3

INPUT/OUTPUT

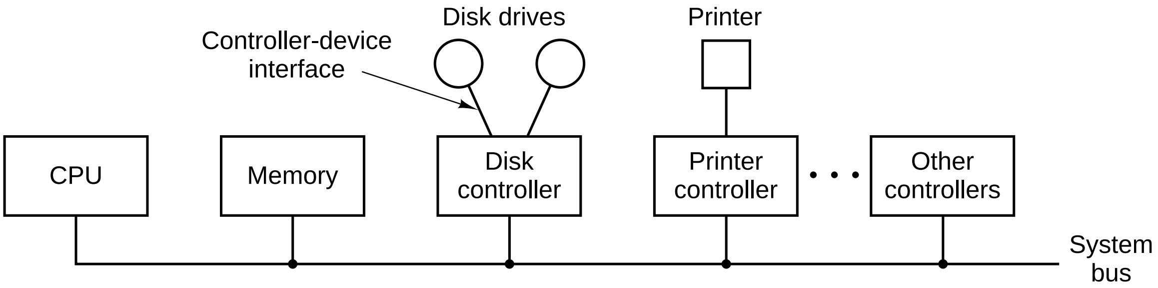

Figure 3-1. A model for connecting the CPU, memory, controllers, and I/O devices.

I/O controller	I/O address	Hardware IRQ	Interrupt vector
Clock	040 – 043	0	8
Keyboard	060 – 063	1	9
Hard disk	1F0 – 1F7	14	118
Secondary RS232	2F8 – 2FF	3	11
Printer	378 – 37F	7	15
Floppy disk	3F0 – 3F7	6	14
Primary RS232	3F8 – 3FF	4	12

Figure 3-2. Some examples of controllers, their I/O addresses, their hardware interrupt lines, and their interrupt vectors on a typical PC running MS-DOS.

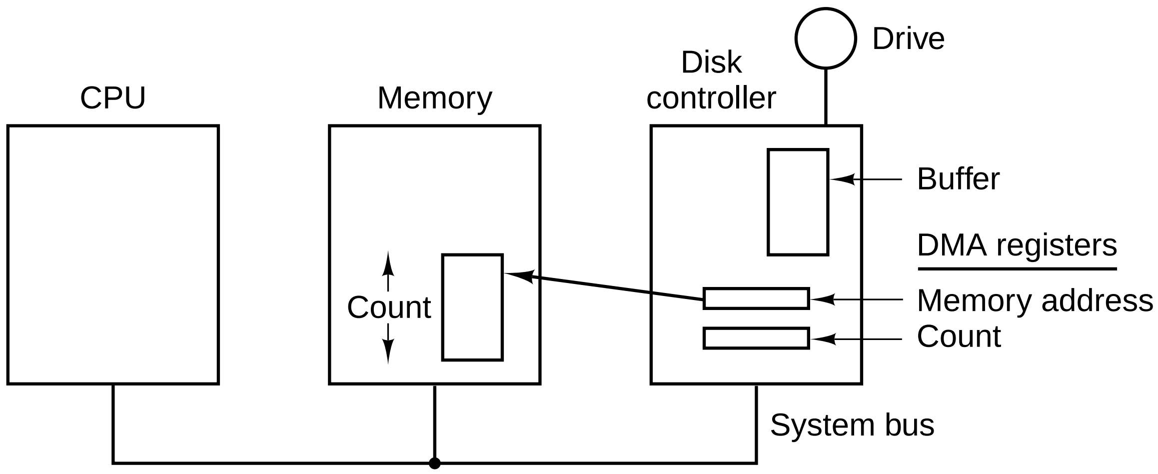

Figure 3-3. A DMA transfer is done entirely by the controller.

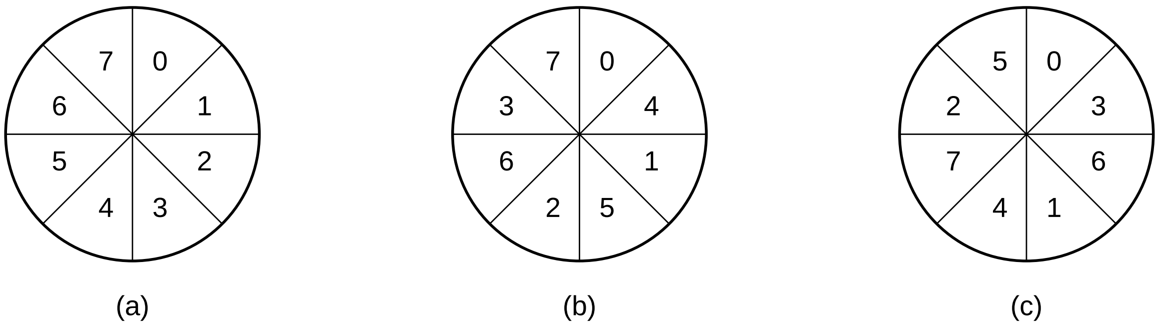

Figure 3-4. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.

Uniform interfacing for device drivers
Device naming
Device protection
Providing a device-independent block size
Buffering
Storage allocation on block devices
Allocating and releasing dedicated devices
Error reporting

Figure 3-5. Functions of the device-independent I/O software.

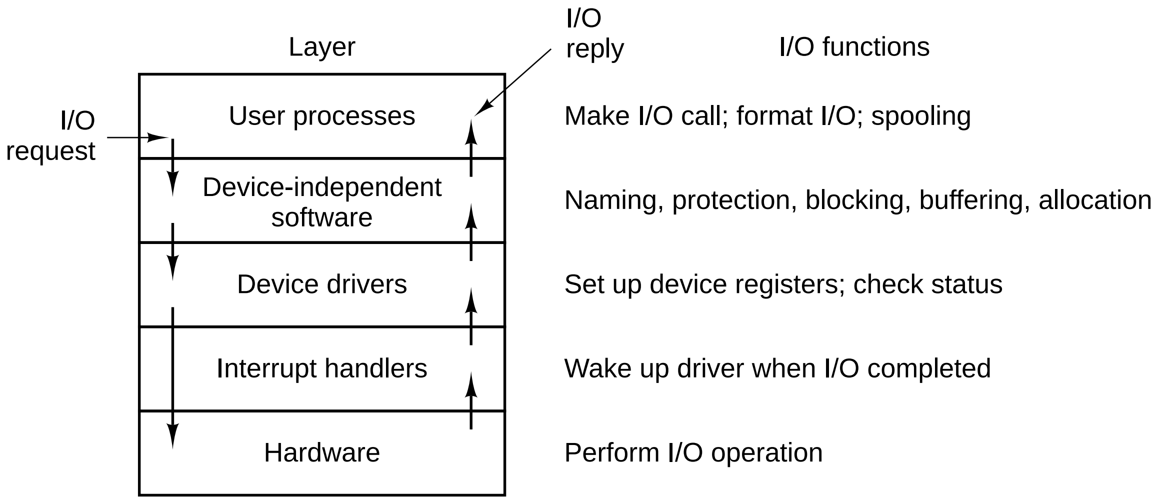

Figure 3-6. Layers of the I/O system and the main functions of each layer.

