TicTacToe Using the /Proc File System

Jonathan Hunter

Jackson Millsaps

Kevin Smith

The /Proc File System

The /proc file system is a virtual file system that allows us to contribute to the linux kernel in a number of meaningful ways. For our project, the /proc file system presents two major advantages for implementing our tictactoe game in a clean and efficient manner.

The first of these advantages is the ability to build our project into a loadable kernel module(lkm). As an lkm, our project is easily loaded or removed from the kernel with little to no effort from the user. In our case, the user may simply load the lkm whenever he wishes to play a game of tictactoe. The necessary game files are then initialized for every user on the system, allowing anyone who wishes to start a game against any other user. If no one wishes to play any longer, the last user can easily unload the module and all files are cleaned up in the process. In addition, this idea of a self contained module that can be easily loaded and unloaded at will gives us a highly mobile game of tictactoe that can be taken and played on any linux system.

The second of these advantages is the ability to work in kernel space and to facilitate the passing of data between kernel and user space. Using the /proc system, we are able to read and writes to a file almost like linux shell commands, where the content being written to the file affects the game state and reading from the file displays the game state to the user. /Proc provides a number of methods for facilitating this passage of data between the user and the kernel, and even allows us to associate files within the /proc system with the methods that we want to run when they are read from or written to.

TicTacToe

With the /proc file system as the backbone of our game and our Architect Jonathan at the lead, we began fleshing out the basic structure of our game. Once the module is loaded into the system, each user will have his own directory within the /proc file system with game and opponent files associated with it.

File System

The opponent file is where a user can initialize new games. Every user starts off with an empty game board against every other user. If two users begin playing a game, but decide they want to start over, they ask the game file to remove their game for them and it will do so. At that point the user is free to start a new game with whoever he wishes. Alternatively, the user can read from the opponent file and it will tell them who their opponent is.

The game file is where a user can make commands and interact with other users. Upon writing to the game file, the module will first check to ensure that it is a valid command. After that, it will check the state of the game between the current user and the user he is playing against, and ensure that it is the current users turn to make a move. After his move is made, he can then read the game file to see the game board.

Table 1
List of all actions available to the player.

Actions	Function
cat /proc/user/opponents	Tells the player if he has an opponent and who it is if he does.
echo “user” >> /proc/user/opponents	Initializes a game with the user if neither user is already in a game.
cat /proc/user/game	If the player is in a game, prints out his game board to the console.
echo “move a” >> /proc/user/game	If it is users turn, it updates his game board so that he now is in possession of the spot ‘a’. If not, it tells the user it is not his turn.
echo “delete” >> /proc/user/game	Deletes the game pointer so that the user is free to start a new game.
echo “help” >> /proc/user/game	

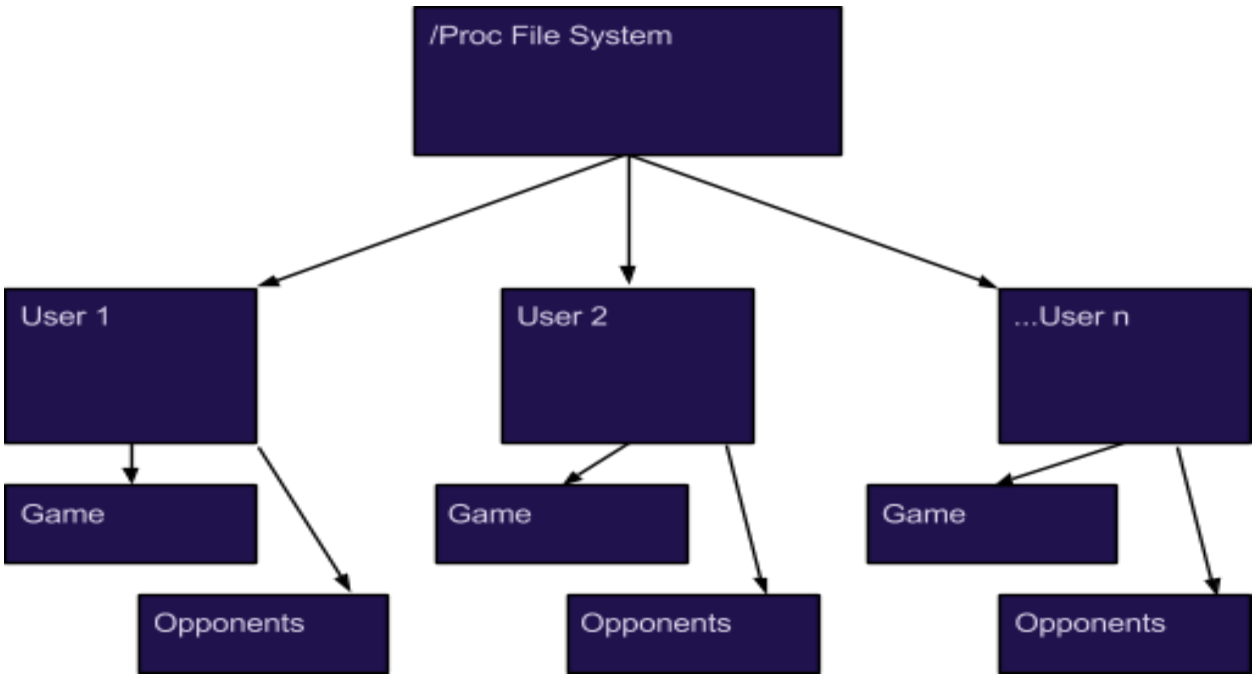

Figure 1. The basic architecture of our tictactoe game using the /proc file system as a backbone.

