

PegasusOS

An ARMv8, 64-bit Operating System for the Raspberry Pi

Group 23

Kenny Alvarez, Jacqueline Godier, Giancarlo Guillen,
Jacob Thomas, Christopher Walen

Credits

Project Manager and Founder:
Christopher Walen

Kernel Team:
Christopher Walen
Giancarlo Guillen
Kenny Alvarez

Hardware Team:
Jacqueline Godier
Jacob Thomas

Shell Team:
Giancarlo Guillen
Christopher Walen

Table of Contents

PegasOS	
▫ Cover Page	1
▫ Credits	2
▫ Table of Contents	3
 PegasOS: 1. Introduction	 10
▫ 1.1 Executive Summary	11
▫ 1.2 Personal Motivation/Interests	12
▫ 1.3 Broader Impacts	14
 PegasOS: 2. Legal, Ethical, and Technical	 16
▫ 2.1 Legal, Ethical, and Privacy Issues	17
↳ 2.1.1 Legal - Licensing	
↳ 2.1.2 Legal - Source Code Usage	
↳ 2.1.3 Legal - UCF and CECS Affiliation	
↳ 2.1.4 Ethical & Privacy	18
 PegasOS: 3. Technical Content	 19
▫ 3.1 Project-Specific Content	20
↳ 3.1.1 Project Goal	
↳ 3.1.2 Objectives	
↳ 3.1.3 Specifications and Requirements	
↳ Must-Haves	
↳ Like-to-Haves	22
▫ 3.2 Program and System Ideas	23
↳ 3.2.1 Group Ideas	
↳ 3.2.2 Christopher's Ideas	
↳ 3.2.3 Jacob's Ideas	24
↳ 3.2.4 Giancarlo's Ideas	
↳ 3.2.5 Jacqueline's Ideas	
↳ 3.2.6 Kenny's Ideas	25
▫ 3.3 Initial Research and Design	26
↳ 3.3.1 Design Overview	
↳ 3.3.2 Cross-Compiler	27
↳ 3.3.3 Setting Up the Build for the Cross-Compiler	
↳ 3.3.4 Setting Up the Prebuilt Cross-Compiler	28

▫ 3.4 Build, Prototype, Test, and Release Plan	30
↳ 3.4.1 Equipment	
↳ 3.4.2 Prototypes	
↳ 3.4.3 Building	31
↳ 3.4.4 Testing	
↳ 3.4.5 Shipping/Release	32
↳ 3.4.6 Contributions	
↳ 3.4.7 Source Code Modification	33
PegasOS: 4. Hardware	34
▫ 4.1 Raspberry Pi Hardware	35
↳ 4.1.1 General-Purpose Input/Output (GPIO)	36
↳ 4.1.2 Bootloaders	37
↳ 4.1.3 EEPROM	39
↳ 4.1.4 Basic Kernel Test	41
↳ 4.1.5 Drivers	42
↳ 4.1.6 I/O	43
↳ 4.1.7 Framebuffer	45
↳ 4.1.8 Mailbox Peripheral	
↳ 4.1.9 Page Table	46
↳ 4.1.10 Translation Lookaside Buffer (TLB)	48
↳ 4.1.11 Page Coloring	49
↳ 4.1.12 ARM Registers and General Research	50
↳ 4.1.13 ARMv8 Processor States	52
↳ 4.1.14 System Registers	53
↳ 4.1.15 Endianness	
↳ 4.1.16 Exception Handling	
↳ 4.1.17 Multi-Core Processors	54
↳ 4.1.18 Caches	55
↳ 4.1.19 Cache Controller	56
↳ 4.1.20 Cache Policies	
↳ 4.1.21 Cache Maintenance	57
↳ 4.1.22 Memory Management Unit	58
↳ 4.1.23 Virtual to Physical	
↳ 4.1.24 Access Permissions	59
↳ 4.1.25 Cacheable Memory	
↳ 4.1.26 Translating a Virtual Address to a Physical Address	
↳ 4.1.27 Configuring the MMU	60
▫ 4.2 I/O Manager	61

PegasOS: 5. System	63
▫ 5.1 System Directories	64
↳ 5.1.1 Root Directory	
↳ 5.1.2 Root Directory Overview	65
▫ 5.2 System Set-Up	68
↳ 5.2.1 System Language	
↳ 5.2.2 System Location	
↳ 5.2.3 System Keyboard Layout	
↳ 5.2.4 System Date and Time	69
↳ 5.2.5 Master Password	
↳ 5.2.6 Create New User	
▫ 5.3 System Localization	70
↳ 5.3.1 International Localization	
↳ 5.3.2 Localization File Standards	
↳ 5.3.3 Loading Translated Text	71
↳ 5.3.4 Localization Support	72
▫ 5.4 System Users	73
↳ 5.4.1 User Registry File	
↳ 5.4.2 Usernames	74
↳ 5.4.3 Username Storage	
↳ 5.4.4 User Passwords	
↳ 5.4.5 User Password Storage	77
↳ 5.4.6 User Permissions	78
▫ 5.5 User Subsystem	79
↳ 5.5.1 Log-In Menu	80
↳ Existing User	81
↳ New User	84
↳ Registry	86
↳ Quit	
↳ 5.5.2 User Menu	87
↳ Change User Details	
↳ Logout	92
↳ Switch User	
↳ Quit	93
▫ 5.6 System Calls	94
↳ 5.6.1 System Diagnostics	
↳ get_clocktime()	
↳ set_clocktime()	

↳ set_clockzone()	
↳ 5.6.2 Process Creation and Deletion	95
↳ p_clone()	
↳ p_destroy()	
↳ p_wait()	
↳ 5.6.3 Process Information	96
↳ get_p_id()	
↳ get_p_name()	
↳ get_p_uptime()	
↳ set_p_id()	
↳ set_p_priority()	
↳ 5.6.4 File Manipulation	98
↳ close()	
↳ open()	
↳ read()	
↳ write()	
↳ print()	
▫ 5.7 Implementation of System Calls	99
PegasOS: 6. Kernel	102
▫ 6.1 Kernel Design	103
▫ 6.2 Kernel Components	106
↳ 6.2.1 Scheduler	
↳ 6.2.2 Program Execution	109
↳ 6.2.3 Interrupt Handler	112
↳ 6.2.4 Memory Manager	
↳ 6.2.5 Page Manager	113
↳ 6.2.6 Virtual Memory Manager	
↳ 6.2.7 Cache Management	
▫ 6.3 File Systems	114
↳ 6.3.1 File System Research	
▫ 6.4 FAT32	
↳ 6.4.1 Boot Record	115
↳ 6.4.2 BIOS Parameter Block (BPB)	116
↳ 6.4.3 Extended Boot Record	117
↳ 6.4.4 File System Info Structure	118
↳ 6.4.5 File Allocation Table	119
↳ 6.4.6 Directory	120
↳ 6.4.7 Standard 8.3 Format	

↳ 6.4.8 Long File Names	122
▫ 6.5 Ext2	123
↳ 6.5.1 Base Superblock Fields	125
↳ 6.5.2 Extended Superblock Fields	126
↳ 6.5.3 Block Group Descriptor Table	127
↳ 6.5.4 Block Group Descriptor Table Structure	128
↳ 6.5.5 Inodes	
↳ 6.5.6 Inode Data Structure	129
↳ 6.5.7 Directories	130
↳ 6.5.8 Directory Entry Structure	131
↳ 6.5.9 Creating a Customized File System	132
▫ 6.6 File System Design	135
PegasOS: 7. Shell	139
▫ 7.1 Shell Background Research	140
↳ 7.1.1 What is a Shell?	
↳ 7.1.2 Graphical User Interface (GUI)	
↳ 7.1.3 Command Line Interface (CLI)	141
↳ 7.1.4 PegasOS Interface	
↳ 7.1.5 Common Shell Components	142
↳ 7.1.6 Common Shell Commands	144
▫ 7.2 Shell Commands for PegasOS	148
↳ backgroundpalette	
↳ cd	
↳ clear	
↳ concat	149
↳ copy	
↳ createdir	150
↳ createfile	
↳ currentdir	151
↳ delete	152
↳ deletedir	
↳ echo	
↳ filespace	153
↳ find	
↳ head	
↳ hello	154
↳ login	
↳ mount	

↳ move	155
↳ power	
↳ systeminfo	
↳ tail	156
↳ tasklist	
↳ terminatetask	157
↳ textpalette	
↳ uninstall	
□ 7.3 Basic Shell Architecture	159
↳ 7.3.1 Lexical Analysis	
↳ 7.3.2 Parser	160
↳ 7.3.3 Shell Architecture for PegasOS	161
↳ 7.3.4 Process Flow	162
↳ 7.3.5 Process Flow for Shell	163
□ 7.4 Shell Interactions With Kernel	167
PegasOS: 8. Administrative Content	168
□ 8.1 Budgeting and Finance	169
↳ 8.1.1 Initial Project Gantt Chart	170
□ 8.2 Project Roadmap and Milestones	171
↳ 8.2.1 Spring Semester	
↳ 8.2.2 Summer Semester (Original Schedule)	172
↳ 8.2.3 Fall Semester (Original Schedule)	
□ 8.3 COVID-19	173
↳ 8.3.1 Remainder of Spring Semester	
↳ 8.3.2 Summer (May - Mid August)	
↳ 8.3.3 Fall (Late August - Early December)	174
□ 8.4 Project Summary and Conclusions	176
↳ 8.4.1 Summary	
↳ 8.4.2 Conclusions	177
PegasOS: 9. Appendices	178
□ 9.1 Copyright	179
↳ 9.1.1 KnightOS	
↳ 9.1.2 PegasOS	
↳ 9.1.3 GNU General Public License	180
□ 9.2 Software	193
↳ 9.2.1 Diagrams	
↳ 9.2.2 Source Code and Documentation	

↳ 9.2.3 Discord	194
▫ 9.3 Miscellaneous	195
↳ 9.3.1 Hashing Algorithm	
PegasOS: 10. References	196
▫ 10.1 References	197
▫ 10.2 Uncategorized References	201

PegasOS

1. Introduction

```
+hN
.y+yN+
os.ysmY
:h--d+hmy
.yd`odhyhdm
`os:Myhssh:hh
/y--m+`-h++hmo      .:oy-
.y+`+hhs  -yohM-      `:+ss+/sM`
+h- `+ss`  -osdh      -/sys/.-/odNo
+h`y-`yNs  :ohM-  `-/syshm-/oyhyym`
`m-ss `sN:  :+hs+sy+:odmy+-..oosdd`
:m m:`s/sy  oNd+-`:/shhd-  /ssdh.
`--`:/+:-:/:/o os`M` :om` -ms`./so:my-` -sshh.
:so/oo/...:/ymm:.mo:m -o+m` :m+`ohssys/` `+ohmh`
`ss//+oo++o++ddhyodM+oy .yd- +d:`ddo-dM:  /oyNy-
.:+ohh+-..`...`...:+s+-..ydy:~od.oh.`hd++yyo` -:shd/`
oNNhdmhh/      .+s:+MNy-:-ys`yhdsd/  /ymNo.
`hs.++mmh. :/      .ossNmy:d+ -hsds////+osdh+.
/ysm:s+ .N+`      `-/smm/-+ssss+yyo.odho`
./yohs ohoy      oh+N: -.~ohyoysyyys/:~
+d yh` m:      yy.+:++h//sdhh+-`...-/o+//o+o+:`
m- sy ss  ./  `ossyyodyhho:.      :d+-` `:o+`
+yysmN. .d- ys      ``.---`      /mdh- :y/`
-sho-  /N+ oy      :mhdd `sy.
.M- N: `      ...      M:mm+ /Nd`
/md- d+:s`      `++oy-      /M`d/ys: .ys`
h/-yhymS`  -` ``-/osyshm+      om-N` `so +ohh
` :msso+/-` .yy//oso+yd-` .ho`  `+:h-` `y+` :Nmm/
-yooY--:/:++osoo+/:.  oy-` /-md+.``  `sso- +y:y/.
-shs/yyomdyoy-/hhs:  ` /sy/hsh+/ooo/.. /myN  :+oh/
-s+sNhshhyyydy:+/dNmy.  ` .+sooyysy  -+ooomM`
`om-/d-  `:-:/:/ooo:.  `:os+:./yo+.  /h+my
:mMMN/      :d:+oss+-      -d.+h`
:++/-      -Ndyh.`      -d./h`
      .NMNy.      .d`-d-
      ``      :h/:do`
      :dMMs
      :hM/
```

1.1 Executive Summary

Operating Systems are amazing pieces of software. They are something that many of us take for granted in the modern digital age, and yet the amount of work that goes into a functional operating system, let alone a fully-featured and usable operating system is a monument to the decades of design and progress that has gone into the field. From the early days of operating systems that had no display or interface whatsoever, to the early text-based systems, to rudimentary graphic interfaces and finally the modern systems that we are used to - immense amounts of work have gone into optimizing these systems, and making the design as intuitive to the user as possible.

And just as operating systems are amazing pieces of software, they are nothing without the incredible hardware that they run on. We are truly lucky to live in an age with such a wide array of computers to choose from; from powerful full-sized tower PCs to slim and elegant Macbooks, credit-card sized Single-Board Computers and everything in between. Of particular note is the ever-growing field of Single-Board Computers, which contains processing power that blows away anything from as early as 10 years ago for the form factor. No discussion about Single-Board Computers can be complete without the mention of what is perhaps the most significant contributor to the field: the Raspberry Pi.

First released in February of 2012, the small computer boasted 256Mb of RAM, a 700MHz processor, 2 USB 2.0 Ports, a 3.5mm Jack, an HDMI port, and a 250MHz VideoCore IV GPU - all for the low price of \$35. It was immediately popular and has become a staple of early computing education and Linux education ever since. The low cost of the computer for the power that it provides allows for a wide array of applications, and many choose to use them as cheap Linux machines as a result.

However, we have a different plan in mind: developing a new operating system specifically for the Raspberry Pi Model 4, and using our development process as a guide for others interested in the world of systems software and operating system development. This new operating system will make as much use out of the hardware as is possible, to show what the Pi 4 can really do. This new operating system is called:

PegasOS.

1.2 Personal Motivation/Interests

System software is one of the branches in computer science that has interested me ever since I started my computer science degree. I'd always been interested in both hardware and software and the field of system software was a nice mix of both hardware and software. I'd also wanted to expand my knowledge of the area in system software and joining this project seemed like an excellent opportunity to learn more about operating systems and system software.

- Jacob Thomas

Up until recently, I had wondered what kind of specialization I'd want to have within Computer Science, and never really had a preference. But after taking a lot of time with the Systems Software class that's offered here at UCF, I began to really appreciate how the software interacts with the hardware, and what kinds of crazy optimizations and algorithms are running behind the scenes to make that happen. Eventually, I decided that I wanted to go above and beyond within the realm of Systems Software, and try doing the most challenging problem in that realm: building an OS from scratch. It's going to be an amazing learning experience, and a great project to be able to point to and say "*We made that from the ground up!*". There's just so much potential with PegasOS that it makes me very excited to be actually working on it.

