Operating System for the Raspberry Pi

A Unix-like Kernel, Shell, and Coreutils for the 64-bit Raspberry Pi 3b

Sam Whitehead - 14325283
psysrw@nottingham.ac.uk
Msci Computer Science

COMP4029: Individual Programming Project
Project Supervisor: Steve Bagley
University of Nottingham

1 Introduction

TODO: describe / introduce the project.

I am aiming to produce a Unix-like Operating System (OS) which is capable of being used as a general purpose hobbyist OS. The objectives of the project are as follows.

2 Motivation and Background

Motivation for this project

My motivation for choosing to create an Operating System (OS) for this project stems from my interest in low-level programming. I have an interest in understanding how computers work at the lowest level, and the best way to learn how an OS is implemented and the reasons for design decisions will be to implement my own OS at that low level.

Background: the Raspberry Pi and Operating Systems

The Raspberry Pi (RPi) is a cheap Single Board Computer (SBC) powered by an arm System-On-a-Chip (SOC). This project focuses on the RPi 3, the first iteration of the RPi to use a 64-bit Central Processing Unit (CPU). The RPi is ideal for a project like this, as it is complex enough that creating an OS for it is a challenge, but also modern enough that no out-of-date technologies will be necessary to work with it. The RPi board includes a number of different components that will require drivers to function, so there is plenty of room for additional features if the project moves faster than anticipated.

The RPi is popular among hobbyist OS developers, and its hardware schematics and technical manuals are made available, so there are plenty of resources available online for documentation of technical details. The OS I develop will be a learning resource for others who wish to discover more about OS design. The source code will be made available under an open source license, and I will create documentation of how each component of the final system works.

OSes are programs which run directly on top of computer hardware and allow other software to run on the computer. They come in many different varieties, but this project will be based on the "microkernel" design. Unix was an OS developed at Bell Labs of AT&T in the 1970s. It was originally written in assembly language for the specific target machine, the PDP7, but for its 4th version it was rewritten in C, a new language at the time (also created at Bell Labs). Being written in C made the OS very portable, as only the compiler had to be ported for the entire system to run on a new machine. This made Unix very popular, and soon there were many Unix-like OSes. "A Unix-like OS is one that behaves in a similar manner to a Unix system"*. There are many examples of Unix-like OSes, including some very popular ones like Linux, MacOS, and BSD. The OS I create in this project will aim to be Unix-like in its behaviour.

3 Related work

- A second stage bootloader for the Raspberry Pi (RPi) [5].
- Learning Operating System (OS) development using Linux kernel and RPi [2].
- Bare metal RPi 3 tutorials [3].
- The Linux kernel [6].
- NetBSD [1] (and the rest of the BSDs).
- RiscOS [4] – the first operating system written for arm Central Processing Units (CPUs), kept updated to run on the RPi. Written in assembly language.

TODO: Describe each project and how I will be using them for inspiration / code snippets.

References for the Related Works

[1] *NetBSD*. URL: `https://github.com/NetBSD/src` (visited on 2021-11-28).

[2] *`raspberry-pi-os`: Learning OS development using Linux kernel and RPi*. URL: `https://github.com/s-matyukevich/raspberry-pi-os` (visited on 2021-11-28).

[3] *`raspi3-tutorial`: Bare metal RPi 3 tutorials*. URL: `https://github.com/bztsrc/raspi3-tutorial` (visited on 2021-11-28).

[4] *RiscOS*. Version 5.29. URL: `https://www.riscosopen.org/content/` (visited on 2021-11-28).

[5] *`rpi-boot`: A second stage bootloader for the RPi*. URL: `https://github.com/jncronin/rpi-boot` (visited on 2021-11-28).

[6] *The Linux kernel*. Version v5.15. URL: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/snapshot/linux-5.15.tar.gz` (visited on 2021-11-28).

*Wikipedia, 'Unix-like': `en.wikipedia.org/wiki/Unix-like`

Task	Duration (weeks)
Kernel	
Bootloader	1
Graphics driver	2
Syscalls	6
Filesystem	3
Statically linked ELF Loader	2
Memory management	4
Process control	8
Init system and service manager	5
Threads and multithreading	4
Shell	
Debug shell	2
`pwd`, `cd`, `ls`, `stat`, etc.	2
`export`, variables, `set`, etc.	2
`if`, `while`, `for`, `case`, globs, etc.	3
Command substitution	4
IO pipes and output redirection	1
Misc parts	
Setup cross-compiler and build environment	1
IO and strings libraries	2
libc	8
`cat`, `head`, `less`, etc.	2
`roff`	1
`man`	1
Documents	
Interrim report	1
Documentation	5
Dissertation	7

Table 1: The plan of work for the project, including how many weeks I think each component will take.