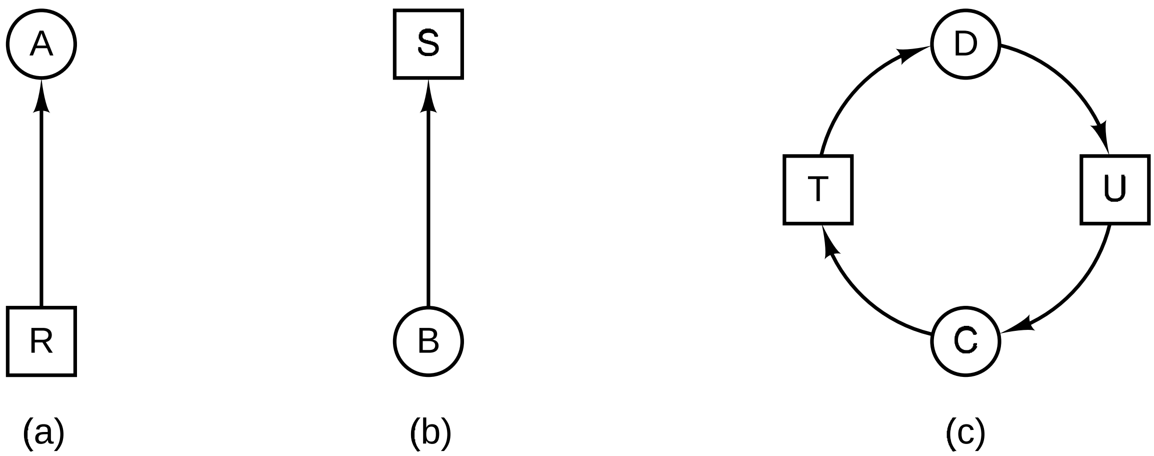

Figure 3-7. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

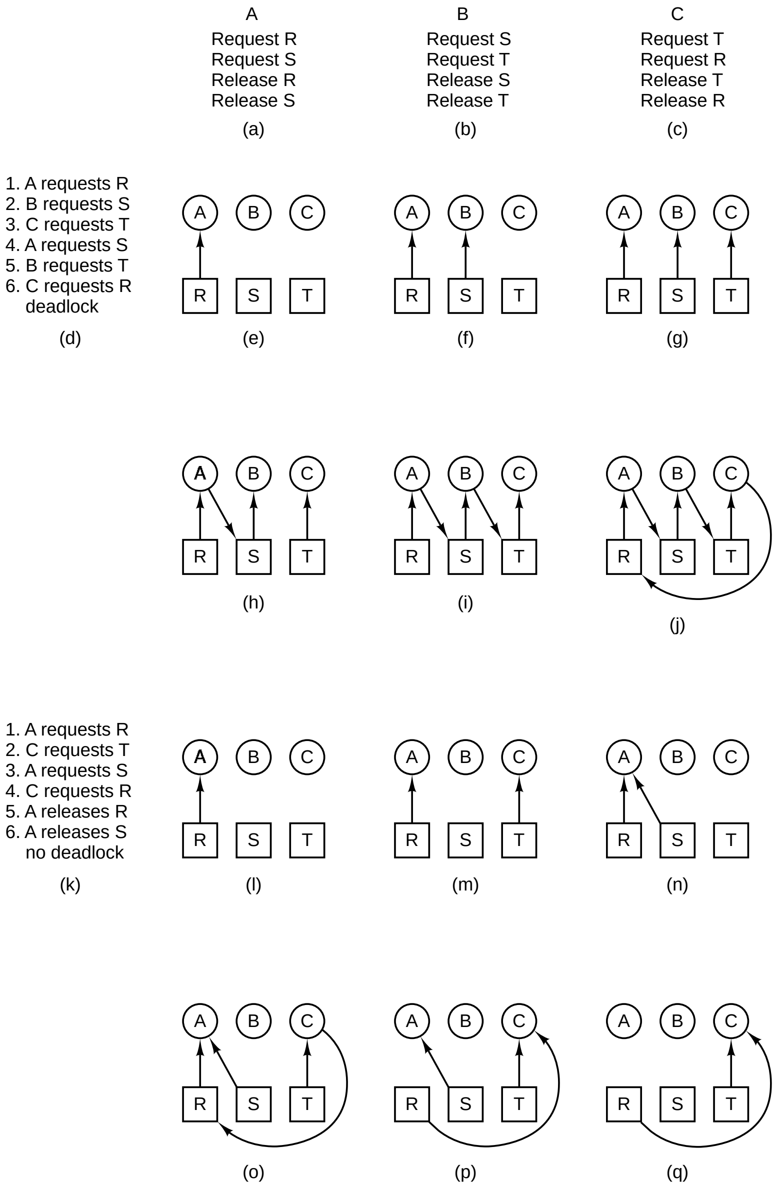

Figure 3-8. An example of how deadlock occurs and how it can be avoided.

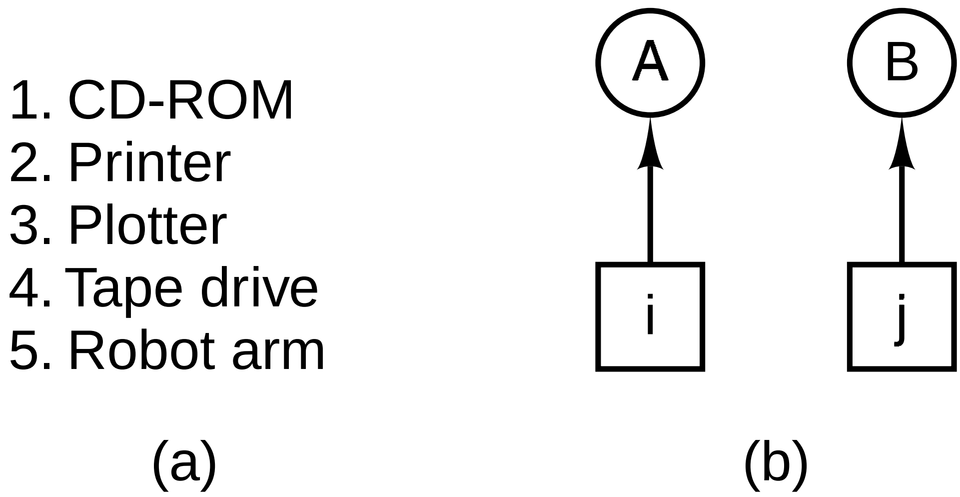

Figure 3-9. (a) Numerically ordered resources. (b) A resource graph.

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

Figure 3-10. Summary of approaches to deadlock prevention.

Name	Used	Maximum
Andy	0	6
Barbara	0	5
Marvin	0	4
Suzanne	0	7

Available: 10

(a)

Name	Used	Maximum
Andy	1	6
Barbara	1	5
Marvin	2	4
Suzanne	4	7

Available: 2

(b)

Name	Used	Maximum
Andy	1	6
Barbara	2	5
Marvin	2	4
Suzanne	4	7

Available: 1

(c)