- Christopher Walen

Having been in the CS program and to be coming into my 4th year it has finally hit me that the vast majority of software engineer positions are dealt with website development/maintenance and dealing with Full Stack components, like front-end and back-end development. This was made even more clear when a good majority of the senior design pitches were Web Dev. and none of them I found interesting until our PM Chris pitched his idea. The main reason I chose to tackle this great idea of a project is because I wanted to help create some products that will be used for others, monetarily or educationally. The thing with Websites these days is that a vast majority of companies don't make it past their 2nd year or sustain long enough past 10 years. With this project I felt that I could contribute to creating an OS that could be later optimized for the Raspberry Pi and which will lead to better projects being created using the Pi. Also another reason I chose to tackle this project is that I wanted to "get my feet wet" in this area of CS. The concept and execution of operating systems have always intrigued me and I thought "why not learn from the ground up and help create one for others to use".

- Giancarlo Guillen

I have been working with a lot of high-level gaming software for quite some time, but I always had an interest in the lower-level side of things. I really enjoyed Systems Software and after building my first computer I wanted to learn more about how software interacts with the hardware to make an interface anyone can understand. I have also been working with a lot of low-level programming recently and thought this would be a great way to expand my skill set.

- Jacqueline Godier

Ever since I took system software at UCF my interest on how software interacts with hardware grew from there. Almost all of the project pitches in senior design played out similarly as most of them had vague demands on what the end product should be and many of them didn't really interest me. The PegasOS project was one of few that was different from the rest of the projects and I think it's a good opportunity to really understand operating systems and gain more knowledge about system software.

- Kenny Alvarez

1.3 Broader Impacts

PegasOS is about creating a brand new operating system for the Raspberry Pi, one of the first mainstream and affordable Single Board Computers on the market. For a low \$30-35, you have a surprisingly powerful computer that can fit the niche for a number of projects that require a dedicated computer you might otherwise not have. However, there is already a number of operating systems available for the Raspberry Pi. Why bother with this gargantuan task? What purpose does it serve? This requires a little bit of explanation.

More and more operating systems are becoming available for the Raspberry Pi. Traditionally, when we look at desktop or laptop computers, there are three main choices for operating systems: Windows, Mac, and Linux. Windows and Mac are self-explanatory, as they are very popular options for computer operating systems. Linux is rather complicated for a novice. Linux operating systems are made up of three pieces: the Linux Kernel, the Distribution, and the Desktop Environment. When you hear about OS's like Debian, Ubuntu, Arch, Kali, and so on, those are what we call distributions of Linux. All that means is that the heart of the OS is the Linux Kernel, and then the rest of the system's programs, functions, and management are built around that in various ways. Packaged together, this makes a distribution. Sitting on the very top of this, is the Desktop Environment, which is essentially the GUI interface that makes working with the OS that much easier. That is great and all, but how does that relate to the Raspberry Pi?

Raspberry Pi's are essentially Linux machines. While Windows does have a modified Windows 10 compiled for the Raspberry Pi and a few other SBCs, just about every OS for the Pi is Linux-based. There are exceptions to this, namely the UNIX-based OS's like RISC OS, OpenBSD, Android, and ChromiumOS (among others), but the overwhelming majority is some form of Linux distribution. Perhaps the most popular option among the Linux distributions is that of Raspbian, which is a Debian-based distribution rewritten specifically for the Pi by the Pi Foundation. It is a 32-bit operating system that comes pre-baked with an LXDE Desktop Environment, featuring options for pre-installed programs or more lightweight offerings. We could go on about the various distributions that are available to the Pi. However, like Raspbian here, the majority are written and compiled to 32-bit mode. This may not seem significant, until you consider the fact that since as early as the later models of Raspberry Pi 2, that the Pi has had an ARM chip containing 64-bit support.

The later Pi 2's and the standard Pi 3 models contain the model BCM2837 processor. This processor contains a Quad-Core ARM Cortex A53 at 1.2Ghz, which supports ARMv8 - a 64-bit instruction set. The Pi 3A+ and 3B+ contain a model BCM2837B0 processor, which also contains a Quad-Core ARM Cortex A53 with a higher frequency of 1.4Ghz - meaning this also supports ARMv8. Lastly, the Pi 4 contains a model BCM2711, which contains a Quad-Core ARM Cortex A72 at 1.5Ghz - making it the fastest processor of the bunch that also supports ARMv8.

The majority of operating systems for the Pi contain either ARMv6 or ARMv7 instruction sets, which are backward-compatible with ARMv8. This makes it an ideal choice as far as portability is concerned because the old Pi 1/2/Zero models do not support ARMv8, so having your distribution compile to 32-bit and simply telling the newer models to run in 32-bit mode means you can keep versions across all the Pi's and leave no one out. However, if you are running these processors in 32-bit mode by using the ARMv6 or ARMv7 instruction set, you are simply wasting the instruction space that the processor is giving you, which is a greatly missed opportunity that is ever-so-slowly being capitalized on.

And this is where PegasOS comes in. Our goal is to make a true 64-bit, ARMv8 OS for the Raspberry Pi, and since this is compiling to ARMv8, potentially any machine that runs a 64-bit ARM processor could run this OS as well. We want to make the most use of the processor that the hardware is giving us, and to use that to write a fast and efficient OS that offers something a bit apart from the standard Linux and UNIX systems.

Because PegasOS will be FOSS (Free and Open Source Software) at the end of the project, it will be publicly available for anyone who wants to use it or build on to it. The GitHub repositories are already public and can be forked or pulled to for other contributors past the scope of the project during this Spring/Fall term. In conjunction with our requirements for our capstone here at UCF, the amount of documentation that will go along with the building of this OS will serve as an educational tool for those interested in systems software and operating systems development, of which there is a lack of sources in this regard apart from *wiki.OSDev.org*. While on this journey of building PegasOS, there are also opportunities to create new and improved algorithms related to OS development, that could be used in other OS projects. The PegasOS team believes that Systems Software is incredibly engaging, challenging, and rewarding as a result, and we want to share that with others out there that have that same spark that we all have.

2. Legal, Ethical, and Technical

[illegible]

2.1 Legal, Ethical, and Privacy Issues

2.1.1 Legal - Licensing

PegasOS is planning on using the GNU General Public License, wherein any versions of the license including version 3 are applicable to the releases of PegasOS. This would allow us to monetize aspects of the operating system - such as allowing for donations to the project - but still, keep the goal of Free and Open Source Software front and center. This allows anyone to download the source code and/or operating system for free with no obligation to pay and allow them to make any modifications to said source code on their own behalf. Part of this licensing will require getting an agreement with UCF and CECS to disclaim the copyright on PegasOS, such that PegasOS remains free and open source. For your convenience, a copy of the GNU General Public License (version 3) is included in the *Appendices* section.

2.1.2 Legal - Source Code Usage

Most if not all code in PegasOS will be procured completely by the PegasOS team, including future teams from UCF and/or future contributors to the repository. In the event that code for the project is not written by the team, it will be code that is open source itself or freely donated to the project by an individual contributor(s) with proper credits to the author(s) of that code. If necessary, contact will be made with those individuals to ensure that their code is not mishandled in any way, and removed upon request.

2.1.3 Legal - UCF and CECS Affiliation

Any affiliation with UCF and its mascot, colors, or other trademarks owned and maintained by UCF will be used with their explicit approval that is written and copied into the licensing of the system and all releases of the system. UCF and CECS may assign future teams at their discretion from Senior Design to continue work on the operating system, whether it is from the initial release of PegasOS or the most current release of PegasOS. In this regard, it is recommended that any future teams assigned to this project allow for a Spring to Fall semester timeline, to maximize the amount of time dedicated to modifying the system and ensuring PegasOS is functioning at full capacity.

2.1.4 Ethical & Privacy

PegasOS will remain free of any type of mandatory spyware or hidden record collection of any kind. This is defined as follows:

- There will be no such software that tracks the users actions or otherwise collects data based on the users actions, without the users explicit approval and understanding of said software.
- There will be no such software that transmits the current state of the system without the explicit approval of the user, including but not limited to; general system diagnostics such as up-time, power usage, security settings and status, currently installed or running programs, and disk usage.
- There will be no such software that records and stores the user's inputs into the system for analytical purposes, where input is defined as including but not limited to; text inputs, voice inputs, and video or motion-tracking inputs. This excludes software such as Discord, Skype, or other communication software written by individuals, teams, or companies not affiliated with PegasOS, and any risk from using these software(s) is the responsibility of the user and not PegasOS or its team. This also excludes software such as but not limited to; live text editors, audio streaming services, and video streaming services.
- There will be no such software that records, releases, modifies, copies, deletes, or in any other way affects personal information from the user. PegasOS is not liable for any personal information kept within the system, and the storing, modifying, or otherwise placing of said personal information on the system is at the user's risk and theirs alone.

In addition to this, PegasOS and its team are not liable for data theft of any kind from the system. During the software's lifetime and in particular, during its infancy, we cannot guarantee 100% security of the system from malicious users attempting to damage the system, steal information from the system, or otherwise modify the system in such a way that is detrimental to the user against or to their knowledge. To the best of our ability, we will provide a secure and stable system, and it is up to the user's discretion to use the system safely.

3. Technical Content

[illegible]

3.1 Project-Specific Content

3.1.1 Project Goal

Our goal for this project is to create an operating system that can utilize the ARM 64-bit architecture that is native to the chip, since 32-bit systems are legacy support with the instruction set itself. Furthermore, we do not want this to be a fork of another operating system like many Linux distributions out there, but rather its own system. We want to create an operating system that has the essential programs that a user would need, plus some fun UCF-themed applications.

3.1.2 Objectives

1. To create a functioning operating system, called PegasOS, that uses purpose-built components by the PegasOS team.
2. To learn more about the design and process that goes into making systems software, and in particular operating systems.
3. To create a stable system that can be expanded upon, whether by individual contributors or future CECS Senior Design teams.
4. To fully release all code, documents, and related software components upon Senior Design's completion, so that the operating system can be accessed, modified, and used by anyone who wishes to do so.

3.1.3 Specifications and Requirements

Must-Haves

1. PegasOS shall have a custom kernel, so called The Kingdom, that is not based on an existing kernel.
2. PegasOS shall have an attached shell terminal to the kernel, so called The Royal Messenger, from which the user can issue commands to interact with the operating system.
3. PegasOS shall have at least 20 built-in shell commands, so called The Decrees, which allow the user to interact with the various managers and systems within the kernel and operating system.

4. PegasOS shall have driver support for the components built into the board, so called Tim.
5. PegasOS shall have at least 5 pre-built programs that allow the user to have a degree of control within the shell terminal.
6. PegasOS shall have a File Manager, so named The Armory, from which the user can change directories, create and destroy directories, create and destroy files, copy and move files, read and write to files, and it shall manage the partitions of the disks so that files and directories are preserved between startups.
7. PegasOS shall have a Scheduler, so named The Warden, which will manage the processes that are instantiated within the operating system's run, so that they all get execution time on the CPU.
8. PegasOS shall have an Input/Output Manager, so named The Gatekeeper, which shall handle all input from physical controllers, and disperse output to the appropriate devices that request it.
9. PegasOS shall have an Interrupt Handler, so named The Executioner, which shall handle the system's interrupts, which includes the minimum of: Input/Output Interrupts, Timeout Interrupts, Insert Interrupts, Suspend Interrupts, ProcessMurder Interrupts.
10. PegasOS shall have a Page Manager, so named The Scribe, which shall transfer memory pages between Swap, RAM, DRAM, Hard Disk, Solid State, and other such storage devices.

Like-to-Haves

1. PegasOS may have an animated booting screen, so called The Gate, which includes a Knight of some description, that is not alike to the UCF mascot Knightro unless explicit permission is granted.
2. PegasOS may have a usable set of compilers, so called The Dragons.
3. PegasOS may have a set of non-mandatory programs included that will elevate the user's experience with the operating system.
4. PegasOS may have a customizable shell terminal screen, with a dedicated section for terminal feed, past successful commands, and the current directory and file path, so called The Record.

3.2 Program and System Ideas/Decisions

Since this is a student project, we wanted to challenge ourselves but also not overwhelm our team. For this project, we need an operating system that has the essential components to any operating system while also having additional programs that the user can use. The following programs were agreed upon by the team as either required programs or 'like-to-have' programs, in a brainstorming session during a physical meeting. This list is neither comprehensive nor final.

3.2.1 Group Ideas

File Manager

This program allows for a user to manage files and folders. The user can perform the most basic operation on files such as creating, opening, renaming, moving, copying, deleting, and searching for files. They are also able to modify attributes, properties, and file permissions.

Calculator

A simple calculator that allows for basic operations such as addition, subtraction, multiplication, and division, with a programmer-friendly syntax

3.2.2 Christopher's Ideas

GoKnightScreamer

A simple program that reacts to the user typing "Go Knights" into the shell, printing out a banner with the words 'Charge On!' and potentially a sound effect to go with it

ZorkClone

One of the earliest interactive video games created by Tim Anderson, Marc Blank, David Lebling and Bruce Daniels in the late 1970s.

RoguePort

A custom port of some description of the classic game Rogue, a terminal-based randomly-generated turn-based hack-and-slash game.

3.2.3 Jacob's Ideas

Parking Lot Simulator

A simulation of the average parking situation on campus at UCF, presented in a charming console format

Hangman

Another pen and paper game where one player tries to guess the word or phrase by suggesting a letter given a certain amount of guesses.

3.2.4 Giancarlo's Ideas

TicTacToe

A pen and paper game where players take turns marking down X's or O's in a 3x3 grid. First player to get 3 marks in a vertical, horizontal, or diagonal row wins.

Texteditor

A text editor is a computer program that allows the user to read, edit, and save plain text files. The user can create text files to create text files or other programming language file types. It should be noted that this program is not a word processor that can create rich text format and is only a plain text editor.

3.2.5 Jacqueline's Ideas

Users & WhoAmI

A system within the operating system that allows for user accounts that separate data, files, and programs with different permissions for system access

KnightQuestions

Ask the shell any questions, and it will try to answer them

Tell Me a Story

The user asks the operating system for a story and it chooses a set of small short stories randomly.

