Mini OS and a Battleship Game

Natasha Dudler and Adrian Prokopczyk

University of Basel
Operating Systems (FS22)
https://github.com/NatashaDudi/MiniOSforRaspi4

1 Abstract

In this project our goal was to create a mini Operating System. By doing this task we wanted to learn more about the fundamentals that are necessary to use a hardware like a Raspberry Pi 4 and work on programs on kernel level in the C and Assembly Language. We focused on an Operating System that starts when powered on and that only has the functionality of a battleship game that can be played through user input.

2 Introduction

Seeing all the Operating Systems (MacOS, Windows, Linux,...) available at the moment, we want to look behind the curtain by creating our own Operating System. Since it is nearly impossible to reinvent the wheel, we took inspiration from isometimes[1]. We followed his milestones until part 6 including the following stages:

Part 1 **"Bootstrapping"** where we created a file called 'kernel.c' in the C Language. Than we used Assembly Code to make sure to let our kernel be processed on the master core and link this with a linking file to our 'kernel.c' file.

Part 2 **"Linking"** deals with creating a Makefile and creating our first kernel image that can be exchanged with the image already on the SD Card.

Part 3 In **"hello world"** we use PuTTY and a TTL to USB cable to get the first output from the Raspberry Pi 4 to a Windows machine.

Part 4 **"Mini UART"** makes sure to use PuTTY and a TTL to USB cable to get user inputs from a Windows machine to the Raspberry Pi 4.

Part 5 **"Framebuffer"** displays different shapes like circles and rectangles on a screen connected to the Raspberry Pi 4 through HDMI .

Part 6 **"Breakout"** creates a simple game which was changed almost entirely by us to a battleship game.

When our Raspberry Pi 4 is plugged into a power source and connected with a screen via HDMI as well as when it is connected with a Windows machine using Putty via a TTL to USB cable, then users are able to play a battle ship game by pressing keys while using PuTTY with the settings described in io.c at line 100.

[1] https://www.rpi4os.com

3 Proceedings

For an Operating System to have all its basic functionality, we defined the following things as important to be in our project: receiving / analysing user input and creating output to a screen. Therefore we made it our goal to implement those two things.

We started by following the instructions of jsandler18[2]. With Ubuntu we tried to install all necessary requirements for a specific version of a cross-compiler called 'gcc-arm-none-eabi'. It took us several days to do so. Getting up to part 6 where the framebuffer was looked at, we realized that no output on a screen was shown. Therefore we had to search for a new tutorial. Then we found isometimes' project[3] where we luckily could use the same cross-compiler or install homebrew for Mac users.
Giving it another go, we restarted our project by putting Raspbian on a SD Card that we later inserted into the Raspberry Pi 4 model. We put Raspbian on the SD Card by using the Raspberry Pi Imager[4]. Then we followed along with the tutorial and tried to understand the code as well as possible, adding comments on the way. The ".c" files used for the kernel image are the following:

(a) **"fb.c"** is the framebuffer where all methods to connect and draw on a screen can be found. The screen shows the drawn objects if it is connected with the Raspberry Pi 4 via HDMI.
(b) **"io.c"** is for the I/O methods using pins from the Raspberry Pi 4.
(c) **"kernel.c"** is where the main method can be found. This file includes our game and makes sure that the connection to I/O is initialized.
(d) **"mb.c"** is the mailbox class where details concerning messages for the framebuffer are stored.

There was no way to continue the tutorial after part 6 because isometimes uses the 'iohook' and 'bleno' modules from Node.js[5] for MacOS which are no longer working. He used thoses modules in part 8 to create user input through a bluetooth connection that he created in part 7. Even after using a virtual machine with the same OS version isometimes used, there was an error that prevented us from proceeding. Because of this our only way to generate user input was by using a TTL to USB cable. Since we had to buy this cable before we could work on user inputs, we designed the visuals for our game first.

[2] `https://github.com/jsandler18/raspi-kernel`, `https://jsandler18.github.io/tutorial/dev-env.html`
[3] `https://www.rpi4os.com`
[4] `https://www.raspberrypi.com/software/operating-systems/`
[5] `https://nodejs.org/en/download/`

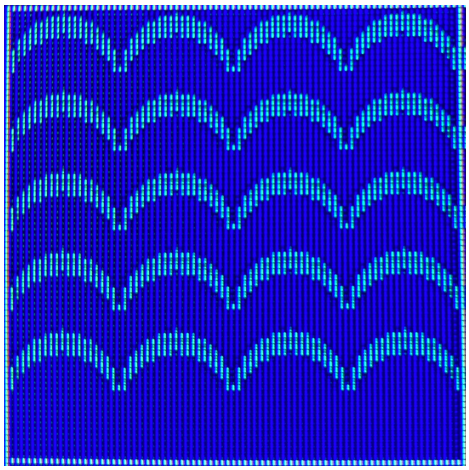

Fig. 1

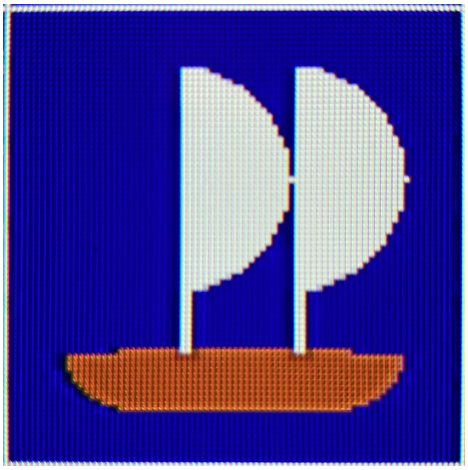

Fig. 2

We created two basic fields including one with waves and one with a boat only using simple shapes like circles and rectangles (see Fig. 1, Fig. 2). For two boards - one for the player and one for the opponent - we added wave fields next to each other until we had two fields of size 10x10.