Figure 3-11. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

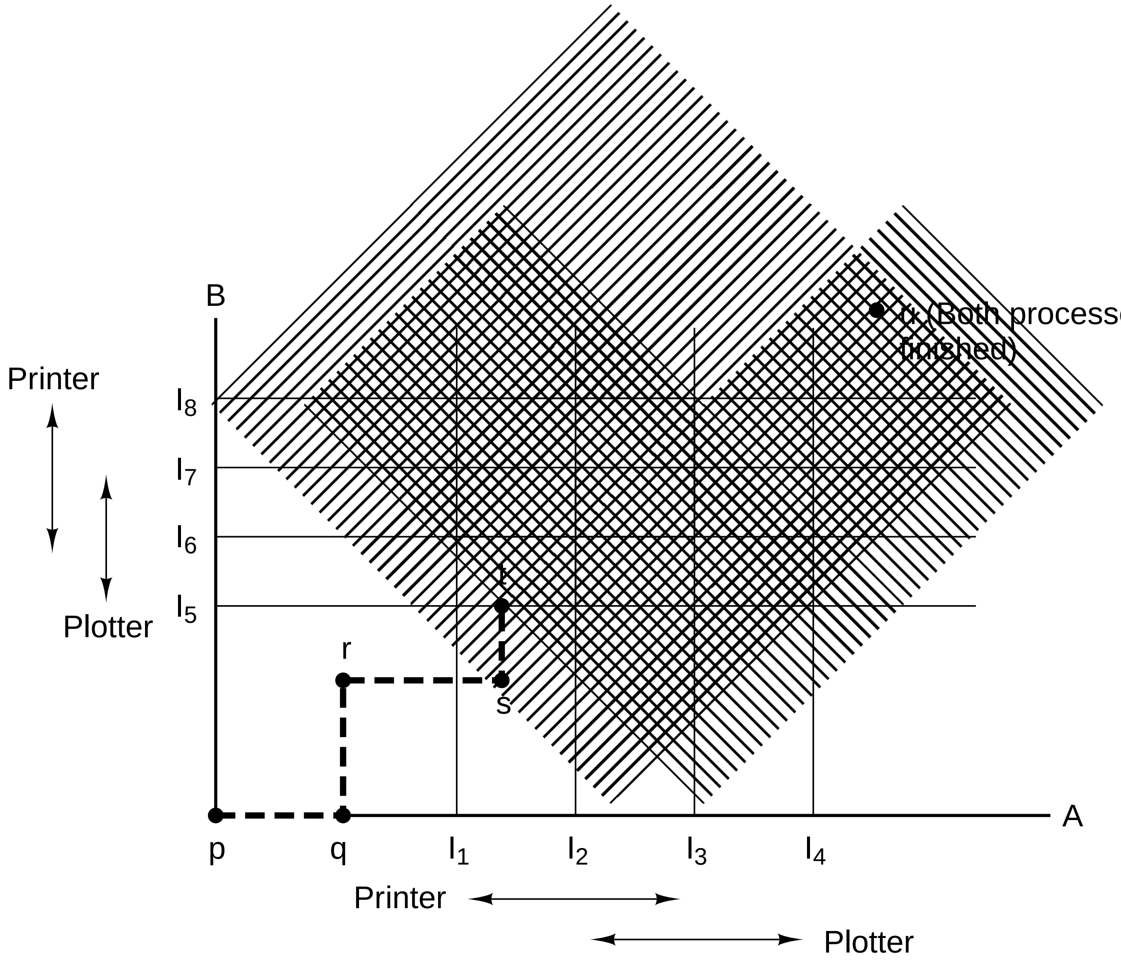

Figure 3-12. Two process resource trajectories.

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 3-13. The banker's algorithm with multiple resources.

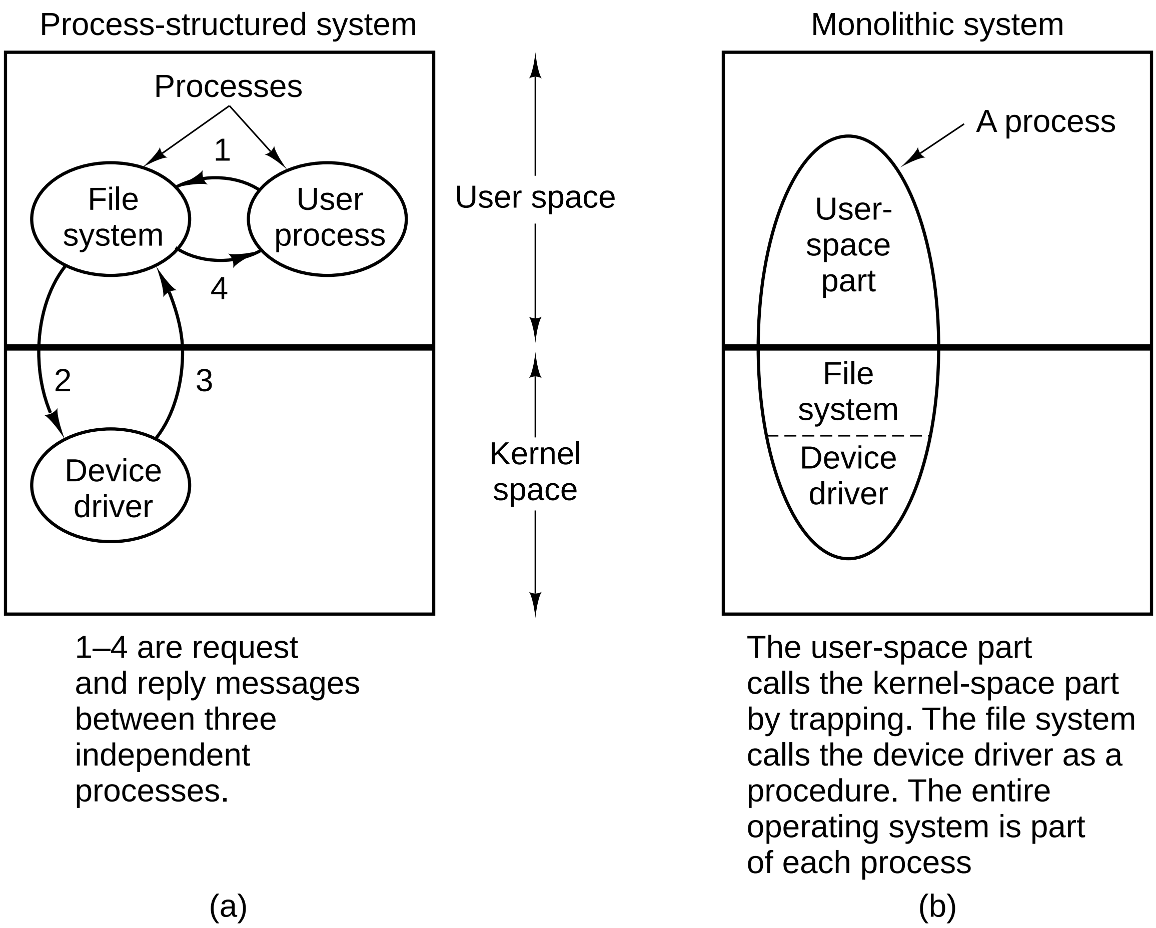

Figure 3-14. Two ways of structuring user-system communication.

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 TASK_REPLY
m.REP_PROC_NR	int	Same as PROC_NR in request
m.REP_STATUS	int	Bytes transferred or error number

Figure 3-15. Fields of the messages sent by the file system to the block device drivers and fields of the replies sent back.

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

void io_task() {
  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 = TASK_REPLY;
        mess.status = rcode; /* result code */
        send(caller, &mess); /* send reply message back to caller */
  }
}
```

Figure 3-16. Outline of the main procedure of an I/O task.

```
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 3-17. A shared I/O task main procedure using indirect calls.

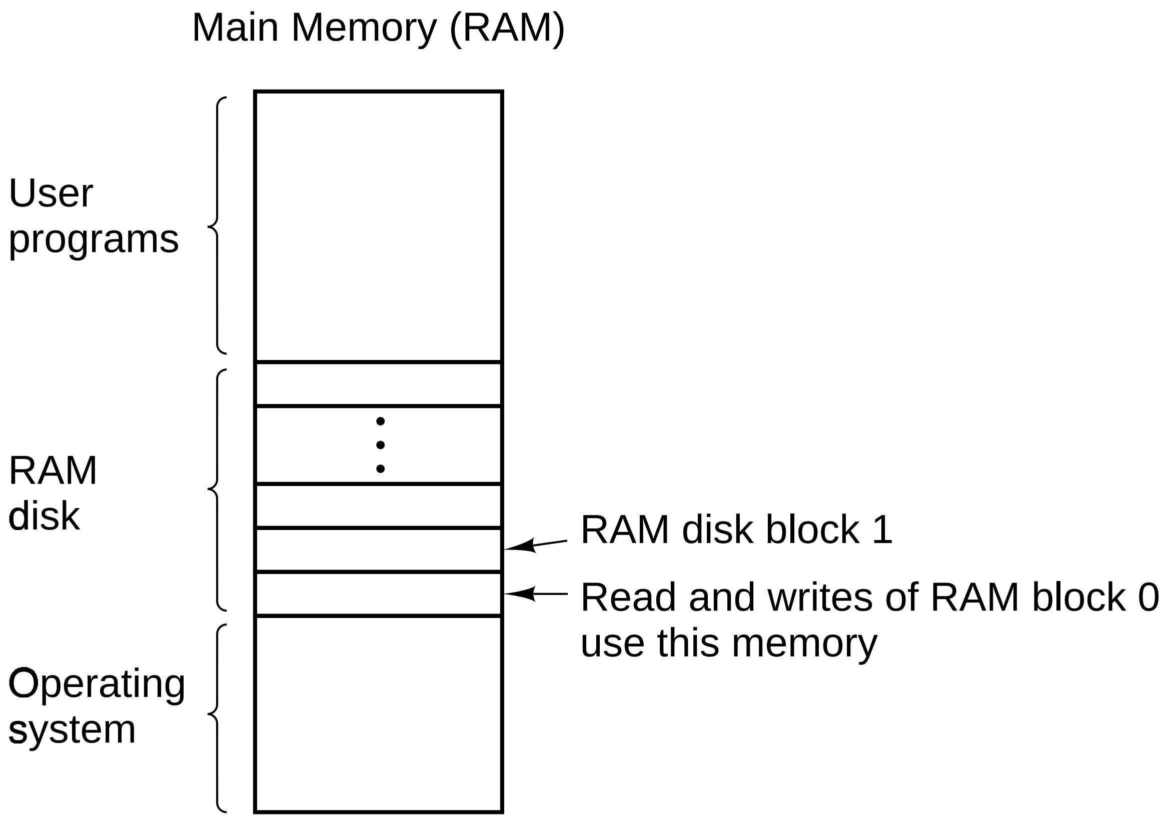

Figure 3-18. A RAM disk.