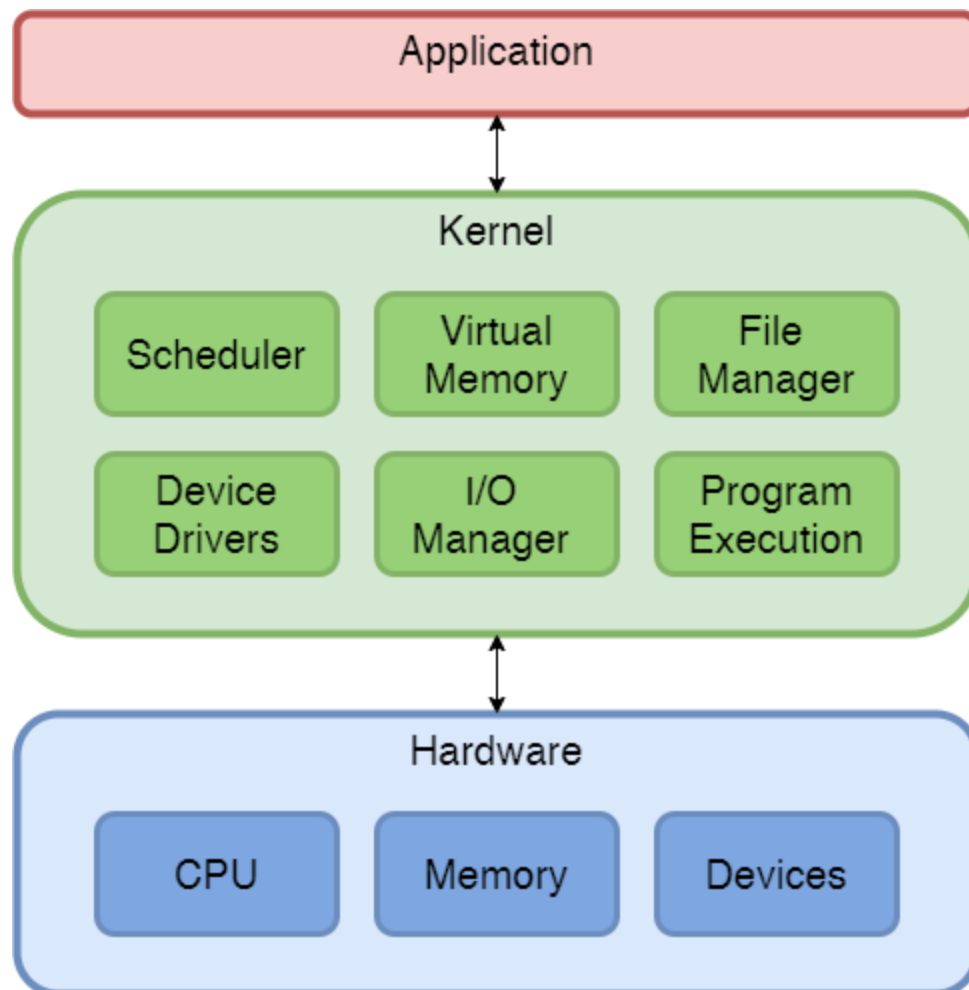
3.2.6 Kenny's Ideas

Display Settings

A PegasOS feature that will allow the user to change the display settings. The user can modify settings such as resolution, text font, and text size. The user can also modify the terminal to have add or remove the file path window and previous command lists window.

3.3 Initial Research and Design

3.3.1 Design Overview



3.3.2 Cross-Compiler

There are two main methods of building the kernel for our Raspberry Pis. We can either build the program locally on our Raspberry Pis or build using a cross-compiler. We opted to use the cross compiler since it will be much quicker to compile. The only downside to this is that it will require more time to set up. So why do we need a cross compiler? If you use the compiler that comes with your system, it wouldn't know it's compiling for something on a different platform. Otherwise a lot of unexpected things can happen because the compiler assumes your building code on your host operating system.

There are two ways that we can set up the cross-compiler. We can build the cross-compiler ourselves. That way we can use the latest GCC compiler on the system. We can also download prebuilt cross-compilers to skip setting up the build for the cross-compiler. It should be noted that this might not use the GCC compiler on the system. In this document, we will be going over both the setup.

3.3.3 Setting Up the Build for the Cross-Compiler

To begin building our cross-compiler we will need to know what platform we want to build on, otherwise, we will run into problems. Our platform will be the aarch64 platform since we want to utilize the 64-bit architecture. To prepare our cross-compiler we will need several dependencies to download to compile our compiler.

GCC Compiler

The GNU Compiler Collection is a compiler system produced by the GNU Project that supports various programming languages

Binutils

Also known as GNU Binary Utilities is a set of programming tools for managing binary programs, object files, libraries, profile data, and assembly source code.

Bison

A general-purpose parser generator that converts an annotated context-free grammar into a deterministic left-right parser or generalized left-right parser employing parser tables

Flex

A fast lexical analysis generator that is used for generating scanners, programs that recognize lexical patterns in text.

GMP

A free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers.

MPC

Another library for the arithmetic of complex numbers with arbitrarily high precision.

MPFR

C library for multiple-precision floating-point computations with correct rounding

After downloading all the dependencies that are required, we will need to find a place to put the cross-compiler. It is generally recommended to install the cross-compiler into `$HOME/opt/cross` if you want to install it locally onto your computer. If you want to install it globally, a good directory would be `/usr/local/cross/`. It should be noted that this should work for Windows Subsystem for Linux.

3.3.4 Setting Up the Prebuilt Cross-Compiler

ARM has a developer section on their website where you can download the GNU Toolchain for your cross-compiler. For our purposes, we want to download the GNU-A Toolchain since our Raspberry Pi uses the Cortex-A processor. The specific GNU Toolchain we used for this project is the AArch64 ELF bare-metal target (aarch64-none-elf). Our reason for choosing this version is that our OS won't be built on top of Linux, so we need to use the bare-metal target.

After downloading the prebuilt cross-compiler, create a directory to put your new cross-compiler into. It's generally recommended not to install it into system directories. A good place to put your new cross compiler is in `$HOME/opt/cross` if you want for just your system. After creating the directory move files from the tar file into the destination you chose to put your cross-compiler.

Now you need to edit the path environment variable. For Linux, all you need to do is edit the `~/.profile` found in the home directory. You can do this by using nano to edit the file. Scroll down to the bottom and paste this line of text `export PATH="$HOME/opt/cross/bin:$PATH"`. Restart the computer and your system should be able to compile with ARM.

3.4 Build, Prototype, Test, and Release Plans

This section details our various testing methods and plans for testing the validity of the system. This is broken up into various sections in the order that the testing will be taking place.

3.4.1 Equipment

Testing equipment for PegasOS will consist of the following:

- Raspberry Pi 4
- 3.5A USB-C Power Adapter
- Micro HDMI to HDMI Cable
- USB Micro SD Card Reader
- Raspberry Pi 4 Heat Sinks
- HDMI-Compatible Display
- USB-to-TTL Cable

The SD Card, Power Adapter, HDMI Cable, and Raspberry Pi itself are the bare minimum to get the system up and running. For our initial testing, we need the USB-to-TTL Cable to get input readings from the GPIO pins to test our framebuffer. More information about this testing can be found in our *Hardware* section.

Once our framebuffer is working, and output is being printed to the screen, we can make use of the HDMI cable to test future iterations of the operating system on a traditional display.

3.4.2 Prototypes

As we develop the code base of PegasOS, first we will need to develop compartmentalized prototypes of various components within the system, before we integrate the prototype component into the main system. These prototypes will undergo unit tests of their own, before their integration into the rest of PegasOS, where it will undergo further unit tests to ensure that its integration was a success and has not affected the usage or operation of other aspects of the system.

3.4.3 Building

Building PegasOS will require the usage of the cross-compiler (as detailed previously), as building on the Raspberry Pi will take far too long to be feasible over multiple tests in a short timeframe, even with five separate units across the members of the team. Therefore, building PegasOS will involve compiling the system into ARM binaries that can be bundled and mounted onto the SD Card, which can then be inserted into the Raspberry Pi and run on the hardware.

If a user would like to build the system themselves, they will need to set up their own cross-compiler to target ARMv8 or aarch64. For references on how to do this, please visit our *Cross-Compiler* section for more information. If a user would like to create their own .iso file for the system, they are free to do so.

3.4.4 Testing

To test various aspects of the operating system, we will be designing unit tests for each major component of the system. This includes but is not limited to:

- The Kernel Scheduler
- The Kernel Input/Output Manager
- The Kernel Memory Manager
- The File System
- The PegasOS Shell
- The Kernel's System Calls
- The System's Program Execution

These unit tests will be recording empirical data, such as execution time from process invocation to scheduler insertion, computation time for scheduling algorithms, average CPU burst time, average Page Swaps per minute, average number of processes, and so on. Using these tests, we can determine if components or subsystems of the system are underperforming and evaluate methods to boost the performance of these components and subsystems. Using these tests, we can also create logs of system performance within the system itself, such that if the user wishes to know how the system is performing, the system can report its findings to the user.

3.4.5 Shipping/Release

Once the first official release of PegasOS is ready for public usage, we will release the source code as well as a mountable .img or .iso file to use in installing the operating system. The source code will be available for those who wish to inspect the code, compile the code themselves into an .img or .iso file for mounting on a Micro SD Card. For the convenience of other users, we will also include links to pre-compiled .img and .iso files for use in mounting PegasOS onto Micro SD Cards for installation.

At the current time of writing, there are no plans for a physical release of the operating system on Disks, Flash Drives, SD Cards, Micro SD Cards, or other media storage devices. As we are, at time of writing, not making a profit from this operating system, we cannot supply the funds necessary for a wide release of physical media containing the operating system. We therefore ask that the user take it upon themselves to mount, burn, or otherwise copy the files necessary to use the system onto a media storage device that they own.

The source code will be available on the project's public GitHub repository, from which it can be cloned, downloaded, and forked. We will also provide links to downloads of the .img and .iso files that we have pre-built. These links will most likely be on the GitHub repository as well, and if they are hosted elsewhere we will include links to the new download host(s).

3.4.6 Contributions

As the project matures and is released to the public for usage, there will be individuals that will wish to contribute to the project. One of the goals for this project is to be a community resource into creating operating systems for the Raspberry Pi, and for these reasons we welcome all suggestions and contributions to the system that are beneficial to its development. If an individual or group would like to contribute to the project, they may ask for permission to become a contributor on the project GitHub. When asking to become a contributor to the project, please state your reasoning for why you wish to do so, and what aspects of the system you would like to work on. Upon being accepted as a contributor to the project, you may then make pull requests which can then be reviewed by members of the team for merging into the master branch.

3.4.7 Source Code Modification

As per the provisions granted under the GNU General Public License (version 3 and later), any individual or entity is allowed to download and modify the source code of PegasOS for their own personal usage. If they would like to submit their modifications as an official contribution to PegasOS, they may do so through the public GitHub page where they can apply to become a contributor to the project and have their submission reviewed by the team.

If a user, individual, or group would like to fork the project and make their own modifications to their fork, they are free to do so under the provisions of the GNU General Public License, versions 3 and later. They may also use their fork of the project as a reference in applying to become an official contributor to the project.

PegasOS

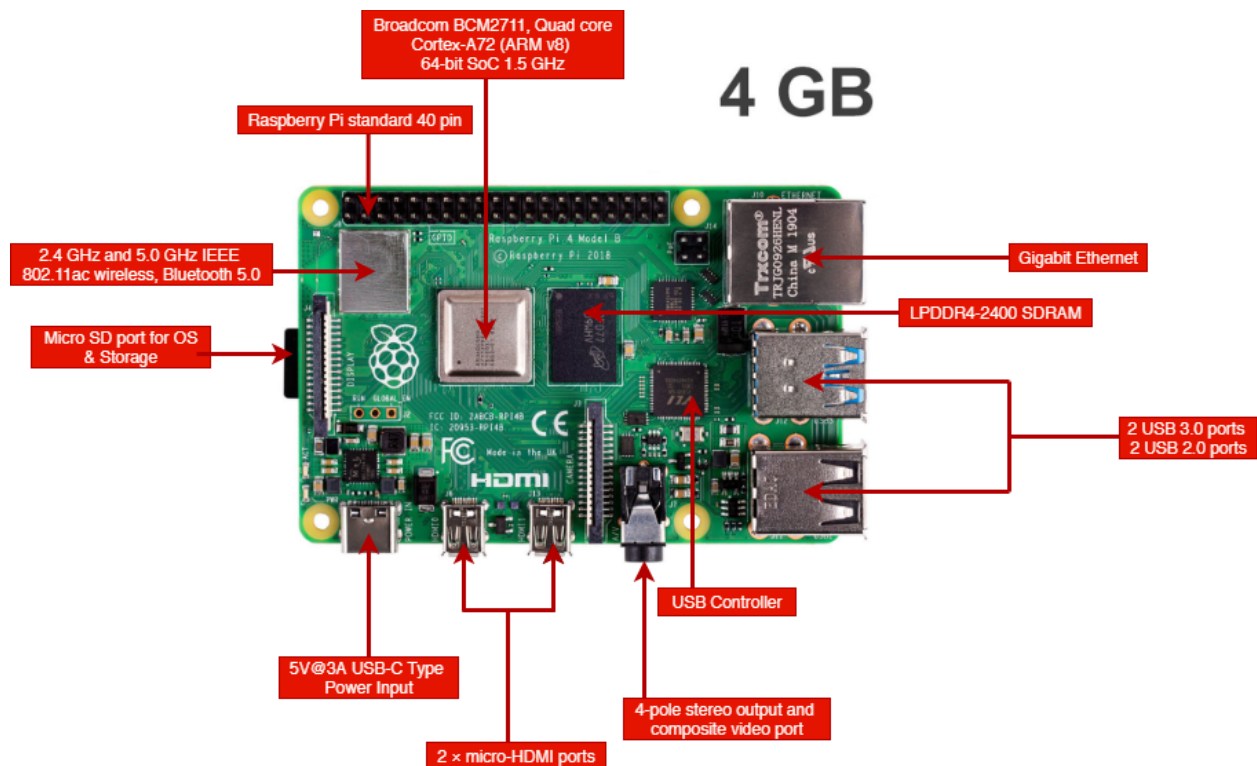
4. Hardware

[illegible]

4.1 Raspberry Pi Hardware

The Raspberry Pis' internal hardware has evolved through several generations. All current Raspberry Pis have a Broadcom SoC (System on Chip) that has an ARM integrated CPU (Central Processing chip) and GPU (Graphics Processing Unit). The Raspberry Pi 4 has an 1.5 GHz 64-bit quad-core processor with a Broadcom VideoCORE VI at 500 MHz. The Raspberry Pi 4 is also equipped with on-board 802.11ac Wifi, Bluetooth 5.0, gigabit Ethernet, two USB 2.0 ports, two USB 3.0 ports and dual Micro HDMI ports for dual monitor support.

