

# PCS 2042 – Sistemas Operacionais

## *Relatório de Resolução do Exercício do MINIX*



Autores:

Daniel da Cunha Porto  
Guilherme Yokaichiya  
Neury Dias de Freitas  
Paulo Henrique Pereira de Morais  
Yuri Koldaev

Data de emissão:

21/03/2005

Revisor:

Neury Dias de Freitas

Data de revisão:

22/03/2005

## FOLHA DE CONTROLE DE REVISÕES

<b>Número da versão</b>	<b>Data de emissão</b>	<b>Registro de modificações</b>
00	21/03/2005	Criação do documento
01	22/03/2005	Revisão Final

# Sumário

<b>1. Objetivo do Documento .....</b>	<b>4</b>
<b>2. Resolução do problema .....</b>	<b>5</b>
<b>2.1. Arquivos modificados .....</b>	<b>5</b>
<b>2.2. Solução proposta .....</b>	<b>6</b>
<b>2.3. Código fonte com alterações .....</b>	<b>8</b>
<b>2.3.1. termios.h .....</b>	<b>8</b>
<b>2.3.2. tty.c .....</b>	<b>12</b>
<b>3. Referências .....</b>	<b>17</b>
<b>3.1. Internet .....</b>	<b>17</b>
<b>3.2. Livros.....</b>	<b>17</b>

## 1. Objetivo do Documento

O objetivo deste documento é descrever como foi realizada a resolução do exercício 3.37, cujo enunciado, traduzido, segue abaixo:

3.37 – Modifique o driver do terminal para que o mesmo passe a ter, além de uma tecla especial para apagar o último caractere, uma tecla que apague a última palavra.

## 2. Resolução do problema

### 2.1. *Arquivos modificados*

- termios.h      Caminho: /usr/include
- tty.c          Caminho: /usr/src/kernel

## **2.2. Solução proposta**

Através da inspeção do código foi possível perceber que existe uma rotina que remove o último caractere digitado (`back_over`). Em função disso percebemos que, para solucionar o nosso problema (apagar uma palavra), bastava chamar essa rotina já existente 'n' vezes, onde n = quantidade de caracteres da palavra.

A função `back_over` retira do *buffer* de entrada o último caractere digitado, analogamente podemos pensar que esta rotina faz um POP da pilha de entrada.

Existe a estrutura `tty_t`, que está no arquivo `tty.h` (Caminho: `/usr/src/kernel`), e é nesta estrutura que existe `tty_inbuf`, que é o buffer, no qual são inseridos e removidos os caracteres digitados pelo usuário.

Esta solução foi realizada para funcionar quando o terminal estiver operando no modo Canônico, ou seja, quando o mesmo só executa o que foi digitado após a entrada de um caractere de fim de linha.

```
/* Canonical mode? */
if (tp->tty_termios.c_lflag & ICANON) {

    /* Erase processing (rub out of last character). */
    if (ch == tp->tty_termios.c_cc[VERASE]) {
        (void) back_over(tp);
        if (!(tp->tty_termios.c_lflag & ECHOE)) {
            (void) echo(tp, ch);
        }
        continue;
    }
}
...

```

O código acima mostra o que é executado quando se deseja excluir um caractere. Neste caso, verificamos se o comando digitado pelo usuário é o comando para apagar um caractere (Ctrl + H). Este comando foi definido como `VERASE`.

Para que seja possível interpretar o comando que escolhermos para remover uma palavra (Ctrl + W) como um caractere de final de linha e, portanto, executar o que estiver antes, precisamos inserir este comando na lista de comandos executáveis.