Parameter	IBM 360-KB floppy disk	WD 540-MB hard disk
Number of cylinders	40	1048
Tracks per cylinder	2	4
Sectors per track	9	252
Sectors per disk	720	1056384
Bytes per sector	512	512
Bytes per disk	368640	540868608
Seek time (adjacent cylinders)	6 msec	4 msec
Seek time (average case)	77 msec	11 msec
Rotation time	200 msec	13 msec
Motor stop/start time	250 msec	9 sec
Time to transfer 1 sector	22 msec	53 μsec

Figure 3-19. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD AC2540 540-MB hard disk.

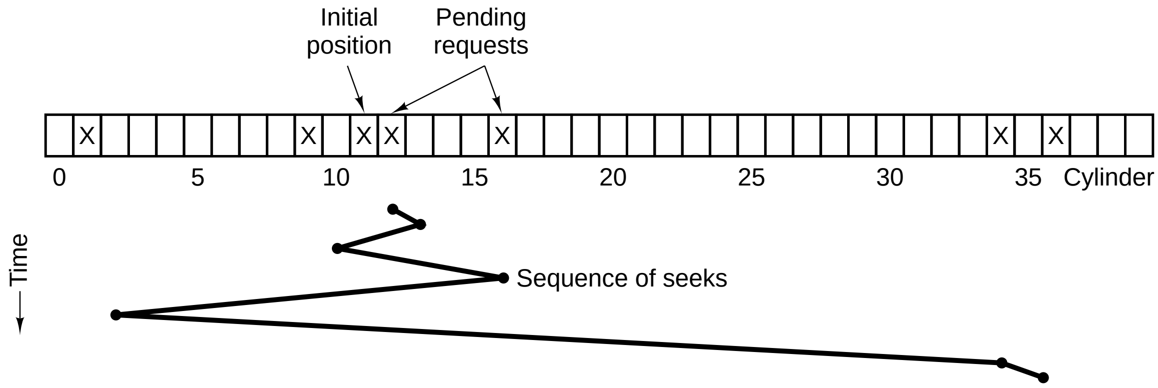

Figure 3-20. Shortest Seek First (SSF) disk scheduling algorithm.

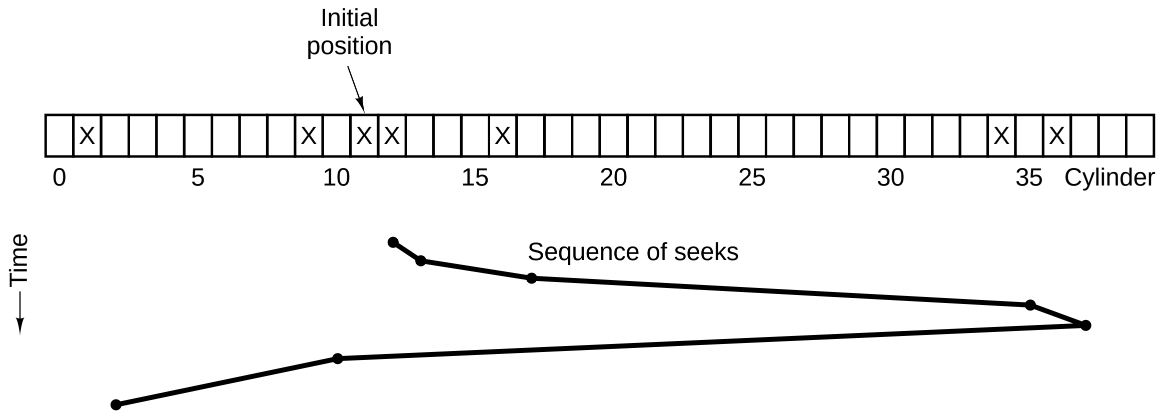

Figure 3-21. The elevator algorithm for scheduling disk requests.

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)

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 3-22. (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. (b) The fields of the Select Drive/Head register.

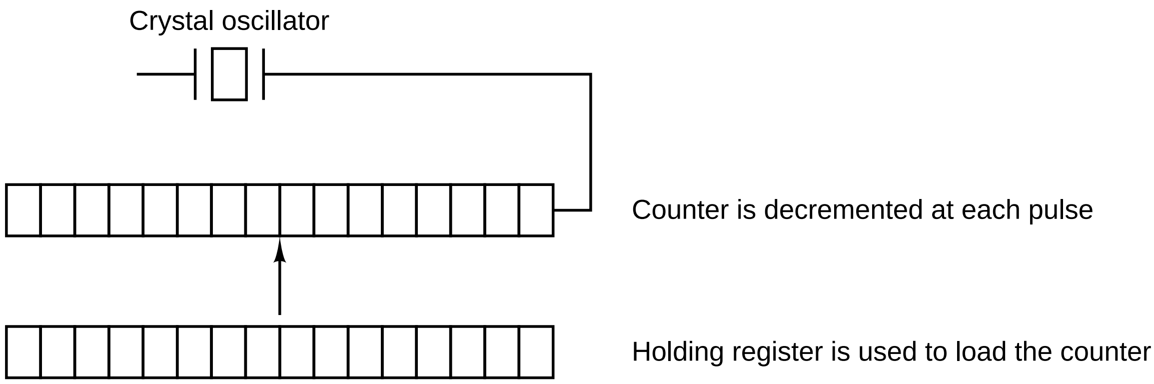

Figure 3-23. A programmable clock.

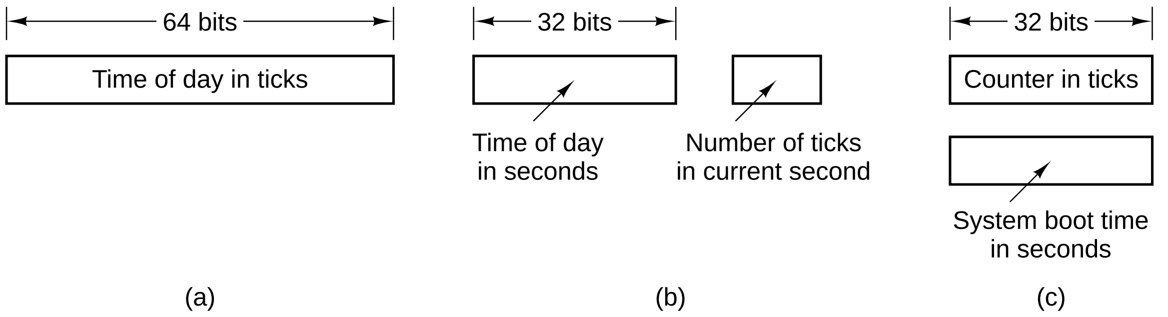

Figure 3-24. Three ways to maintain the time of day.

