Design Document
Muhammed Muktar
CMSC 421: Project 3
April 7, 2021

In my understanding, the main piece of the project is the development of the virtual character device. The project is to implement a character device that will take in commands given in user space and read and write data to the device file that will be /dev/reversi. A character device is a device that functions through transmission of reads and writes of characters. In terms of hardware, the driver would help organize data from user space and kernel space through the device file to work with the device. In the project we will be creating a virtual character device will be able to compute the commands and play the game of Reversi from the module (driver) that is working in kernel space.

The basic function requires an initialization function for the device and a cleanup. During module initialization, the device will need to be registered using register_chrdev. In clean up the device will need to be unregistered using unregister_chrdev (Salzman, Burian, & Pomerantz). Next, the main file operations needed to communicate is .read, .write, .open, .release. The operations, similar to:

static struct file_operations fops

> **.owner = THIS_MODULE, .read = read function, .write = write function, .open = signal open function, .release = signal release function.**
>
> (Salzman, Burian, & Pomerantz)

Read Function – Device file is being read from; module will need to copy to user the response message. The message will respond accordingly with respect to the current command.

Write function – Device file is being written to; module will need to copy from user the command that is given from user space. The said command will be used in later functions to be executed properly.

Open function – Device file is being opened; module will need to keep track of device being opened to specify occupied status.

Release function – Device file is being closed; module will need to update status of device of being unoccupied.

For the actual game, I will split the game into different functions. The functions planned include runCommand, printBoard, updateBoard, computerRun, checkMove, checkPossibleMove, and checkWin.

runCommand – After device file has been written to command will be passed to this function. Command will be parsed in this function and complete the command specified. If no command is found message will respond accordingly.

printBoard – runCommnad will call print board that will turn the board into a character string, stored as the message.

updateBoard – runCommand will call updateBoard. The function will determine if the board location specified by the user is legal with checkMove. If the move is legal, the function checkMove will give a direction to update the board. The function will convert the computer's pieces within the player's pieces to the player's pieces and vice versa.

computerRun – a randomMove will the generated and checked with checkMove, this will continue as long as there is a possible move. The function will then call updateBoard.

checkMove – To check if the location is legal. The location specified will be checked. The move is legal if the location specified is empty and there is another player's piece in between the opposing player's piece. The function will then check each direction for an opposing player's piece, if it found it will continue to move in that direction checking if the opposing player's is in that direction. It will end checking that direction if the current player's piece is found, else the location specified is not legal and will return an illegal message.

checkPossibleMove – the function will run through the board looking for the current player's piece, the function will the checkMove for that current location. However, while checking in each direction the location must end in an empty space as it means there is a possible location. If there are no more current player's pieces to check the function will keep track through a variable if there are any possible moves for the current player.

checkWin – If the board is filled up, the function will run through the board counting the number of pieces. If the board contains no more of the opposite player's piece. The function will then update the message specifying if the player won, if the computer won, or if there was a tie.

Data Storage:

For communication there will be a message variable and a pointer that will be set to a response when the device file is read from. The module will also have a command variable that will store commands that are given in user space to the module when the device file is written to. The variables will be updated and used within the functions required to play the game

static char message[buffer len] , *msgPtr | static char command[buffer len], *commandPtr

For the game, the game board will be stored in a 2d cstring. The pieces of the player, computer, and the current player will also be stored in a character variable.

Lines of code estimation:

Initialization function; 30 lines.

Clean up; 10 lines.

File Operations:

Read function; 15 lines. Write function; 15 lines. Open function; 15 lines. Release function; 15 lines.

Game Functions:

runCommand; 50 lines. printBoard; 15 lines. updateBoard; 50 lines. computerRun; 30 lines. checkMove; 30 lines of code. checkPossibleMove; 40 lines. checkWin; 30 lines.

Total Number of Lines: 345

References

Rules for Reversi and Othello. (n.d.). Retrieved April 04, 2021, from https://www.mastersofgames.com/rules/reversi-othello-rules.htm

Salzman, P. J., Burian, M., & Pomerantz, O. (2005, December 31). 4.1. character device drivers. Retrieved April 04, 2021, from https://linux.die.net/lkmpg/x569.html

Tang, J. (n.d.). Lecture 18: Device Drivers. Retrieved April 04, 2021, from https://www.csee.umbc.edu/~jtang/archives/cs421.f19/lectures/L18DeviceDrivers.pdf