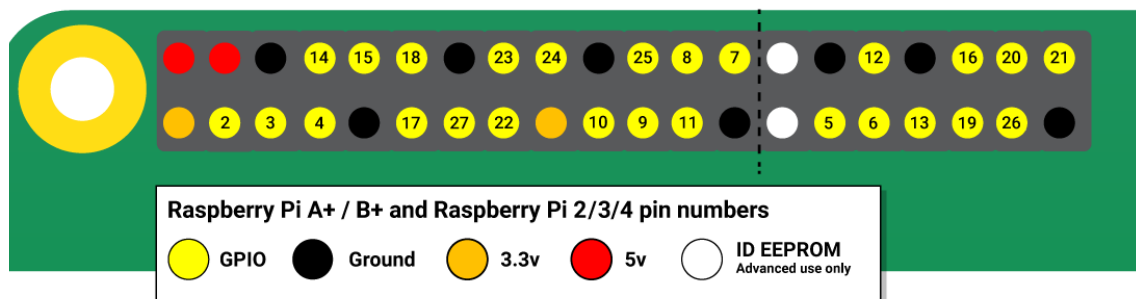
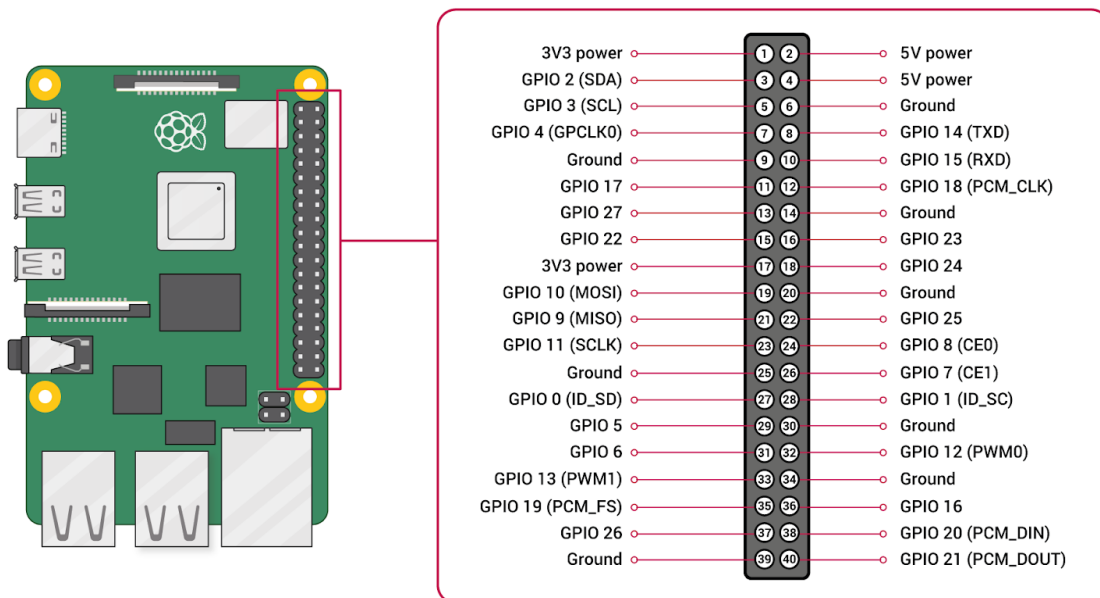
A system on a chip is essentially all of the parts of a computer in a single chip. This is great for things like the Pi 4 because it uses up much less power than multi-chip designs with similar functionality. The Pi 4 uses a Broadcom BCM 2711 SoC. It uses an ARM 72 core, making it more powerful than its predecessors. Furthermore, its GPU is improved to process input and output faster. This SoC is capable of addressing more memory, continues to use heat spreading technology, and a natively attached Ethernet controller.



Credit to: Raspberry Pi Company

4.1.1 General-Purpose Input/Output (GPIO)

One feature of the Raspberry Pi is the GPIO (General-Purpose Input/Output) pins along the top edge of the board. Most Raspberry Pi boards have a 40-pin GPIO header except Pi 1 Model B+ (2014). Each GPIO pin on the board is used for a wide range of purposes. Here is a diagram showing how the GPIO pins are assigned.



Credit to: Raspberry Pi Company

4.1.2 Bootloaders

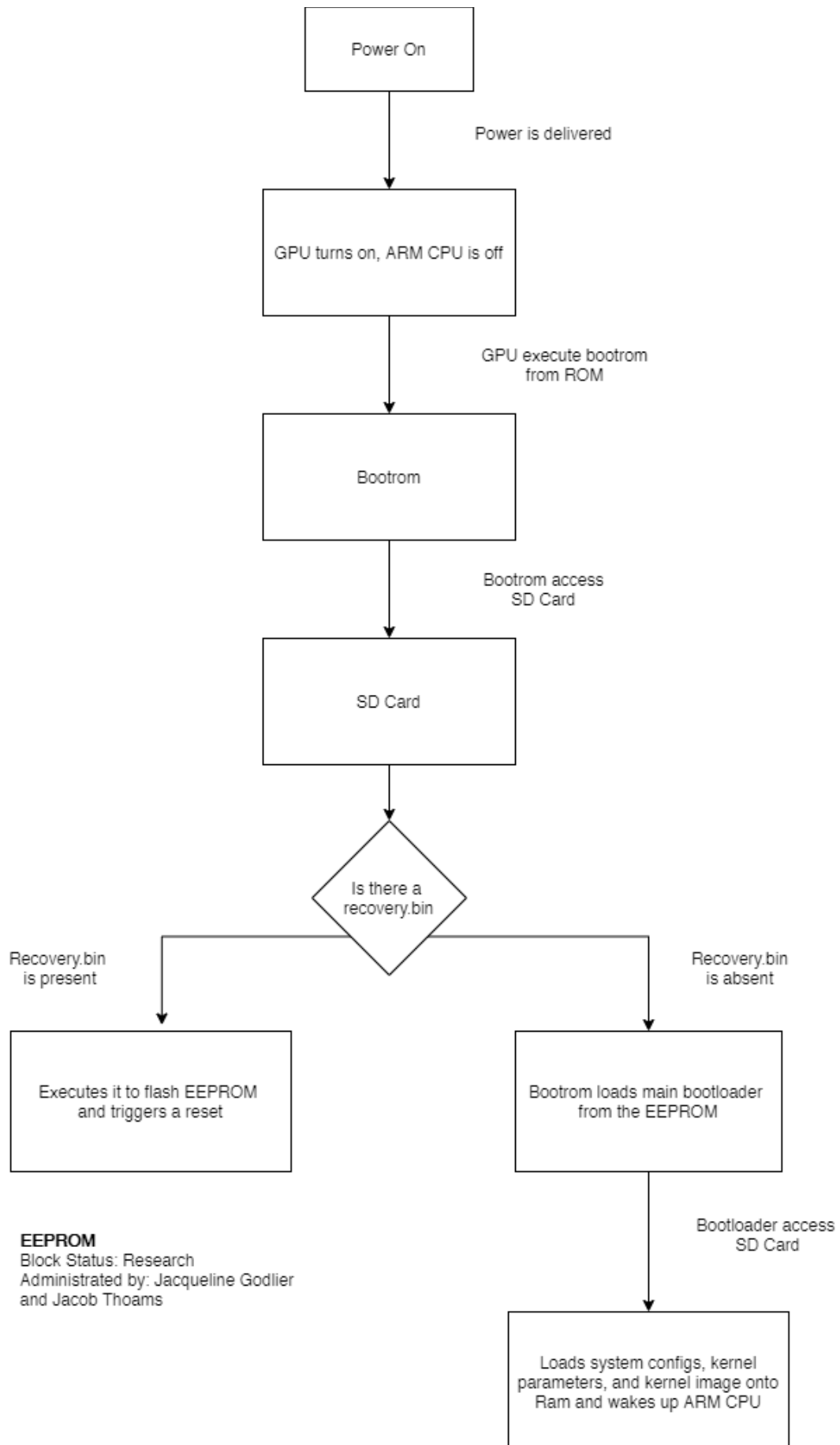
Before we begin developing the kernel for the OS, it would be best to get a better understanding of how the Raspberry Pi works by learning more about the booting sequence. This way, we will know how to boot up the kernel properly.

Not every iteration of the Raspberry Pi boots up the same way. Before the Pi boots up, power must be applied to the Raspberry Pi. Once the power is delivered, the Pi will begin to boot the GPU to execute the first stage boot loader. This is where we begin to see the differences in the booting sequences between the older Raspberry Pi's (Versions 1, 2, 3) and the Pi 4.

In the older Raspberry Pi's, the first-stage bootloader mounts to the SD card and loads the second stage bootloader (bootcode.bin). It then loads it into the SoC's (system on chip) L2 cache. The second stage loader would then load and execute the start*.elf files.

The Pi 4 does a different method, because it loads the code contained in an EEPROM, which is in the SoC, and executes it. The EEPROM is an electrically erasable programmable read-only memory, that is nonvolatile memory integrated into many electronic devices to store relatively small amounts of data. The purpose of this EEPROM boot code is to load the GPU and to execute the next boot stage. It serves the same purpose as the bootcode.bin but instead loads the code in the EEPROM instead of the SD card.

The start.elf files reads the system configurations, loads the kernel image, reads the kernel parameters, and releases the reset signal on the arm to boot up the kernel. It also should be noted that prior to October 2012, the 2nd stage bootloader would load a 3rd stage bootloader called the loader.bin file. The loader.bin file handles the elf files, but has since then been removed in the booting process because the bootcode.bin file has been updated to load the elf files instead.



4.1.3 EEPROM

The EEPROM is an electrically erasable programmable read-only memory. It is non-volatile memory, which means that it does not need constant power to retain data. EEPROM is commonly used in microcontrollers and smartcards.

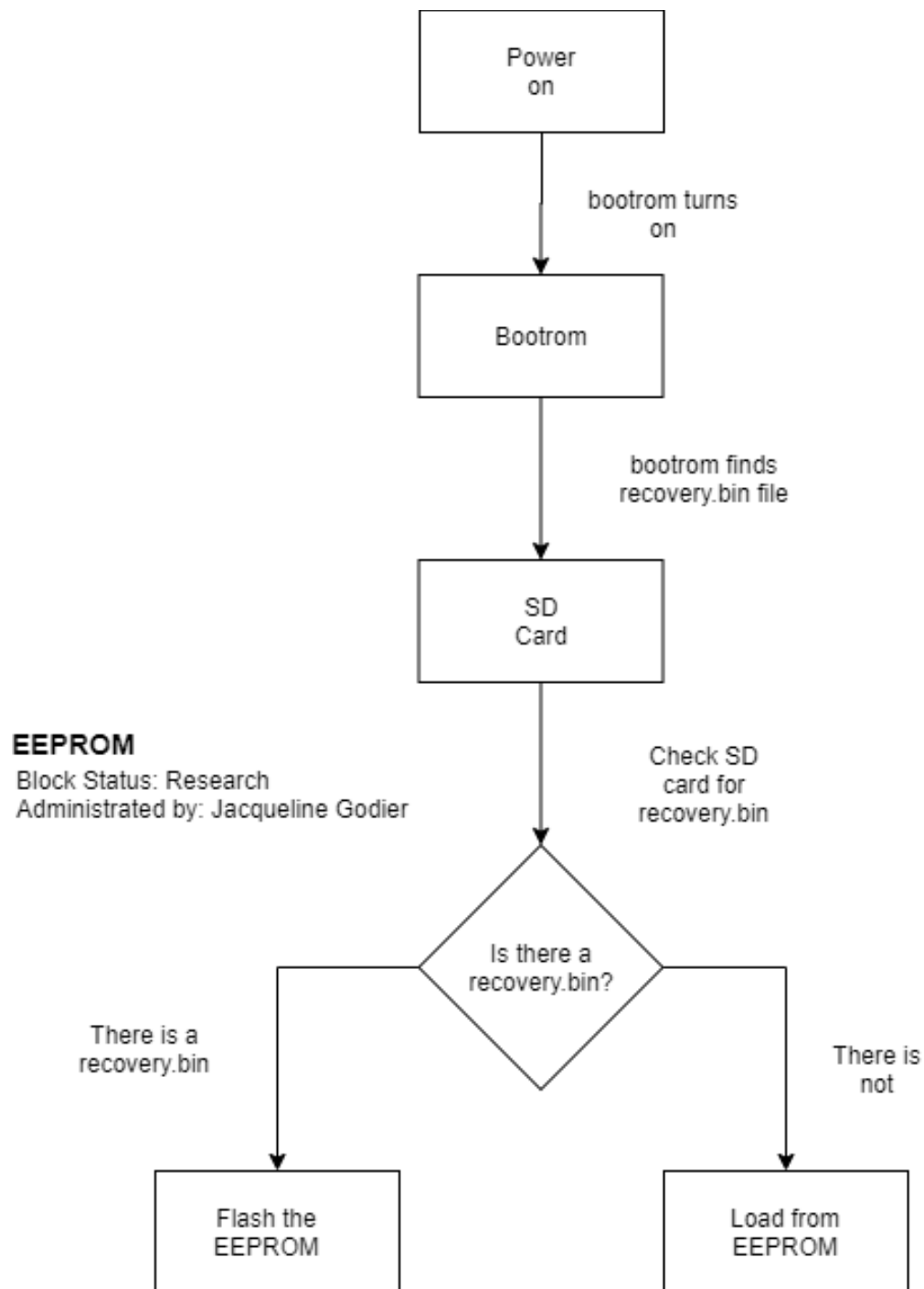
The EEPROM was included for many reasons. First, the boot sequence for the Pi 4 is more complicated, so there is more room for error in the read-only memory on the System on a Chip. After further research we noticed that most open-source kernels for the Pi 4 went straight to writing the kernel. We found the EEPROM is constantly being updated, with some current updates bringing down the processor temperature.

The EEPROM is not easily editable. As it stands, there is some sort of bootloader inside of the EEPROM, and is not something you can overwrite without flashing the EEPROM. This is a bad idea for multiple reasons, one being that the EEPROM has only so many times it can be overwritten, and also flashing the EEPROM would need some additional hardware.

However, there is also a workaround for the EEPROM. In the case the EEPROM gets corrupted, Raspberry Pi has a `recovery.bin` file you can use on it to reset the EEPROM. Updates can be made to the bootloader using that. However, it is easier said than done, because although the EEPROM has no write restrictions it requires very specific formatting to get anything to work properly.

Since the goal of this project is to install an OS out of the box of any Pi 4, we have decided to keep the current code instead of EEPROM. This will reduce the amount of hardware we will need for the project, which brings down our costs. Furthermore, since the EEPROM is constantly being updated, any rewrites we would make would just be overridden the next time someone updates it.

This is not an end-all to any sort of customization, however. The EEPROM still has a readily available config file. From here, we will do most of our low-level edits to it to prevent any previously stated roadblocks.