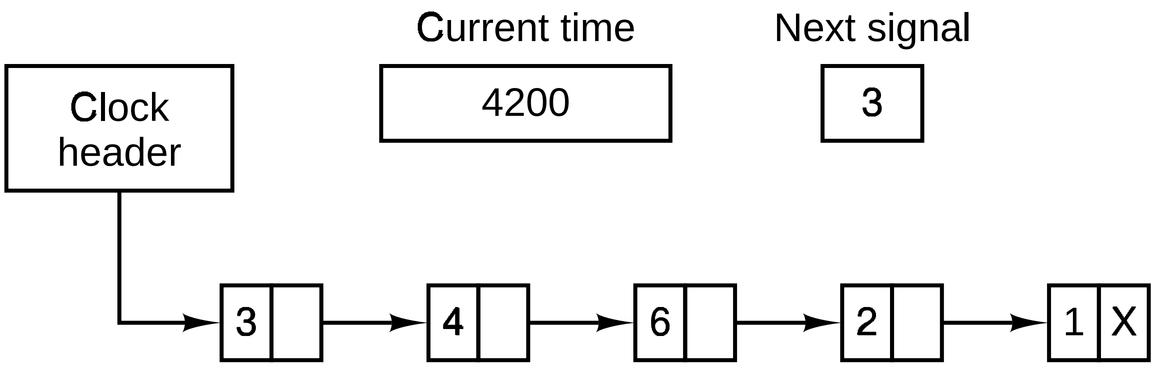

Figure 3-25. Simulating multiple timers with a single clock.

Service	Access	Response	Clients
Gettime	System call	Message	Any process
Uptime	System call	Message	Any process
Uptime	Function call	Function value	Kernel or task
Alarm	System call	Signal	Any process
Alarm	System call	Watchdog activation	Task
Synchronous alarm	System call	Message	Server process
Milli_delay	Function call	Busy wait	Kernel or task
Milli_elapsed	Function call	Function value	Kernel or task

Figure 3-26. The clock code supports a number of time-related services.

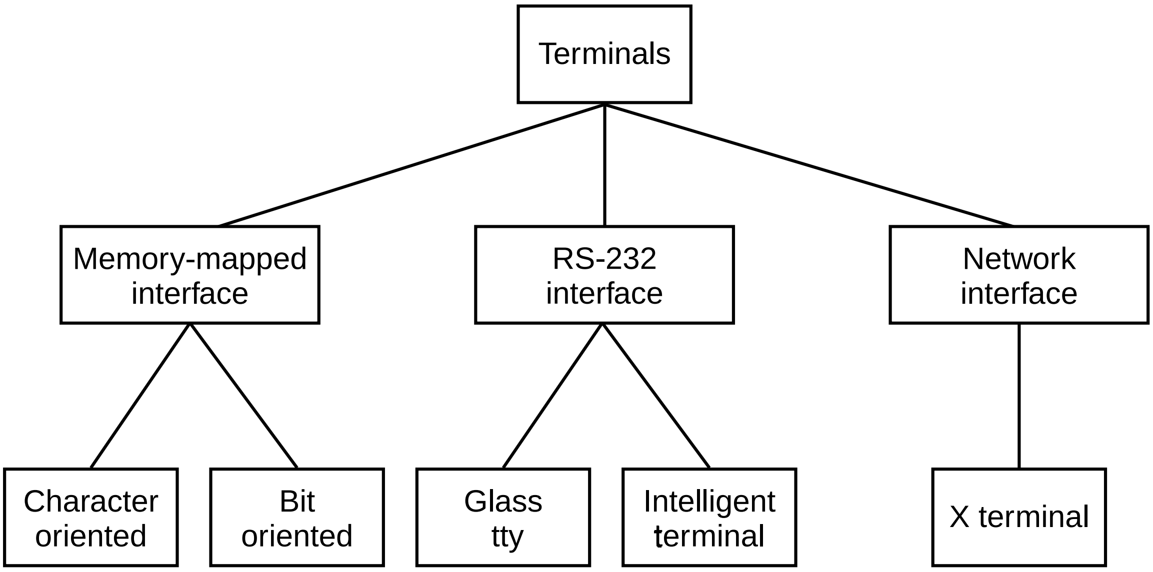

Figure 3-27. Terminal types.

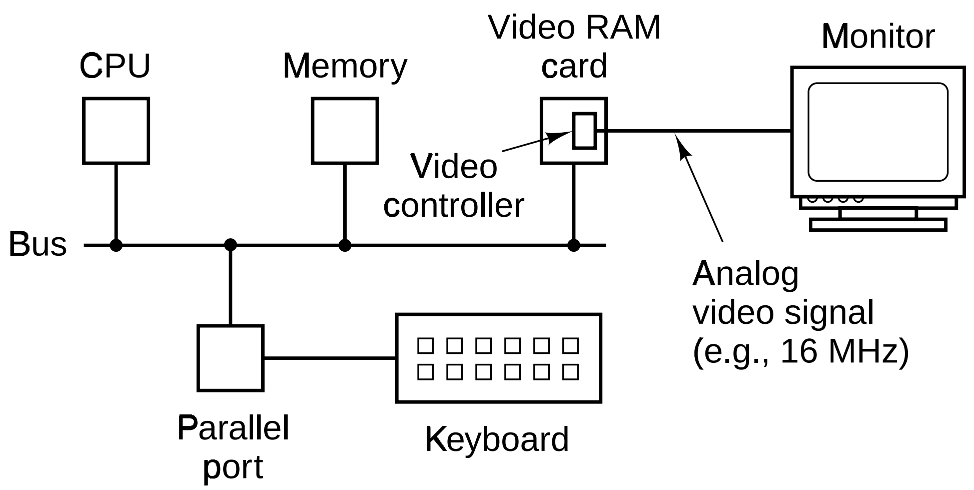

Figure 3-28. Memory-mapped terminals write directly into video RAM.

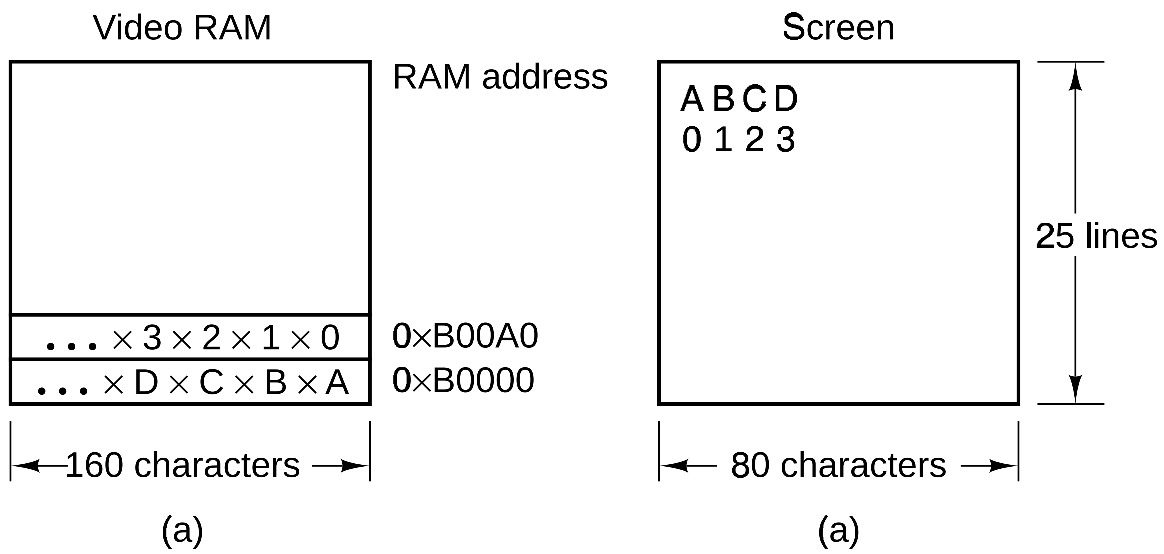

Figure 3-29. (a) A video RAM image for the IBM monochrome display. (b) The corresponding screen. The ×s are attribute bytes.

