FPS

A simple program which uses forks, pipes and signals to communicate between interdependent processes

Abstract

This document will explore the design and implementation of the fps program which was written in C.

Justin Cervantes

Cervantes.jfa@gmail.com

Contents

About FPS

This FPS application is a simple console application with 3 main functions which run on 3 different processes. Processes run different methods based on an if-else structure which checks for the return value of the fork call to ensure each of the three processes runs a designated function. The program starts by disabling all normal Linux terminal processing, and then proceeds to listen for user input onto the keyboard – once messages are received they are piped to a output process for immediate printing and also a translate process which stores a buffer that is translated and piped to the output process for printing when the ‘E’ character is received.

The three process roles are:

1) To receive input from the user on a character by character basis
2) To immediately print the input received from the first process, and if ‘E’ was detected, read from the last process and print its payload
3) To receive input from the input process on a character by character basis, and to transform the input into a translated message based on certain criteria.

The application was built using Visual Studio Code for Linux console applications (or Cygwin).

Translation criteria:

-- 'E' will act as the return key

-- 'a' will be replaced with 'z'

-- 'X' will act as backspace

-- 'K' will kill all characters proceeding it's call

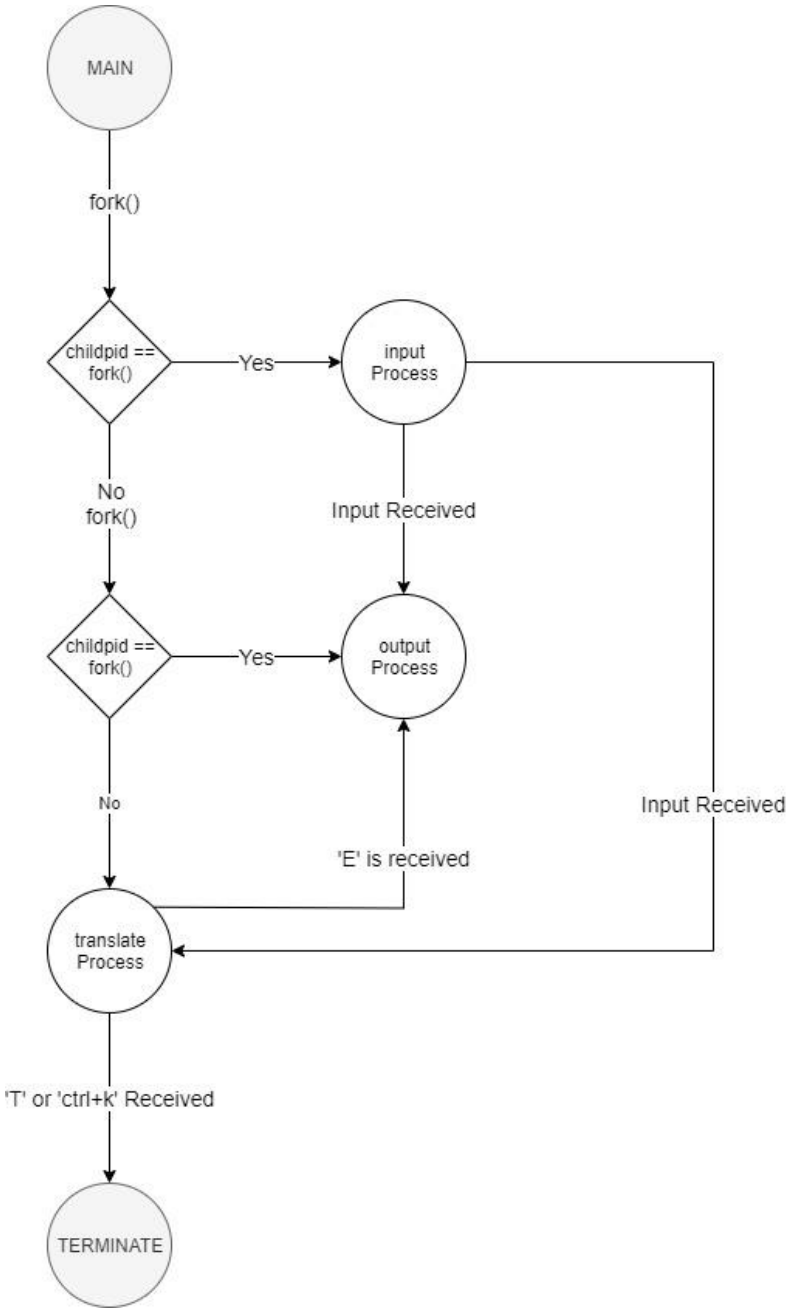
-- 'T' will kill the program

-- 'ctrl+k' will force kill the program without restoring sanity

Diagram

In this diagram, there are three states to represent the three processes. The processes are labelled according to a path of forking (parent/child). As can be seen from the structure here, the forking process follows a simple chain structure which will produce a chain of parent process IDs.

It is important to note that the fork() api will return 0 in the child process and the child process' process ID in the parent.



Pseudocode

Main (fps.cpp)

Disable keyboard processing
Stop normal interruptions, so ‘ctrl+c’ no longer exits the program, ‘enter’ doesn’t return the line, etc.

Building the processes
Process construction follows a chain format.

The first process will fork and have the return value be a non-zero integer value. This will happen twice for the parent, causing it to run the translateProcesses function.

The first spawned process, which returned 0 on the first fork, will run inputProcess.

The third process is a result of the second fork, which returned a zero-value on fork and executes translateProcess.

inputProcess
Run an infinite loop
Wait for character input
Once character input is received, write the input to a pipe which sends the character in a buffer to both the outputProcess and the translateProcess

outputProcess
Run an infinite loop
Print the contents out of the pipe
If ‘E’ was received, read the pipe that connects translate and output and print out the payload in green.
Flush the output

translateProcess
Run an infinite loop
Read from the pipe which has data from the inputProcess
A cursor will be used to help construct the buffer as string manipulation is required
Parse the received input character and construct a buffer based on translated codes:

-- 'E' will act as the return key

-- 'a' will be replaced with 'z'

-- 'X' will act as backspace

-- 'K' will kill all characters proceeding it's call

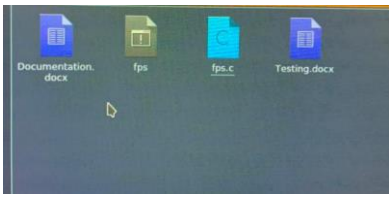
-- 'T' will kill the program

-- 'ctrl+k' will force kill the program without restoring sanity

For normal keys, add them incrementally to the buffer
If ‘E’ is received, send the buffer to the outputProcess through a dedicated pipe, then reset the buffer by moving the cursor back to 0 and running a for loop which sets the buffer to all 0’s.
If ‘a’ is received, add it to the current pointer on the buffer but write ‘z’ instead
If ‘X’ is received, add a 0 to the current location and move the pointer one space backwards
If ‘K’ is received, put a K at index 0, increment the cursor to array index 1, and run a for loop through the rest of the buffer and clear it out to 0.
If ‘T’ is received, restore sanity to the keyboard then kill all the processes.
If ‘ctrl+k’ is received, kill the processes without restoring sanity to the keyboard.

File Structure

The application will have one driver which will be the entry point into the program called fps.cpp. Once the file is compiled, it produces an executable called fps.



Starting the Program

You must be using a Linux machine to start this program, or be running the program through a Linux emulator like Cygwin. To start the program, open a BASH/Konsole and enter ./fps from the path of the file.

To compile the program, navigate to the directory and call ‘gcc -Wall -o fps fps.exe’ to get the executable.