Open Computing “Hands-On”: “Wizard’s Grabbag” Column: February 1994: Listings

Listing: The `upload` program collects input characters, writing them to an output file.

A. Listing of the `upload` program:

```
#ifdef TERMIO                   /* System V or BSD4.3+ */
#include <termio.h>             /* Declares struct termio and flags */
#define STDIN_FILENO    0       /* Standard-input file descriptor */
struct termio   oldterm,        /* Original driver settings */
                newterm;        /* "raw" I/O driver settings */
#else                           /* Use POSIX serial I/O interface */
#define _POSIX_SOURCE   1       /* Hide vendor extensions to POSIX */
#include <stdlib.h>             /* Declares atoi(), exit(); */
#include <unistd.h>             /* close(), isatty(), read(), write(),
                                    tcgetattr(), and tcsetattr() */
#include <termios.h>            /* Declares termios, CRTL(c) */
struct termios  oldterm,        /* Original driver settings */
                newterm;        /* "Raw" I/O driver settings */
#endif  /* POSIX */

#include <stdio.h>              /* Read these with _POSIX_SOURCE set */
#include <fcntl.h>              /* Defines open() symbolic constants */

#define CTRL(c)     ((c)&037)   /* Macro to get control char code */
#define CR          CTRL(’m’)   /* Carriage return is Control-M */
#define CZ          CTRL(’z’)   /* DOS EOF character is Control Z */
#define FALSE       0           /* Logical falsehood */
#define TRUE        1           /* Logical truth */
static char         *prog;      /* Points to name of this program */

static char SccsID[] = "@(#) upload.c, Frank Lees, Jan. 89; \
POSIX port by Becca Thomas, read time out by Ray Swartz, Oct. 93";

void reset_term()   /* Reset serial driver */
{
#ifdef TERMIO
    if (ioctl(STDIN_FILENO, TCSETAF, &oldterm) < 0)
#else   /* Assume POSIX */
    if (tcsetattr(STDIN_FILENO, TCSAFLUSH, &oldterm) < 0)
#endif
        perror("Error resetting serial I/O flags");
}

void usage()    /* Display correct command-line usage */
{
    fprintf(stderr,
"\nUsage: %s [ -b ] [ -m ] [ -q secs ] [ -z ] output-file\n\
    -b   Binary transfer  (8 data bits; default: 7 data bits)\n\
    -m   Pass Control-Ms to output (default: block them)\n\
    -q # Quit time (is rounded up to next 10 secs; default 10 sec)\n\
    -z   Pass Control-Zs to output (default: block them)\n\
    output-file  Won’t overwrite existing file with same path name\n",
        prog);
}

