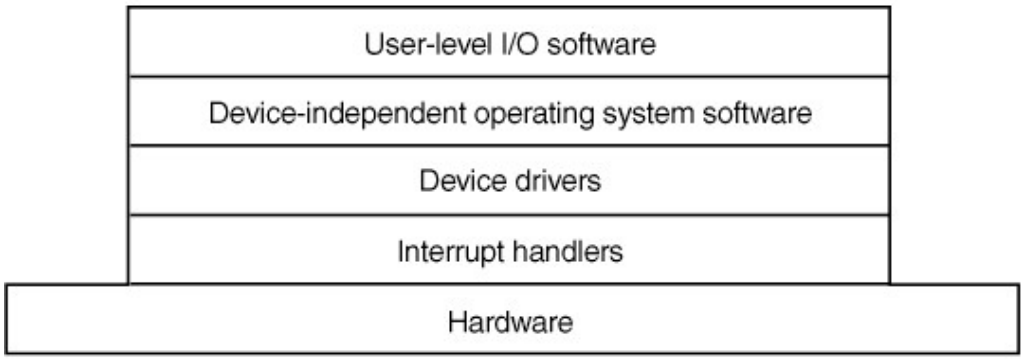
Today: Minix I/O Subsystem

- Device-independent Minix Layer
- Device Drivers in Minix
-

Minix I/O Subsystem

Layers of the I/O software system.



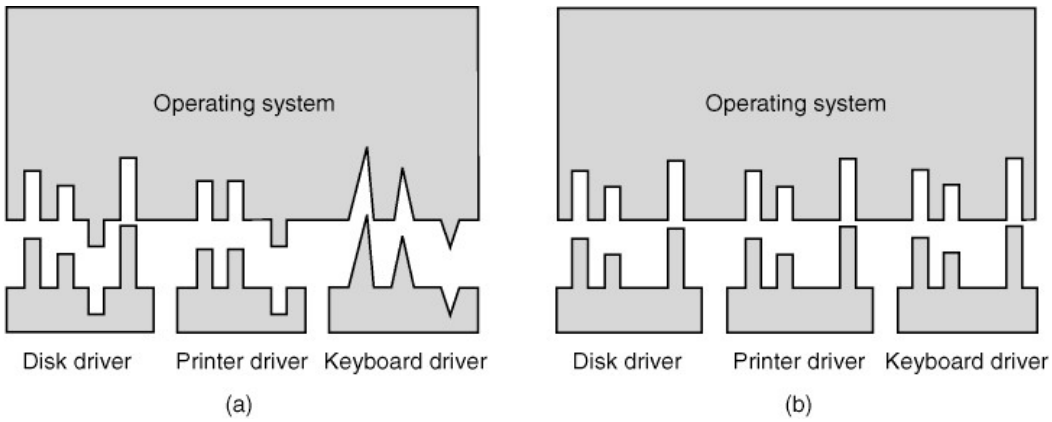
Device-Independent I/O Software

Functions of the device-independent I/O software.