Game Data

For each user, any pertinent game related information is put into a struct and the stored into an arraylist[1] using kmalloc. It is important to note that we use an arraylist because arraylists are dynamic and we have no idea how many users we may need to store. In addition to that, each user has an associated index that is saved with the other game data. If we have the index, then retrieving the rest of the game data is a simple matter of indexing into the arraylist using that index. This saves us the logistical troubles as well as the performance hit of searching through the entire list looking for a user. We chose not to do this in our code, but if we wanted to allow multiple games between users, we could also add a game name opponent and then just keep a list of game names, each with their own board and move history.

While most of the game data is easily stored as strings or ints, the game board could have a number of different representations. We needed a way to represent the game board that was quick, easy, and intuitive to both the coders and the users. The typical tictactoe board (and thus the one that we will be supporting) consists of a 2 dimension array with 3 rows and 3 columns. Each 'cell', say [1,1], can consist of 3 states: X, O, or empty. To represent this in our code, we decided that each users game data will only store the moves that he has made. This allows us to cut down the board to two states, X and empty. With only two states to consider, we can represent the board as a unsigned short. That short can then be bitwise compared with the short of the opponents game board, and a 9 character string created to represent the two boards together. The user viewing the board will always see his moves on the board as X and his opponents as O. Furthermore, empty slots in this new string will be listed as [a..i] so that when users are making their moves, it is clear exactly what space they are marking in.

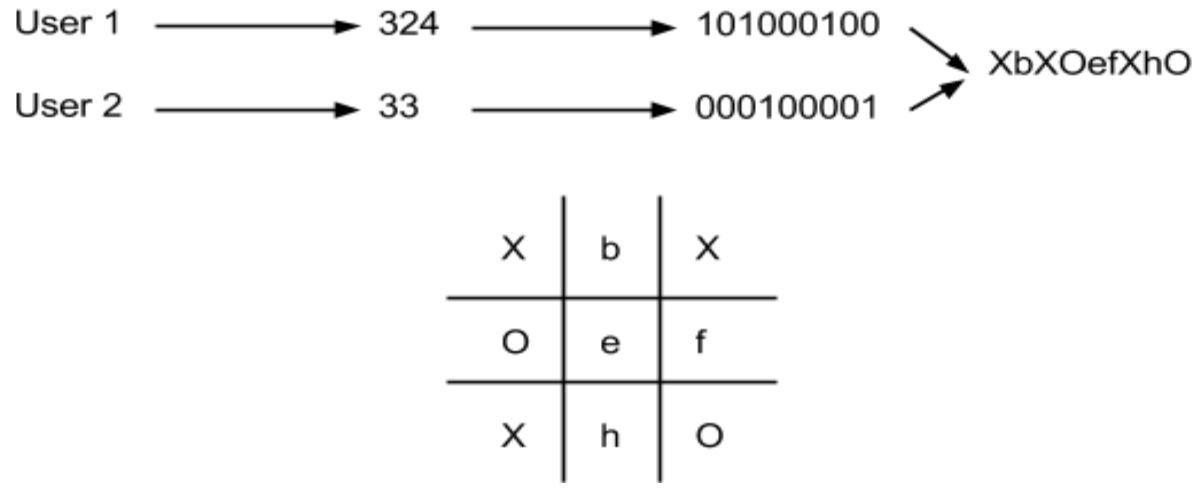

Figure 2. The evolution of each users game board from a short to a unified string representation.

[1] We downloaded and used a precoded arraylist from a public github repo: https://github.com/json-c/json-c

Implementation

Implementation of our game was overseen by our developer, Jackson Millsaps. As our codebase grew, we decided to split the module up into a number of cohesive c files, most with an accompanying header file. This helped us to keep our code base clean and easy to follow. A makefile was also included so that we only needed to run one command to compile our the module.

Table 2

List of files and the function for each file.

Files	Function
ArrayList.c and ArrayList.h	Implements a dynamic arraylist that we used to hold user data.
FileIO.c and FileIO.h	Handles file I/O, more specifically the retrieval and parsing of the file “/etc/passwd” which is where we get a list of all users.
OpponentIO.c	Handles the interaction between the current user and his opponent.
gamelib.c and gamelib.h	Separate game library we wrote to handle basic game operations.
tictactoe_init.c and tictactoe.h	Handles all the /proc functions. This is where the init and cleanup functions are held, as well as the read and write functions used for communicating with the user.