## 2.3. Código fonte com alterações

Há explicação em todas as partes do código aonde foram adicionadas linhas de código através de comentários no formato:

```
/*
 * TODO: alterado para resolucao do exercicio
 * AQUI VAI A EXPLICAÇÃO DA ALTERAÇÃO
 *
 * */
```

### 2.3.1. termios.h

```
/* The <termios.h> header is used for controlling tty modes. */

#ifndef _TERMIOS_H
#define _TERMIOS_H

typedef unsigned short tcflag_t;
typedef unsigned char cc_t;
typedef unsigned int speed_t;

#define NCCS 20 /* size of cc_c array, some extra space
 * for extensions. */

/* Primary terminal control structure. POSIX Table 7-1. */
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 */
};

/* Values for termios c_iflag bit map. POSIX Table 7-2. */
#define BRKINT 0x0001 /* signal interrupt on break */
#define ICRNL 0x0002 /* map CR to NL on input */
#define IGNBRK 0x0004 /* ignore break */
#define IGNCR 0x0008 /* ignore CR */
#define IGNPAR 0x0010 /* ignore characters with parity errors */
#define INLCR 0x0020 /* map NL to CR on input */
#define INPCK 0x0040 /* enable input parity check */
#define ISTRIP 0x0080 /* mask off 8th bit */
#define IXOFF 0x0100 /* enable start/stop input control */
#define IXON 0x0200 /* enable start/stop output control */
#define PARMRK 0x0400 /* mark parity errors in the input queue */

/* Values for termios c_oflag bit map. POSIX Sec. 7.1.2.3. */
#define OPOST 0x0001 /* perform output processing */

/* Values for termios c_cflag bit map. POSIX Table 7-3. */
#define CLOCAL 0x0001 /* ignore modem status lines */
#define CREAD 0x0002 /* enable receiver */
#define CSIZE 0x000C /* number of bits per character */
```



```

#define CS5 0x0000 /* if CSIZE is CS5, characters are 5 bits */
#define CS6 0x0004 /* if CSIZE is CS6, characters are 6 bits */
#define CS7 0x0008 /* if CSIZE is CS7, characters are 7 bits */
#define CS8 0x000C /* if CSIZE is CS8, characters are 8 bits */
#define CSTOPB 0x0010 /* send 2 stop bits if set, else 1 */
#define HUPCL 0x0020 /* hang up on last close */
#define PARENB 0x0040 /* enable parity on output */
#define PARODD 0x0080 /* use odd parity if set, else even */

/* Values for termios c_lflag bit map. POSIX Table 7-4. */
#define ECHO 0x0001 /* enable echoing of input characters */
#define ECHOE 0x0002 /* echo ERASE as backspace */
#define ECHOK 0x0004 /* echo KILL */
#define ECHONL 0x0008 /* echo NL */
#define ICANON 0x0010 /* canonical input (erase and kill enabled) */
#define IEXTEN 0x0020 /* enable extended functions */
#define ISIG 0x0040 /* enable signals */
#define NOFLSH 0x0080 /* disable flush after interrupt or quit */
#define TOSTOP 0x0100 /* send SIGTTOU (job control, not implemented) */

/* Indices into c_cc array. Default values in parentheses. POSIX Table 7-5. */
#define VEOF 0 /* cc_c[VEOF] = EOF char (^D) */
#define VEOL 1 /* cc_c[VEOL] = EOL char (undef) */
#define VERASE 2 /* cc_c[VERASE] = ERASE char (^H) */
#define VINTR 3 /* cc_c[VINTR] = INTR char (DEL) */
#define VKILL 4 /* cc_c[VKILL] = KILL char (^U) */
#define VMIN 5 /* cc_c[VMIN] = MIN value for timer */
#define VQUIT 6 /* cc_c[VQUIT] = QUIT char (^\) */
#define VTIME 7 /* cc_c[VTIME] = TIME value for timer */
#define VSUSP 8 /* cc_c[VSUSP] = SUSP (^Z, ignored) */
#define VSTART 9 /* cc_c[VSTART] = START char (^S) */
#define VSTOP 10 /* cc_c[VSTOP] = STOP char (^Q) */

/*
 * TODO: alterado para resolucao do exercicio
 * indice do comando para apagar palavra
 *
 * */
#define VPALAVRA 14 /* cc_c[VPALAVRA] = APAGA PALAVRA - ^W */

#define _POSIX_VDISABLE (cc_t)0xFF /* You can't even generate this
 * character with 'normal' keyboards.
 * But some language specific keyboards
 * can generate 0xFF. It seems that all
 * 256 are used, so cc_t should be a
 * short...
 */

/* Values for the baud rate settings. POSIX Table 7-6. */
#define B0 0x0000 /* hang up the line */
#define B50 0x1000 /* 50 baud */
#define B75 0x2000 /* 75 baud */
#define B110 0x3000 /* 110 baud */
#define B134 0x4000 /* 134.5 baud */
#define B150 0x5000 /* 150 baud */
#define B200 0x6000 /* 200 baud */
#define B300 0x7000 /* 300 baud */
#define B600 0x8000 /* 600 baud */
#define B1200 0x9000 /* 1200 baud */
#define B1800 0xA000 /* 1800 baud */
#define B2400 0xB000 /* 2400 baud */
#define B4800 0xC000 /* 4800 baud */
#define B9600 0xD000 /* 9600 baud */