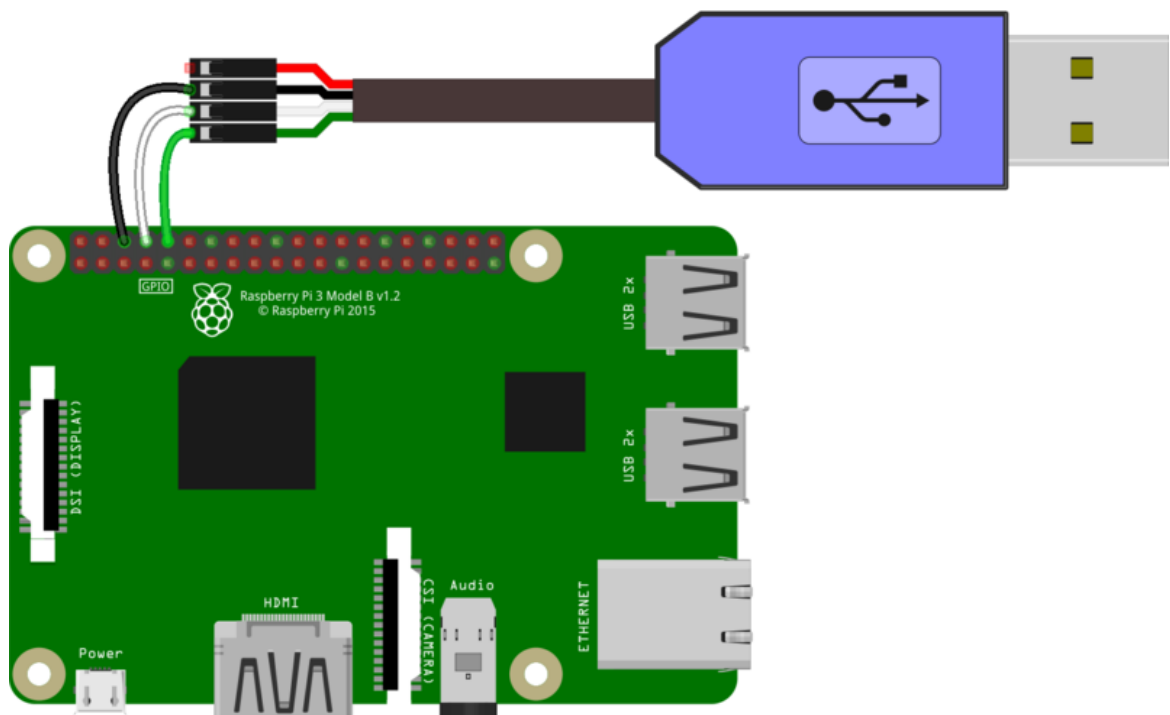


4.1.4 Basic Kernel Test

With this in mind, to test out our theory about the booting sequence we tried loading a basic Kernel onto it. This presented some challenges. First off, many setups we were using were configured to take the Linux kernel specifically. Since we are writing our own kernel, we could not just load a Linux-based kernel. As such, we found a cross compiler from the ARM company that did not specifically need a Linux kernel. We tried to get it to load to the screen but then found nothing would show first. After that, we realized that we needed a frame buffer.

A frame buffer is a portion of memory that contains the frames that are sent to the screen. To print something to the screen, we would need to convert the I/O from the chip to something that can be readily read by the screen. However, for a computer to find a frame buffer, it first needs a mailbox peripheral to find it from.

To test our kernel without programming a framebuffer, we bought a USB-to-TTL cable that will send input into the serial console. Following the picture below, the black lead needs to attach to the GND pin, the white lead to the TXD pin, and green lead to the RXD pin.



fritzing

Credit to: Adafruit

After finishing connecting the serial cable to the Raspberry Pi, plug in the USB to your computer. Your computer will need a serial console, so I would recommend using a program called PuTTY. Change your connection type to serial and put COM3 as your serial line and 115200 for your speed. Now you need to flash your SD Card for the Raspberry Pi. To do this, download any Raspberry Pi operating system and flash onto the SD card. Once that is done, replace the OS's .img file and put your compiled .img file onto the SD card. All that's left is to insert your SD card in the Raspberry Pi and power it on. You should be getting a "Hello World" on your serial terminal.

4.1.5 Drivers

Our PegasOS project will require some basic drivers in order for our operating system to function properly. A keyboard, mouse, and sound output driver are essential components for this project. Our keyboard drivers will enable the user to type inputs into their keyboard. The OS also needs a mouse driver for the user to click on the screen. Finally, we will need sound output to play the go knights scream at the startup. We originally thought of making the drivers ourselves since it will give us a learning experience on how to make drivers and give us a better understanding of our Operating System. Unfortunately, there is not a lot of information on how to make USB drivers for the Raspberry Pi. For the sake of our time and sanity, we will be using an open-source repository for our USB drivers. This will prevent us from having to reinvent the wheel and also gives us more time to work on more important matters with the hardware.

<https://github.com/rsta2/uspi>

This URL link is the USB driver repository we will be using for our operating system. It's called USPi and it's a bare metal USB driver for the Raspberry Pi in C. It was originally ported from the Circle USB library which was written in C++. The functional drivers support USB keyboard, mice, MIDI instruments, gamepads, USB flash drives, and on-board Ethernet controllers. This repository also comes with an environment library that provides all required functions to get USPi running. In order to use this repository, we will have to make some changes in the configuration file in *include/usprios.h*

4.1.6 I/O

GPIO Registers

Address	Field Name	Description	Size	Read/Write
0xXX20000	GPFSEL0	GPIO Function Select 0	32	R/W
0xXX20004	GPFSEL1	GPIO Function Select 1	32	R/W
0xXX20008	GPFSEL2	GPIO Function Select 2	32	R/W
0xXX2000C	GPFSEL3	GPIO Function Select 3	32	R/W
0xXX20010	GPFSEL4	GPIO Function Select 4	32	R/W
0xXX20014	GPFSEL5	GPIO Function Select 5	32	R/W
0xXX2001C	GPSET0	GPIO Pin Output Set 0	32	W
0xXX20020	GPSET1	GPIO Pin Output Set 1	32	W
0xXX20028	GPCLR0	GPIO Pin Output Clear 0	32	W
0xXX20034	GPCLR1	GPIO Pin Output Clear 1	32	W
0xXX20038	GPLEV0	GPIO Pin Level 0	32	R
0xXX20040	GPLEV1	GPIO Pin Level 1	32	R
0xXX20044	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W
0xXX20064	GPEDS1	GPIO Pin Event Detect Status 1	32	R/W
0xXX20068	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W
0xXX20064	GPREN1	GPIO Pin Rising Edge Detect Enable 1	32	R/W
0xXX20098	GPPUDCLK0	GPIO Pin Pull-up/down Enable Clock 0	32	R/W
0xXX2009C	GPPUDCLK1	GPIO Pin Pull-up/down Enable Clock 1	32	R/W

For more information about each register, please see the BCM ARM Peripheral Manual.

UART Address Map

Address Offset	Register Name	Description	Size
0x0	DR	Data Register	32
0x4	RSRECR	Flag Register	32
0x18	FR	Flag Register	32
0x20	ILPR	No use	32
0x24	IBRD	Integer Baud Rate	32
0x28	FBRD	Fractional Baud Rate Divisor	32
0x2c	LCRH	Line Control Register	32
0x30	CR	Control Register	32
0x34	IFLS	Interrupt Mask Set Clear Register	32
0x38	IMSC	Raw Interrupt Status Register	32
0x3c	RIS	Interrupt Clear Register	32
0x40	MIS	DMA Control Register	32
0x44	ICR	Test Control Register	32
0x48	DMACR	DMA Control Register	32
0x80	ITCR	DMA	32
0x84	ITIP	Integration Test Input Register	32
0x88	ITOP	Integration Test Output REgister	32
0x8c	TDR	Test Data Register	32

For more information about each register, please see the BCM ARM Peripheral Manual.

There are a couple of pressing matters to consider with the I/O. These include the Page manager, the TLB, and how ARM registers are set up. Without these components, we cannot hope to get even the simplest of algorithms written for our OS, as they are crucial to interact with the hardware.

4.1.7 Framebuffer

To display on the screen without any special hardware, we need a framebuffer. A framebuffer is a portion of RAM that uses a bitmap to feed to the video display. It is a piece of data that both the GPU and CPU use. Most modern video cards have some sort of device to convert bitmap to video signal, but there is a portion of code that needs to be written to feed the data to it. The data is written as RGB, and then is fed to the GPU.

To initialize the framebuffer, you need to print a fully black screen. Then, you need to write pixels individually to the screen. Each character has a different bitmap. After this data is written in a bitmap, it is then fed to a mailbox peripheral to be fed to the hardware.

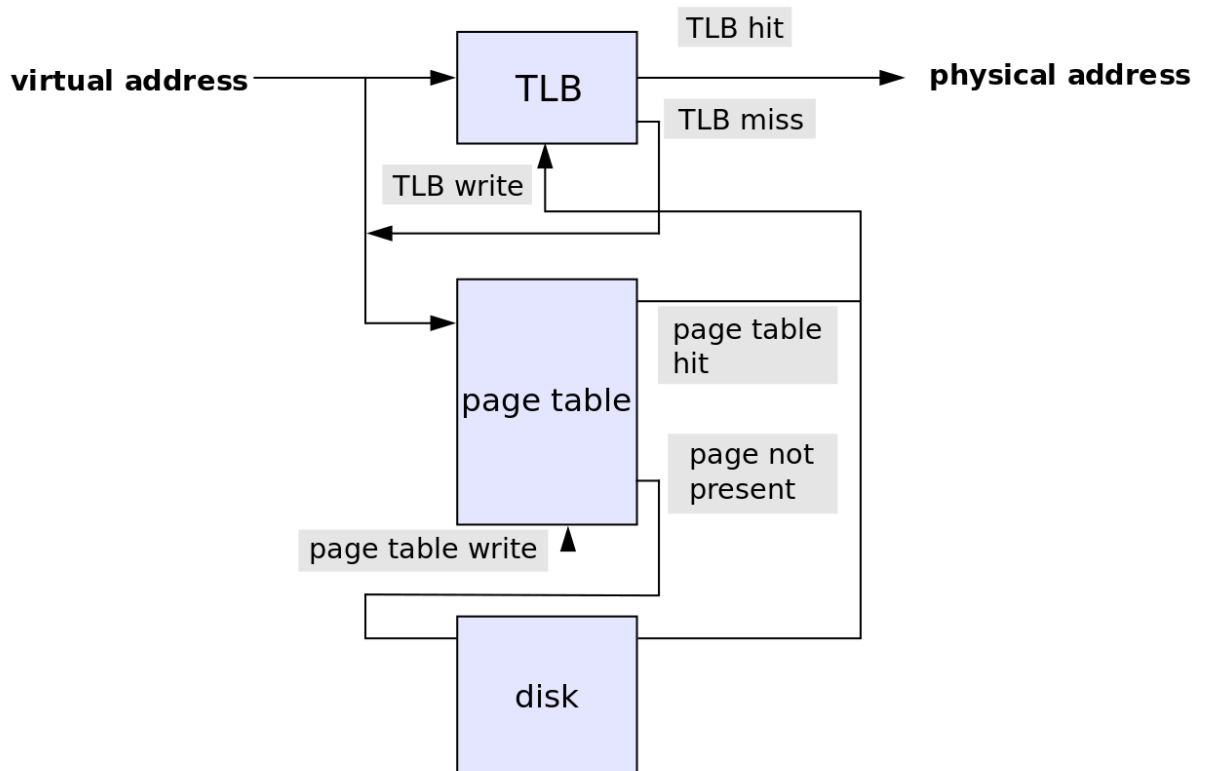
4.1.8 Mailbox Peripheral

The mailbox peripheral is how the ARM GPU talks to the Video GPU. It is the closest to bare metal we will get in terms of the display. The mailbox has three registers: read, write, and status. The Read register is at offset 0x00 from the mailbox base address and it facilitates reading messages from the GPU. It's made up of 4 lower bits that say which channel the message is from and 28 high bits are our data. The Status register is at offset 0x18 from the mailbox base address and it tells you whether the read register is empty or tells you whether the write register is full. It has two bits: bit 30 tells you if the read register is empty and bit 31 tells you if the write register is full. The Write register is at 0x20 from the mailbox base address and it works similar to the Read register. It has 4 low bits that tell which channel and the 28 high bits are our data.

The mailbox has 8 channels. A channel is a number that tells you and the GPU what information is being sent through the mailbox. Means. For this project, we will be only needing channel 1, the framebuffer channel, and channel 8, the property channel (ARM -> VideoCore).

4.1.9 Page Table

The page table maps virtual memory to physical memory. The processes in memory use virtual memory while the hardware uses physical, and the page table is what reconciles the two. The page table first checks the TLB to see if the memory has been recently used and stored in the cache. If it is not there, a TLB miss occurs and the mapping is looked up in the page table. This is the tricky part, because there are a couple different page tables you could use.

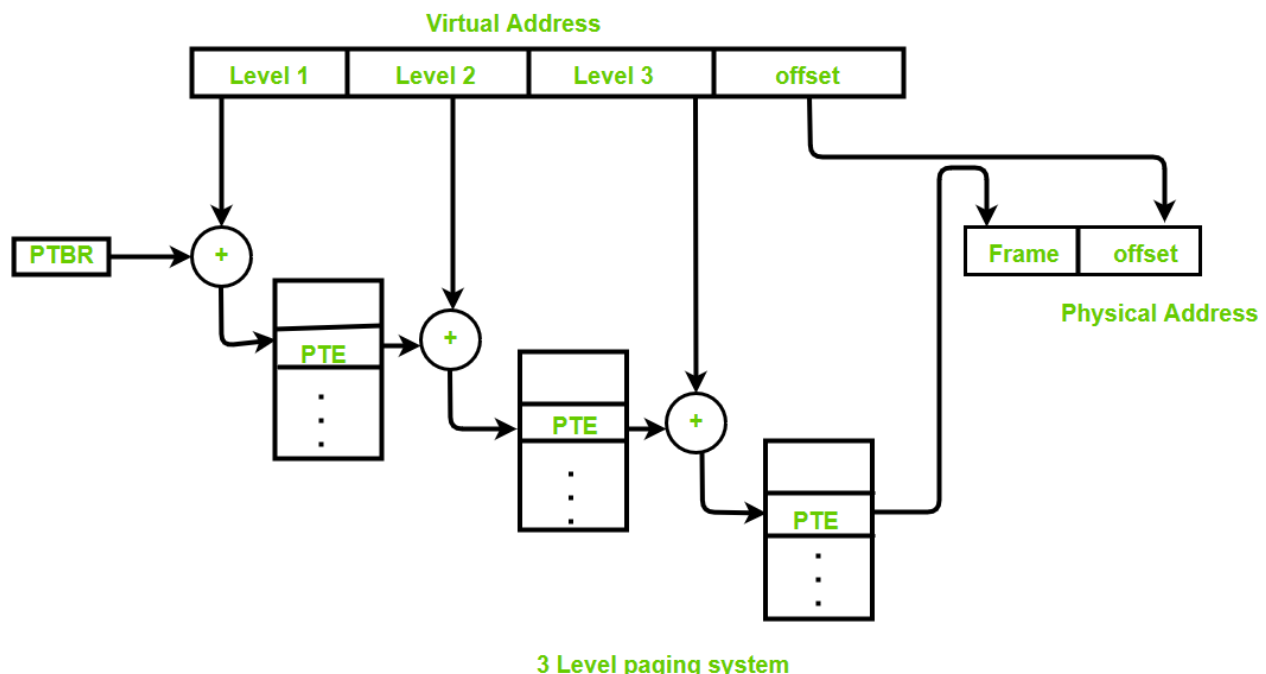


Source: https://en.wikipedia.org/wiki/Page_table

Before discussing the different implementations of page tables, it should be important to talk about page sizes. The default size for a page table is 4KB, but can be increased or decreased depending on how process space is needed. Having a large page size, like 2MB or 1GB, can result in a number of problems like having too much internal fragmentation(i.e unused space) and can lead to less flexibility for placing data near processors accessing them the most. But having very small page sizes, like 512 B, can also lead to some major issues including high overhead for bookkeeping translation.

