Linux Terminal Simulator

A Guide to Breaking the Program.

Table of Contents

Tyler Trepanier

Introduction

So you want to break this Linux Terminal Simulator? Well then, there's a few things you should know. Number one, this program was built and mostly tested on Ubuntu and Fedora systems. First thing you should know is that the "control-\" character is definite signal that I've researched and have unable to find a method to catch it. If you wanted to break my program, there you go! You're free to close this file.

Oh, you're still reading this… well then, I'll have some more tests for you!

All the tests to be done are inputted manually and will first test the basic functionality.

1. ***Since we are dealing with processes, the program will create three processes that will be assigned to the LinuxTerminal application.***
   Success conditions*:*
   a. *will have three processes on execution*
   b. *will have zero processes on regular program termination*
   c. *will have zero processes on regular program termination (no zombie processes)*

2. ***Each character has a specified task and will perform their assigned functionality.***
   Success conditions
   a. *'X' character erases the formatted text but will appear in the RAW output.*
   b. *'T' performs the normal termination of the program. It will also allow each of the child processes to perform their assigned tasks before exiting.*
   c. *'K' (That's an upper-case 'k') removes all buffered input.*
   d. *'ctrl-k' immediately kills the program. (Hint: the message "killed" is displayed)*
   e. *'E' functions similar to the enter key and sends the message buffer to the output to be displayed in raw and formatted forms.*
   f. *'a' characters are formatted as 'z' characters while other normal characters are untouched.*
   g. *All other normal characters are inserted into the buffer.*

3. ***This is the stress testing portion. There are various tests that have their own defined success conditions in the tests themselves:***
   a. *Inputting characters larger than the buffersize (for this test, the buffersize will be decreased artificially from 4098 to 24).*
   b. *Attempting other control signals into the program.*
   c. *Closing the program while the terminal is open. (Note, don't do this. It may create zombies that can eat your brians [yes, Brian, as in the person… it's a joke]).*
   d. *Randomly whacking the keyboard and occasionally hitting shift-E.*

Summary

The terminal does not accept any special characters of its own ASIDE from ctrl and dash which makes the window smaller. I am unable to increase the size of the window using ctrl and plus so I had to close the window itself.

All of the special characters perform their specified function

- 'T' terminates the program
- 'K' Erases all previous buffered input
- 'X' Erases the previous character.
- 'ctrl-k' Immediately terminates the program.

All control signals were successfully handled and I programmed our own “control signal” which was the special character 'ctrl-k'.

Test Cases

Test Type: Are there processes? Should they still be alive?

Test Number	**Scenario**	**Expectations**	**Results**	**Conclusion**
1	Executing the program starts the application.	Three processes will be created that are associated to the Linux Terminal.	Three processes are created: Parent: 2665 Child1: 2668 Child2: 2669	PASS
2	Program started and the 'T' character is inputted. No previous input.	There will be no processes associated to the Linux Terminal in the "current processes" list. -No output!	No processes exist with the ids of: 2665 /2668 /2669	PASS
3	Program started and the 'T' character is inputted. Previous input. "Hello There"	There will be no processes associated to the Linux Terminal in the "current processes" list. -Will output whatever was in the input and the new formatted string.	Raw: Hello T Format: Hello *Note, the remaining input went into the terminal bash	PASS
4	Program started and 'ctrl-k' is inputted. No previous input.	There will be no processes associated to the Linux Terminal in the "current processes" list. -Display the "Killed" message (spoiler alert)	Start pids: 2721, 2724, 2725. pids are fully termianted.	PASS
5	Program started and the 'ctrl-k' character is inputted. Previous input. "Hello [ctrl-k] There"	There will be no processes associated to the Linux Terminal in the "current processes" list. -Display the "Killed" message (spoiler alert)	Start pids: 2745, 2748, 2749. pids are killed and all the input before [ctrl-k] is not shown	PASS
1	Executing the program starts the application.	Three processes will be created that are associated to the Linux Terminal.	Three processes are created: Parent: 2665 Child1: 2668 Child2: 2669	PASS

Test Type: Do the characters accomplish their assigned goal?