```

```
main(argc, argv)
int argc; char *argv[];
{
    char            c;                  /* Character read/written */
    int             binary = FALSE,     /* Default is text transfer */
                    block_M = TRUE,     /* Don’t pass carriage retns */
                    block_Z = TRUE,     /* Don’t pass control-Z */
                    fdo,                /* Output-file descriptor */
                    loop_count,         /* Outer loop counter */
                    optletter,          /* Option letter */
                    quit_time = 10,     /* Max wait time betwn chars */
                    status = 0;         /* Exit status */
    extern char     *optarg;            /* Option argument value */
    extern int      optind;             /* Index of next argument */

    prog = argv[0];                     /* Point to program’s name */

    /* Process command-line options: */
    while ((optletter = getopt(argc, argv, "bmq:z")) != EOF) {
        switch(optletter) {
            case ’b’:   binary = TRUE; break;
            case ’m’:   block_M = FALSE; break;
            case ’q’:   if ((quit_time = atoi(optarg)) < 10) {
                            fprintf(stderr,
                                "Quit time must greater than 10 sec\n");
                            usage(); exit(1);
                        }
                        /* round up to next highest multiple of ten: */
                        quit_time = (10 + ((quit_time - 1)/10) * 10);
                        break;
            case ’z’:   block_Z = FALSE; break;
            case ’?’:   usage(); exit(1);   /* unknown option */
        }   /* end of switch statement */
    }       /* end of while loop */

    /* Check remaining (non-option) argument(s): */
    switch (argc - optind) {            /* number of remaining args */
        case 0:     (void)fprintf(stderr,
                        "Must specify output-file name argument.\n");
                    usage(); exit(1);
        case 1:     break;              /* one argument is okay */
        default:    (void)fprintf(stderr,
                        "Only one non-option argument is allowed.\n");
                    usage(); exit(1);
    }

    /* Open output-file for writing: */
    if ((fdo = open(argv[optind], O_WRONLY|O_CREAT|O_EXCL, 0600)) < 0) {
        perror("Error opening output file"); exit(2);
    }

    /* Make sure standard input denotes a terminal: */
    if (!isatty(STDIN_FILENO)) {
        (void)fprintf(stderr,
            "Error, standard input is not a terminal device.\n");
        exit(3);
    }

    /* Get original serial I/O settings: */
#ifdef TERMIO
    if (ioctl(STDIN_FILENO, TCGETA, &oldterm) < 0) {
```

```
112  #else   /* Assume POSIX */
113      if (tcgetattr(STDIN_FILENO, &oldterm) < 0) {
114  #endif
115          perror("Error getting serial I/O flags"); exit(4);
116      }
117
118      newterm = oldterm;              /* structure copy */
119      /*  Reset option flags for non-canonical operation:
120          (most SYSV, BSD4.3+, and POSIX option flags are the same) */
121      newterm.c_lflag &= ~(ECHO    | ICANON | ISIG);
122      newterm.c_iflag &= ~(BRKINT  | ICRNL  | INPCK | ISTRIP | IXON);
123      newterm.c_cflag &= ~(CSIZE   | PARENB);
124      newterm.c_cflag |= CS8;
125      newterm.c_oflag &= ~(OPOST);
126      newterm.c_cc[VMIN] = 0;
127      newterm.c_cc[VTIME] = 100;  /* Times out after 100 * 0.1 secs */
128
129      /* Set "raw" mode operation, flushing input queue: */
130  #ifdef TERMIO
131      if (ioctl(STDIN_FILENO, TCSETAF, &newterm) < 0) {
132  #else   /* Assume POSIX */
133       if (tcsetattr(STDIN_FILENO, TCSAFLUSH, &newterm) < 0) {
134  #endif
135          perror("Error setting raw serial I/O"); exit(5);
136      }
137
138      /* Collect input characters and write to output file: */
139      (void)printf("START of Transfer\r\n");
140      loop_count = quit_time/10;
141      while (loop_count-- > 0) {
142          while (read(STDIN_FILENO, &c, 1)) {
143              if (!binary)
144                  c &= 0x7f;              /* Mask bit 8 if not binary */
145              /* Pass the character if one of these tests is successful:*/
146              if (binary || ((c != CR) && (c != CZ)) ||
147                  ((c == CR) && !block_M) || ((c == CZ) && !block_Z))
148              if (write(fdo, &c, 1) < 0) {
149                  perror("Error writing to output file");
150                  reset_term(); exit(6);
151              }
152              loop_count = (quit_time/10 - 1); /* reset if read */
153          }
154      }
155
156      if (close(fdo) < 0) {   /* Done with output file */
157          perror("Error closing output file"); reset_term(); exit(7);
158      }
159
160      (void)printf("END of Transfer\007\r\n");
161      reset_term(); exit(0);
162  }
```

B. A command line to compile for ‘‘termio’’-structure environments:

```
$ cc -D TERMIO -o upload upload.c
```

C. Alternate call that will open an existing file without an error indication:

```
98    if ((fdo = open(argv[optind], O_WRONLY|O_CREAT|O_TRUNC, 0600)) < 0 {
```

D. The ‘‘termios’’-structure definition:

```
struct termios {
    tcflag_t    c_iflag;    /* input modes */
    tcflag_t    c_oflag;    /* output modes */
    tcflag_t    c_cflag;    /* control modes */
    tcflag_t    c_lflag;    /* line discipline modes */
    cc_t        c_cc[NCCS]; /* control chars */
};
```

E. The ‘‘termio’’-structure definition:

```
struct termio {
    unsigned short  c_iflag;    /* input modes */
    unsigned short  c_oflag;    /* output modes */
    unsigned short  c_cflag;    /* control modes */
    unsigned short  c_lflag;    /* line discipline modes */
    char            c_line;     /* line discipline */
    unsigned char   c_cc[NCC];  /* control chars */
};
```

TABLE: Serial-driver structure flags manipulated by `upload`.

ECHO	Input characters echoed back to source
ICANON	Assemble input into lines, recognize control characters
ISIG	Recognize control characters that cause signals
BRKINT	Generate SIGINT (interrupt signal) on "break"
ICRNL	Received carriage return is translated into newline character
ISTRIP	Strip most-significant bit of each character
IXON	Use start-stop control on input
CSIZE	Character size mask
PARENB	Perform parity checking on incoming characters
CS8	Use all eight bits
OPOST	Use implementation-dependent output processing
VMIN	Minimum number of bytes read before "read" call returns
VTIME	Number of tenths-of-a-second to wait before "read" returns

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com