There are several ways of implementing a page table, each having different tradeoffs between one another. One type of page table is a linear page table. This page table is linearly allocated for each application starting from a specific address, thus it being impractical to use. Another type of page table we could have used is an inverted page table. One page table is shared among all the processes and we can use a hash function to speed up the lookup process. One benefit of this is just managing a one-page table for potential low-storage overhead. However, the spatial locality is not considered and no physical pages are shared between applications. Our last option is to use a multi-level page table, which is the implementation we choose to use. It has the benefits of being able to grow dynamically similar to a tree, each application has its own page table, and the page table entries are dynamically allocated with paging.

ARM naturally supports multi-level page tables. This means that the page table is split into levels that point to lower ones. Only the lowest level have frames. This also means that accessing one piece of memory takes several different level accesses to reach. This may seem counterintuitive, but having multi-level paging means that the page table can be split into many parts and page itself. This is especially useful if the entirety of the page table cannot fit in one part of memory. In general, 64-bit systems use 4-page tables.

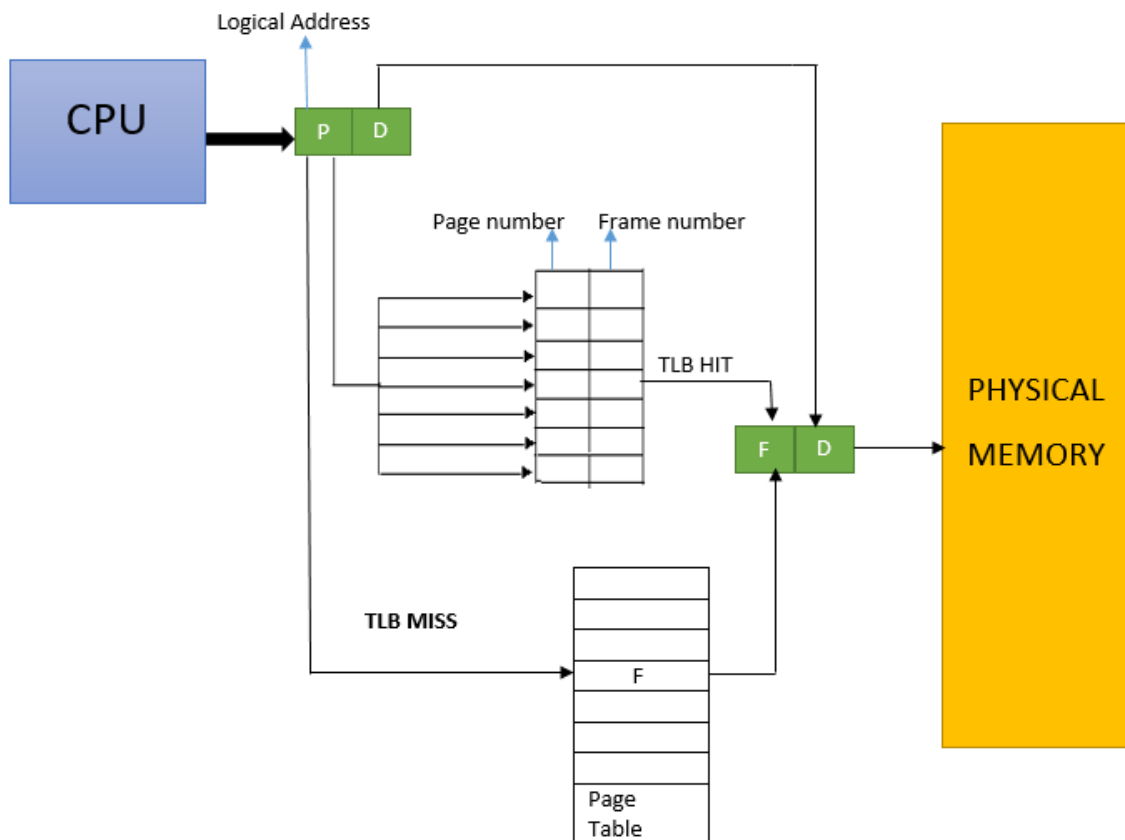


Source: <https://www.geeksforgeeks.org/multilevel-paging-in-operating-system/>

4.1.10 Translation Lookaside Buffer (TLB)

Although using a page table can help store vast amounts of memory addresses, the one downside to using this method is slower access to memory. One solution to this problem would be using a translation lookaside buffer (TLB). The TLB is a memory cache that reduces the amount of time it takes to access a memory location. It stores the most recent translations from virtual memory to physical memory. Each entry will consist of two parts: a page number and a frame number. When the CPU initially looks for an address, it will go to the TLB first. If the data is already there, it will have a TLB hit and bring the data back in. If not, it will have a TLB miss and go into the page table as an interrupt, look up the data, and put it into the TLB. It will then go through the TLB and have a TLB hit.

Although the TLB is not absolutely required, not having one means you are almost working with bare metal, which is very insecure and can lead to some unfortunate errors. Also, having a TLB speeds up the lookup process if you already have certain commonly used addresses stay inside the cache.



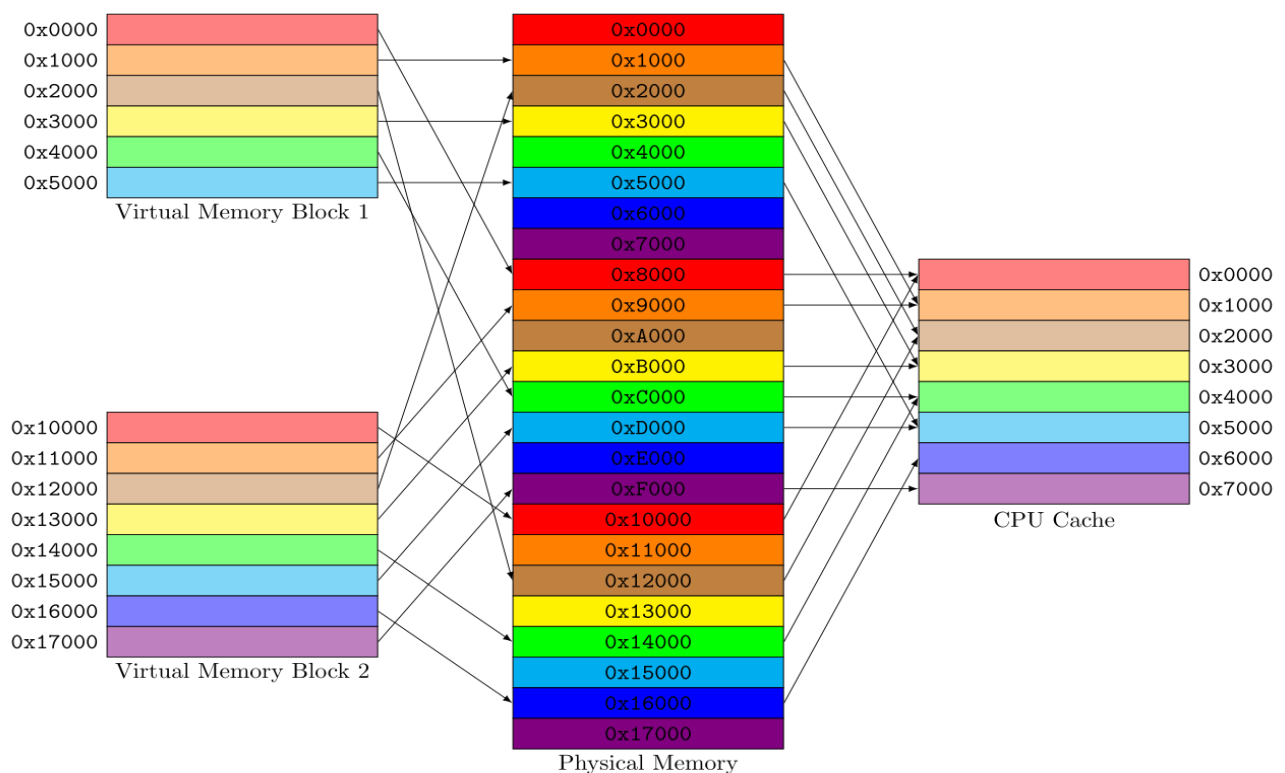
Silberschatz, Galvin, Gagne, Abraham, Peter B. , Greg (2009). *Operating Systems Concepts*. United States of America: John Wiley & Sons. INC. ISBN 978-0-470-12872-5.

4.1.11 Page Coloring

Page coloring is a way to make virtual memory look contiguous. It is in the same vein of thinking as a multi-level page table- meaning that it “colors” pages that look contiguous in virtual memory and maps them to physical memory non-contiguously. Although it is not required to have a functioning page table, it is very closely related to multi-level page tables and would be a great way to bring the performance of PegasOS to a more efficient and deterministic state.

For our project, we decided to keep this out of scope so as to not overwhelm the team. Our operating system will be very bare-bones, and as such will not have super complex memory to manage. However, this will be a get thing to add as a possible stretch goal, or for a future team.

Because we are not implementing page coloring, our process will be a little more streamlined. The addresses will not have a separate virtual and physical address, and will instead be the same on both fronts. It is important to note that this is very secure to program, but due to the scope of our project we are limiting functionality on this feature.



Source: https://en.wikipedia.org/wiki/Cache_coloring

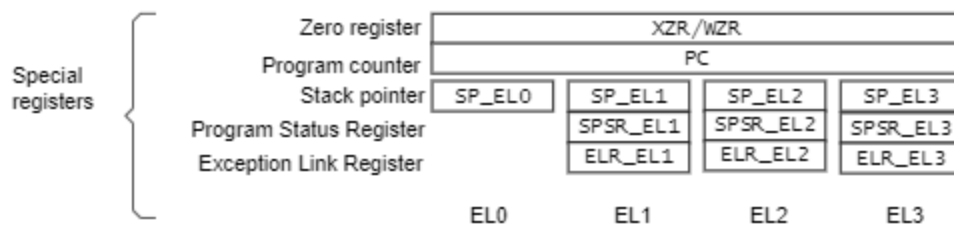
4.1.12 ARM Registers and General Research

Reference for this section: <https://developer.arm.com/docs/den0024/a/armv8-registers>

	X0/W0
	X1/W1
	X2/W2
	X3/W3
	X4/W4
	X5/W5
	X6/W6
	X7/W7
	X8/W8
	X9/W9
	X10/W10
	X11/W11
	X12/W12
	X13/W13
	X14/W14
	X15/W15
	X16/W16
	X17/W17
	X18/W18
	X19/W19
	X20/W20
	X21/W21
	X22/W22
	X23/W23
	X24/W24
	X25/W25
	X26/W26
	X27/W27
	X28/W28
Frame pointer	X29/W29
Procedure link register	X30/W30
	EL0, EL1, EL2, EL3

There are 31 x64 bit general purpose registers accessible at all exception levels. Each register has a 32-bit form for backward compatibility. There are also several special registers:

Figure 4.3. AArch64 special registers



Zero Register

A zero register is a register that is a hard wired 0. This is a very common register component because 0 is a very useful constant.

Program Counter

The program counter houses the current instruction. This is not easily accessed and really should not be directly accessed.

StackPointer

The stack pointer points to the current point on the stack.

Program Status Register

This register contains information on the current state of the program being executed and the state of the processors.

Exception Link Register

This contains the link to return to once a function call completes. There is a dedicated SP with every exception level, but it does not hold the return state.

Saved Process Status Register

Holds the value of the process right before it was interrupted.

4.1.13 ARMv8 Processor States

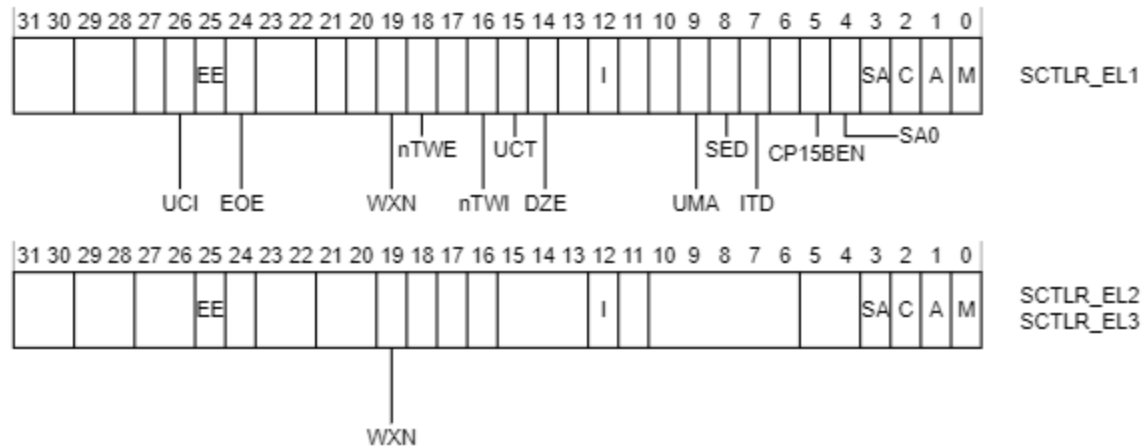
Name	Description
N	Negative Condition
Z	Zero Condition
C	Carry Condition
V	Overflow Condition
D	Debug Mask bit
A	SError mask bit
I	IRQ mask bit
F	FIQ mask bit
SS	Software Step bit
IL	Illegal execution state bit
EL(2)	Exception level
nRW	Execution state 0= 64 1= 32
SP	Stack Pointer selector 0 = SP_EL0 1 = SP_ELn

Unlike ARMv7, ARMv8 does not have a current program status register. The components of the old register are not accessible under these different processor states. N, Z, C, and V can be accessed from EL0. All others are undefined at EL0, but can be accessed from EL1 and up.

4.1.14 System Registers

The System Control Register (SCTLR): Controls standard memory, system facilities, and provides status for functions in the core.

Figure 4.5. SCTLR bit assignments



4.1.15 Endianness

In computer systems, there are two ways of reading a string of bits: little and big-endian. In little-endian, the byte is read from the least significant bit first. In big-endian, the byte is read from the most significant bit first. There are different advantages and disadvantages to both ways, but it is very important to understand which way a system reads them in order to write and read code and bits. In ARMv8, data can be set to read in both little and big-endian, but instruction fetches are strictly little-endian. Endianness is configured in the last step of setting up the MMU.

4.1.16 Exception Handling

Exceptions are ARM's version of interrupts. There are two types of interrupts: IRQ (interrupt request) and FIQ (fast interrupt request). IRQ's are just regular interrupts in the system. FIQ's however, have higher priority over other interrupts, and cannot be masked by interrupts. FIQ's main functionality is to bring events such as keyboard or mouse inputs to the forefront. Context saves are not required, and only one FIQ at a time is supported.