```

```

#define B19200          0xE000 /* 19200 baud */
#define B38400          0xF000 /* 38400 baud */

/* Optional actions for tcsetattr().  POSIX Sec. 7.2.1.2. */
#define TCSANOW        1      /* changes take effect immediately */
#define TCSADRAIN      2      /* changes take effect after output is done */
#define TCSAFLUSH      3      /* wait for output to finish and flush input */

/* Queue_selector values for tcflush().  POSIX Sec. 7.2.2.2. */
#define TCIFLUSH       1      /* flush accumulated input data */
#define TCOFLUSH       2      /* flush accumulated output data */
#define TCIOFLUSH     3      /* flush accumulated input and output data */

/* Action values for tcflow().  POSIX Sec. 7.2.2.2. */
#define TCOOFF         1      /* suspend output */
#define TCOON         2      /* restart suspended output */
#define TCIOFF        3      /* transmit a STOP character on the line */
#define TCION         4      /* transmit a START character on the line */

/* Function Prototypes. */
#ifdef _ANSI_H
#include <ansi.h>
#endif

_PROTOTYPE( int tcsendbreak, (int _fildes, int _duration) );
_PROTOTYPE( int tcdrain, (int _fildes) );
_PROTOTYPE( int tcflush, (int _fildes, int _queue_selector) );
_PROTOTYPE( int tcflow, (int _fildes, int _action) );
_PROTOTYPE( speed_t cfgetispeed, (const struct termios *_termios_p) );
_PROTOTYPE( speed_t cfgetospeed, (const struct termios *_termios_p) );
_PROTOTYPE( int cfsetispeed, (struct termios *_termios_p, speed_t _speed) );
_PROTOTYPE( int cfsetospeed, (struct termios *_termios_p, speed_t _speed) );
_PROTOTYPE( int tcgetattr, (int _fildes, struct termios *_termios_p) );
_PROTOTYPE( int tcsetattr, \
    (int _fildes, int _opt_actions, const struct termios *_termios_p) );

#define cfgetispeed(termios_p) ((termios_p)->c_ispeed)
#define cfgetospeed(termios_p) ((termios_p)->c_ospeed)
#define cfsetispeed(termios_p, speed) ((termios_p)->c_ispeed = (speed), 0)
#define cfsetospeed(termios_p, speed) ((termios_p)->c_ospeed = (speed), 0)

#ifdef _MINIX
/* Here are the local extensions to the POSIX standard for Minix. Posix
 * conforming programs are not able to access these, and therefore they are
 * only defined when a Minix program is compiled.
 */

/* Extensions to the termios c_iflag bit map. */
#define IXANY          0x0800 /* allow any key to continue ouput */

/* Extensions to the termios c_oflag bit map. They are only active iff
 * OPOST is enabled. */
#define ONLCR          0x0002 /* Map NL to CR-NL on output */
#define XTABS          0x0004 /* Expand tabs to spaces */
#define ONOEOT         0x0008 /* discard EOT's (^D) on output) */

/* Extensions to the termios c_lflag bit map. */
#define LFLUSHO        0x0200 /* Flush output. */

/* Extensions to the c_cc array. */
#define VREPRINT       11 /* cc_c[VREPRINT] (^R) */
#define VLNEXT         12 /* cc_c[VLNEXT] (^V) */

