OPERATING SYSTEMS
DESIGN AND IMPLEMENTATION
Third Edition
ANDREW S. TANENBAUM
ALBERT S. WOODHULL

Chapter 3
Input/Output

I/O Devices

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	4 MB/sec
52x CD-ROM	8 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
Gigabit Ethernet	125 MB/sec
Serial ATA disk	200 MB/sec
SCSI Ultrawide 4 disk	320 MB/sec
PCI bus	528 MB/sec

Figure 3-1. Some typical device, network, and bus data rates.

Device Controllers

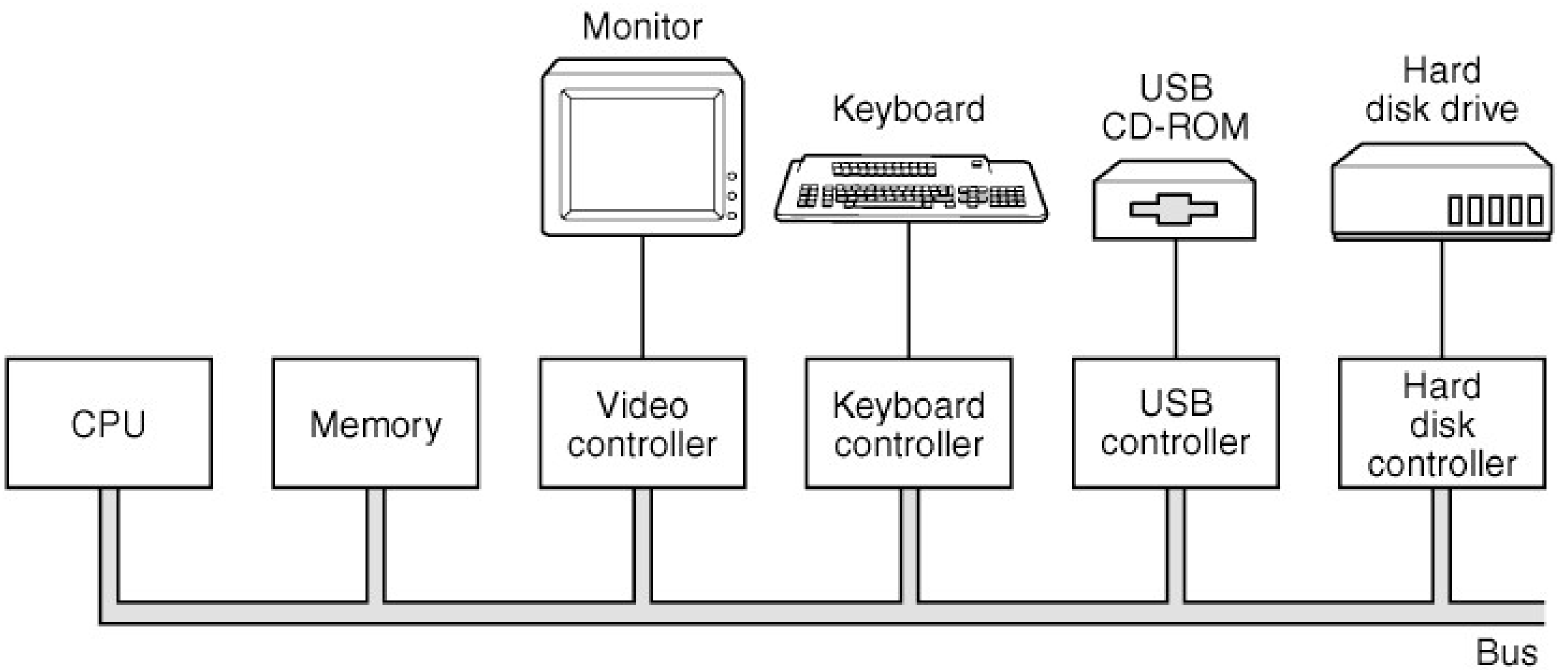


Figure 3-2. A model for connecting the CPU, memory, controllers, and I/O devices.

Memory-Mapped I/O

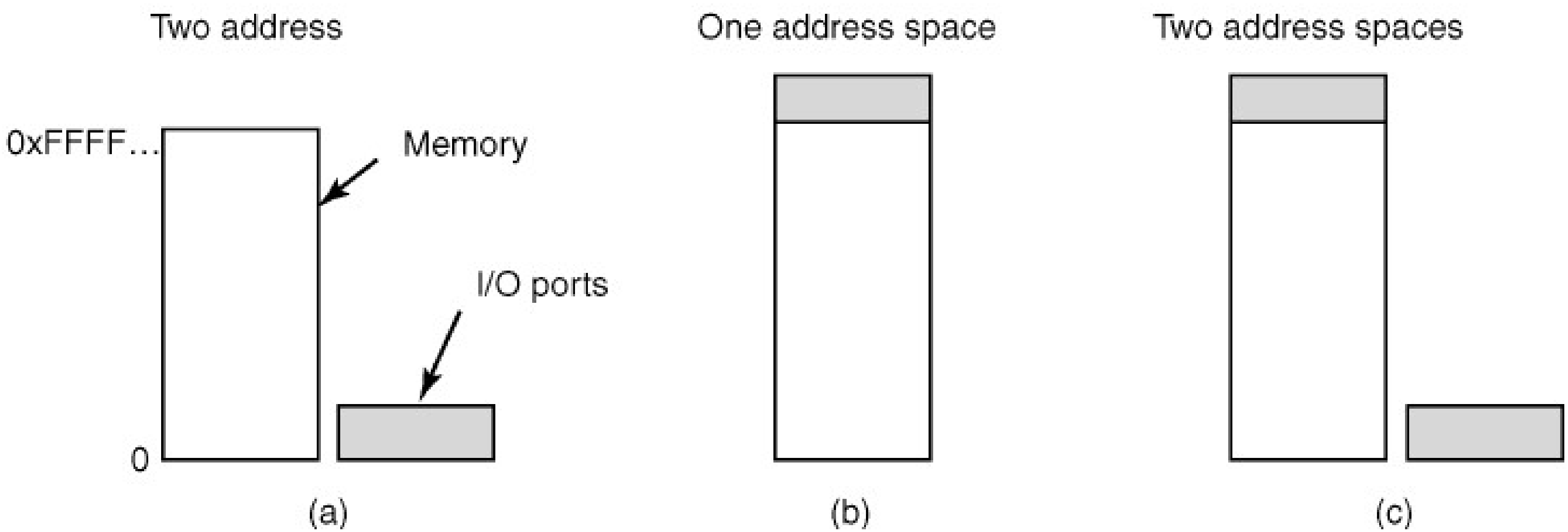


Figure 3-3. (a) Separate I/O and memory space.
(b) Memory-mapped I/O. (c) Hybrid.

Direct Memory Access (DMA)

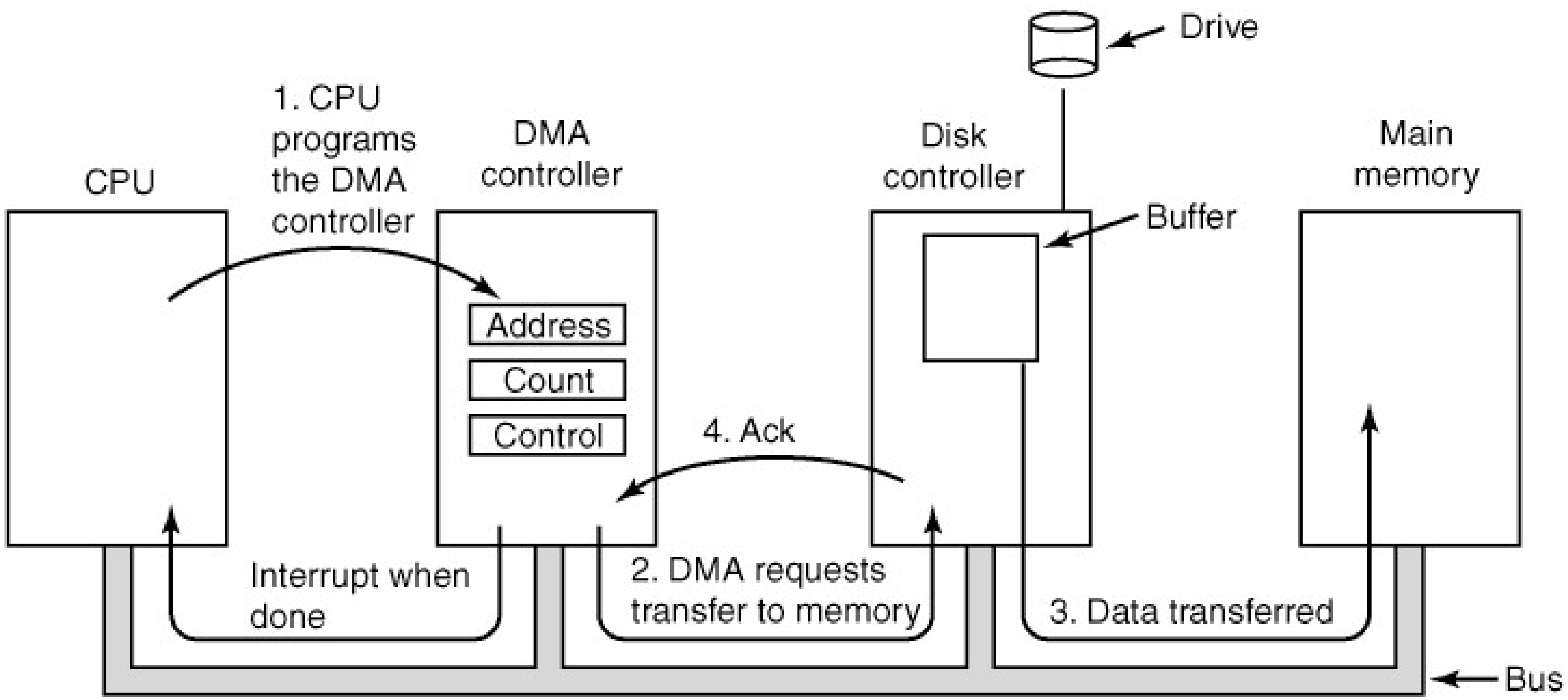


Figure 3-4. Operation of a DMA transfer.