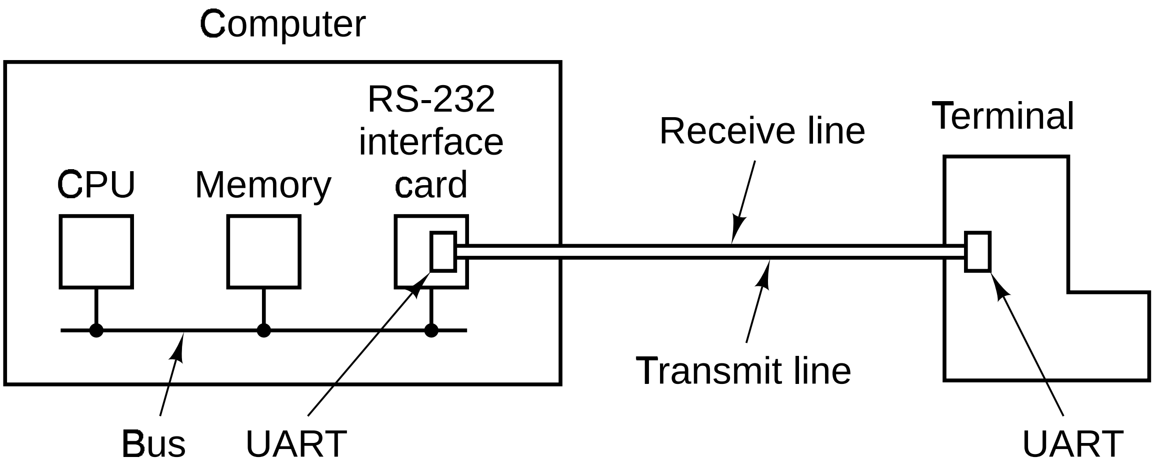

Figure 3-30. An RS-232 terminal communicates with a computer over a communication line, one bit at a time. The computer and the terminal are completely independent.

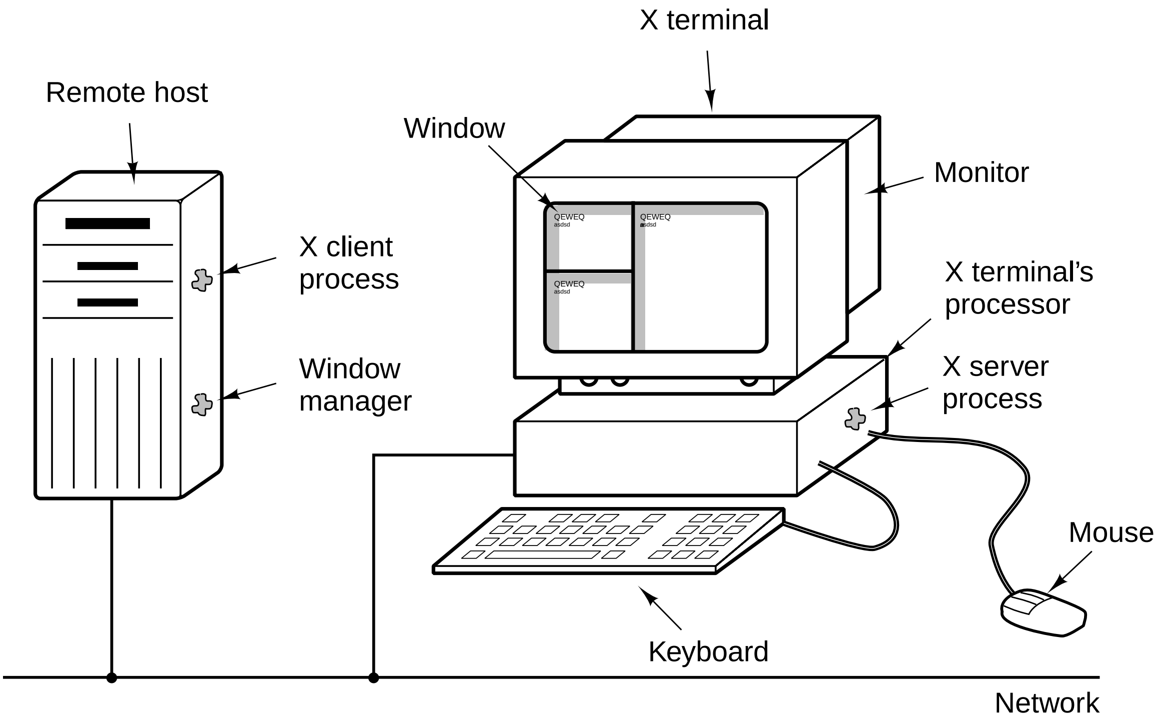

Figure 3-31. Clients and servers in the M.I.T. X Window System.

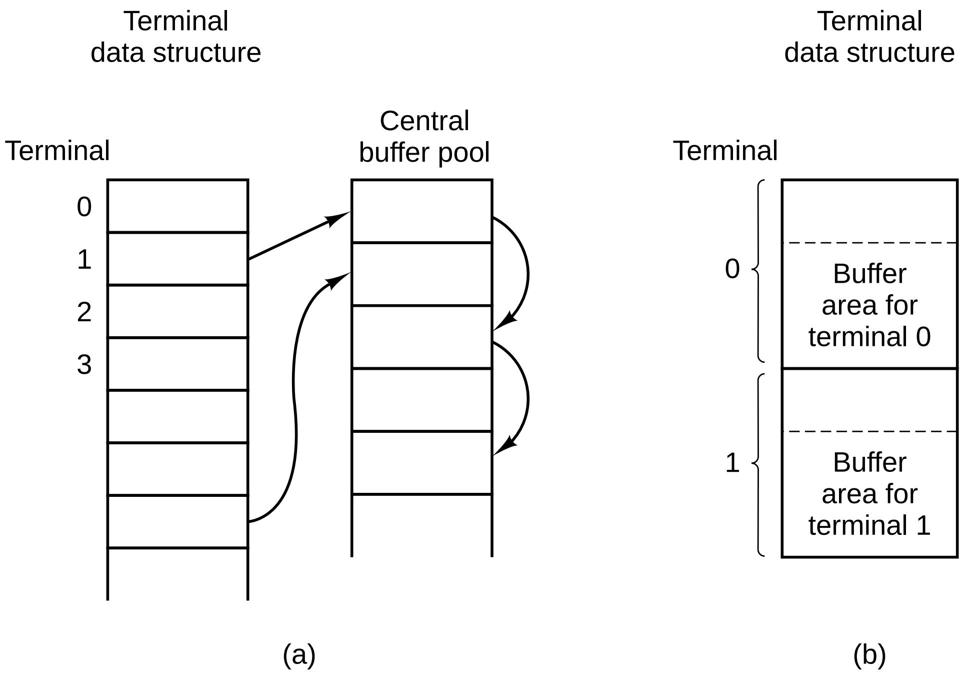

Figure 3-32. (a) Central buffer pool. (b) Dedicated buffer for each terminal.

