Politechnika Świętokrzyska
Kielce University of Technology

Operating systems 1

Terminal support

Tanmay Nandanikar | Lab report 8 | 12/11/20

dr inż. Karol Tomaszewski

INTRODUCTION

This lab primarily focused on Terminal support using the Terminal API found in Linux systems. The report contains a summary of the theory required to perform tasks from the guide in the first part, results for tasks performed in the second part, a brief conclusion in the third part, and the source code for all the tasks in the final part.

1 BRIEF OVERVIEW OF THEORY

1.1 TERMINALS

A terminal provides an interface into which users can type commands and that can print text. The behaviour of terminals can be controlled using the Linux Termial API. The Terminal can work in the canonical and the non-canonical mode. A set of default behaviours are followed in the canonical mode and some standard characters have pre-defined special meanings. In the non-canonical mode, the program has complete control over key bindings. VIM is a good example of non-canonical mode in the terminal.

1.2 TERMINAL API (THE TERMIOS STRUCTURE)

The `termios` structure is used for terminal support and it always contains the following fields:

`c_iflag` - field specifying input mode flags,

`c_oflag` - field specifying output mode flags,

`c_cflag` - field specifying control mode flags,

`c_lflag` - field specifying local mode flags,

`cc_t c_cc[NCCS]` - table specifying which characters are associated with control functions.

1.3 TERMINAL API (SETTINGS)

Following are the functions and macros primarily used to manage terminal settings:

`isatty()` - checks whether the file descriptor is associated with the terminal device file. If so, it returns 1 and if not 0.

`ttyname()` - returns a pointer to a string that is the name of the terminal associated with the descriptor passed to it as arguments, or null in case of error.

`ttyname_r()` - the function accepts three call arguments. The first is the descriptor associated with the terminal, the second is the pointer to the character table, and the third is the size of the table. The second argument (a table of characters) is the name of the terminal associated with the descriptor. If this operation is successful, the function returns 0, otherwise a non-zero value.

`tcgetattr()` - the function takes two call arguments: a file descriptor associated with the terminal and a pointer to the structure of the `termios`. It stores the settings of the terminal associated with the descriptor in this structure. If the function succeeds, it returns 0 and otherwise -1.

`tcsetattr()` - this function is used to change the terminal settings. It accepts three arguments to invoke: the file descriptor associated with the terminal, a constant variable specifying when the change will occur, and a pointer to the `termion` structure containing new settings for the terminal. The second argument can be one of the following:

`TCSANOW` - the change will be made immediately,

`TCSADRAIN` - the change will occur when all output data stored to the terminal file has been processed. This constant should be used only if the changes affect the terminal output.

`TCSAFLUSH` - changes will be made after processing the output data, all unprocessed input data will be discarded. The function returns 0 if the change succeeds, and -1 otherwise.

`VMIN` - constant (macro), used for configuring non-canonical mode. Specifies the element of `c_cc` array (`termios` structure field) that defines how many bytes must be placed in the input queue before the read operation completes.

`VTIME` - constant (marko) also used for configuring non-canonical mode. Specifies the element of c `c_cc` array (`termios` field) that defines how long the system will wait before it finishes the input operation. The time unit is 0,1 second.

2 EXERCISES

2.1 TASK 1

A program that checks if the terminal is associated with a standard input descriptor, then prints the name of the terminal and two of its flags. Source code can be found in the source section (here). Output for this task is shown below:

```
name: /dev/pts/1
Echo is on
OLCUC is off
```

Output for task 1

2.2 Task 2

Program that toggles echo on/off when run. Source code can be found in the source section (here). Output for this task is shown below:

```
Echo is on for you, will turn it off!
Echo has been turned off!
sprices@pop-os:~/Documents/operating systems/lab8$ Echo is off for you,
will turn it on!
Echo has been turned on!
```

Output for task 2

2.3 Task 3

Terminal working in a non-canonical mode. When '.' key is pressed, it prints a message without checking for ENTER key. Source code can be found in the source section (here). Output for this task is shown below:

```
Execution successful!!!
```