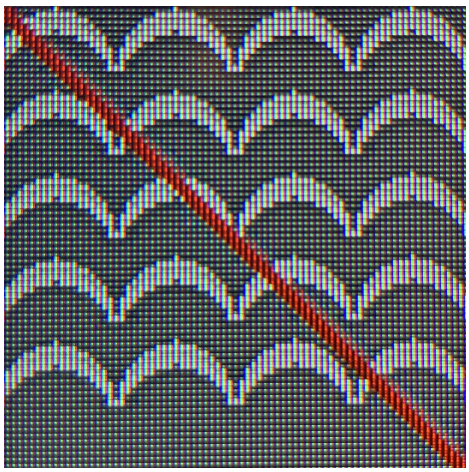

Fig. 3

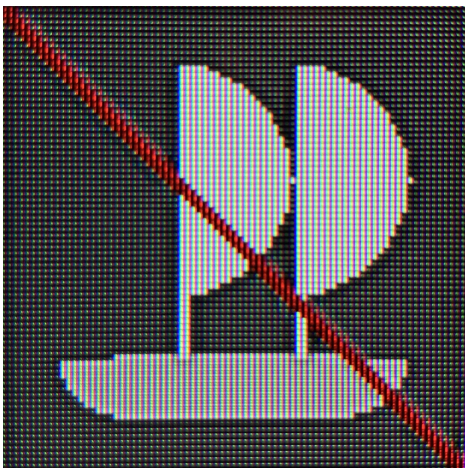

Fig. 4

Fields that were chosen to be looked at are shown in black and white and have a red diagonal line (Fig. 3 + Fig. 4).

Later we added the game logic where we included steps from the opponent on how to play the game. As soon as our TTL to USB cable arrived, we made tests on user inputs by using the cable to connect the Raspberry Pi 4 with our laptop. For that we downloaded PuTTY and started a virtual machine to use it on Windows when necessary.

4 Conclusion

Creating an Operating System is a challenge that learned us a lot. Not only did we find out how linker files and Assembly Code is used in combination with C code, we also learned how difficult it is to program on kernel level which has a limited set of types (no boolean for example) and which does not use any user libraries. Testing our code always took a lot of time since we had to connect all the different I/O devices physically with the Raspberry Pi 4. Nevertheless it was interesting to see how we learned to handle all these cables and how we started to confidently plug cables in and out and were more familiar with the Raspberry Pi 4. Getting inspiration was very difficult since we mostly found whole projects or people that would not recommend to create an Operating System at all. There was no in between. Working close to the hardware also made some problems because our TTL to USB cable seemed to be broken, only producing very strange symbols. To get a '3' at the Raspberry Pi 4 we had to type in 'sdf' than press ENTER. But at least we created our own Operating System where we can play our battle ship game.

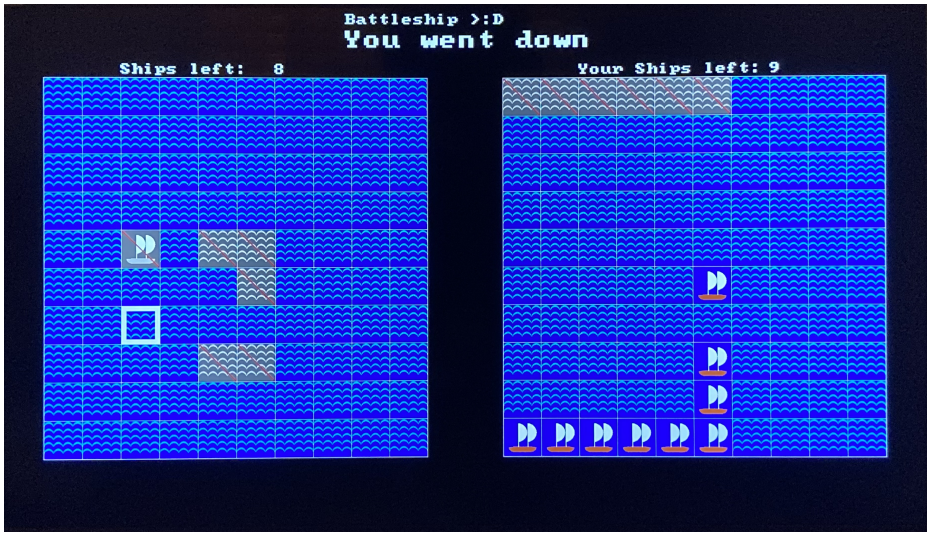

Fig. 5: Screenshot of a screen connected to our Raspberry Pi 4