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

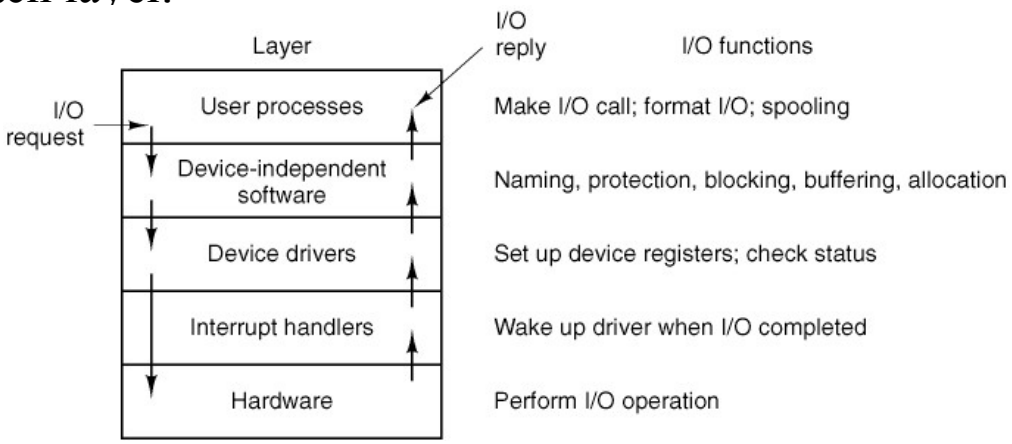
Uniform Interfacing for Device Drivers

(a) Without a standard driver interface.
(b) With a standard driver interface.



User-Space I/O Software

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



Minix Interrupt Handlers and I/O

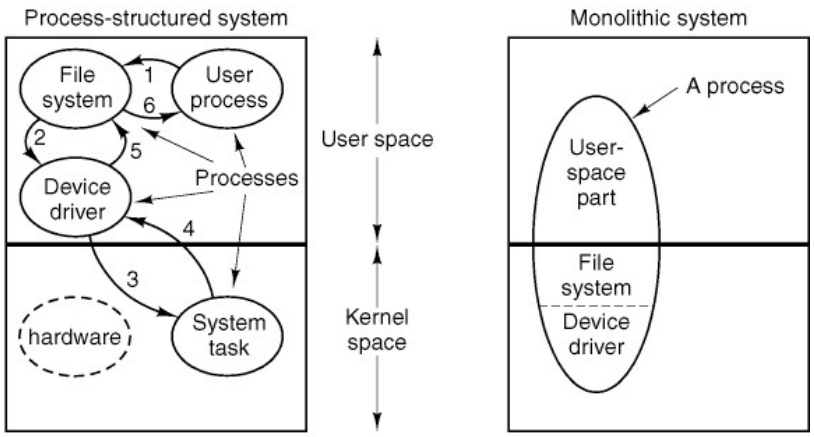
- Minix device drivers run in user space
  - Can not access kernel memory or I/O ports
- User-space drivers need different levels of access
  - Need access to memory outside its data space
    - RAM disk driver
  - Read or write to I/O ports
    - code to do so available in kernel, not user space; disk driver
  - Need to respond to predictable interrupts
    - Disk driver needs to handle interrupt upon I/O completion
  - Handle unpredictable interrupts
    - keyboard driver
- All handled by kernel calls via SYSTEM task

Driver access to kernel

- System task can allow user process to access segments from other address spaces
  - Used to allow memory driver to access RAM disk;
  - Console driver to access video display adapter
- Minix kernel calls support access to I/O ports
  - System task does actual I/O on behalf of user driver (used by hard disk driver)
- Handling predictable interrupts: system task
  - generic_handler in system task: converts interrupt into notification message; device driver uses *receive*
- Unpredictable interrupts: keystrokes, network packets
  - Can't "receive" from one source: interrupts come from anywhere
  - Fast handling, non-blocking receive, non-blocking send
  -

Device Drivers in MINIX 3

Minix versus Unix/monolithic system device drivers



1–6 are request and reply messages between four independent processes.

(a)

The user-space part calls the kernel-space part by trapping. The file system calls the device driver as a procedure. The entire operating system is part of each process

(b)

Minix drivers

- Separate device driver for each class of I/O device
- Drivers run as full fledged user processes
- Communicate with system task using message passing
- Simple drivers : written in one source file
- Accept requests from other processes and carry them out
  - Strictly sequential: no concurrency to keep them simple
    - Receive message, process, send reply
  - Drivers can receive work requests as well as interrupts
  - Driver uses *receive* to accept interrupt messages
    - New work requests are kept pending during interrupt processing or processing of work requests

Device Driver Messages

Fields of the messages sent by the file system to the block device drivers and fields of the replies sent back.

