David Kravets

CMSC 441 Algorithms Project 3

Preliminary design document

In this project, a virtual device driver will be created in the form of a Linux loadable kernel module. This kernel module will implement a virtual character device to perform basic read and write operations. This driver will be used to play a game of Reversi in user space, with calls interacting with the kernel. This game of Reversi will be a single player, with the computer playing the “opponent”. The character board will be stored in a char string in kernel space. The driver function will run all operations of the game, and change the board state char string by writing to this char string. The driver function will also handle checking for win conditions, generating moves for the cpu, and verifying all inputs and outputs are correct and properly parsed and sanitized.

Character devices are devices that do not have physically addressable storage media, such as tape drives or serial ports, where I/O is normally performed in a byte stream. In this program, the board state char string will be stored in a character device. This string will be read/written to by a device driver which will execute program commands and modify the string based on the implemented functions.

Associated with each device driver is a dev_ops(9S) structure, which in turn refers to a cb_ops(9S) structure. These structures contain pointers to the driver entry points. Note that some of these entry points can be replaced with nodev(9F) or nulldev(9F) as appropriate. (2)

“sudo ./cuse -f --name=reversi” should create a /dev/reversi file within the kernel module. This will receive all the open, close, read, write, ioctl calls on it. To "unmount" the device the process must be killed. (3)

Kernel Module functions:

char read_usr_cmd(char){}** (40 lines code)

-calls sanitize_input to parse commands. returns an error if input invalid. Otherwise executes input and returns the effect of the input on the game to the user. This function will either start game (01) (calls start_game), specify a player move (02 C R)/(04) (calls move()), or ask the computer for a move (03) (calls generate_move()).

char sanitize_input(char){}** (40 lines code)

-validates input and returns whether the input is valid

start_game(){} (01 input) (20 lines code)

-when called, this function begins a new game of reversi. This will create a new char string with 66 empty board spots symbolized by 64 "-" characters and ending with a horizontal tab ('\t') character followed by either an 'X' or an 'O' which represents the player who has the next move. (I'm not sure if this should be a randomized start or if the player should always go first). **This char string should be stored in kernel memory until deleted by end_game**

end_game(){} (20 lines code)

-stops the current game of reversi, cleans up memory (**deletes the char** string** that holds the board/player state.

bool check_board_state(char){}** (80 lines code)

-checks if an input char** string is a valid board state. Will return 0 if valid, 1 if invalid. **Does not modify board state**

int generate_move(){} (03) input (10-500 lines of code)

-used to generate a move for the cpu. Should return 0-63 denoting the move that is decided on. Can be randomized choice or programmed logic (based on time left to implement). Calls move().

Char move(int C, int R, bool whoMove){} (80 lines of code)

-takes in a move and verifies that it is a valid move. Verifies that the passed in bool (signifying if the move is generated by player X or O) matches the player currently allowed to play (checks end of char** board string against bool WhoMove). **Modifies the board char** string** to add the new move and flip the player turn. Returns either an error if illegal move or out of turn, or a “ok” if legal move. Runs win_check function.d

win_check(){} (80 lines of code)

-checks for win conditions, and outputs WIN, LOSE, or TIE if one of these conditions is found. **Does not modify board state**, but can call end_game if win condition is reached

1. Brandeburg, J. (n.d.). *Developing Out-of-Tree Drivers alongside In-Kernel Drivers*. Intel Corporation.
2. Chapter 10 drivers for character devices. (n.d.). Retrieved April 08, 2021, from https://docs.oracle.com/cd/E19683-01/806-5222/character-21002/index.html
3. Free_hatfree_hat, & Meuhmeuh. (2018, December 01). Emulating a character device from userspace. Retrieved April 08, 2021, from https://unix.stackexchange.com/questions/477117/emulating-a-character-device-from-userspace