Character	POSIX name	Comment
CTRL-D	EOF	End of file
	EOL	End of line (undefined)
CTRL-H	ERASE	Backspace one character
DEL	INTR	Interrupt process (SIGINT)
CTRL-U	KILL	Erase entire line being typed
CTRL-\	QUIT	Force core dump (SIGQUIT)
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 3-33. Characters that are handled specially in canonical mode.

```
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 3-34. The termios structure. In MINIX tc_flag_t is a short, speed_t is an int, cc_t is a char.

	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 3-35. *MIN* and *TIME* determine when a call to read returns in noncanonical mode. *N* is the number of bytes requested.

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 (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 from start, 2 all)
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 3-36. 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.

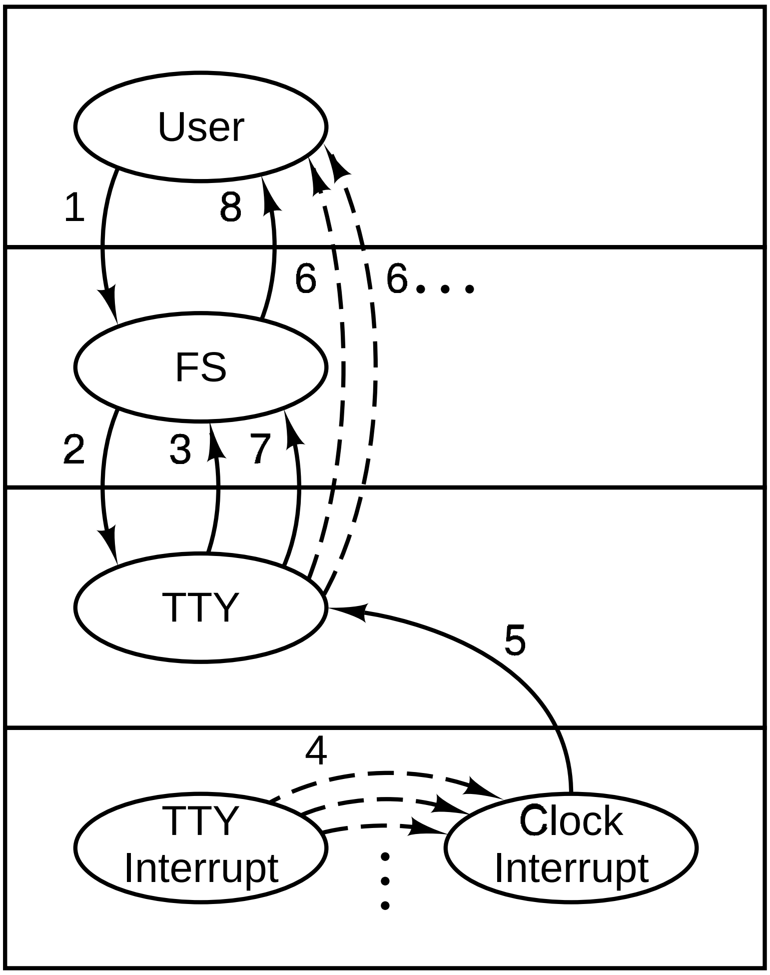

Figure 3-37. Read request from terminal when no characters are pending. FS is the file system. TTY is the terminal task. The interrupt handler for the terminal queues characters as they are entered, but it is the clock interrupt handler that awakens TTY.

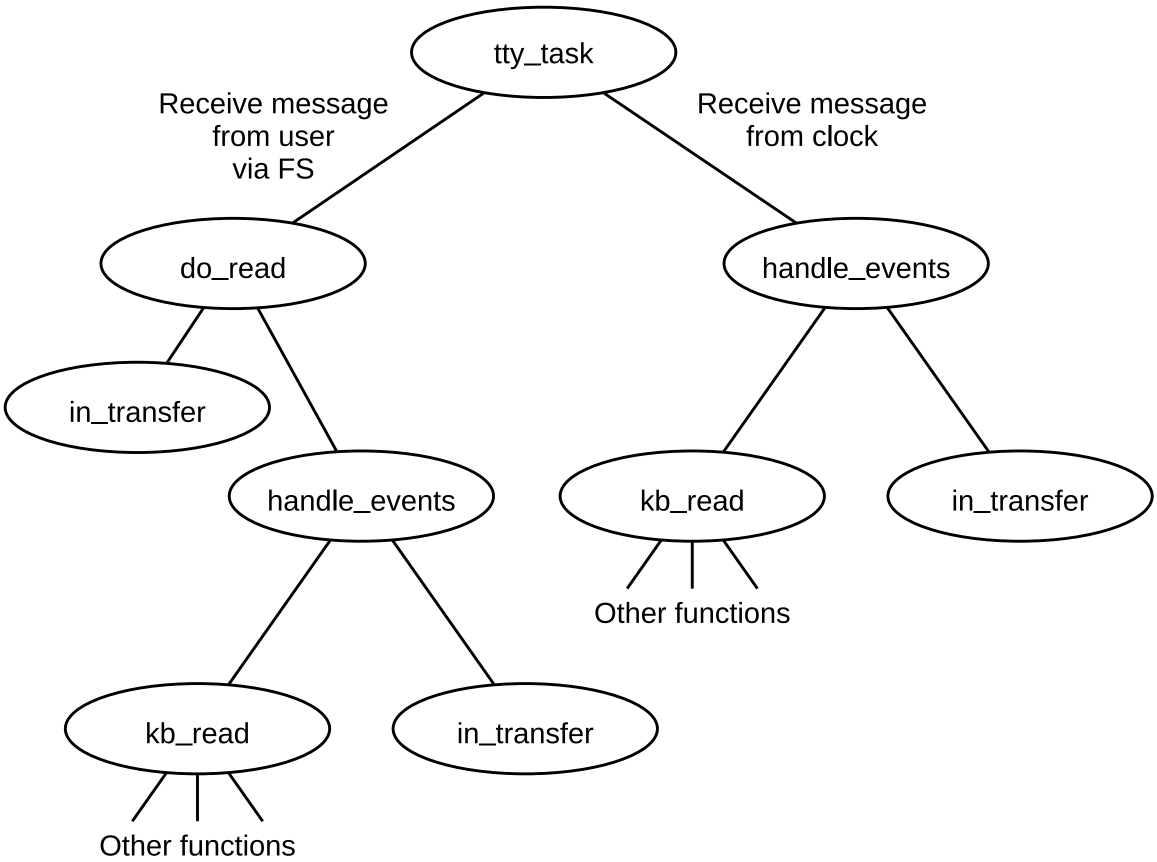

Figure 3-38. Input handling in the terminal driver. The left branch of the tree is taken to process a request to read characters. The right branch is taken when a character-has-been-typed message is sent to the driver.

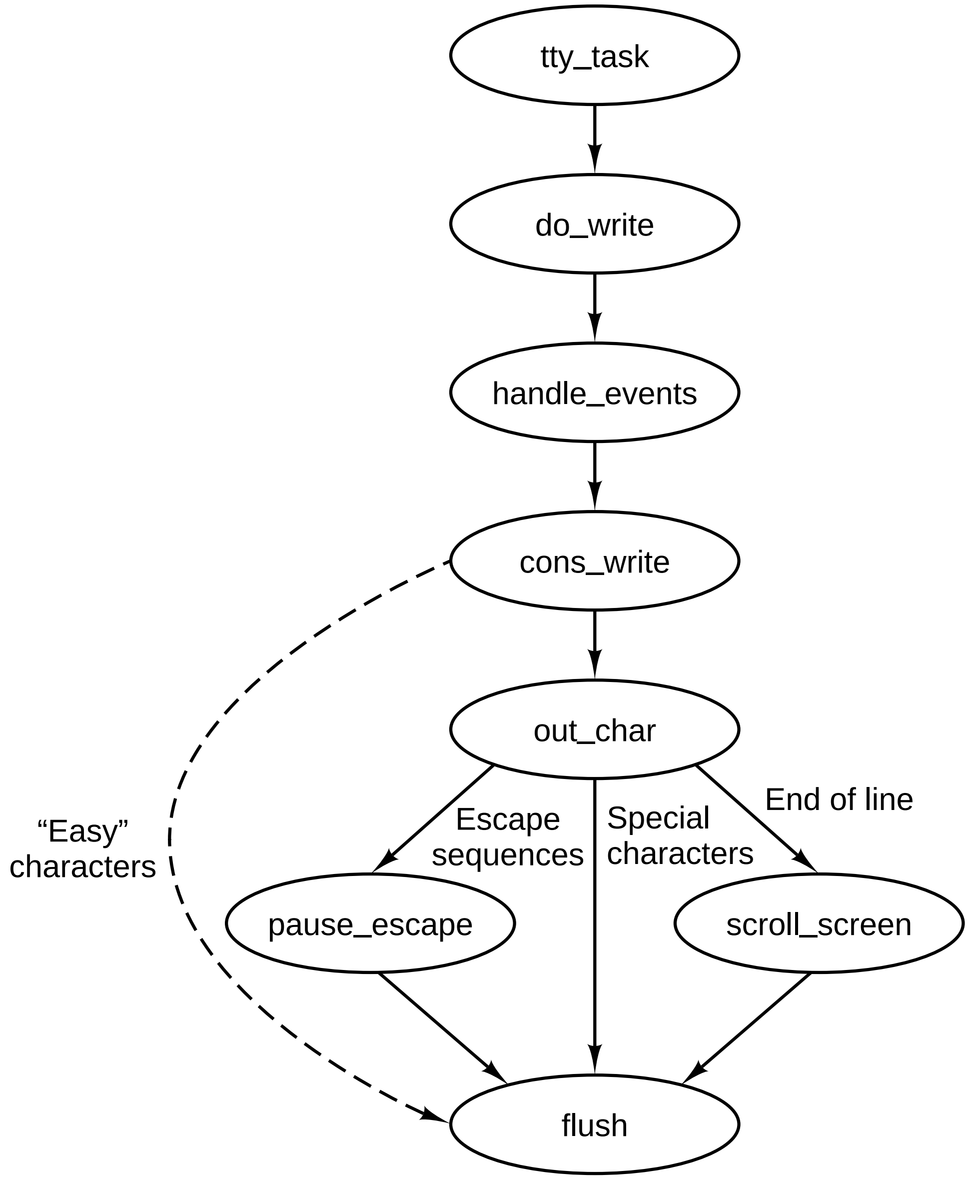

Figure 3-39. Major procedures used on terminal output. The dashed line indicates characters copied directly to *ramqueue* by *cons_write*.

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 3-40. Fields of the console structure that relate to the current screen position.

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 3-41. A few entries from a keymap source file.

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 3-42. Default termios flag values.

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 3-43. POSIX calls and IOCTL operations.

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 3-44. The fields in a character code as it is placed into the input queue.