Requests		
Field	**Type**	**Meaning**
m.m_type	int	Operation requested
m.DEVICE	int	Minor device to use
m.PROC_NR	int	Process requesting the I/O
m.COUNT	int	Byte count or ioctl code
m.POSITION	long	Position on device
m.ADDRESS	char*	Address within requesting process

Replies		
Field	**Type**	**Meaning**
m.m_type	int	Always DRIVER_REPLY
m.REP_PROC_NR	int	Same as PROC_NR in request
m.REP_STATUS	int	Bytes transferred or error number

Typical Driver Pseudo-code

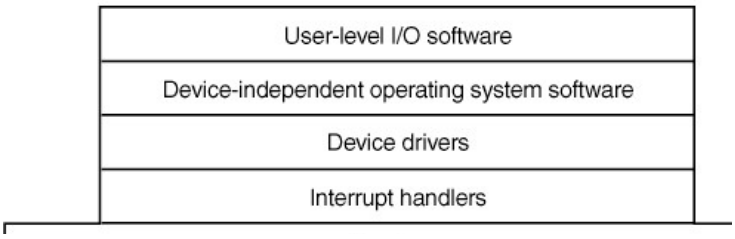
```
message mess;                          /* message buffer */

void io_driver() {
  initialize();                        /* only done once, during system init. */
  while (TRUE) {
        receive(ANY, &mess);           /* wait for a request for work */
        caller = mess.source;          /* process from whom message came */
        switch(mess.type) {
            case READ:      rcode = dev_read(&mess); break;
            case WRITE:     rcode = dev_write(&mess); break;
            /* Other cases go here, including OPEN, CLOSE, and IOCTL */
             default:       rcode = ERROR;
        }
        mess.type = DRIVER_REPLY;
        mess.status = rcode;           /* result code */
        send(caller, &mess);           /* send reply message back to caller */
  }
}
```


Driver Examples

- Block Device Driver
  - RAM disk device driver
  - Hard disk driver
- Terminal Driver (character device driver)
- Keyboard driver (character driver)

Device-independent I/O

- File system process (vfs) contain all device-independent code
  - I/O system closely related to file system and merged into one process
- VFS implements the device independent layer in Minix
  - drivers implement device-specific layer and kernel facilitates interrupt handling



Block Device Drivers in MINIX 3

Generic structure of a block device driver

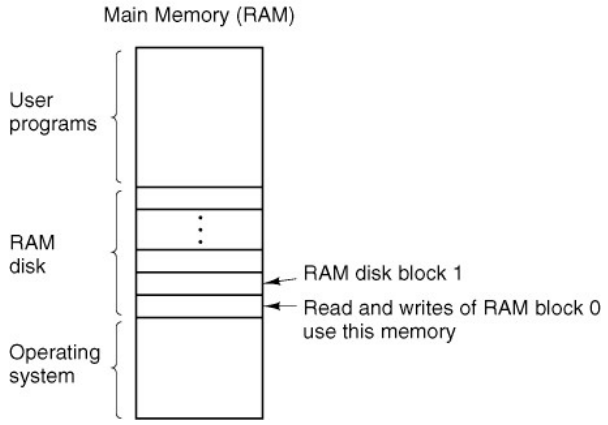
```
message mess;                                   /* message buffer */

void shared_io_task(struct driver_table *entry_points) {
/* initialization is done by each task before calling this */
  while (TRUE) {
        receive(ANY, &mess);
        caller = mess.source;
        switch(mess.type) {
            case READ:      rcode = (*entry_points->dev_read)(&mess); break;
            case WRITE:     rcode = (*entry_points->dev_write)(&mess); break;
            /* Other cases go here, including OPEN, CLOSE, and IOCTL */
            default:        rcode = ERROR;
        }
        mess.type = TASK_REPLY;
        mess.status = rcode;                    /* result code */
        send(caller, &mess);
  }
}
```

Device Driver Operations

- Each driver needs to handle 6 operations:
  - OPEN
  - CLOSE
  - READ
  - WRITE
  - IOCTL
  - SCATTERED_IO
- ***ioctl (I/O control):*** change device parameters (e.g., transmission speed, parity)
- **Scatter-gather I/O:** read/write multiple blocks