Goals of the I/O Software

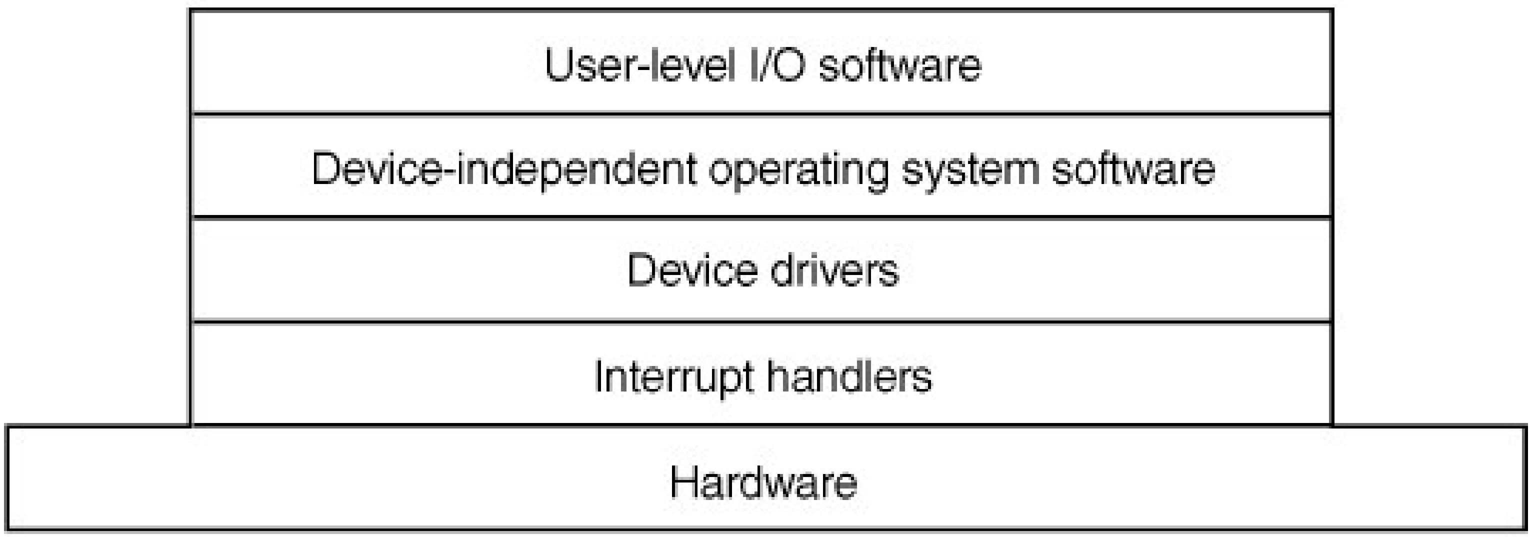


Figure 3-5. Layers of the I/O software system.

Device-Independent I/O Software

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

Figure 3-6. Functions of the device-independent I/O software.

Uniform Interfacing for Device Drivers

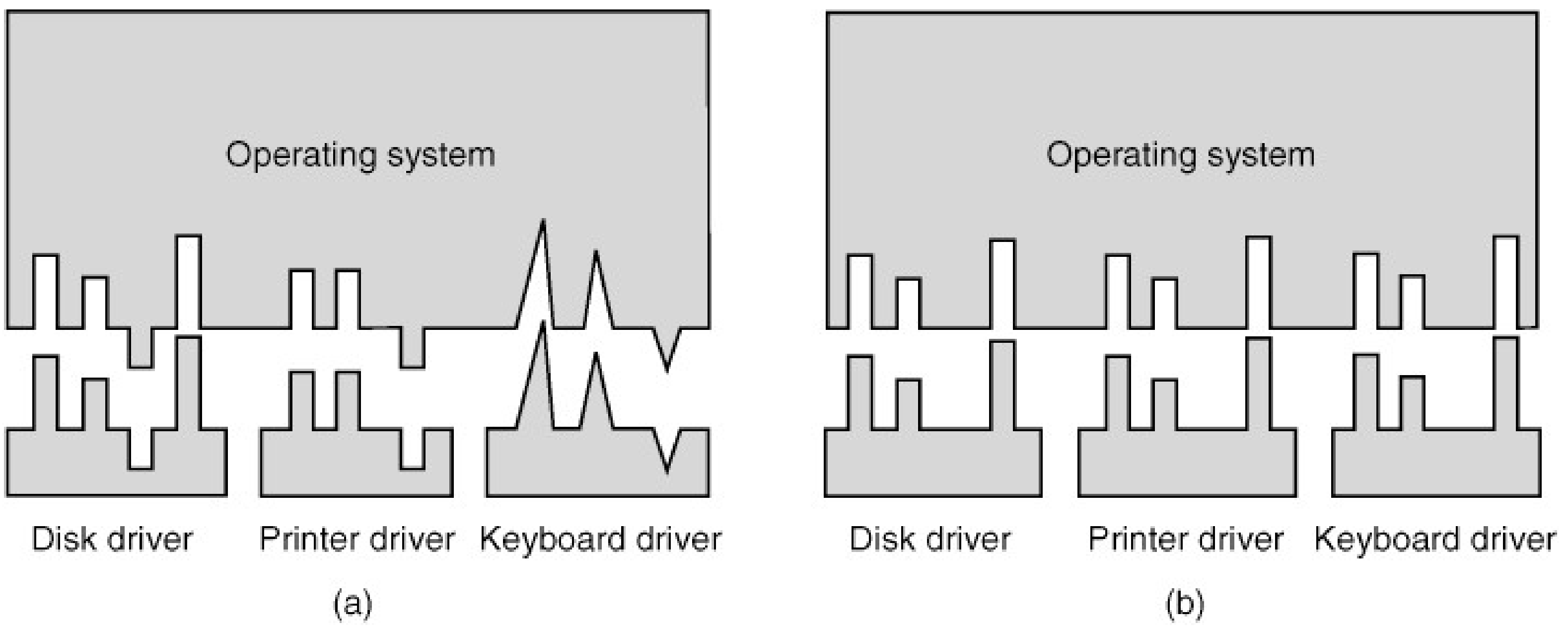


Figure 3-7. (a) Without a standard driver interface.
(b) With a standard driver interface.

User-Space I/O Software

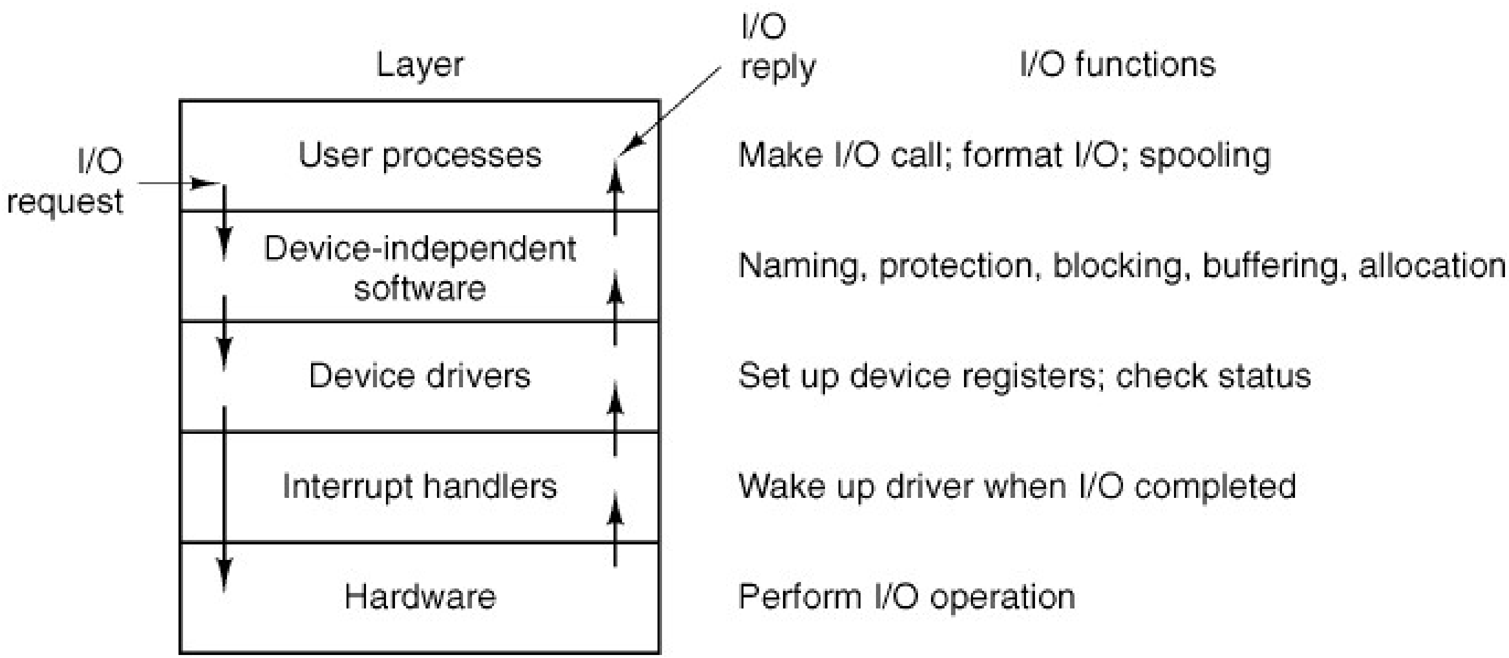


Figure 3-8. Layers of the I/O system and the main functions of each layer.

Resources

The sequence of events required to use a resource:

3. Request the resource.
4. Use the resource.
5. Release the resource.

Definition of Deadlock

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Conditions for Deadlock

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

Deadlock Modeling

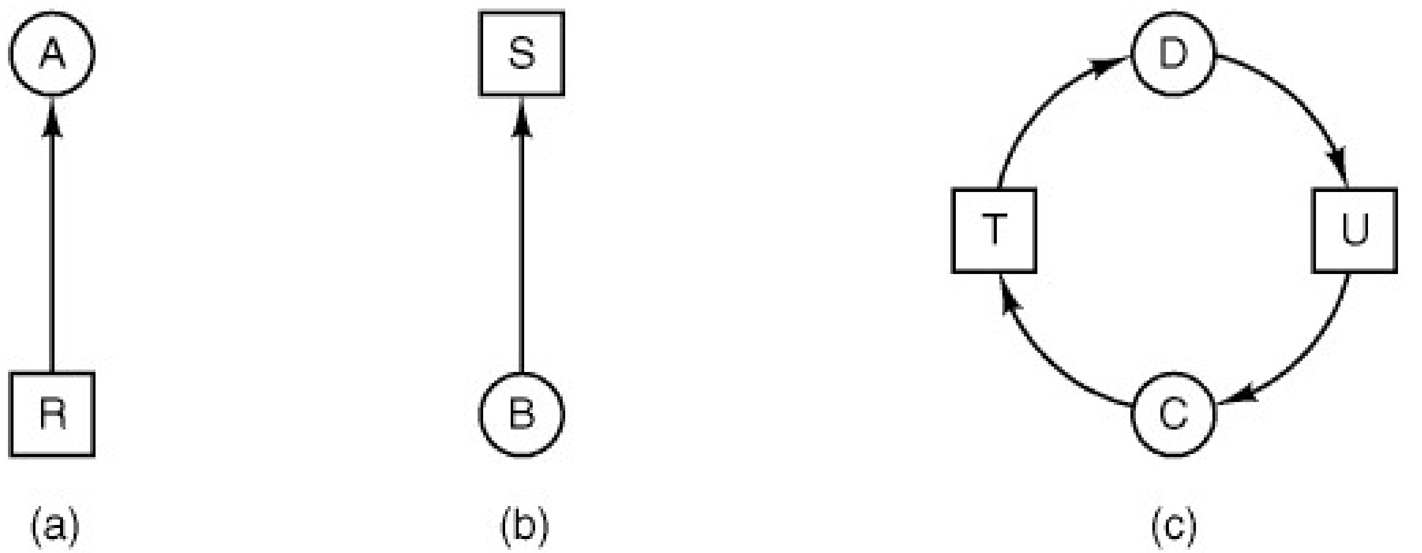


Figure 3-9. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Deadlock Handling Strategies