```

```

#define VDISCARD          13      /* cc_c[VDISCARD] (^O) */

/* Extensions to baud rate settings. */
#define B57600            0x0100 /* 57600 baud */
#define B115200          0x0200 /* 115200 baud */

/* These are the default settings used by the kernel and by 'stty sane' */

#define TCTRL_DEF        (CREAD | CS8 | HUPCL)
#define TINPUT_DEF       (BRKINT | ICRNL | IXON | IXANY)
#define TOUTPUT_DEF      (OPOST | ONLCR)
#define TLOCAL_DEF       (ISIG | IEXTEN | ICANON | ECHO | ECHOE)
#define TSPEED_DEF       B9600

#define TEOF_DEF         '\4'    /* ^D */
#define TEOL_DEF         _POSIX_VDISABLE
#define TERASE_DEF       '\10'   /* ^H */
#define TINTR_DEF        '\177'  /* ^? */
#define TKILL_DEF        '\25'   /* ^U */
#define TMIN_DEF         1
#define TQUIT_DEF        '\34'   /* ^\ */
#define TSTART_DEF       '\21'   /* ^Q */
#define TSTOP_DEF        '\23'   /* ^S */
#define TSUSP_DEF        '\32'   /* ^Z */
#define TTIME_DEF        0
#define TREPRINT_DEF     '\22'   /* ^R */
#define TLNEXT_DEF       '\26'   /* ^V */
#define TDISCARD_DEF     '\17'   /* ^O */

/*
 * TODO: alterado para resolucao do exercicio
 * comando para apagar palavra
 *
 * */
#define TPALAVRA_DEF     '\27'   /* ^W */

/* Window size. This information is stored in the TTY driver but not used.
 * This can be used for screen based applications in a window environment.
 * The ioctls TIOCGWINSZ and TIOCSWINSZ can be used to get and set this
 * information.
 */

struct winsize
{
    unsigned short      ws_row;           /* rows, in characters */
    unsigned short      ws_col;           /* columns, in characters */
    unsigned short      ws_xpixel;       /* horizontal size, pixels */
    unsigned short      ws_ypixel;       /* vertical size, pixels */
};
#endif /* _MINIX */

#endif /* _TERMIOS_H */

```

### 2.3.2. tty.c

```

#include "kernel.h"
#include <termios.h>
#if ENABLE_SRCCOMPAT || ENABLE_BINCOMPAT
#include <sgtty.h>
#endif
#include <sys/ioctl.h>
#include <signal.h>
#include <minix/callnr.h>
#include <minix/com.h>
#if (CHIP == INTEL)
#include <minix/keymap.h>
#endif
#include "tty.h"
#include "proc.h"

/* Address of a tty structure. */
#define tty_addr(line) (&tty_table[line])

... /*eliminado por simplicidade*/

/*
 * TODO: alterado para resolucao do exercicio
 * funcao que apaga a palavra do buffer
 *
 * */
FORWARD _PROTOTYPE( int back_over_palavra, (tty_t *tp) );

/* Default attributes. */
PRIVATE struct termios termios_defaults = {
    TINPUT_DEF, TOUTPUT_DEF, TCTRL_DEF, TLOCAL_DEF, TSPEED_DEF, TSPEED_DEF,
    {
        TEOF_DEF, TEOL_DEF, TERASE_DEF, TINTR_DEF, TKILL_DEF, TMIN_DEF,
        TQUIT_DEF, TTIME_DEF, TSUSP_DEF, TSTART_DEF, TSTOP_DEF,
        TREPRINT_DEF, TLNEXT_DEF, TDISCARD_DEF, TPALAVRA_DEF
    }
    /*
     * TODO: alterado para resolucao do exercicio
     * TPALAVRA_DEF - identificador da tecla que aciona
     * a remocao da palavra do buffer
     *
     * */
},
};
PRIVATE struct winsize winsize_defaults; /* = all zeroes */

... /*eliminado por simplicidade*/

/*=====
 *
 * in_process
 *=====*/
PUBLIC int in_process(tp, buf, count)
register tty_t *tp; /* terminal on which character has arrived */
char *buf; /* buffer with input characters */
int count; /* number of input characters */
{
    /* Characters have just been typed in. Process, save, and echo them. Return
     * the number of characters processed.
     */
}