4 Description of the work

TODO: Modules / work packages.

Tasks

Tasks are the individual components of the project, separated into the components of the Kernel and Shell, the miscellaneous parts, and the documents I will need to produce for the project. These tasks are enumerated in Table 1 and then assembled into a Gantt chart in Figure 1.

Work Packages

A "Work Package" is a collection of tasks with a well-defined deliverable. Not every task will be part of a work package, and some tasks will be work packages on their own. Table 2 contains all of the work packages for this project.

5 Methodology

TODO

6 Design

TODO: design the structure of the Operating System (OS) - modules overview (graphviz?).

TODO: reword this section.

OS components/features I expect to complete

- Microkernel with UART serial output and graphics driver for the Raspberry Pi (RPi)'s HDMI video output. User interaction will use the display and read from the keyboard.
- Shell (hopefully POSIX-compliant) with scripting capabilities. Depending on time constraints, this may be a port of Dash [2].

Work package	Description of the deliverable
Debug shell using the graphics driver	A debug shell which is printed on the Raspberry Pi (RPi)'s display and which uses keyboard input. The debug shell should accept several commands for debugging the processor status (including, but not limited to, printing the contents of memory, writing to memory, and jumping to a given memory address).
A basic kernel which can load (from disk) and jump into a compiled Executable and Linkable Format (ELF) program.	A program that can load a statically compiled ELF executable binary into memory and then jump to its entry point. This requires a filesystem library to be implemented, and also includes the ELF loader task.
Scheduling algorithm	A scheduling algorithm with support for multiple processes running on the system concurrently. This will include an implementation of `fork(3)` or a similar function in order to spawn new processes and `execve(2)` to replace the current program with a new process.
Init system and service manager	An init system which is run at boot and starts all necessary parts of the Operating System (OS) and then spawns the root shell. The service manager ensures that all desired daemons and processes are started at the correct times and remain running, restarting them if they die.
A shell	A (POSIX-compliant) shell **or** a port of a simple shell like Dash[2]. If I decide that implementing my own POSIX-compliant shell is too large of a task, then I will port the source code for Dash to run on my OS, implementing the required syscalls to get it working. This shell will be the main way users interact with the OS.
Documentation	This package contains several parts. I will create documentation for how each part of the OS works, and a guide for how new programmers can get started developing programs for the system. The documentation will be completed in stages as I develop the OS, and I will go back to keep it up-to-date as I make changes to components and add new components.
Multithreading support and multi-core scheduling	The RPi 3 has 4 Central Processing Unit (CPU) cores. This work package will enable users to take advantage of the additional processing power of the other 3 cores for their programs. This includes a threading library similar to `pthread` on Linux systems, and improvements to the scheduler so that it can give different CPU cores to multiple threads owned by the same process. **This work package is optional**.

Table 2: The work packages of the project and their deliverables.

