A basic description of the project itself and what you are expected to produce.

In this project, I must implement a linux driver. This driver must take the form of a loadable kernel module. Inside the loadable kernel module, I must implement a virtual character device. Naturally kernels have many virtual character devices, and ours must enable the user to play a game of Reversi(Othello) against the CPU.

A brief description of what a character device is in the Linux kernel, how they interact with user-space, and the required functions to implement (you will need to do a little bit of research on your own here).

Streams of characters (bytes, octets) are read from and written to Character Devices, as opposed to blocks of data. These streams are exchanged between the hardware space, where many character devices live, and an associated character driver; between the character driver and the user program. Think of a terminal or a printer receiving your command to print from a text editor. All those characters have to pass from your program to a character driver, and then to the printer.

In User Space an application performs file operations on a character device file and, through the kernel space character driver, reaches into hardware space and affects the byte oriented character driver.

User Space | Kernel | Hardware

Application <-->{Character Device file}<--> (Character Driver)<-->[Character Device]

The character device driver receives unaltered system calls, and so, in order to implement any device driver, we must write functions to handle the system calls the virtual file system will throw at the character driver.

In this project, we expect to implement functions to load and unload the device/module to the kernel, like my_dev_init and my_dev_exit.

Since we need to initialize our module with default data, as well as implement access control, we plan to use my_dev_open and my_dev_release to guard my_dev_init and my_dev_exit.

Lastly, we take in user input and return the state of the game. Thus we will implement read and write operations, or my_dev_read and my_dev_write.

Ideas for how to implement the required algorithms that will be used to accomplish your device driver's task.

Random move: The computer will select a random valid move to play. Returns valid column and row tuple.

Human move: get user input from user space and return column and row tuple

Legal moves: evaluate board for legal moves and write a column & row tuple to legalMoves array. Returns array of legal moves.

Next player: takes in board, previous player, and if there are legal moves available for the opponent, then returns opponent of previous player. If there are no moves for the opponent, then return previous player.

Any legal moves: takes in player, and board, return one(1) if there are legal moves available to player, and returns zero(0) if there are none.

Make a move: take in col&row tuple, player, and board, modifies board with correct token, repeatedly calls flipTokens (up to 8 times), and returns nothing.

Flip Tokens: takes in col&row tuple, player, and board, and a direction to check relative to the token location. Calls lookForBracket(returns 0 or tuple of players bracketing token). If lookForBracket evaluates to true, adjust col&row tuple in the direction indicated by direction, and flip the opponent piece. Adjust col&row tuple in the correct direction, check if new location is the bracket piece position, flip if false, stop if true.

Legal move: take in tuple(col&row), player, and board. Evaluates tuple for validMove. If validMove is false, return zero. If validMove is true, check for empty. If empty is true evaluate the directions around move, looking for bracketing token. If one is found, return one(1). Otherwise, return zero(0).

Look for bracket: take in tuple(col&row), player, the board, and a direction. While the tuple evaluates as held by the opponent piece, step the tuple in the direction given. Check if the new location has the player’s piece. If player’s bracketing piece is found, return the location. If empty space or board-edge found, return zero.

Valid move: take in location tuple(col&row), and check that it is inside the board. Return one(1) if true, zero(0) if false.

Print board: take in raw board, add next player information to string, push to user space.

countReversi: take in player and board. Return count of player’s tokens.

Initial board: set default starting pieces, return board.

Copy board: take in board info, kmalloc memory, assign board values, return board.

Who opponent: take in player, return opponent id.

My_dev_read: take in inode and file.

My_dev_write : take in inode and file.

My_dev_open: take in inode and file, call my_dev_init with access control.

My_dev_release: take in inode and file, call my_dev_exit with access control.

My_dev_init: take in void, return void.

My_dev_exit: take in void, return void.

Ideas of how to store the data required to accomplish your driver's task.

Struct to hold the default board. This will be loaded when the device is loaded to the kernel. Kmalloc memory using info from default board. Second structs to hold list of locations for black and white tokens. Third struct to hold respective counts of black and white tokens.

A rough estimate of how many lines of code it will take to implement the driver, divided up by various functions you will be required to implement.

200 lines divided by 21 functions for about 10 lines per function.

Any references you have cited in preparing the design document.

Character device drivers¶. (n.d.). Retrieved April 15, 2021, from

https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html

Corbet, J., Rubini, A., & Kroah-Hartman, G. (n.d.). *Linux Device Drivers, Third Edition*

[LWN.net]. Https://Lwn.Net. Retrieved April 7, 2021, from

https://lwn.net/Kernel/LDD3/

Kutkov, O. (2021, February 05). Simple Linux character device driver. Retrieved April 15, 2021, from https://olegkutkov.me/2018/03/14/simple-linux-character-device-driver/

Sieg, P. (2017, January 13). Petersieg/c/othello.c. Retrieved April 15, 2021, from https://github.com/petersieg/c/blob/master/othello.c