Google Summer of Code 2020

NetBSD

Curses library automated testing

Curses is an important part of NetBSD OS with many applications relying on it. This project aims at creating a robust test suite for the curses library in NetBSD.

Introduction

Overview

Testing a library is as important as writing it. It ensures that the functionality offered by the library actually works and is not broken at the user end. Also, it enables the developers to fix pet bugs much before their part of code goes into production. Usually, tests are useful during development but it also serves as a smoke test for newly installed systems. Automatic testing frameworks have made writing tests much more efficient and working on these lines, this project aims to write a test suite for the curses library. Curses library is an important part of the NetBSD operating system as many applications rely on the correct functioning of the library. It is a complex piece of software. Performing modifications on the curses library can be difficult because the effects of the change may be subtle and can introduce bugs that are not detected for a long time.

NetBSD currently has a testframe that provides the curses library with the terminal interface it needs simultaneously allowing ATF to manage tests. Some tests have already been written. This project aims at writing other tests for the curses library and extending the tests to encompass every aspect of library functionality along with documentation and possible improvement of the testframe.

Potential Mentor

Brett Lymn (blymn@NetBSD.org)

Project Description

About curses library

Curses is a terminal control library for the NetBSD system, enabling the development of TUI applications without writing directly for any specific terminal type. Based on the terminal type it sends the correct control character. NetBSD curses follows SUSv2 (The Single Unix Specification, Version 2) specification with some additional ncurses extension. The curses library being the crucial part of the NetBSD operating system needs to be robust and the bugs should be figured out as early as possible. The library supports wide-characters and complex characters as well. This brings in the need for a test suite for the library. The issue with the integration of libcurses tests under the ATF framework lies with the library itself expecting to read and write from the terminal.

About testframe

The testframe for curses consists of 2 programs: director and slave. The director parses the test command file using an interpreter (tests are written in a custom language defined by formal grammar) to gather the function to execute along with its arguments. It then passes it to the slave program which is capable of running any curses function with arguments provided by the director. These two programs communicate using two pipes, one pipe allows the director to pass commands and arguments to the slave, while the other allows the slave to pass return status and values back to the director. Data from the slave can be compared against the expected output and any differences are highlighted to the tester.

The framework has to be provided with a test suite consisting of robust tests evaluating each aspect of library functionality. Some of the tests for it have already been written. This project aims to complete the suite with full SUSv2 specifications supported by libcurses. The tests written so far covers basic functionality. The test for remaining functions including the ones for window operations, wide-character routines, and complex character routines needs to be done. The complex character is a set of associated characters that may include a spacing character and may also include some non-spacing characters associated with it. The wide-character is a data type that has a size greater than 8-bit to support larger character sets. Having

a robust test suite for wide-character support would be very beneficial for the users of NetBSD in countries using wide character sets.

Tests that are to be intended to be integrated with this test suite can be classified under:

- ➢ Basic functionalities covering window operations, attribute manipulation, terminal manipulation, etc.
- ➢ Wide character routines covering basic operations along with cursor movement for the specified window.
- ➢ Complex character routines covering setting/getting a complex-character from string and attribute, basic operations like insertion, deletion along with cursor movement for the specified window, and miscellaneous functions using it.

Moreover, now the check files for the tests have to be written manually which is quite a mechanical task. This project will automate the check file generation by passing a flag to the director (master) program and adding the required functions to it.

There need to be some fixes in the testframe. Consider the test command `check temp OK`. This command is intended to validate the content of the temp variable to be OK. This command fails for OK and ERR which are not properly covered in the check command while the call function handles it gracefully. This bug though not very critical, needs to be fixed to make the test frame robust. In the course of the project, I will try to fix all such errors.

Deliverables

Deliverables for the first evaluation

- ➢ Automating the task of the generation of check files by flag passing to the director.
- ➢ Implementation of tests for wide-character routines covering its basic operations and the ones involving interaction with other functions.
- ➢ Documentation on the usage of automated check file generation and the tests added.

Deliverables for the second evaluation

- ➢ Integrate more tests in the framework, which includes basic terminal functionalities:
  - attribute manipulation routines like bkgrnd, termattrs, etc.

    - terminal manipulation routines like savetty, idcok, etc.
    - window related routings like scroll, clearok, etc.
- ➢ Integrate tests related to complex-character set, which includes:
    - complex-character handling: setting and getting with its wide-character string and its renditions.
    - basic operations like insert, delete, scan of complex-character like add_wchstr, mvin_wchstr, etc.
    - miscellaneous operations that use complex-character like border_set, box_set.
- ➢ Documentation on the usage and behavior that these tests evaluate.

Deliverables for the final evaluation

- ➢ Integrate more complex tests, which include an interaction between various functionalities and data types.
- ➢ Fixing the testframe for the bugs found throughout the project.
- ➢ Documentation extending to these new tests.
- ➢ **Stretch goals:** If time permits, then working towards testing the suite for another modification of pseudo-tty (based on xterm) and adding it to terminfo database.

Implementation Details

Various technologies and the testing framework that is planned to be used are discussed in this section. It is also discussed how these technologies are to be used and by what way they are useful for the project. ATF (Automatic Testing Framework) is to be used as the framework for automated testing.