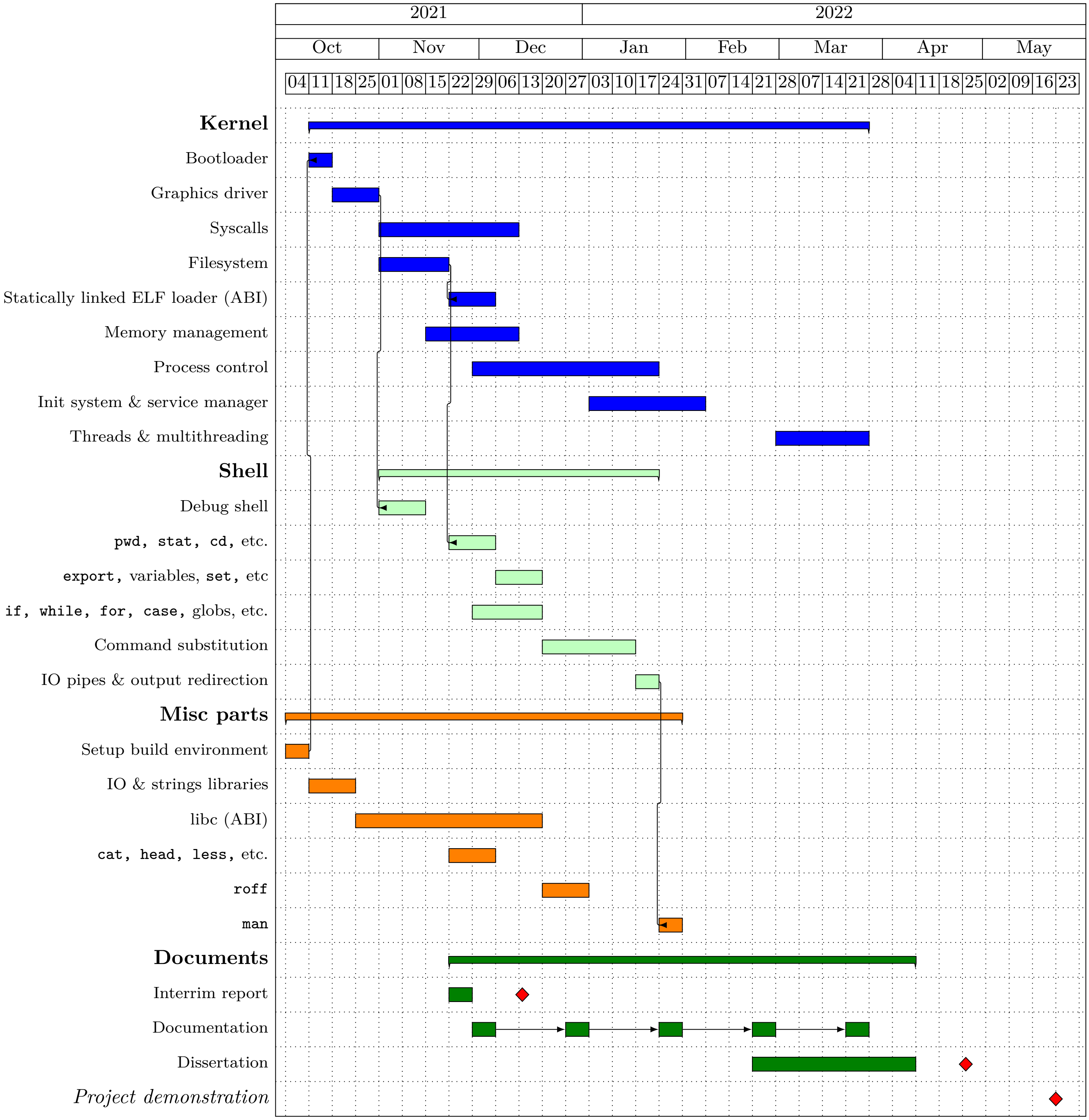

Figure 1: A Gantt chart breaking down the proposed schedule of work for this project.

- Multi-process system allowing multiple processes to time-share the CPU with a process scheduler.
- An implementation of the System V ABI for executable files (statically compiled ELF files).
- Suite of core utilities (like those provided by GNU) - these may be partially or totally ported from the GNU coreutils [3].
- An init program for starting the OS's essentials and for starting the root* shell.
- A service manager for starting services at boot and ensuring that they remain running, restarting them if they fail. This may be bundled with the init system, like with systemd on most Linux distributions, or may be a separate program.

Nice-to-haves

- A threads system with multithreading support (multiple threads owned by one process).
- A multi-user system with basic passwords and access control on files (file/directory ownership).
- Runs on hardware (not just a VM/emulator).

7 Implementation (if there is any)

TODO: describe the current implementation - challenges encountered during development so far, how I tackled problems.

8 Progress

8.1 Project management

TODO: **Project management** covering the tasks as a part of your **work plan as presented in your final proposal** and progress as well as how time and resources are managed. Your interaction with the external sponsor. The need for reviewing the tasks and explanation of/reasons for any adjustment made to the future work plan. Inclusion of a Gantt chart is strongly recommended as a visual indicating the progress of the project.

8.2 Contributions and reflections

TODO: **Contributions and reflections** providing the details of your achievements and contributions (including the contributions from the external sponsor) up to date as well as a personal reflection on the plan and your experience of the project (a critical appraisal of how the project has been progressing).

Acronyms

arm Arm Microprocessors Ltd.

CPU Central Processing Unit

ELF Executable and Linkable Format

OS Operating System

RPi Raspberry Pi

SBC Single Board Computer

SOC System-On-a-Chip

*Here, "root" means PID 0, not a root user (although if users are implemented, it would be both).

References

[1] Maurice J. Bach. *The Design of the UNIX Operating System*. Prentice-Hall, Inc., 1986. ISBN: 978-0132017992.

[2] *Debian Almquist Shell (Dash)*. URL: https://git.kernel.org/pub/scm/utils/dash/dash.git/ (visited on 2021-10-26).

[3] *GNU Coreutils*. URL: https://www.gnu.org/software/coreutils/ (visited on 2021-10-26).

[4] Brian W. Kernighan and Rob Pike. *The UNIX Programming Environment*. Prentice-Hall, Inc., 1984. ISBN: 978-0139376818.

[5] *The OSDev Wiki*. URL: wiki.osdev.org (visited on 2021-10-20).