```

```

int ch, sig, ct;
int timeset = FALSE;
static unsigned char csize_mask[] = { 0x1F, 0x3F, 0x7F, 0xFF };

for (ct = 0; ct < count; ct++) {
    /* Take one character. */
    ch = *buf++ & BYTE;

    /* Strip to seven bits? */
    if (tp->tty_termios.c_iflag & ISTRIP) ch &= 0x7F;

    /* Input extensions? */
    if (tp->tty_termios.c_lflag & IEXTEN) {

        /* Previous character was a character escape? */
        if (tp->tty_escaped) {
            tp->tty_escaped = NOT_ESCAPED;
            ch |= IN_ESC; /* protect character */
        }

        /* LNEXT (^V) to escape the next character? */
        if (ch == tp->tty_termios.c_cc[VLNEXT]) {
            tp->tty_escaped = ESCAPED;
            rawecho(tp, '^');
            rawecho(tp, '\b');
            continue; /* do not store the escape */
        }

        /* REPRINT (^R) to reprint echoed characters? */
        if (ch == tp->tty_termios.c_cc[VREPRINT]) {
            reprint(tp);
            continue;
        }
    }

    /* _POSIX_VDISABLE is a normal character value, so better escape it. */
    if (ch == _POSIX_VDISABLE) ch |= IN_ESC;

    /* Map CR to LF, ignore CR, or map LF to CR. */
    if (ch == '\r') {
        if (tp->tty_termios.c_iflag & IGNCR) continue;
        if (tp->tty_termios.c_iflag & ICRNL) ch = '\n';
    } else
    if (ch == '\n') {
        if (tp->tty_termios.c_iflag & INLCR) ch = '\r';
    }

    /* Canonical mode? */
    if (tp->tty_termios.c_lflag & ICANON) {

        /* Erase processing (rub out of last character). */
        if (ch == tp->tty_termios.c_cc[VERASE]) {
            (void) back_over(tp);
            if (!(tp->tty_termios.c_lflag & ECHOE)) {
                (void) echo(tp, ch);
            }
            continue;
        }
    }

    /*
     * TODO: alterado para resolucao do exercicio
     * verifica se o comando digitado é o de apagar palavra

```

```

*
* */
if (ch == tp->tty_termios.c_cc[VPALAVRA] ) {
    (void) back_over_palavra(tp);
    if (!(tp->tty_termios.c_lflag & ECHOE)) {
        (void) echo(tp, ch);
    }
    continue;
}

/* Kill processing (remove current line). */
if (ch == tp->tty_termios.c_cc[VKILL]) {
    while (back_over(tp)) {}
    if (!(tp->tty_termios.c_lflag & ECHOE)) {
        (void) echo(tp, ch);
        if (tp->tty_termios.c_lflag & ECHOK)
            rawecho(tp, '\n');
    }
    continue;
}

/* EOF (^D) means end-of-file, an invisible "line break". */
if (ch == tp->tty_termios.c_cc[VEOF]) ch |= IN_EOT | IN_EOF;

/* The line may be returned to the user after an LF. */
if (ch == '\n') ch |= IN_EOT;

/* Same thing with EOL, whatever it may be. */
if (ch == tp->tty_termios.c_cc[VEOL]) ch |= IN_EOT;
}

/* Start/stop input control? */
if (tp->tty_termios.c_iflag & IXON) {

    /* Output stops on STOP (^S). */
    if (ch == tp->tty_termios.c_cc[VSTOP]) {
        tp->tty_inhibited = STOPPED;
        tp->tty_events = 1;
        continue;
    }

    /* Output restarts on START (^Q) or any character if IXANY. */
    if (tp->tty_inhibited) {
        if (ch == tp->tty_termios.c_cc[VSTART]
            || (tp->tty_termios.c_iflag & IXANY)) {
            tp->tty_inhibited = RUNNING;
            tp->tty_events = 1;
            if (ch == tp->tty_termios.c_cc[VSTART])
                continue;
        }
    }
}

if (tp->tty_termios.c_lflag & ISIG) {
    /* Check for INTR (^?) and QUIT (^\) characters. */
    if (ch == tp->tty_termios.c_cc[VINTR]
        || ch == tp->tty_termios.c_cc[VQUIT]) {
        sig = SIGINT;
        if (ch == tp->tty_termios.c_cc[VQUIT]) sig = SIGQUIT;
        sigchar(tp, sig);
    }
}