RAM Disk

A RAM disk uses a portion of RAM to look like a small disk. RAM is volatile, so RAM disks are temporary storage — used during boot process to create root device



RAM Disk Driver

- Typical block driver: read or write a block
- RAM Disk: preallocated RAM to store blocks
- Minix RAM disk driver uses 6 drivers in one
  - Each message uses a specific device
    - /dev/ram - actual RAM disk
    - /dev/mem - device allows reading or writing to physical memory
    - /dev/kmem - byte offset 0 allows read/write to kernel mem
    - /dev/boot - holds boot image
    - /dev/null, /dev/zero
- Code is in "drivers" under "ramdisk" and "memory" folders

Hard Disk Driver

Disk geometry: cylinder, track, sector

Typical disk address: cylinder#, track#, sector#

Logical Block Addressing (LBA) sequentially number sectors

Code in drivers/at_wini ("winchester drives")

PCI Bus holds IDE disk or SATA (Serial AT Attachment) disks - can have upto 4 disks

Disks are numbered /dev/c0d0 to /dev/c0d3

Hard Disk Driver Controller

The control registers of an IDE hard disk controller. The numbers in parentheses are the bits of the logical block address selected by each register in LBA mode.

Register	Read Function	Write Function
0	Data	Data
1	Error	Write Precompensation
2	Sector Count	Sector Count
3	Sector Number (0-7)	Sector Number (0-7)
4	Cylinder Low (8-15)	Cylinder Low (8-15)
5	Cylinder High (16-23)	Cylinder High (16-23)
6	Select Drive/Head (24-27)	Select Drive/Head (24-27)
7	Status	Command

(a)

Hard Disk Driver Addressing

The fields of the Select Drive/Head register.

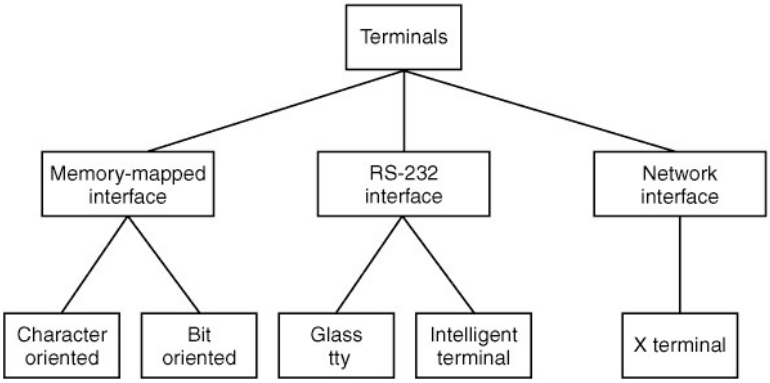
7	6	5	4	3	2	1	0
1	LBA	1	D	HS3	HS2	HS1	HS0

LBA: 0 = Cylinder/Head/Sector Mode
1 = Logical Block Addressing Mode
D: 0 = master drive
1 = slave drive
HSn: CHS mode: Head select in CHS mode
LBA mode: Block select bits 24 - 27

(b)