1. Ignore the problem altogether
2. Detection and recovery
3. Avoidance by careful resource allocation
4. Prevention by negating one of the four necessary conditions

Deadlock Avoidance (1)

A	B	C
Request R Request S Release R Release S	Request S Request T Release S Release T	Request T Request R Release T Release R
(a)	(b)	(c)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

(d)

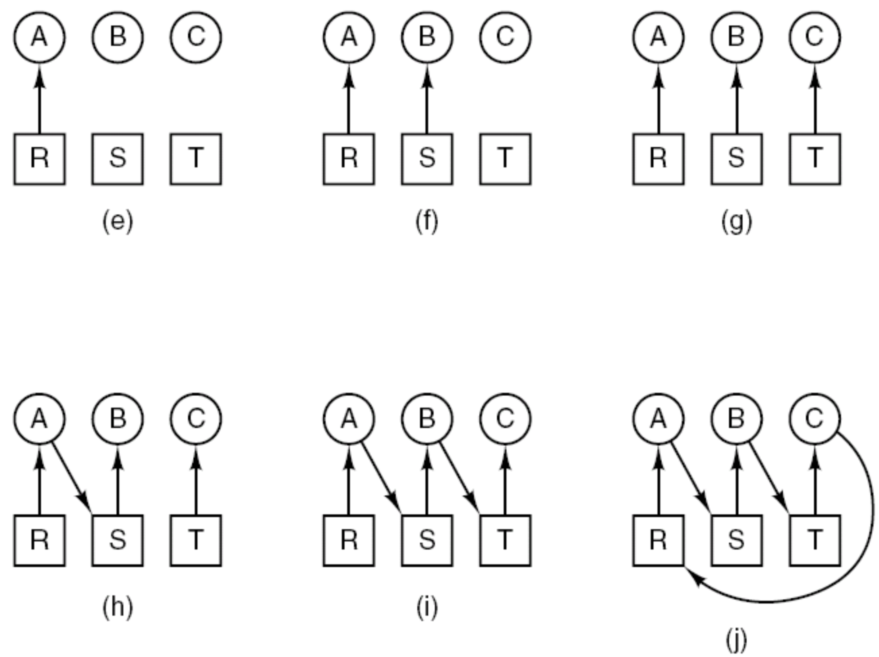


Figure 3-10. An example of how deadlock occurs and how it can be avoided.

Deadlock Avoidance (2)

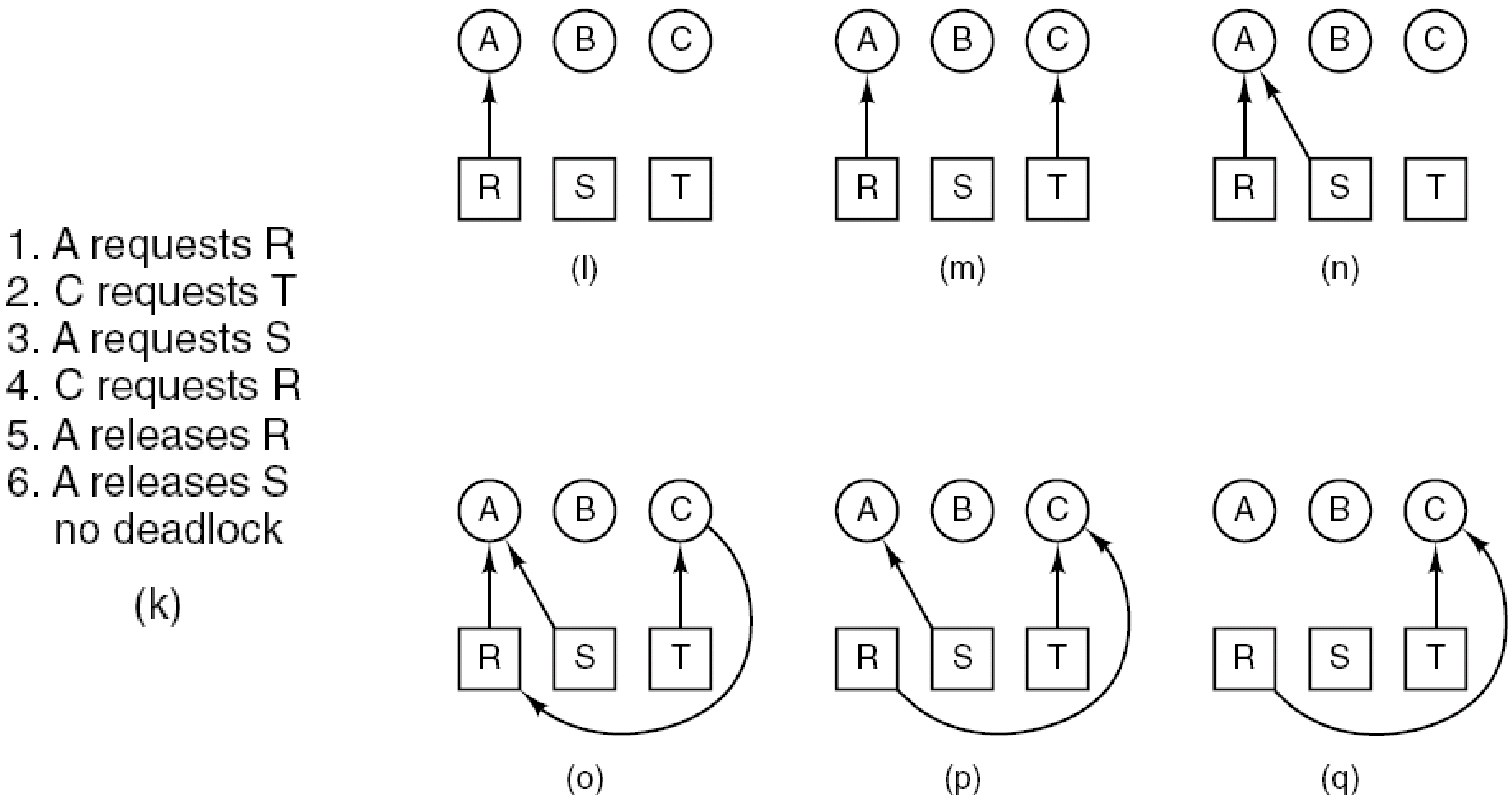
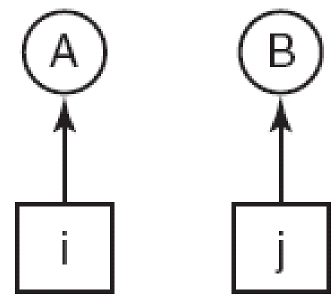


Figure 3-10. An example of how deadlock occurs and how it can be avoided.

Deadlock Prevention (1)

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

(a)



(b)

Figure 3-11. (a) Numerically ordered resources.
(b) A resource graph.

Deadlock Prevention (2)

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Figure 3-12. Summary of approaches to deadlock prevention.

The Banker's Algorithm for a Single Resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Figure 3-13. Three resource allocation states:
(a) Safe. (b) Safe. (c) Unsafe.

Resource Trajectories

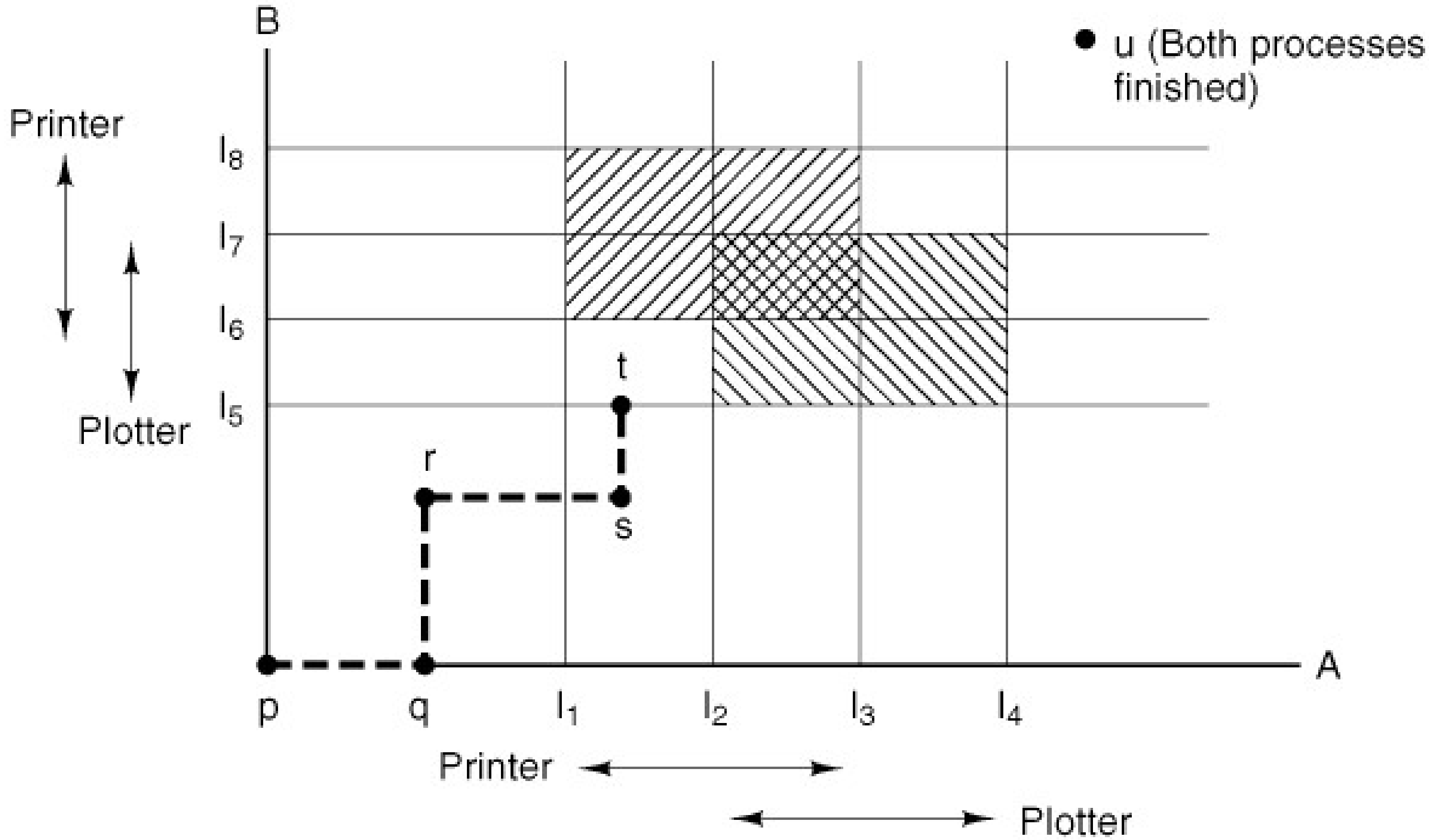


Figure 3-14. Two process resource trajectories.

The Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Figure 3-15. The banker's algorithm with multiple resources.

Safe State Checking Algorithm

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.
2. Assume the process of the row chosen requests all the resources it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated, in which case the initial state was safe, or until a deadlock occurs, in which case it was not.

Device Drivers in MINIX 3 (1)

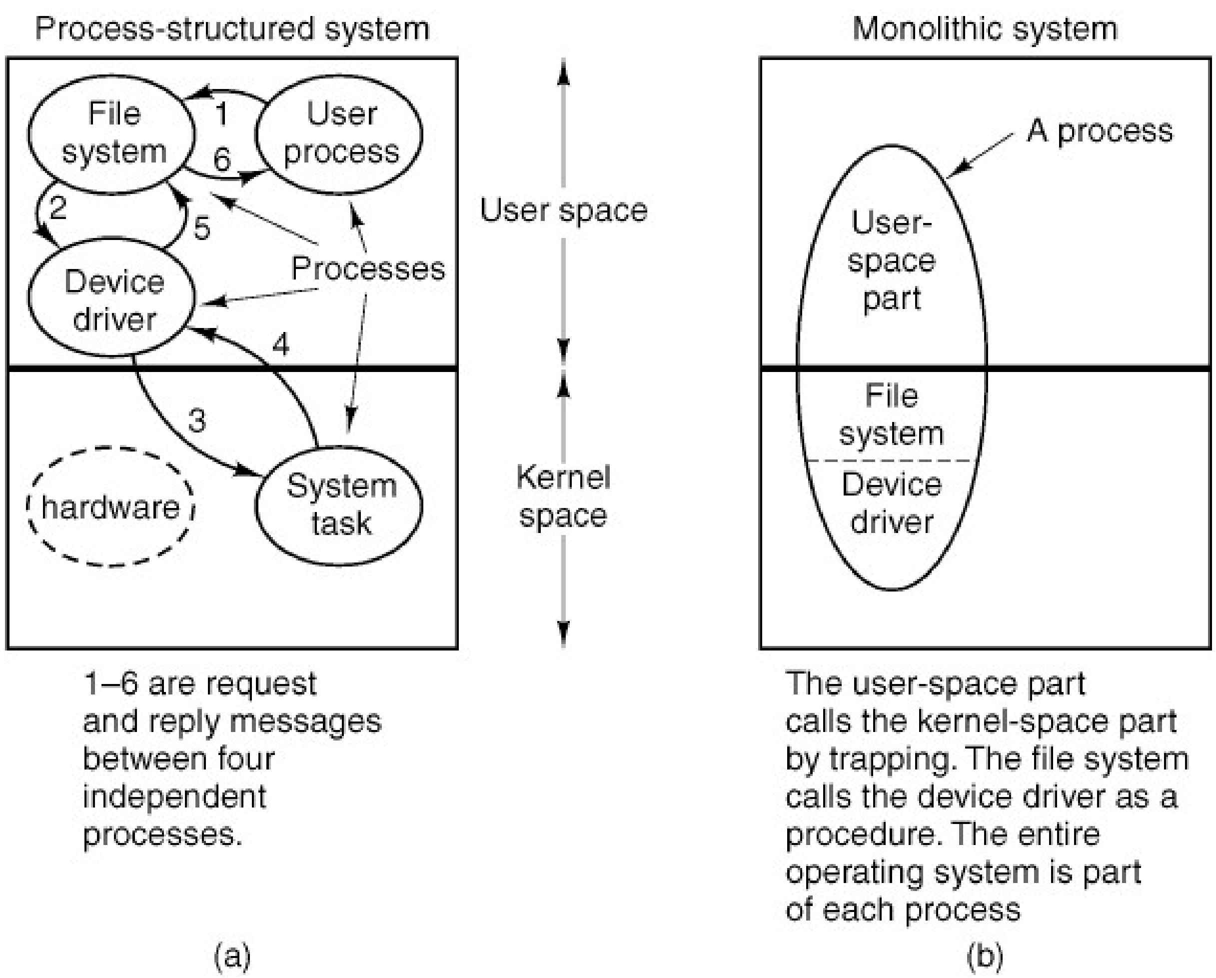


Figure 3-16. Two ways of structuring user-system communication.

Device Drivers in MINIX 3 (2)

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

Figure 3-17. Fields of the messages sent by the file system to the block device drivers and fields of the replies sent back.

Device-Independent I/O Software in MINIX 3

```
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 */
  }
}
```

Figure 3-18. Outline of the main procedure of an I/O device driver.

Block Device Drivers in MINIX 3

```
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);
  }
}
```

Figure 3-19. A shared I/O task main procedure using indirect calls.

Device Driver Operations

1. OPEN
2. CLOSE
3. READ
4. WRITE
5. IOCTL
6. SCATTERED_IO

RAM Disk Hardware and Software

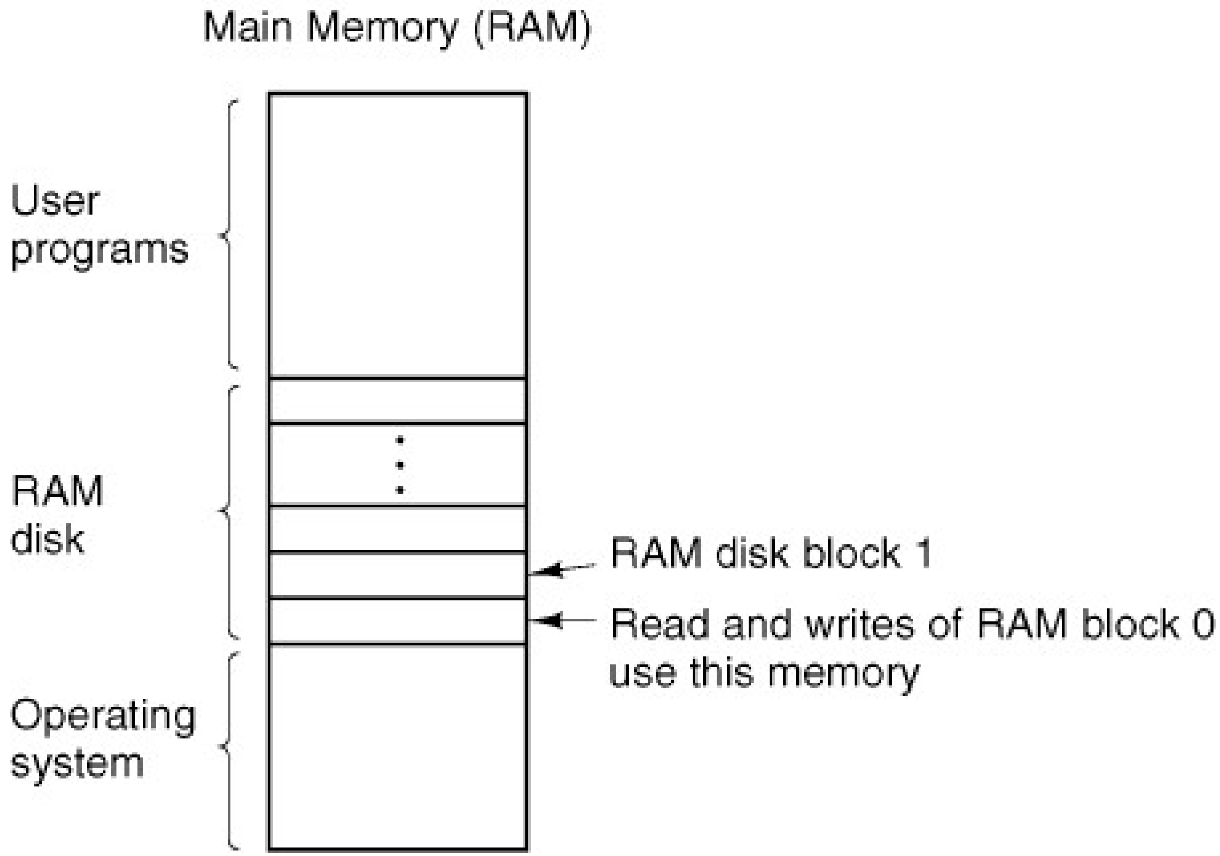


Figure 3-20. A RAM disk.

Disk Software

Read/Write timing factors:

3. Seek time
4. Rotational delay
5. Data transfer time

Disk Arm Scheduling Algorithms (1)

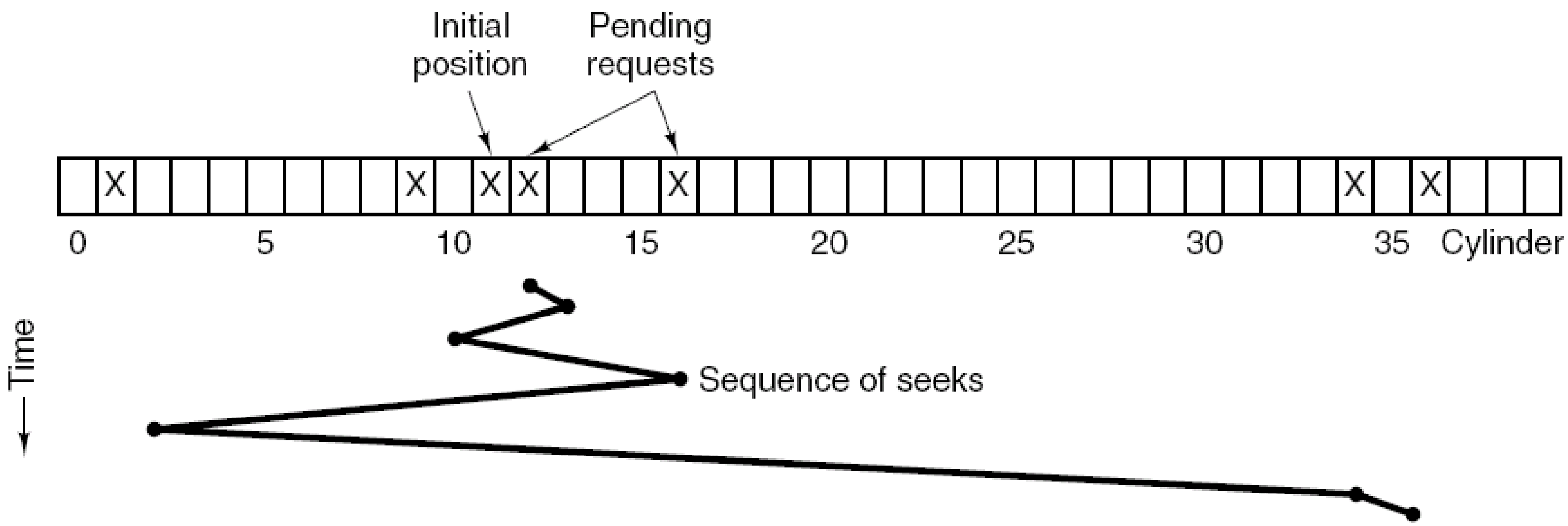


Figure 3-21. Shortest Seek First (SSF) disk scheduling algorithm.