```

```

        (void) echo(tp, ch);
        continue;
    }
}

/* Is there space in the input buffer? */
if (tp->tty_incount == buflen(tp->tty_inbuf)) {
    /* No space; discard in canonical mode, keep in raw mode. */
    if (tp->tty_termios.c_lflag & ICANON) continue;
    break;
}

if (!(tp->tty_termios.c_lflag & ICANON)) {
    /* In raw mode all characters are "line breaks". */
    ch |= IN_EOT;

    /* Start an inter-byte timer? */
    if (!timeset && tp->tty_termios.c_cc[VMIN] > 0
        && tp->tty_termios.c_cc[VTIME] > 0) {
        lock();
        settimer(tp, TRUE);
        unlock();
        timeset = TRUE;
    }
}

/* Perform the intricate function of echoing. */
if (tp->tty_termios.c_lflag & (ECHO|ECHONL)) ch = echo(tp, ch);

/* Save the character in the input queue. */
*tp->tty_inhead++ = ch;
if (tp->tty_inhead == bufend(tp->tty_inbuf))
    tp->tty_inhead = tp->tty_inbuf;
tp->tty_incount++;
if (ch & IN_EOT) tp->tty_eotct++;

/* Try to finish input if the queue threatens to overflow. */
if (tp->tty_incount == buflen(tp->tty_inbuf)) in_transfer(tp);
}
return ct;
}

... /*eliminado por simplicidade*/

/*=====
 *                               back_over                               *
 *=====*/
PRIVATE int back_over(tp)
register tty_t *tp;
{
    /* Backspace to previous character on screen and erase it. */
    ul6_t *head;
    int len;

    if (tp->tty_incount == 0) return(0); /* queue empty */
    head = tp->tty_inhead;
    if (head == tp->tty_inbuf) head = bufend(tp->tty_inbuf);
    if (*--head & IN_EOT) return(0); /* can't erase "line breaks" */
    if (tp->tty_reprint) reprint(tp); /* reprint if messed up */
    tp->tty_inhead = head;
    tp->tty_incount--;
    if (tp->tty_termios.c_lflag & ECHOE) {
        len = (*head & IN_LEN) >> IN_LSHIFT;

```

```

        while (len > 0) {
            rawecho(tp, '\b');
            rawecho(tp, ' ');
            rawecho(tp, '\b');
            len--;
        }
    }
    return(1);                /* one character erased */
}

/*
 * TODO: alterado para resolucao do exercicio
 * funcao que apaga uma palavra do buffer
 *
 * */
PRIVATE int back_over_palavra(tp)
register tty_t *tp;
{
    /*contador de caracteres*/
    int c = 0;
    /*ponteiro para o topo do buffer*/
    ul6_t *topoBuffer;
    topoBuffer = tp->tty_inhead;

    /*
     * o topoBuffer tem tamanho 16, os 8 bits menores correspondem
     * ao caracter,
     * para poder fazer uma comparacao com o caractere de espaco (0x0020),
     * precisamos normalizar o topoBuffer, pegando somente os 8 bits menos
     * significativos.
     */

    do{
        /*apaga um caracter do buffer*/
        back_over(tp);
        /* aponta pro proximo caracter do topo do buffer*/
        topoBuffer = tp->tty_inhead;
        /* incrementa o contador*/
        c++;

    } while (((*topoBuffer & 0x00FF) != 0x0020) && (tp->tty_incount>0));
    return c;
}

... /*eliminado por simplicidade*/

```



### 3. Referências

#### 3.1. Internet

MINIX PROGRAMMER'S MANUAL, TTY(4).

<http://www.cs.vu.nl/pub/minix/2.0.0/manuals/CAT4/TTY.4>

Acessado em: 18/03/2005.

MINIX INFORMATION SHEET – Faculty of Sciences – Department of  
Computer Science – VRIJE University – Amsterdam.

<http://www.cs.vu.nl/~ast/minix.html>

Acessado em: 18/03/2005.

#### 3.2. Livros

TANENBAUM, Andrew S., “Operating Systems – Design and Implementation”,  
Prentice Hall, 2<sup>nd</sup> Edition, Chapter 3.