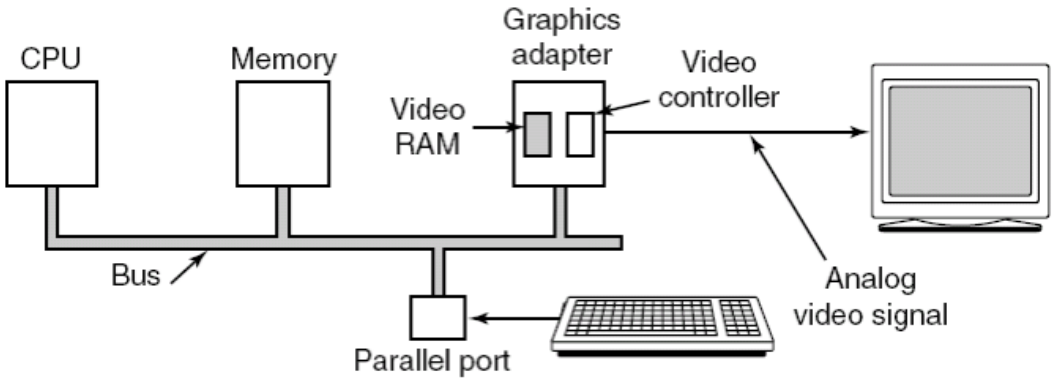
Terminals

Typical Terminal types.



Memory-mapped Terminals

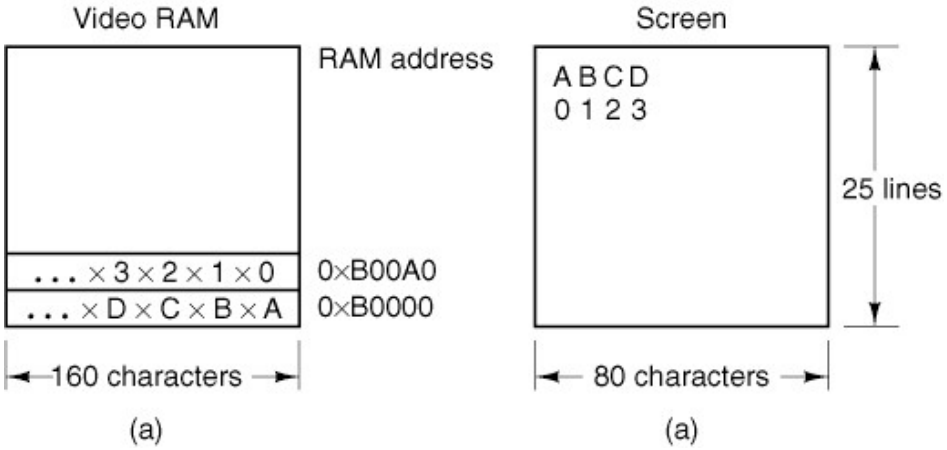
Memory-mapped terminals write directly into video RAM.



Terminals Output

A video RAM image for the IBM character display.

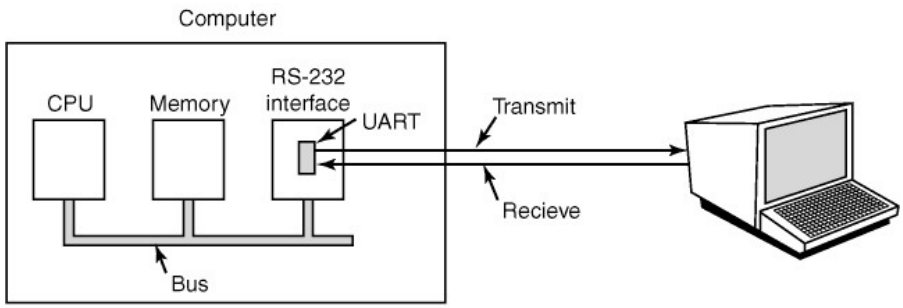
Bitmapped displays are similar: each pixel can be controlled



RS-232 Terminals

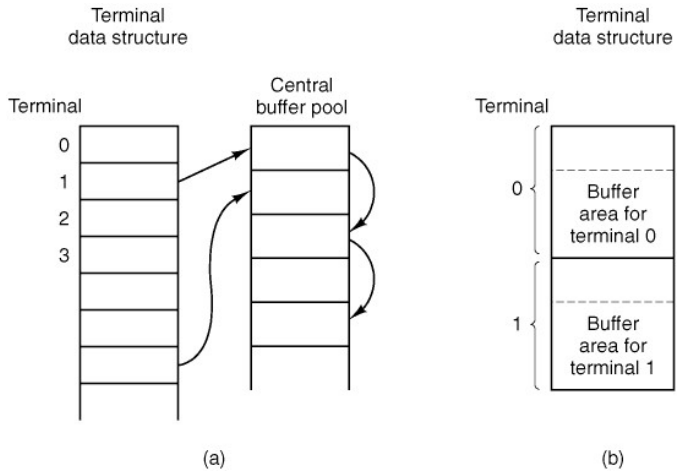
An RS-232 terminal communicates with a computer over a communication line, one bit at a time.