Disk Arm Scheduling Algorithms (2)

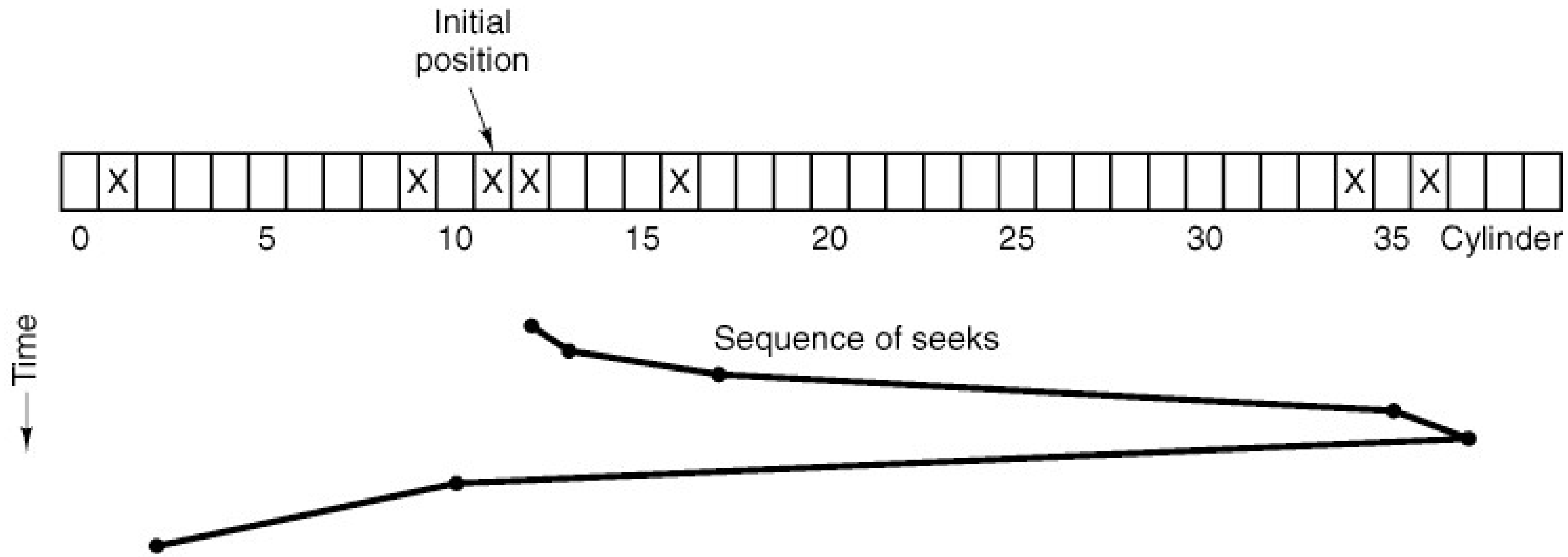


Figure 3-22. The elevator algorithm for scheduling disk requests.

Common Hard Drive Errors

1. Programming error
   - request for nonexistent sector
2. Transient checksum error
   - caused by dust on the head
3. Permanent checksum error
   - disk block physically damaged
4. Seek error
   - the arm sent to cylinder 6 but it went to 7
5. Controller error
   - controller refuses to accept commands

Hard Disk Driver in MINIX 3 (1)

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)

Figure 3-23. (a) 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.

Hard Disk Driver in MINIX 3 (2)

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)

Figure 3-23. (b) The fields of the Select Drive/Head register.

Floppy Disk Drive

Characteristics that complicate the driver:

3. Removable media.
4. Multiple disk formats.
5. Motor control.

Terminals (1)

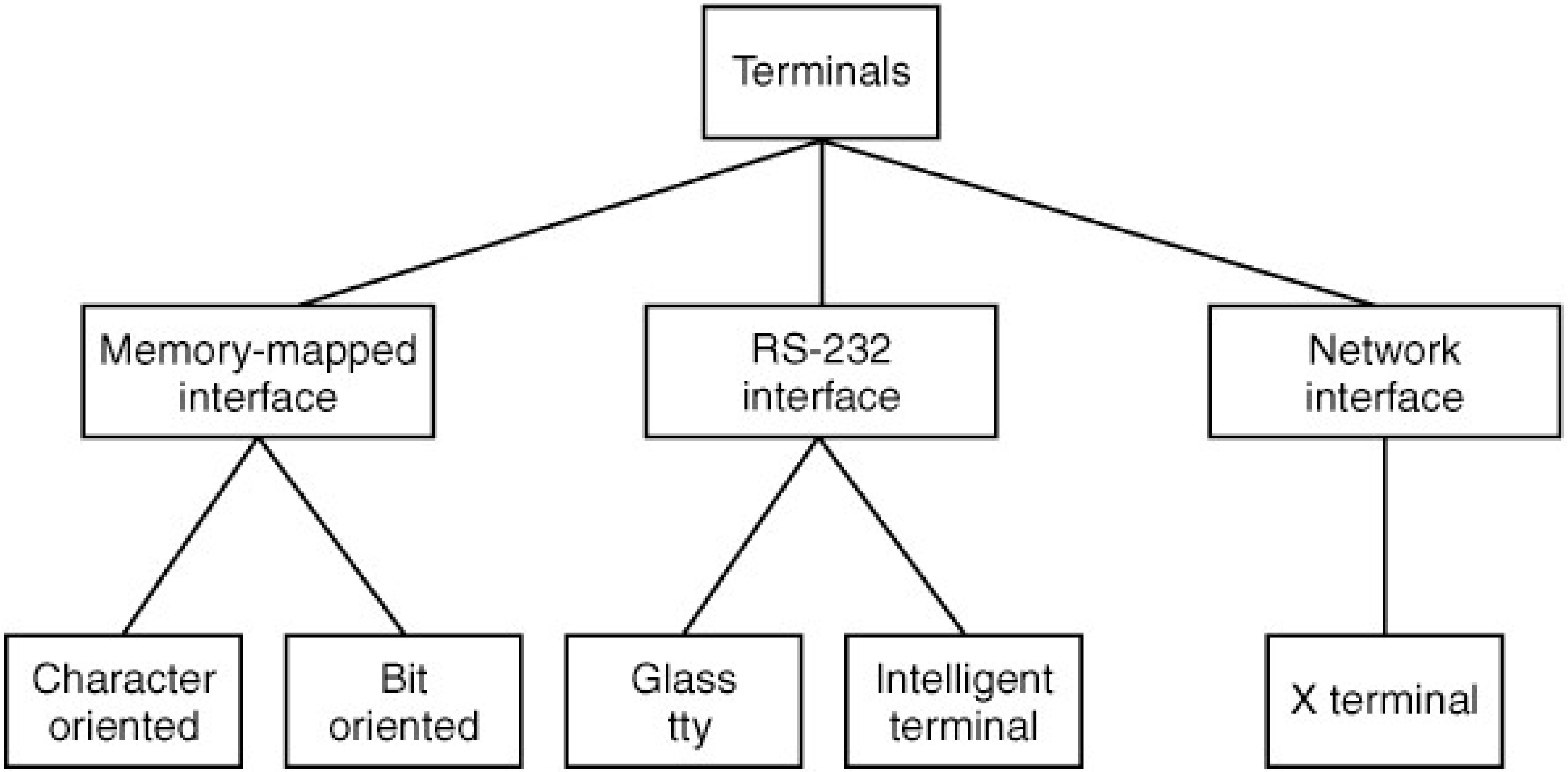


Figure 3-24. Terminal types.

Terminals (2)

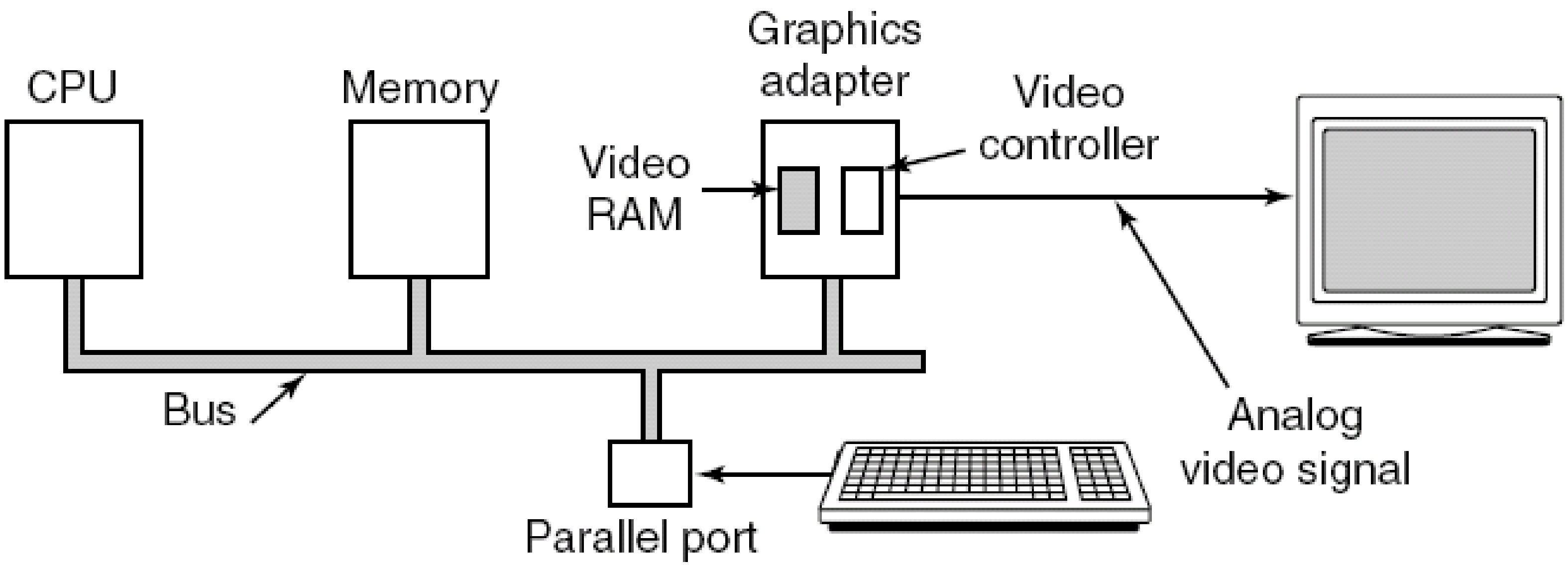


Figure 3-25. Memory-mapped terminals write directly into video RAM.