Writing tests

Basic test:

The tests have to be written in a custom language defined by a formal grammar which is parsed by the director. Having done a course in compiler design essentially proved useful in tracing down to low level intricacies. The test language comprises a small number of commands, like assign, call, check which are described in the testframe.txt available with the source. We can define different types of strings. The quotes which enclose the string defines how the string will be treated. Example test looks like this:

```
include start
call win newwin 10 20 2 5
```

```
check win NON_NULL
assign msg "hello world"
call OK wprintw $win "%s" $msg
call OK wrefresh $win
compare wprintw.chk
```

The above test creates a new window and prints "hello world" in it. The first line includes another test file that initializes the terminal. The second line creates a new window by calling `newwin` passing the arguments and storing the return value in the variable win. The third line does the sanity check of the variable pointer win. The fourth line assigns a string "hello world" to the variable msg. The next line calls a function `wprintw` which prints a formatted string on window win (which was assigned value previously). Then we refresh the window and the slave output is compared with the check file wprintw.chk. The output comparison can be performed wherever we need to validate the output. It need not be done at the end.

Usage of complex-character:

Complex characters in the testframe can be defined using the command `cchar var_name attributes elements` or using a string enclosed in backticks.

```
cchar HCHAR 0x00000100 "H"
assign HCHAR `\001\000\000\000\000H`
```

Both of the above produce the same effect when HCHAR is used as a complex character.

Using atf(7)

atf(7) is an automated testing framework in which tests can be written as shell scripts or as C/C++ code. The role of atf in this project is for automating the runs as the output validation is handled by the master program, director. Hence, the shell script, atf-sh-api version is best suited to test the entire application as compared to atf-c-api due to its language simplicity. The test program relies on some runtime engine which is responsible for the isolation of the test program from the rest of the system and for cleaning the effects of the test program. The template for testing is very well described in atf-sh(3) man pages and the same will be followed. The head function is used to write the configuration for the test case. We will write the test description using `atf_set “descr” “the description here”`. Various tests in our curses testing suite may result in expected failure due to known bugs. We can specify the mode of expectation for a test case using `atf_expect_fail "reason"` in the head function. A complete list of such options is available in man atf-sh-api.

Test Plan

It is very unlikely for the first draft of a huge code to be perfect. The code used for testing other codes is no different and it is very possible that the test code won't do exactly as expected. Testframe provides the debug_test file which logs lots of things. This information can be accessed using

```
./debug_test -v example_test
```

For example, using this for a test of add_wch, which tests for cursor updation on adding a wide-character goes like this:

```
./debug_test -v add_wch
```

This gives a lot of information about the function calls being made, data passed between director and slave using pipes, and comparison done for validation. At times, the output can be analyzed against terminfo for more inference.

Project Schedule

START	END	PERIOD	TASK
4 May	1 June	Community Bonding	Interact with the NetBSD developers, get their opinions on the plan of action. Try to figure out known bugs so that tests can be written for them to ensure that the particular bug never pops up. Familiarise with the accepted conventions in the organization as well as the technologies to be used in the project
1 June	12 June	Automated check file generation	Add necessary code in director to fetch output from slave and dump it to check files. Exploration and deeper understanding of the testing framework.
12 June	29 June	Wide-character routines testing	Write tests related to wide-character routines and integrate it with the framework.
29 June	3 July	Phase 1 evaluations	Checking on the previous work done till now and making the documentation for the work done.
3 July	15 July	Terminal functionality testing	Write tests related to basic terminal functionalities like scrolling, attribute manipulation, terminal manipulation etc.
15 July	27 July	Complex-character	Write tests for complex-character routines, which

		routines testing	includes its handling, basic operations and usage in miscellaneous functions.
27 July	31 July	Phase 2 evaluations	Improving the work done till now and documenting it.
31 July	7 Aug	Fixing the testframe	Fixing the known bugs and ones found while using the testframe.
7 Aug	17 Aug	Miscellaneous testing	Write tests to complete testing for SUSv2 specification.
17 Aug	24 Aug	Wrap it up	Integrate more complex tests involving interaction of various functionalities and some tests figured out on the go.
24 Aug	31 Aug	Final Week	Finishing up the work including the constantly maintained blog based on the progress made.

Biography

I am a final year student studying at Indian Institute of Technology, Kanpur in India pursuing my majors in Computer Science and Engineering. My interests lie particularly in the field of Computer Systems and Algorithms. I have used various operating systems such as Ubuntu, Fedora and for about 2 months I have been using NetBSD. I have experience in building libraries and understand the intricacies of working with large libraries. In an internship at Microsoft, I worked on developing and testing the library for Distributed Tracing.

The curses testframe uses a custom terminfo file that uses the names of the terminfo capabilities instead of normal escape sequences. Having done a course in compiler design helped me clearly understand the test language which may prove very helpful while writing the tests. List of courses relevant to this project includes a course on C (ESC101), Data Structures and Algorithms (CS210), Compiler Design (CS335) and Operating Systems (CS330). I have been in contact with Mr Brett Lymn for about a month and gathered a lot of knowledge on testing and testframe for curses library from him and other people over IRC. I developed the concept of testing and its importance in large codebases and got to know the complexity of writing automated tests. For the last few months, I have been reading various tests written for curses in the NetBSD system and have tried running and tweaking them.

Working Hours

My summer vacations were supposed to start from May 1 and end on July 22. During this period, I will always be available on IRC and hangouts as long as I'm awake. Also, I will start working on the project as soon as the vacation starts (semester is off).

Note: Semester was supposed to end by April-end and then we have summer vacations of 3 months. Due to COVID-19 there may be a change in the academic calendar. But I ensure my full availability during the project.

Personal Information

Name	Naman Jain
E-Mail	namanj@iitk.ac.in jnaman806@gmail.com(Hangouts)
Phone	+91 8168287231
IRC	namanjain(freenode)
Address:	E-217 Hall of Residence - 9 Indian Institute of Technology, Kanpur Kanpur, Uttar Pradesh India - 208016