The computer and the terminal are completely independent - UARTS used for serial communication



Input Software

(a) Central buffer pool. (b) Dedicated buffer for each terminal.



Terminal Driver in MINIX

Terminal driver message types:

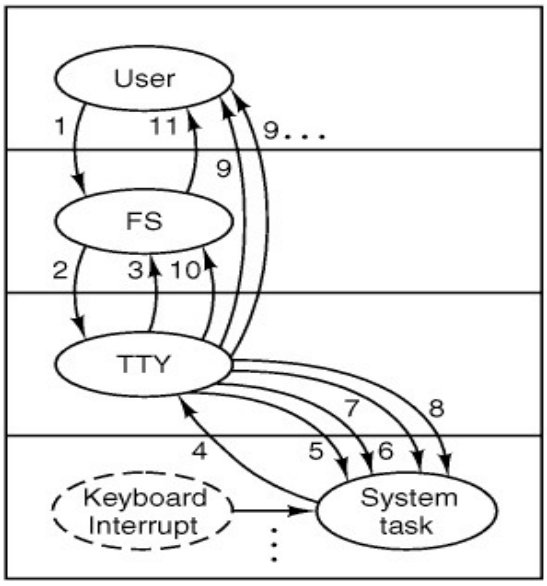
1. Read from the terminal (from VFS on behalf of a user process).
2. Write to the terminal (from VFS on behalf of a user process).
3. Set terminal parameters for IOCTL (from FS on behalf of a user process).

Terminal Driver in MINIX (2)

Terminal driver message types (continued):

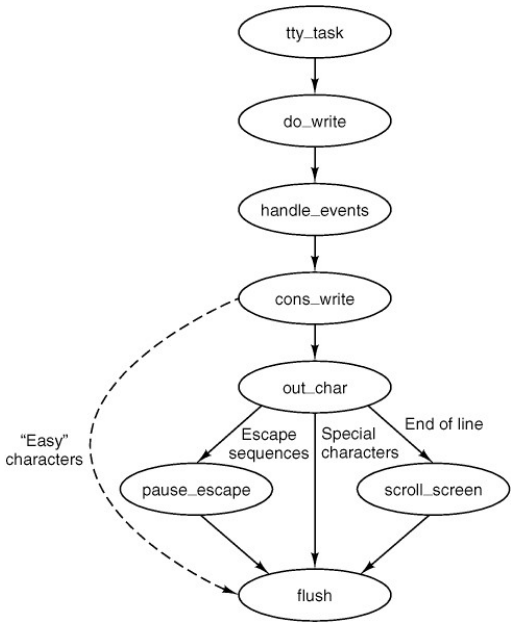
4. I/O occurred during last clock tick (from the clock interrupt).
5. Cancel previous request (from the file system when a signal occurs).
6. Open a device.
7. Close a device.

Terminal Input



Terminal Output

Major procedures used in terminal output.



Keyboard Driver

- Scan codes in the input buffer, with corresponding key actions below, for a line of text entered at the keyboard.
- L and R represent the left and right Shift keys. + and - indicate a key press and a key release.
- The code for a release is 128 more than the code for a press of the same key.

42	35	163	170	18	146	38	166	38	166	24	152	57	185
L+	h+	h-	L-	e+	e-	l+	l-	l+	l-	o+	o-	SP+	SP-

54	17	145	182	24	152	19	147	38	166	32	160	28	156
R+	w+	w-	R-	o+	o-	r+	r-	l+	l-	d+	d-	CR+	CR-