Test Number	**Scenario**	**Expectations**	**Results**	**Conclusion**
1	Input: “workstestXXXXE”	Raw: “workstestXXXXE” Format: “works”	Raw: workstestXXXXE Format: workstest (undesired output)	FAILURE (Fixed Jan 20th)
2	Input is: “I Xam Xa Xbanana.E”	Raw: “I Xam Xa Xbanana.E” Format: “Iamabanana.”	*Hilarious.* Raw: I Xam Xa Xbanana.E Format: Izmzbznznz (Forgot about translation)	PASS
3	Input is: “XXXXXXXXE”	Raw: XXXXXXXXE Format:	Raw: XXXXXXXXE Format: *Note, no text after formatted message	PASS
4	Input is only: “T”	Raw: T Format: Program also quits. *Note the lack of a formatted message.	Raw: T Format: Program quit	PASS
5	Input is: “this is a Test”	“Raw: this is a T“ “Format: this is z “ *Note the lack of a formatted message and space AFTER the Raw output.	Raw: this is a T Format: this is z Remaining output went into the terminal after termination.	PASS

6	Input is: hElElElTJKJKE *Note: this also tests ‘T’ and ‘E’	Raw: hE Format: h Raw: lE Format: l Raw: lE Format: l Raw: lT Format: l	Same as expectation. “JKJKE” went into the terminal after program termination	PASS
7	Input: [ctrl-k] *Note, the brackets (which can be used, indicates the ctrl-k signal)	“Killed”	“Killed”	PASS
8	Input is: “this is futile[ctrl-k]” *Note, the brackets (which can be used, indicates the ctrl-k signal)	“Killed”	“Killed”	PASS
9	Input is: EEEE	Raw: E Format: Raw: E Format: Raw: E Format: Raw: E Format:	Raw: E Format: Raw: E Format: Raw: E Format: Raw: E Format:	PASS
10	Input is: “Since Amy and amanda like apples, I decided to Eat my apple.E”	Raw: Since Amy and amanda like apples, I decided to E Format: Since Amy znd zmzndz like zpples, I decided to Raw: at my apple.E Format: zt my zpple.	Raw: Since Amy and amanda like apples, I decided to E Format: Since Amy znd zmzndz like zpples, I decided to Raw: at my apple.E Format: zt my zpple.	PASS
11	Input is: “these are normal characters that shouldn’t be output onto the screen.”	*Note, there is NO output because there is no ‘E’ character.	*no character output	PASS
12	Input is: “Enter at your own risk, I’m a sign, not a copE”	Raw: E Format: Raw: nter at your own risk, I’m a sign, not a copE Format: nter at your own risk, I’m a sign, not a cop	Raw: E Format: Raw: nter at your own risk, I’m a sign, not a copE Format: nter at your own risk, I’m a sign, not a cop	PASS

Test Type: Can we break it? We have the technology!

Test Number	**Scenario**	**Expectations**	**Results**	**Conclusion**
1	Buffersize changed to MAXIMUM 24 characters and we input more characters than that. Input: 1234567890 1234567890 1234567890E	Raw: 567890E Format: 567890 *Note that the initial characters are completely ignored.	Undefined text behaviour. *** stack smashing detected ***: ./LinuxTerminal terminated Aborted (core dumped)	FAILURE [Fixed Jan 20th]
2	I will attempt EVERY button with ctrl and only output the ctrl buttons that create an operation!	Ctrl-k will not be the only signal that kills the program.	I was wrong, all signals have been ignored. Also ctrl+dash makes the window smaller. Strangely enough, the plus sign doesn't increase the size...	PASS
3	Closing the terminal in progress.	Zombie processes that will consume your Brians.	No zombie processes are leave behind inside the machine.	PASS
4	Randomly whacking the keyboard until some sort of error breaks the program.	Unless ctrl-\ is hit, the program will not end.	Overflow does not break the program; however, it does create extra newline characters occasionally.	FAILURE [Solution: have a big buffer and an alarm clock to wake up the user from nap.]

Conclusion

Initially I thought that signals would still be able to penetrate through the terminal but using the system command <<system("stty raw igncr -echo")>> implements raw access and ignores standard terminal features (i.e. 'ctrl-\' does not work while the program is running). After doing some research the "igncr" flag ignores carriage return on input meaning that you have to manually enter in the '\r' after the newline character. The echo flag simply turns echo off. Interesting. In addition, the control and dash combination made the terminal window smaller... however I did not find a ctrl-button combo to make the window bigger.

Anyways, to the meat of the story. Standard input that is within the buffersize functions completely well. The only issue comes when there is a small buffer and there is constant overflow. This throws off the newline characters and I am unable to fix this issue. Nonetheless it doesn't break the program but simply prints an extra unintended line.

All the special characters perform their tasks as requested. K performs the line-kill, X erases the previous character, T terminates the program and ctrl-k Kills the program. All in all, this is a very good introduction to how terminal processing functions.

Overall, the program works as intended!