42	35	170	18	38	38	24	57	54	17	182	24	19	38	32	28

Figure 3-45. Scan codes in the input buffer, with corresponding key presses below, for a line of text entered at the keyboard. L+, L-, R+, and R- represent, respectively, pressing and releasing the left and right Shift keys. The code for a key release is 128 more than the code for a press of the same key.

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 3-46. 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.

Key	Purpose
F1	Display process table
F2	Display details of process memory use
F3	Toggle between hardware and software scrolling
F5	Show Ethernet statistics (if network support compiled)
CF7	Send SIGQUIT, same effect as CTRL-\
CF8	Send SIGINT, same effect as DEL
CF9	Send SIGKILL, same effect as CTRL-U

Figure 3-47. The function keys detected by *func_key()*.

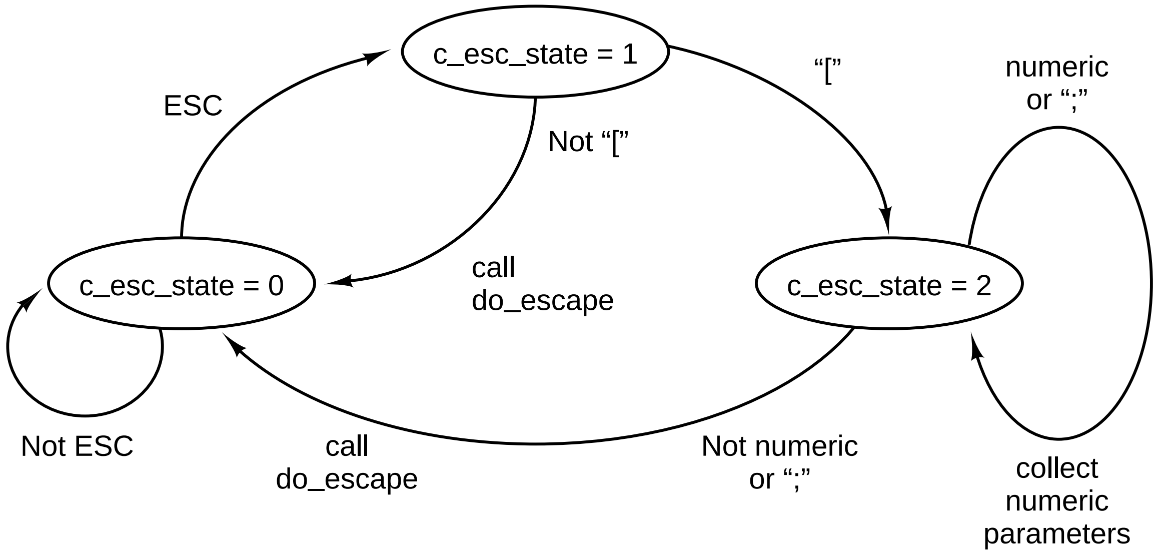

Figure 3-48. Finite state machine for processing escape sequences.

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

Figure 3-49. Some of the 6845's registers.

Message type	From	Meaning
SYS_FORK	MM	A process has forked
SYS_NEWMAP	MM	Install memory map for a new process
SYS_GETMAP	MM	MM wants memory map of a process
SYS_EXEC	MM	Set stack pointer after EXEC call
SYS_XIT	MM	A process has exited
SYS_TIMES	FS	FS wants a process' execution times
SYS_ABORT	Both	Panic: MINIX is unable to continue
SYS_SENDSIG	MM	Send a signal to a process
SYS_SIGRETURN	MM	Cleanup after completion of a signal.
SYS_KILL	FS	Send signal to a process after KILL call
SYS_ENDSIG	MM	Cleanup after a signal from the kernel
SYS_COPY	Both	Copy data between processes
SYS_VCOPY	Both	Copy multiple blocks of data between processes
SYS_GBOOT	FS	Get boot parameters
SYS_MEM	MM	MM wants next free chunk of physical memory
SYS_UMAP	FS	Convert virtual address to physical address
SYS_TRACE	MM	Carry out an operation of the PTRACE call

Figure 3-50. The message types accepted by the system task.

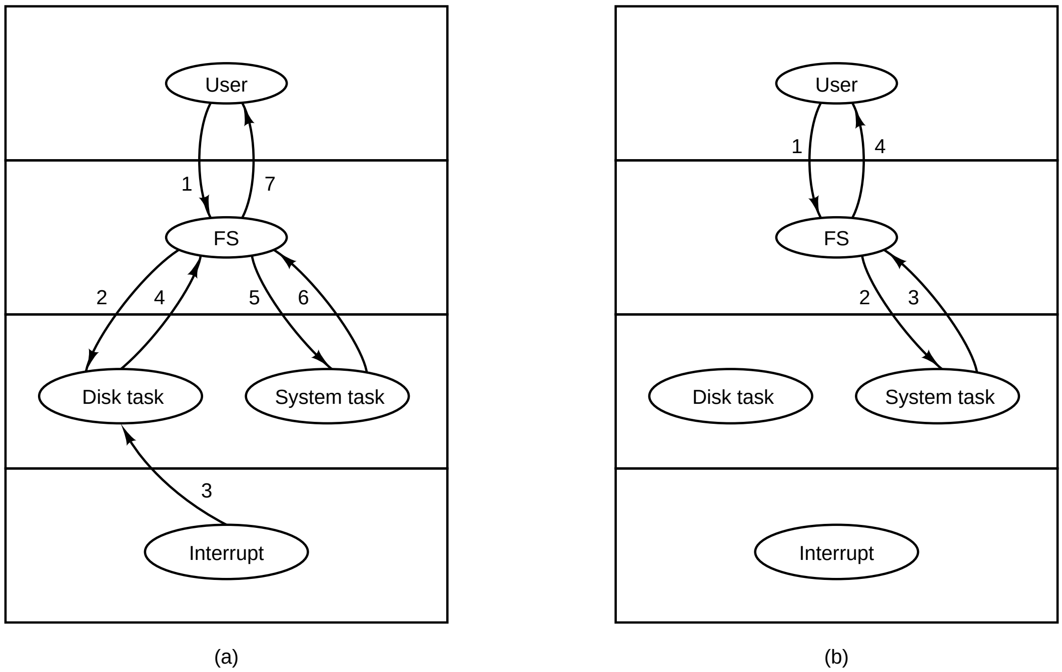

Figure 3-51. (a) Worst case for reading a block requires seven messages. (b) Best case for reading a block requires four messages.