Both IRQ's and FIQ's are asynchronous. Some synchronous exceptions include: instruction aborts from Memory Management Unit, data aborts, and debug exceptions. (more can be found in the ARMv8 manual).

For synchronous exceptions, the following registers are used:

ESR_ELn gives information on the cause of an exception.

FAR_ELn holds the virtual address where it faulted.

ELR_ELn holds the address for aborting data access.

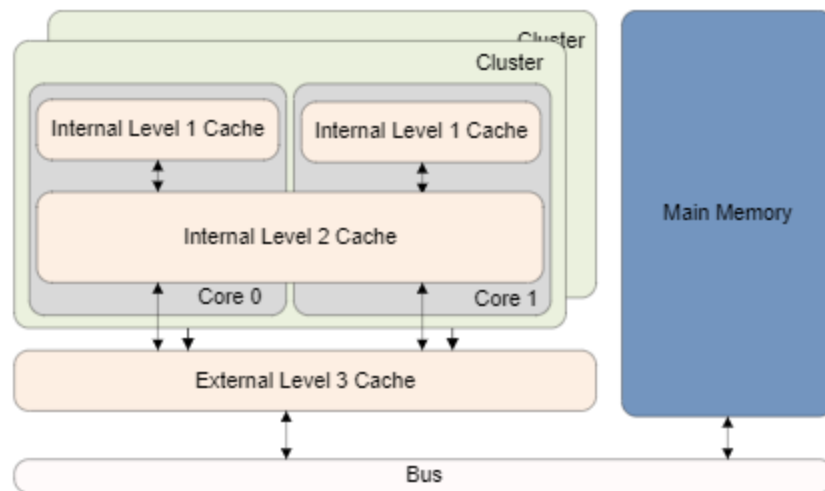
4.1.17 Multi-Core Processors

The Pi 4 has four cores. Each core can be used to run multiple things at once and scheduled and managed through complex algorithms. ARM v8 offers a lot of support for these kinds of processes.

For our sake, we are designing our system to run on only one core for now. In the future, it would be wise to run the system using all cores, but as of right now we have several more pressing features to finish. As of right now however, we are considering multi-core as a possible stretch goal to improve performance.

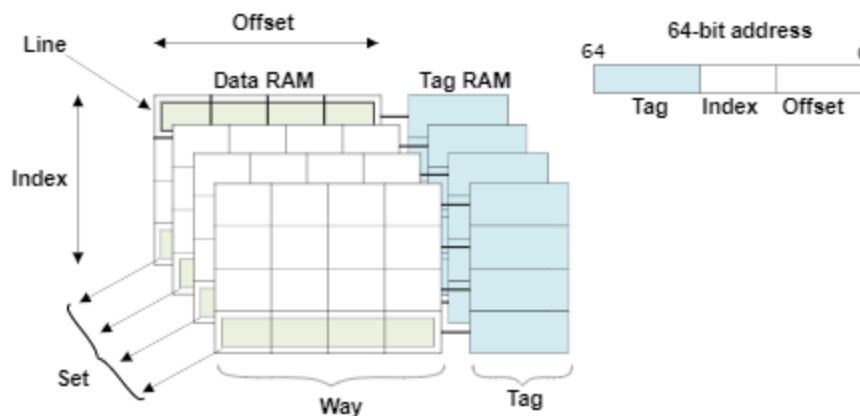
4.1.18 Caches

Figure 11.1. A basic cache arrangement



The cache is a vital part of an OS because it is very quick-access memory. In ARM systems, caches are usually two or more levels. A cache is required to hold three things: address, data information, and status information.

Figure 11.2. Cache terminology



4.1.19 Cache Controller

The cache controller is a piece of hardware that manages cache memory. It is largely invisible to the program, but it automatically caches things from main memory. When the cache finds a piece of data inside of it, it is called a cache look-up. When a cache does not find it, it is a cache miss and goes to external memory (L2) to find it. The core however, does not need to wait for the new data to be stored in the cache to use it. This is because the cache controller passes the critical line to the core. This allows the core to read the data from memory while the cache is saving it in the background.

4.1.20 Cache Policies

Cache policies let us detail what kinds of data gets stored in the cache. The allocation policy are as following:

Write Allocation

Data is loaded into the cache on a write- miss, followed by a write-hit.

Read Allocation

Data is loaded on a read-miss.

The update policies are as followed:

Write-Back (WB)

Write only occurs on the cache and it is marked as dirty. External memory only updated when evicted or cleaned.

Write-Through (WT)

Write updates on both the cache and external memory.

Specific Arm instructions regarding policies are as followed:

PRFM <prfop>, addr : prefetches memory for the cache. <Prfop> is broken down into type (PLD for prefetch for load, PST for prefetch for store), target (L1, L2, or L3), and policy (KEEP for retaining in the cache, STRM for single-use memory that is only used once).

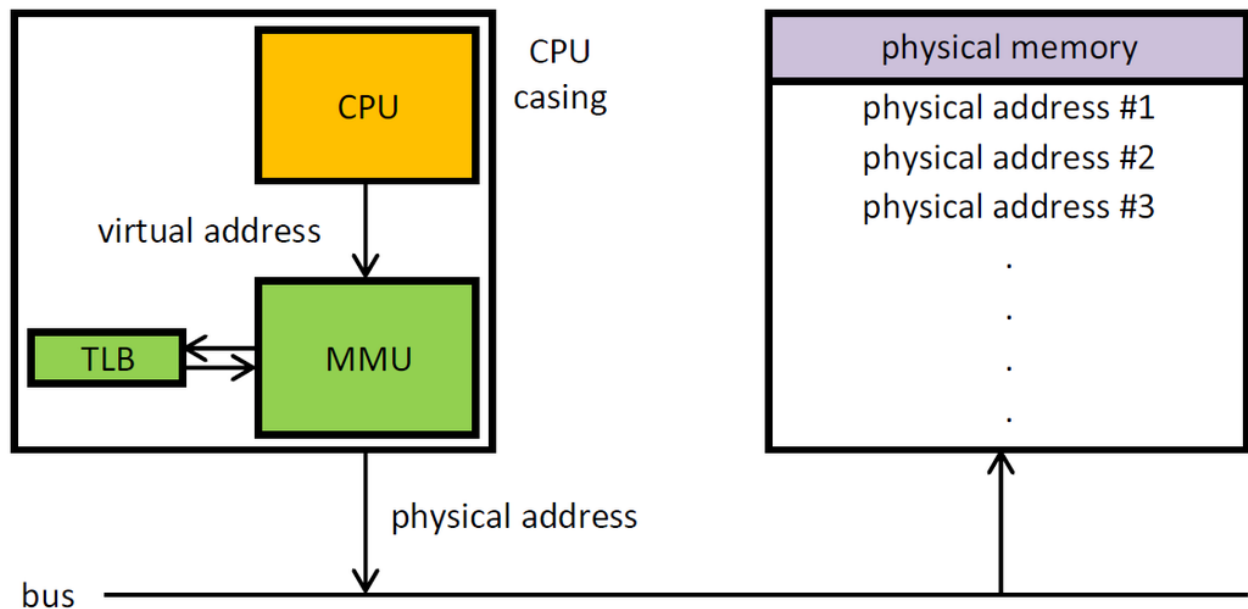
4.1.21 Cache Maintenance

Cache maintenance is required to remove stale data or update MMU changes. Here is a list of ARM commands to maintain the cache:

Cache	Operation	Description
	CISW	Clean and invalidate by Set/Way
	CIVAC	Clean and Invalidate by Virtual Address to Point of Coherency
	CSW	Clean by Set/Way
DC	CVAC	Clean by Virtual Address to Point of Coherency
	CVAU	Clean by Virtual Address to Point of Unification
	ISW	Invalidate by Set/Way
	IVAC	Invalidate by Virtual Address, to Point of Coherency
DC	ZVA	Cache zero by Virtual Address
	IALLUIS	Invalidate all, to Point of Unification, Inner Shareable
IC	IALLU	Invalidate all, to Point of Unification, Inner Shareable
	IVAU	Invalidate by Virtual Address to Point of Unification

4.1.22 Memory Management Unit

The MMU is an actual physical piece of hardware. It is vital, because it translates virtual addresses to physical addresses. It also checks access permissions and gives us control over cacheable memory.



CPU: Central Processing Unit

MMU: Memory Management Unit

TLB: Translation lookaside buffer

Source:

https://en.wikipedia.org/wiki/Memory_management_unit#cite_note-TanenMOS-1

What does all of that mean? Let us break it down.

4.1.23 Virtual to Physical

The actual hardware reads physical addresses. At first glance, it may seem easier for us to read physical addresses as well. The problem that arises if you do that is that you end up wasting a lot of space. For example, if you have 32-bit addresses that you want to store in 4G, you would actually need, in the worst case, 16G to hold 4G worth of physical addresses. As such, it is vital to translate addresses to virtual addresses, and the MMU facilitates this process.

4.1.24 Access Permissions

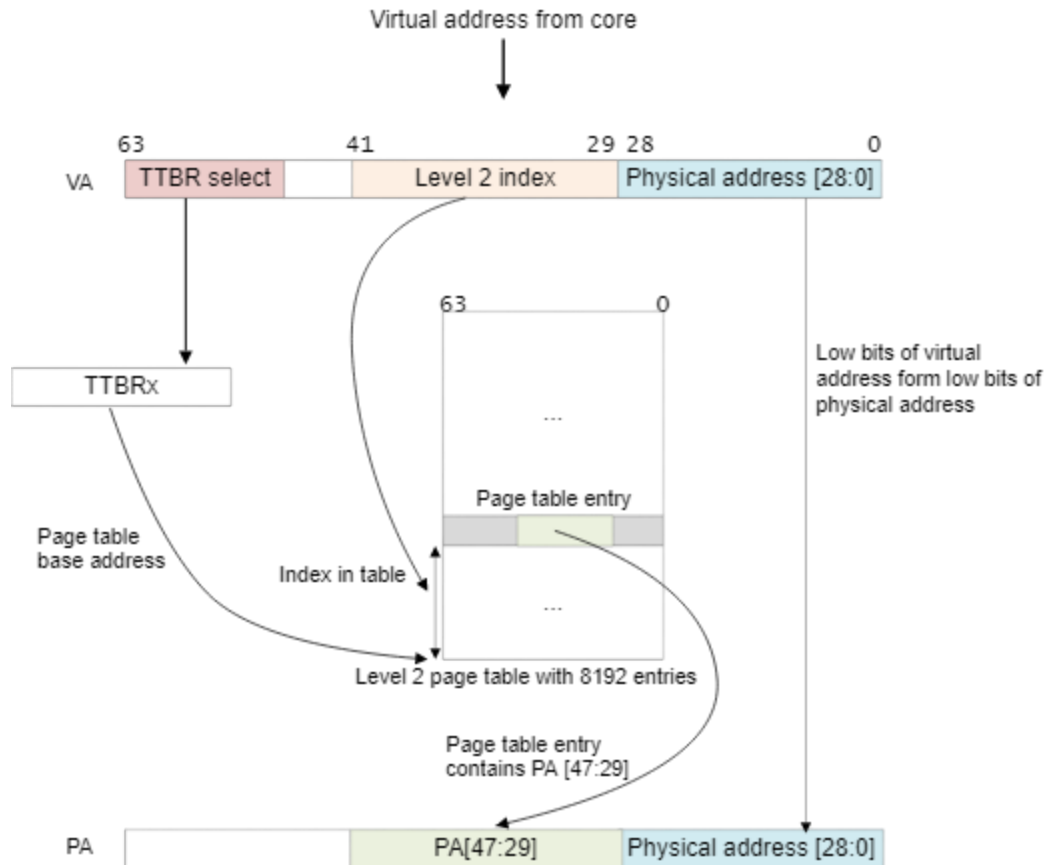
Every program cannot have full reign over all the memory. For instance, a text editor should not be able to access personal information in another window. This is where access permissions come in. If a program tries to access somewhere it is not allowed then the MMU aborts. This, however, will not be fatal. For example, this feature is used to map virtual machines to registers on the hardware they are actually on. When it makes access, it is denied but then given the right address. This can also be used to extend RAM when it runs out for a program because the nature of virtual addresses allows this.

4.1.25 Cacheable Memory

The MMU lets you decide whether or not you want to cache a certain memory. For example, you would not want to cache a peripheral, because if you cache it's value and use that instead of the actual value your program will probably crash or receive several errors (ex: caching a timer). A general rule of thumb is to only make things cacheable if they can only be written to through the cache. That way, you never run into a problem of reading the wrong value.

4.1.26 Translating a Virtual Address to a Physical Address

In AArch64, bits 63-47 must be either all 1s or all 0s. This is the base address and will trigger an error if not done correctly. Bits 41-29 map to a page table entry, which maps to the actual physical address. If the address is valid, it will allow memory access. Bits 28-0 give an offset within the section to combine with the physical address.



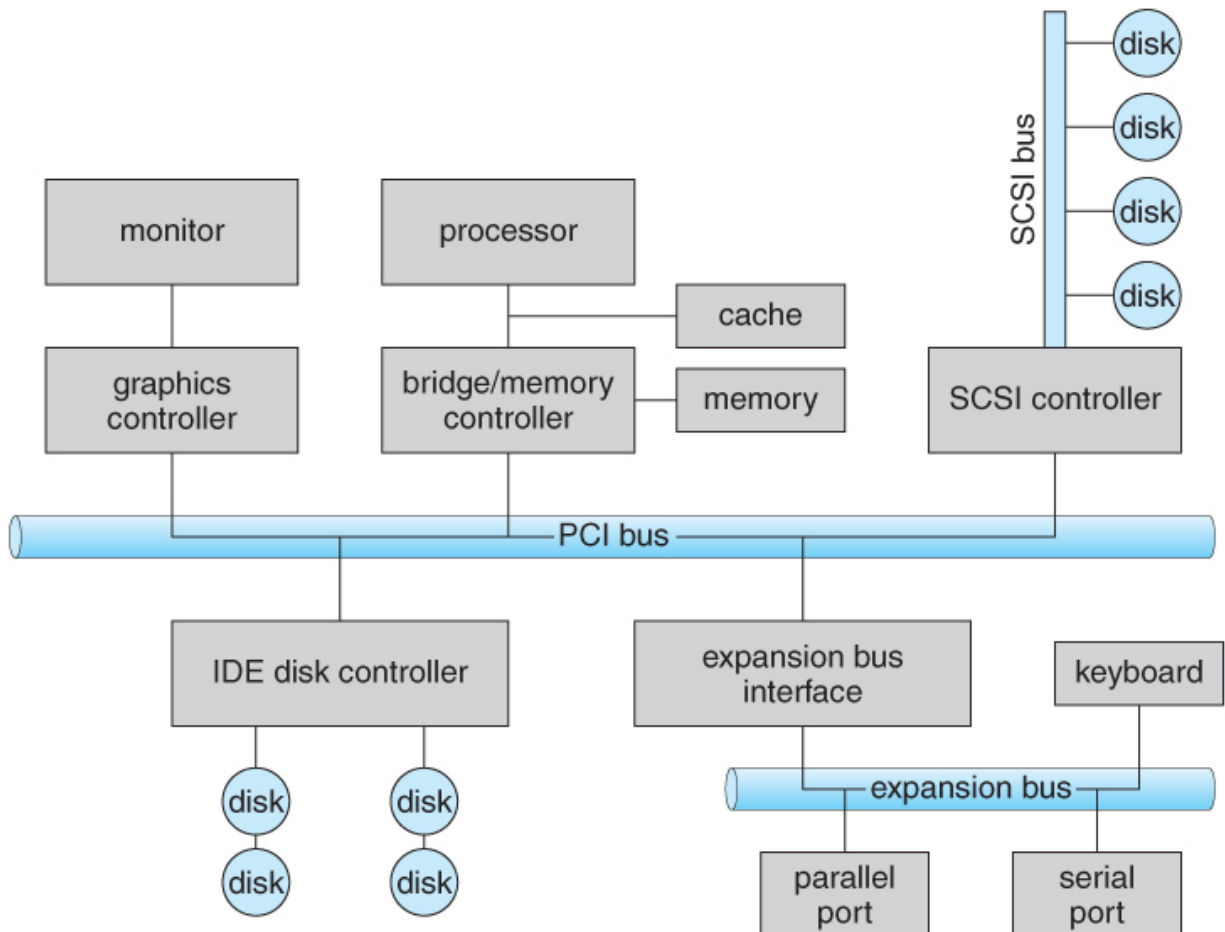
4.1.27 Configuring the MMU

The MMU in the Pi 4 is very similar to the Pi 3, except for a bigger address space. The memory layout begins a bit odd, the first gigabyte of physical RAM you have \$gpu_mem and \$arm_mem that is used by firmware.