Terminals (3)

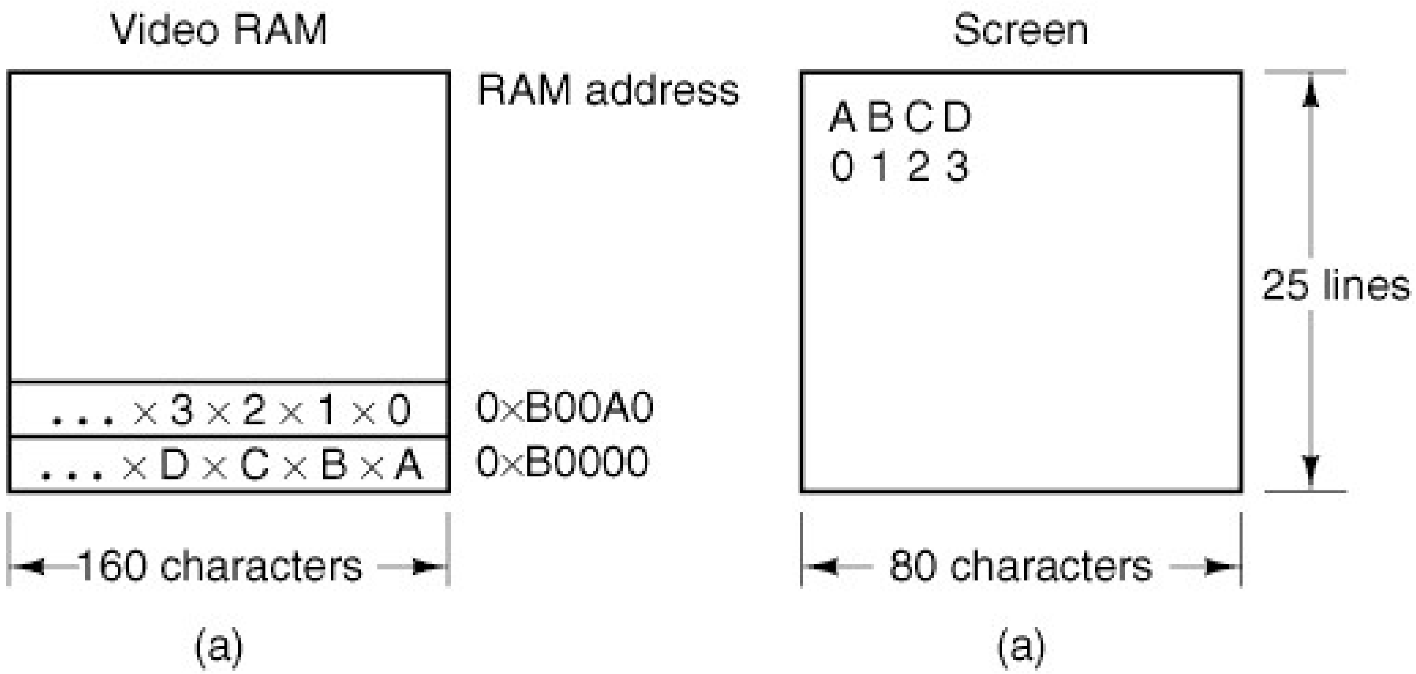


Figure 3-26. (a) A video RAM image for the IBM monochrome display. (b) The corresponding screen.

RS-232 Terminals

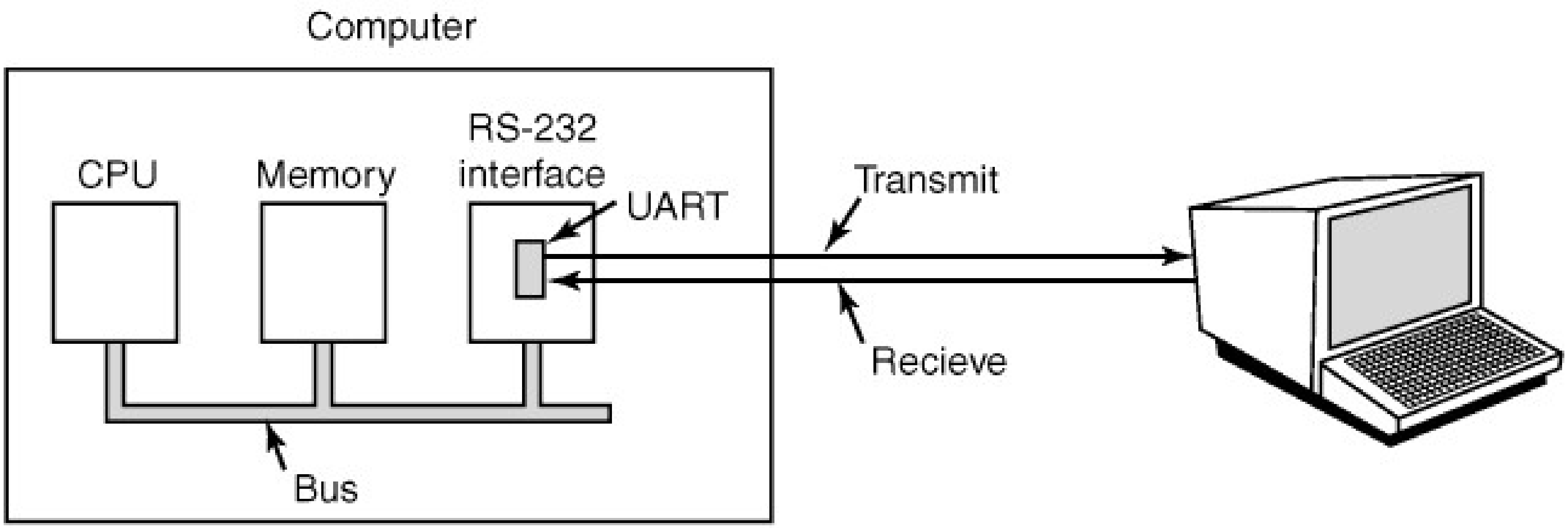


Figure 3-27. An RS-232 terminal communicates with a computer over a communication line, one bit at a time. The computer and the terminal are completely independent.

Input Software (1)

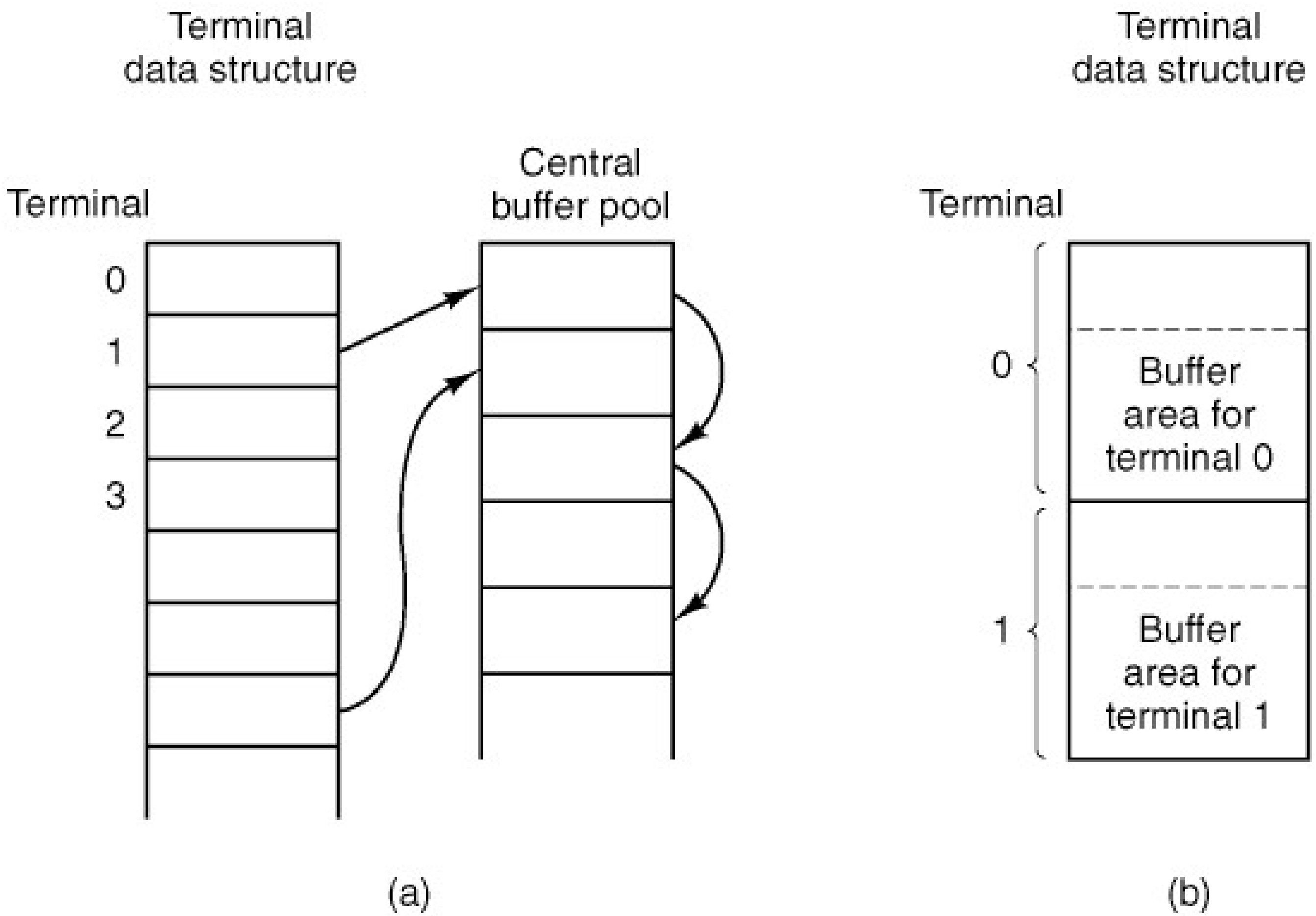


Figure 3-28. (a) Central buffer pool.
(b) Dedicated buffer for each terminal.

