is up t

provi

Some devices deliver data one byte at a time (e.g. mo

others deliver theirs in larger units (e.g. network in

User-Space I/O Software I

Most of the I/O software is within the OS

Small portion of it consists of libraries linked together with user programs, and even whole programs running outside the kernel

Exa

Proc
call

Other procedures may do actual work: e.g. forma

Not all user-level I/O software consists of library procedures → another important category is the spooling system

User-Space I/O Software II

Some devices, such as CD-ROM recorders, can be used only by a single process at any given moment and cannot be shared

- Blocking user access to these devices causes delays and bottlenecks

Spool to intermediate medium (disk file)

Spoo
mult

Spooled devices (e.g. printers)

1. Printer output saved to disk file
2. File printed later by spooler daem
   - Printer only allocated to spooler daemo
   - No normal process allowed direct access

- Provides sharing of non-sharable devices
- Reduces I/O time → gives greater throughput

User-Space I/O Software III

Layers of the I/O system and the main functions of each layer

Linux

Loadable Kernel Module (LKM)

Loadable kernel modules provide device drivers

- Contain object code, loaded on-demand
  - Dynamically linked to running kernel
  -

Kmod

- Kernel subsystem managing modules wi
- Determines module dependencies
- Loads modules on demand

Linux

Basic LKM Module

Every LKM consists of two basic functions (minimum)

```
/* used for all initialisation code */
int init_module (void) {
...
}

/* use
void cleanup_module (void) {
...
}
```

Load module by using the **insmod** command → normally restricted to root

Kernel provides common interface for I/O system calls

Devices grouped into device classes

- Members of each device class perform similar functions
-

Major and minor identification numbers

- Used by device drivers to identify their devic
- Devices with same major number controll
- Minor numbers enable system to distinguish between devices of same class

Linux

Device Drivers and Access

Most devices represented by device special files

Device files accessed via virtual file system (VFS) (/dev)

- System calls pass to VFS, which in turn issues calls to device drivers → most drivers implement common file operations,

List of

Linu

→ retrieving status information from printer

Character Device

- Transmits data as stream of bytes
- Represented by device_struct structure, which contains

file_operations structure

- Maintains operations supported by devic
- Stores functions called by VFS when system call accesses device special file

Block I/O subsystem

- Kernel's block I/O subsystem contains number of layers
-

Two primary strategies used by kernel to minimise amount of time spent accessing block devices

- Caching data
- Clustering I/O operations

Linux

Block Device Caching

When data from block device requested, kernel first searches cache

- If found, data copied to processs address space
-

Dire

- Driver bypasses kernel cache when access
- Important for databases and other applic caching inappropriate and may reduce performance/consistency

Socket (message)	Network interface

Linux

I/O API Calls I

```
fd = create (filename, permission)
```

- Opens file for reading/writing
-
-

```
fd = open (filename, mode)
```

- Mode is 0, 1, 2 for read, write, read/write

```
close (fd)
```

```
numbytesread = read (fd, buffer, numbytes)
```

- Read numbytes from file or device referenced by fd into

- ad

```
numbyteswritten = write (fd, buffer, num
```

- Write numbytes to file referenced by fer
- Returns number of bytes actually written in d

I/O API Calls III

`pipe (&fd[2])`

- Creates pipe
- `fd` is an array of 2 integers (`fd[0]` for reading, `fd[1]` for writing)

`newf`

- [illegible]

`ioctl (fd, operation, &termios)`

- Used to control devices: e.g. [illegible] chars

`fd = mknod (filename, permission, dev)`

- Creates new special file e.g. character or block device

Linux

File Descriptors

Each process has its own file descriptor table

- Each process has 3 file descriptors when created

1	
2	

By default, all three file descriptors refer to terminal from which program was started

Blocking vs. Non-blocking I/O

Blocking I/O

- Call returns when operation completed
- Process suspended   I/O appears “instantaneous”
-

Non

- I/O call returns as much as available (e.g. rea
- Turn on for file descriptor using fcntl system
- Provides application-level polling for I/O

Asynchronous I/O

Process executes in parallel with I/O operation

- No blocking in interface procedure

I/O su
com

-
  . . .

Supports check/wait if I/O operation completed

Very flexible and efficient

Harder to use and potentially less secure