Output for task 3

2.4 Task 4

Program that toggles between all CAPS mode on/off using OLCUC flag. Only works on linux, not all POSIX systems. Source code can be found in the source section (here). Output for this task is shown below:

```
sprices@pop-os:~/Documents/operating systems/lab8$ ./a.out
OLCUC is off for you, will turn it on!
OLCUC HAS BEEN TURNED ON!
$ DOCUMENTS/OPERATING SYSTEMS/LAB8
$ ./A.OUT
OLCUC IS ON FOR YOU, WILL TURN IT OFF!
OLCUC has been turned off!
sprices@pop-os:~/Documents/operating systems/lab8$
```

Output for task 4

3 Conclusion

During this lab, we discussed Terminal support in Linux the Terminal API. We used the `termios` structure to manipulate terminal functions by changing the appropriate flags.

4 SOURCE

4.1 SOURCE CODE FOR EXERCISE ONE

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>

main (){
    int c;
    c = isatty(0);

    if(c == 0 ){
        perror("isatty");
        exit(1);
    }
    if(c == 1 ){
        char *s=ttyname(0);
        printf("name: %s \n", s);
    }


    struct termios termInfo, save;
    int term_0;

    term_0 = tcgetattr(0,&termInfo);

    if(term_0 == -1 ){
        perror("tcgetattr");
        exit(1);
    }
    if(termInfo.c_lflag & ECHO){
        printf("Echo is on\n");
    }else{
        printf("Echo is off\n");
    }

    if(termInfo.c_oflag & OLCUC){
        printf("OLCUC is on\n");
    }else{
        printf("OLCUC is off\n");
    }
}
```

4.2 SOURCE CODE FOR EXERCISE TWO

```c
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>

main (){
    struct termios termInfo, save;
    int c;

    c = tcgetattr(0,&termInfo);

    if(c == -1 ){
        perror("tcgetattr");
        exit(1);
    }
    if(termInfo.c_lflag & ECHO){
        printf("Echo is on for you, will turn it off! \n");
        termInfo.c_lflag &= ~ECHO;
        tcsetattr(0, TCSANOW, &termInfo);
        printf("Echo has been turned off! \n");
    }else{
        printf("Echo is off for you, will turn it on! \n");
        termInfo.c_lflag |= ECHO;
        tcsetattr(0, TCSANOW, &termInfo);
        printf("Echo has been turned on! \n");
    }
}
```

4.3 SOURCE CODE FOR EXERCISE THREE[1]

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>

struct termios saved_attributes;

void
reset_input_mode (void)
{
    tcsetattr (STDIN_FILENO, TCSANOW, &saved_attributes);
}

void
set_input_mode (void)
{
    struct termios tattr;
    char *name;

    if (!isatty (0))
    {
        fprintf (stderr, "Not a terminal.\n");
        exit (EXIT_FAILURE);
    }

    tcgetattr (0, &saved_attributes);
    atexit (reset_input_mode);
```

[1] Code modified from snippet found on https://www.gnu.org

```

    tcgetattr (0, &tattr);
    tattr.c_lflag &= ~(ICANON|ECHO);
    tattr.c_cc[VMIN] = 1;
    tattr.c_cc[VTIME] = 0;
    tcsetattr (0, TCSAFLUSH, &tattr);
}

int
main (void)
{
    char c;

    set_input_mode ();

    while (1)
    {
        read (0, &c, 1);
        if (c == '\056') {
            printf("Execution successful!!!\n");/* C-d */
            break;
        }
        else
            putchar (c);
    }

    return EXIT_SUCCESS;
}
```

4.4 Source code for Exercise Four

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>

main (){

    struct termios termInfo, save;
    int c;

    c = tcgetattr(0,&termInfo);

    if(c == -1 ){
        perror("tcgetattr");
        exit(1);
    }
    if(termInfo.c_oflag & OLCUC){
        printf("OLCUC is on for you, will turn it off! \n");
        termInfo.c_oflag &= ~OLCUC;
        tcsetattr(0, TCSANOW, &termInfo);
        printf("OLCUC has been turned off! \n");
    }else{
        printf("OLCUC is off for you, will turn it on! \n");
        termInfo.c_oflag |= OLCUC;
        tcsetattr(0, TCSANOW, &termInfo);
        printf("OLCUC has been turned on! \n");
    }
}
```