Input Software (2)

Character	POSIX name	Comment
CTRL-D	EOF	End of file
	EOL	End of line (undefined)
CTRL-H	ERASE	Backspace one character
CTRL-C	INTR	Interrupt process (SIGINT)
CTRL-U	KILL	Erase entire line being typed
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-Z	SUSP	Suspend (ignored by MINIX)
CTRL-Q	START	Start output
CTRL-S	STOP	Stop output
CTRL-R	REPRINT	Redisplay input (MINIX extension)
CTRL-V	LNEXT	Literal next (MINIX extension)
CTRL-O	DISCARD	Discard output (MINIX extension)
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Figure 3-29. Characters that are handled specially in canonical mode.

Input Software (3)

```
struct termios {
  tcflag_t c_iflag;                    /* input modes */
  tcflag_t c_oflag;                    /* output modes */
  tcflag_t c_cflag;                    /* control modes */
  tcflag_t c_lflag;                    /* local modes */
  speed_t  c_ispeed;                   /* input speed */
  speed_t  c_ospeed;                   /* output speed */
  cc_t c_cc[NCCS];                     /* control characters */
};
```

Figure 3-30. The termios structure. In MINIX 3 tc_flag _t is a short, speed_t is an int, and cc_t is a char.

Input Software (4)

	TIME = 0	TIME > 0
MIN = 0	Return immediately with whatever is available, 0 to N bytes	Timer starts immediately. Return with first byte entered or with 0 bytes after timeout
MIN > 0	Return with at least MIN and up to N bytes. Possible indefinite block.	Interbyte timer starts after first byte. Return N bytes if received by timeout, or at least 1 byte at timeout. Possible indefinite block

Figure 3-31. MIN and TIME determine when a call to read returns in noncanonical mode. N is the number of bytes requested.

Output Software (1)

Escape sequence	Meaning
ESC [*n*A	Move up *n* lines
ESC [*n*B	Move down *n* lines
ESC [*n*C	Move right *n* spaces
ESC [*n*D	Move left *n* spaces
ESC [*m* ; *n*H	Move cursor to (y = *m*, x = *n*)
ESC [*s*J	Clear screen from cursor (0 to end, 1 from start, 2 all)

Figure 3-32. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and n, m, and s are optional numeric parameters.

Output Software (2)

Escape sequence	Meaning
ESC [*s*K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n*L	Insert *n* lines at cursor
ESC [*n*M	Delete *n* lines at cursor
ESC [*n*P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n*m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Figure 3-32. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and n, m, and s are optional numeric parameters.

Terminal Driver in MINIX (1)

Terminal driver message types:

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

Terminal Driver in MINIX (2)

Terminal driver message types (continued):

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

Terminal Input (1)

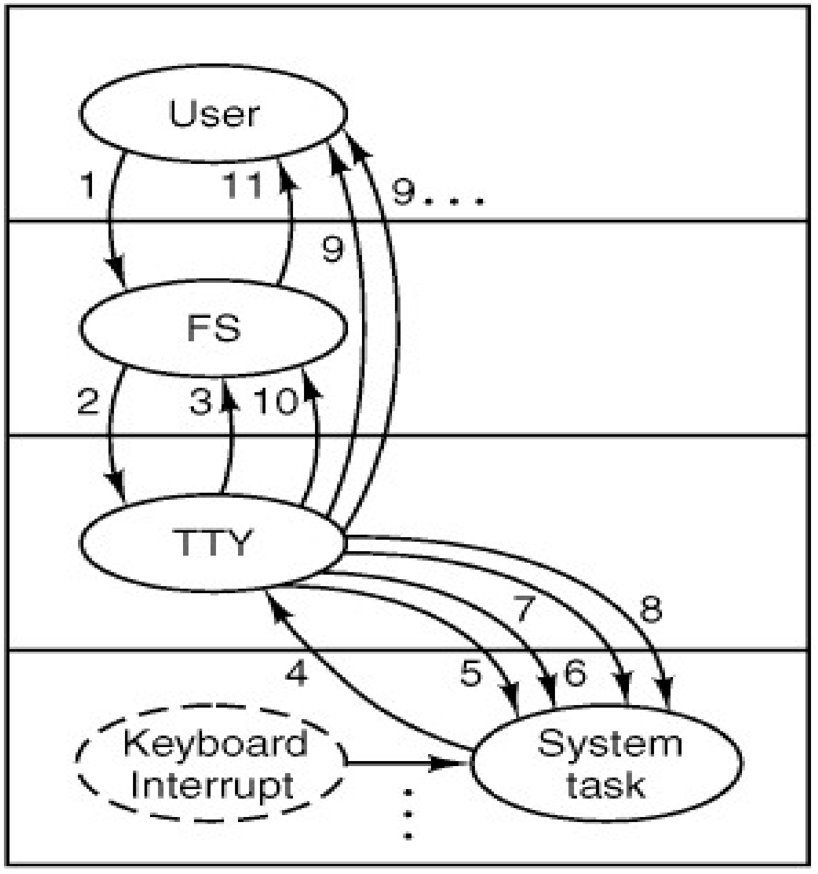


Figure 3-33. Read request from the keyboard when no characters are pending. FS is the file system. TTY is the terminal driver. The TTY receives a message for every keypress and queues scan codes as they are entered. Later these are interpreted and assembled into a buffer of ASCII codes which is copied to the user process.

Terminal Input (2)

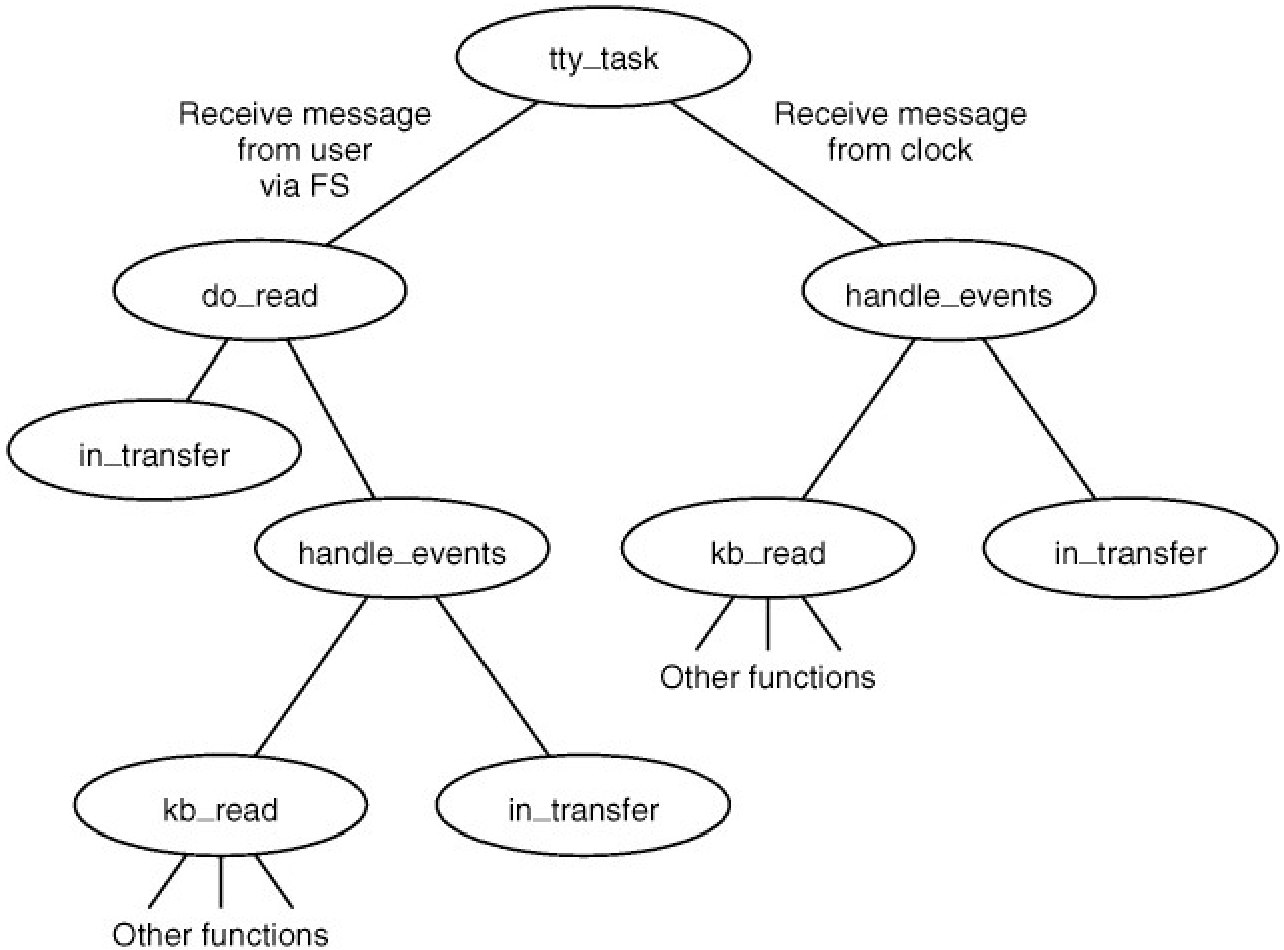


Figure 3-34. Input handling in the terminal driver. The left branch of the tree is taken to process a request to read characters. The right branch is taken when a keyboard message is sent to the driver before a user has requested input.

Terminal Output (1)

