Linux Kernel Character Device Module

Project 3 Design Document

Student: Caleb M. McLaren

Professor: Dr. Maya Larson

Course: CMSC 421, Spring 2021, UMBC

Due: April 25th, 2021

Table of Contents

1 Project Description

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.

2 Module Description

Application

The application is any C program where the virtual files system's character device file is called via open(), close(), read(), or write().

Character Device File

This virtual files system handle will be automatically created by the udev system after creating a module class and a device of the same class inside the module's init() function.

Character Device Driver

The character device driver is loaded into the kernel via cdev_add(), and is the conduit between the Character Device File and the virtual device. All the necessary functions for the character device reside here and enable the character device.

Character Device

This is a virtual device and not a physical device like a mouse or keyboard.

Algorithms Required

setupBoard

* take in nothing
* return char * to initial game board

Count_spaces

* Take in char pointer, return number of space between characters.
* Copied from project 1, file utils.c.
* Utils.c was provided by CMSC 421 project creators,
* Spring 2021, UMBC

Flush_string
* Takes string and replace \n with \0.
* Copied from project 1, file main.c.
* Original authored by Caleb M. McLaren (myself) for project 1,
* CMSC 421, Spring 2021, UMBC.

simpleParse
* Take in user command copied to kernel space and array to hold tokens.
* Parse command into tokens.

returnOpponent
* Take in Player
* Return opposite player or null.

countToken
* take in player and board
* return token count for player

validMove
* Take in board position
* check that move in the board
* return 1 for in the board.

lookForBracket
* take in square on board,
* check in direction for bracket
* return bracket location if found

lookForFlip
* take in move and direction,
* check if neighbor square is an opponent token
* return bracket location or zero

checkForLegal
* take in move,
* check for valid move,
* check for availability of move,
* when move available, check for flips
* no flips, return illegal move
* flips found, return legal move

flipTokens
* call after lookForLegal,
* take in move, player, board, and direction,
* determine bracket location,
* flip opponent tokens

void makeYourMove
* call after checkForLegal
* Take in move, player, board.
* Call flipTokens in all directions

lookForLegalMove
* take in player and board,
* call lookForLegal for every spot on board
* return 1 if even one legal move is found for player

checkNextPlayer
* take in board and previous player,
* check if next/previous player has even 1 legal move.
* return next player, previous player, or cat as appropriate.

tallyLegalMoves
* take in player and board,
* kmalloc to hold all legal moves for player
* return int pointer

chooseRandomMove
* Take in player and board.
* Call tallyLegalMoves for respoective player.
* Free memory allocated in tallyLegalMoves.
* Return random choice of viable moves.

figureWhoWon
* take in player and board.
* iterate through board, counting respective pieces
* return difference. Positive return = player win.
* Negative return = computer win.

Reversi_open
Grab Mutex. Grab pointer to cdev structure. Initialize device data.

Reversi_release

Release Mutex.

Reversi_read

Read the game state by copying to user space the feedbackString.

Reversi_write

The game logic is implemented here.

Reversi_uevent

Specify default permissions for the new device file in virtual file system.

Init_reversi:

/* Receive Major and Minor Numbers
* Create class
* Register Class
* Initialize and Add Device to Kernel

Cleanup_reverse:

* delete device from kernel
  * destroy device
  * unregister class
  * destroy class
  * release major and minor number*/

Data Storage

As the project progressed a structure developed to handle the data storage needs of the reversi game. Originally set up to handle multiple character devices, the implementation was reduced in the interest of simplicity and availability. What remained is the reversi_data structure.

```
struct reversi_data {
    struct cdev reversi_cdev;
    char * the_board; /* pointer to reversi board*/
    char * humanToken;
    char * computerToken;
     char * feedbackString;
    char * prevPlayer;
    int score; }
```

Code Estimation

About 900 lines, including comments, are in the reversi.c document. This is a shocking departure from the originally estimated 200 lines of code. This under evaluation of the task is probably why I am turning it in late and not fully tested. Cutting my losses. See ya during the summer...unless the curve is generous.

References

Character device drivers¶. (n.d.). Retrieved April 25, 2021, from https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html#data-structures-for-a-character-device

Character device drivers¶. (n.d.). Retrieved April 25, 2021, from https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html#registration-and-unregistration-of-character-devices

Corbet, J., Rubini, A., & Kroah-Hartman, G. (n.d.). Linux device DRIVERS, 3rd edition. Retrieved April 25, 2021, from https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch03.html

Kutkov, O. (2021, February 05). Simple Linux character device driver. Retrieved April 25, 2021, from https://olegkutkov.me/2018/03/14/simple-linux-character-device-driver/

Linux device drivers, second edition. (n.d.). Retrieved April 25, 2021, from https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch03s05.html

Pugalia, A. K. (2013, May 01). Playing with systems. Retrieved April 27, 2021, from https://sysplay.in/blog/linux-device-drivers/2013/05/linux-character-drivers/

Sieg, P. (2017, January 13). Petersieg/c/othello.c. Retrieved April 25, 2021, from https://github.com/petersieg/c/blob/master/othello.c

Tuxthink. (n.d.). The working of MACROS MAJOR,MINOR and MKDEV. Retrieved April 26, 2021, from https://tuxthink.blogspot.com/2012/05/working-of-macros-majorminor-and-mkdev.html?m=1