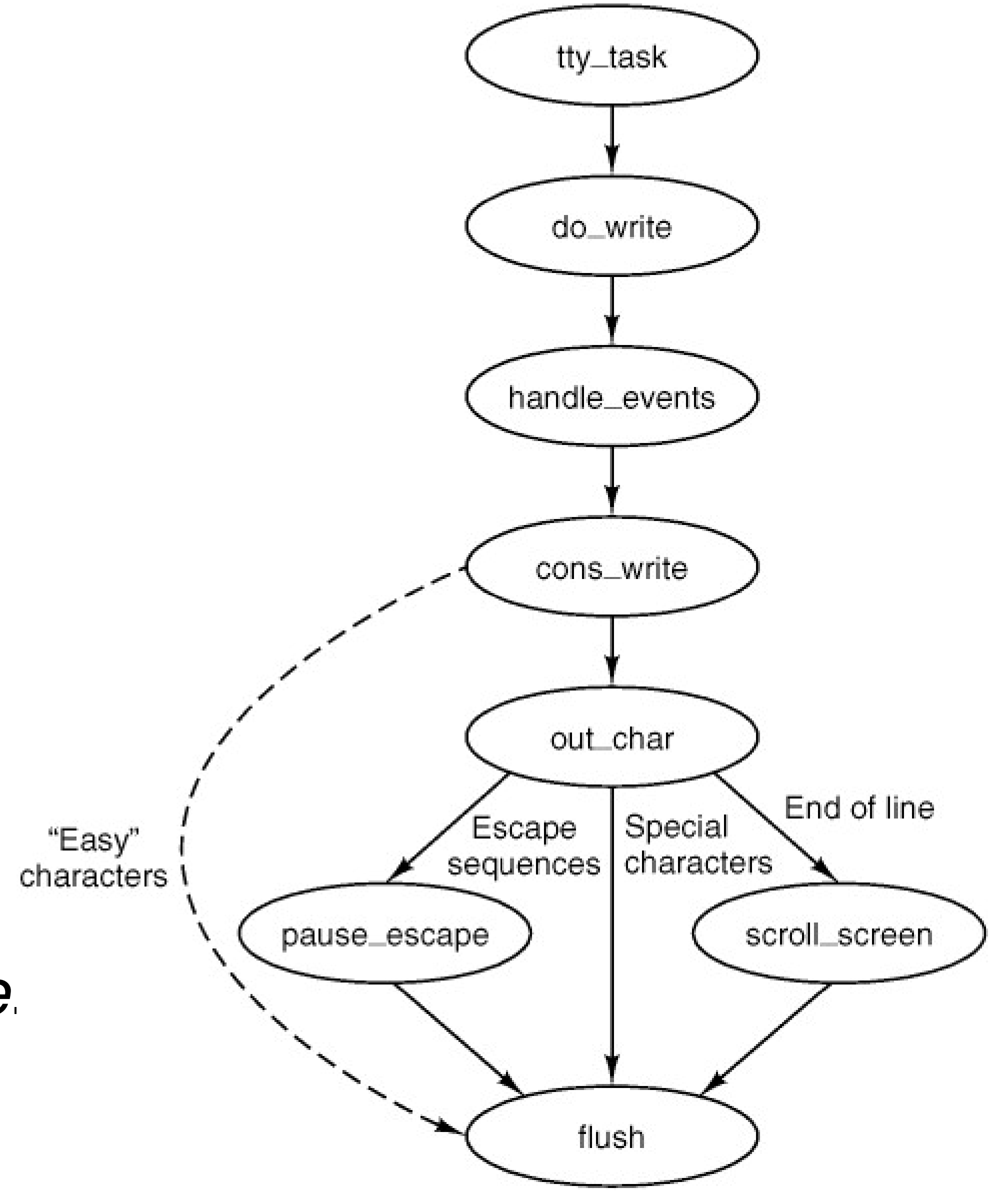


Figure 3-35. Major procedures used in terminal output. The dashed line indicates characters copied directly to *ramqueue* by *cons_write*.

Terminal Output (2)

Field	Meaning
c_start	Start of video memory for this console
c_limit	Limit of video memory for this console
c_column	Current column (0-79) with 0 at left
c_row	Current row (0-24) with 0 at top
c_cur	Offset into video RAM for cursor
c_org	Location in RAM pointed to by 6845 base register

Figure 3-36. Fields of the console structure that relate to the current screen position.

Loadable Keymaps

Scan code	Character	Regular	SHIFT	ALT1	ALT2	ALT+SHIFT	CTRL
00	none	0	0	0	0	0	0
01	ESC	C('[')	C('[')	CA('[')	CA('[')	CA('[')	C('[')
02	'1'	'1'	'!'	A('1')	A('1')	A('!')	C('A')
13	'='	'='	'+'	A('=')	A('=')	A('+')	C('@')
16	'q'	L('q')	'Q'	A('q')	A('q')	A('Q')	C('Q')
28	CR/LF	C('M')	C('M')	CA('M')	CA('M')	CA('M')	C('J')
29	CTRL	CTRL	CTRL	CTRL	CTRL	CTRL	CTRL
59	F1	F1	SF1	AF1	AF1	ASF1	CF1
127	???	0	0	0	0	0	0

Figure 3-37. A few entries from a keymap source file.

Device-Independent Terminal Driver (1)

Field	Default values
c_iflag	BRKINT ICRNL IXON IXANY
c_oflag	OPOST ONLCR
c_cflag	CREAD CS8 HUPCL
c_lflag	ISIG IEXTEN ICANON ECHO ECHOE

Figure 3-38. Default termios flag values.

Device-Independent Terminal Driver (2)

POSIX function	POSIX operation	IOCTL type	IOCTL parameter
tcdrain	(none)	TCDRAIN	(none)
tcflow	TCOOFF	TCFLOW	int=TCOOFF
tcflow	TCOON	TCFLOW	int=TCOON
tcflow	TCIOFF	TCFLOW	int=TCIOFF
tcflow	TCION	TCFLOW	int=TCION
tcflush	TCIFLUSH	TCFLSH	int=TCIFLUSH
tcflush	TCOFLUSH	TCFLSH	int=TCOFLUSH
tcflush	TCIOFLUSH	TCFLSH	int=TCIOFLUSH
tcgetattr	(none)	TCGETS	termios
tcsetattr	TCSANOW	TCSETS	termios
tcsetattr	TCSADRAIN	TCSETSW	termios
tcsetattr	TCSAFLUSH	TCSETSF	termios
tcsendbreak	(none)	TCSBRK	int=duration

Figure 3-39. POSIX calls and IOCTL operations.

Terminal Driver Support Code

0	V	D	N	c	c	c	c	7	6	5	4	3	2	1	0

V: IN_ESC, escaped by LNEXT (CTRL-V)
D: IN_EOF, end of file (CTRL-D)
N: IN_EOT, line break (NL and others)
cccc: count of characters echoed
7: Bit 7, may be zeroed if ISTRIP is set
6-0: Bits 0-6, ASCII code

Figure 3-40. The fields in a character code as it is placed into the input queue.

Keyboard Driver (1)

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-

Figure 3-41. 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.

Keyboard Driver (2)

Key	Scan code	"ASCII"	Escape sequence
Home	71	0x101	ESC [H
Up Arrow	72	0x103	ESC [A
Pg Up	73	0x107	ESC [V
–	74	0x10A	ESC [S
Left Arrow	75	0x105	ESC [D
5	76	0x109	ESC [G
Right Arrow	77	0x106	ESC [C
+	78	0x10B	ESC [T
End	79	0x102	ESC [Y
Down Arrow	80	0x104	ESC [B
Pg Dn	81	0x108	ESC [U
Ins	82	0x10C	ESC [@

Figure 3-42. Escape codes generated by the numeric keypad. When scan codes for ordinary keys are translated into ASCII codes the special keys are assigned "pseudo ASCII" codes with values greater than 0xFF.

Keyboard Driver (3)

Key	Purpose
F1	Kernel process table
F2	Process memory maps
F3	Boot image
F4	Process privileges
F5	Boot monitor parameters
F6	IRQ hooks and policies
F7	Kernel messages
F10	Kernel parameters
F11	Timing details (if enabled)
F12	Scheduling queues
SF1	Process manager process table
SF2	Signals
SF3	File system process table
SF4	Device/driver mapping
SF5	Print key mappings
SF9	Ethernet statistics (RTL8139 only)
CF1	Show key mappings
CF3	Toggle software/hardware console scrolling
CF7	Send SIGQUIT, same effect as CTRL-\
CF8	Send SIGINT, same effect as CTRL-C
CF9	Send SIGKILL, same effect as CTRL-U

Figure 3-43. The function keys detected by func_key().

Display Driver (1)

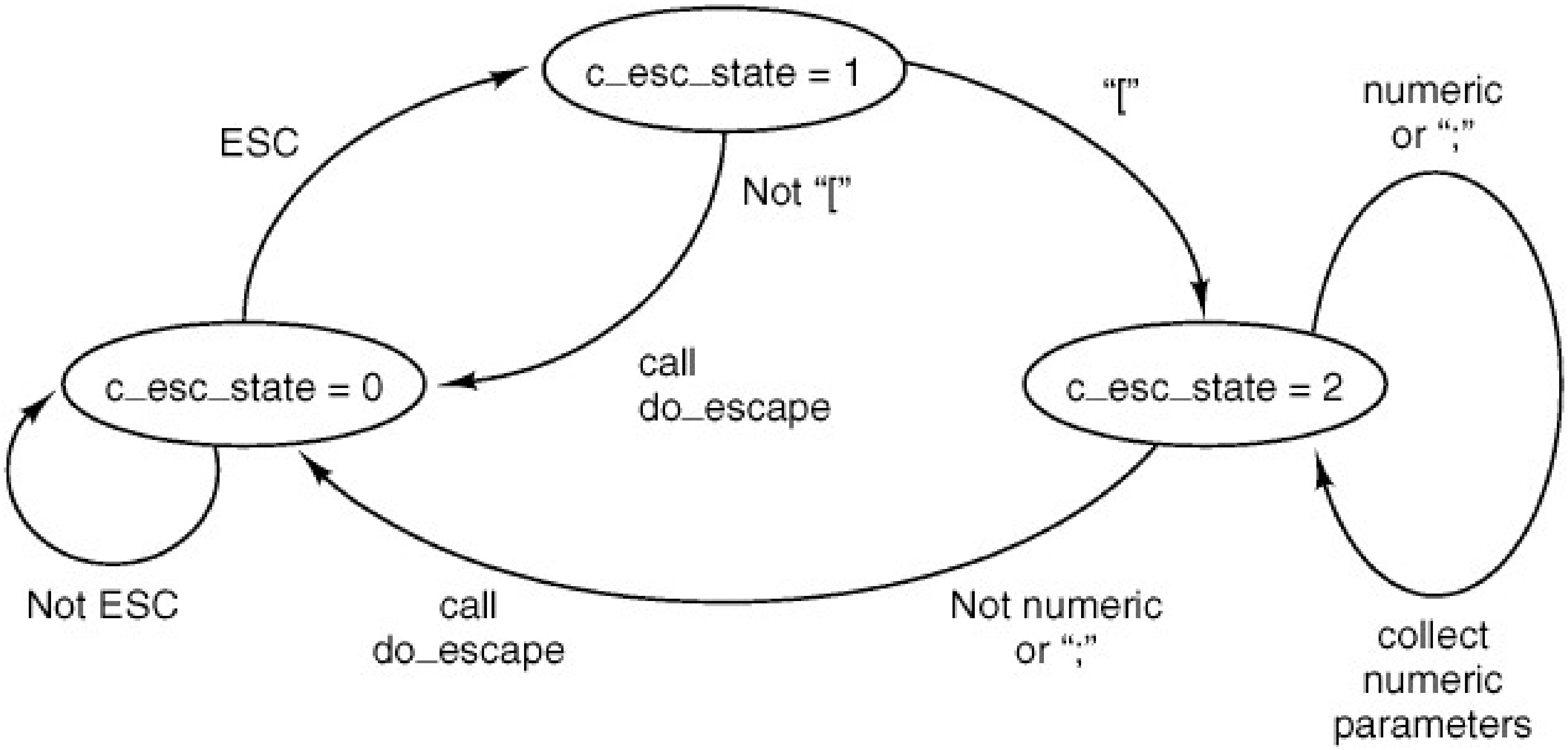


Figure 3-44. Finite state machine for processing escape sequences.

Display Driver (2)

Registers	Function
10 – 11	Cursor size
12 – 13	Start address for drawing screen
14 – 15	Cursor position

Figure 3-45. Some of the 6845's registers.