Gpu_mem can be configured from config.txt. Basically, this is splitting the memory between the GPU and the CPU. The minimum value is 16 but the default is 64. Since the GPU has its own MMU, gpu_mem can be adjusted to lower values if more space is needed in the CPU.

The rest of the MMU needs to be set up by level. EL0 is where the user interface should go. The kernel usually runs on EL1 or EL2. For a small system like ours, it would make more sense to keep the kernel in EL1. EL3 is also available, but is usually set aside for trusted firmware.

4.2 I/O Manager



Source:

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html

A simple way of setting up an I/O Manager is to have registers that the CPU checks every instruction. This is broken down into four registers:

Data-In: Read for input

Data-Out: Write for output

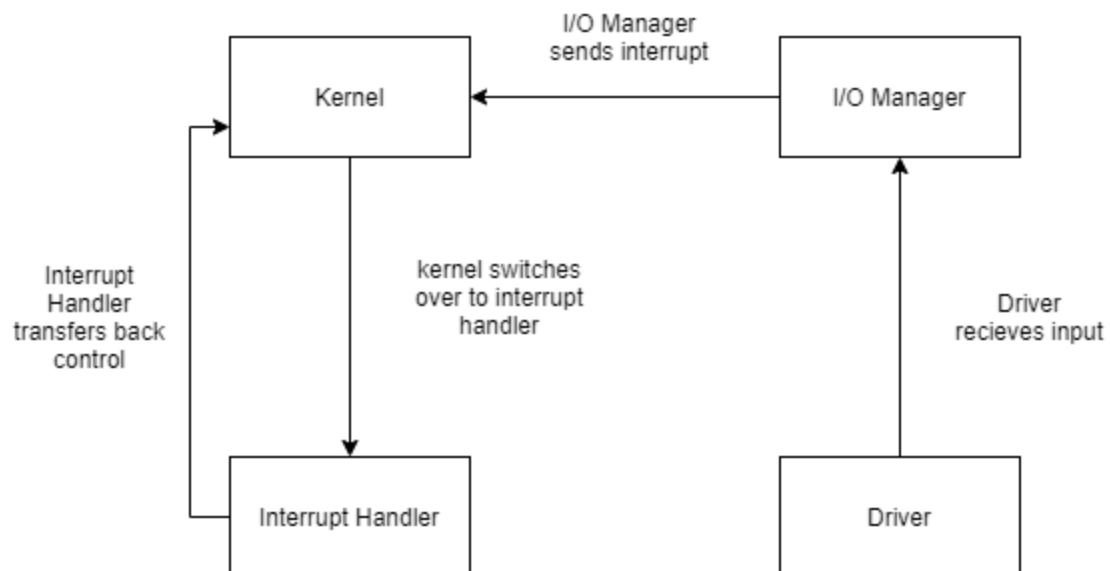
Status: Read for status of devices (idle, ready, busy, error, transaction complete)

Control: has bits to issue commands or change settings (parity checking, word length, etc)

Another way is through memory-mapped I/O. This means portioning out some address space to map to the device. Communication occurs from reading/writing from those areas. This is mostly useful for devices with tons of data to process (ex. A graphics card) and not for low input devices. With this method, however, you will also need to establish access permissions to prevent processes from overwriting it on accident.

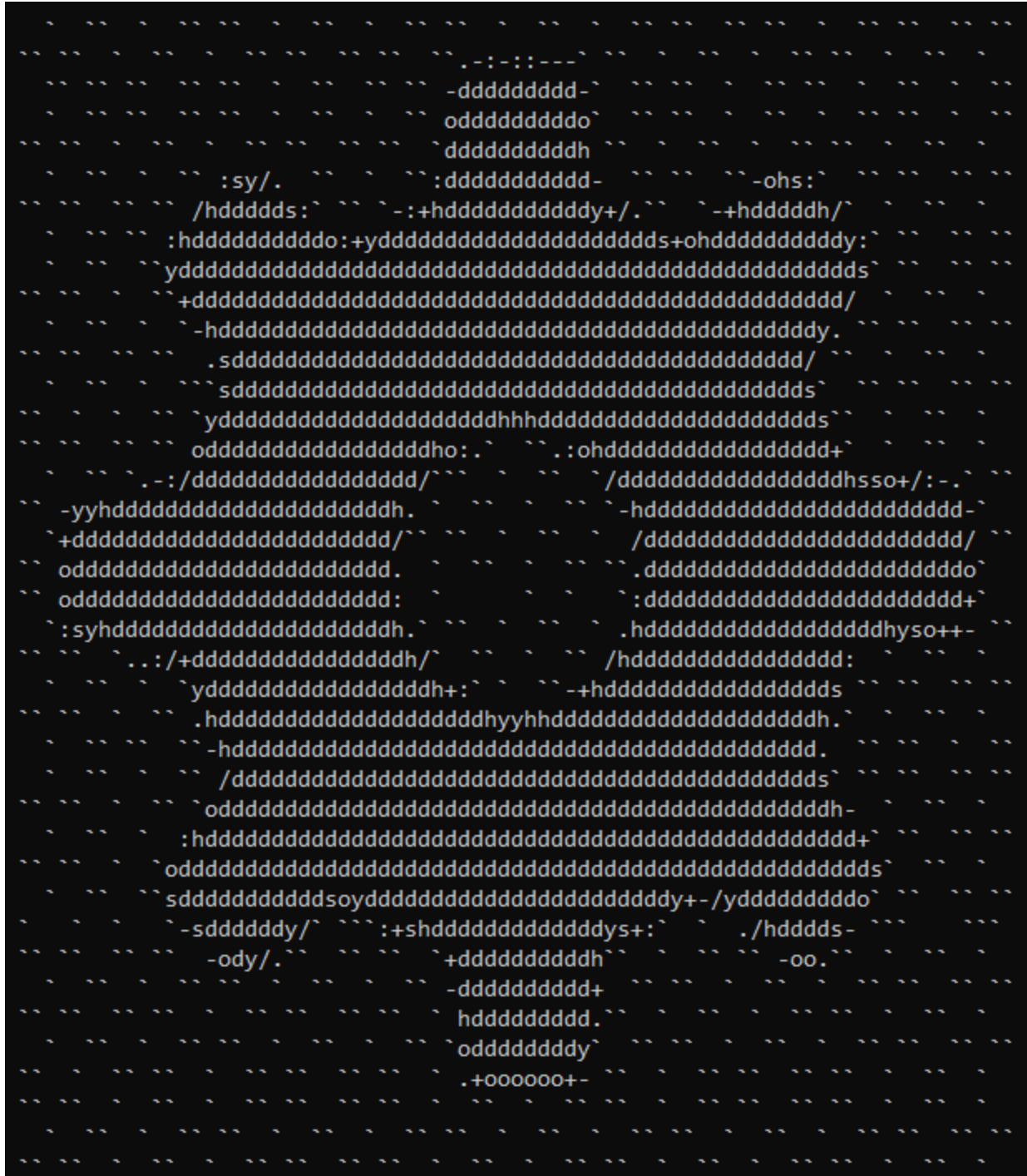
In general, there seem to be these two lines of thinking: constantly check registers or set the drivers up like a process. For our case, it would be more organized to encapsulate all the drivers as processes in an I/O Manager that sends interrupts to the kernel. Any driver, such as a keyboard or a mouse, would send their interrupts to the I/O Manager (as FIQ's), which would organize and send those values to the kernel.

Encapsulating the I/O Manager is better than the registered approach for a couple of reasons: for one, it allows for us to add and remove drivers without having to interact directly with the kernel. Second, it allows the kernel to operate without stopping every instruction to check a register. Third, it creates a single interface that we can update and use to communicate with possible deprecated drivers.



PegasOS

5. System



5.1 System Directories

5.1.1 PegasOS Root Directory

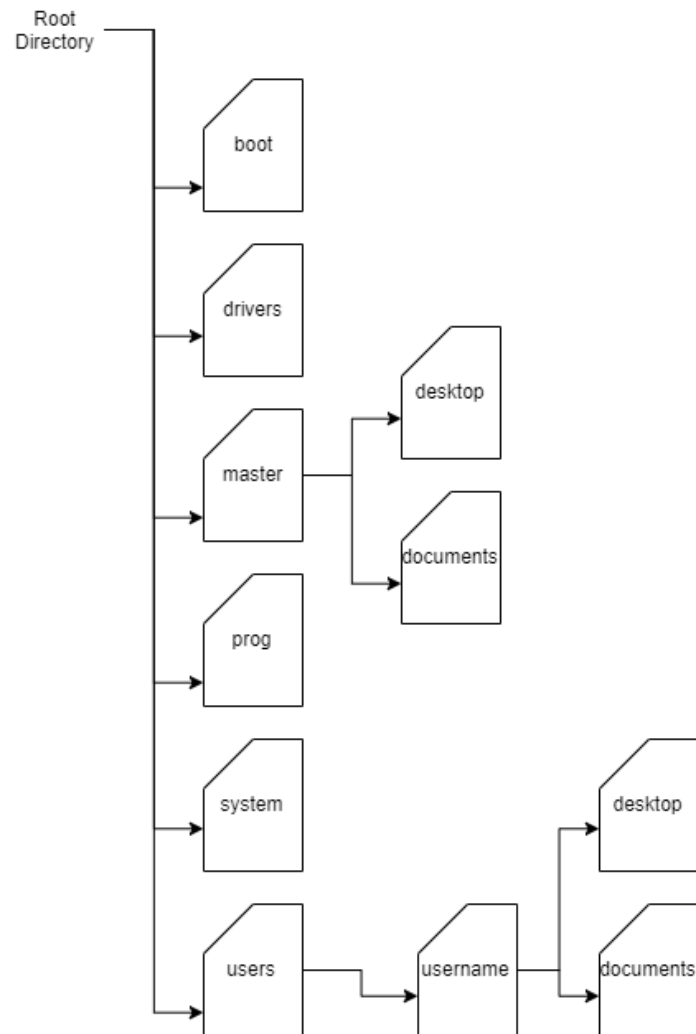
PegasOS will follow the general layout of the Linux systems, with the system's core at the lowest possible root of the file system and additional directories stemming from there. The system will have a minimal root directory that will contain all of the system's files necessary for operation. Included in this is a directory for users of the system, which will allow for multiple users to have their own 'desktops' and files that will be accessible only by them and the system admin or system master.

The root directory of PegasOS will only be able to be modified by the kernel itself - which should not be modifying the structure of the directories - or by the system master. Because the system master will be able to modify this root directory, it is possible that the directory can be tampered with in such a way that the system will no longer be able to function correctly. Should this happen, the system will need to be reinstalled.

The purpose of the root directory is to keep all system-related data as close to the system as possible, which means it will be at the lowest level of the system such that it does not interfere with the user's use of the system, nor can it be accessed by the average user who uses the system. In this way, the workings of the system are protected from accidental tampering, however, it is still accessible to those who wish to have access to it with the caveat that only the system master can modify the root directory. Normal users may still see what files are in these directories.

The structure and contents of the root directory may change over time, however, the aim is to keep the root directory simplistic such that it only contains what is necessary. This will allow us to keep the system's usage of memory to a minimum, and allow the user to use as much remaining space on the SD card as possible. As an added bonus, the less memory that the system takes up, the more it can fit into RAM to keep the system as fast as possible.

5.1.2 PegasOS Root Directory Overview



Boot

This directory will contain all of the necessary files for boot-up, apart from those contained within the EEPROM already on the Raspberry Pi. This will primarily be made up of .elf files that set up the rest of the system for operation. This directory may contain other files or information related to the boot-up of the system.

Drivers

This directory will contain any hardware drivers that the system will use to interact with various pieces of hardware, such as keyboards, speakers, monitors, etc.

Master

This is the admin or system master's directory, which will reflect that of a typical user's directories. The key difference is that the system master will be able to add, modify, and remove files from any location on the system - essentially giving them kernel-level access, much like Linux's root user.

Prog

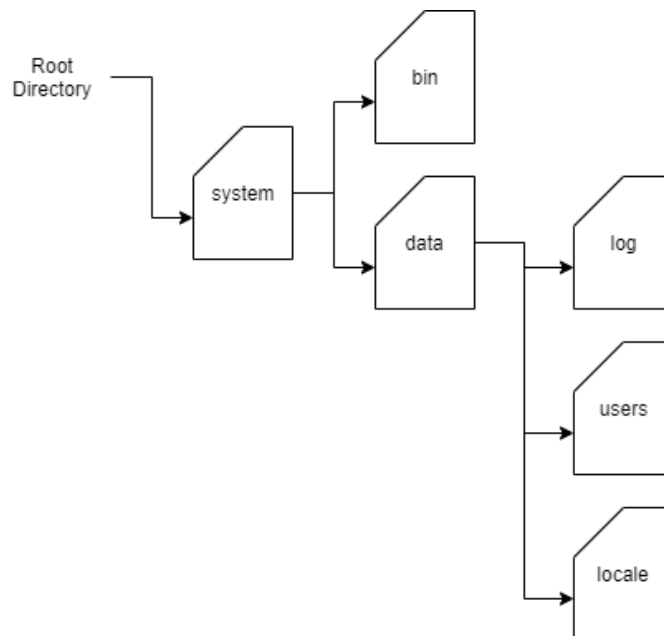
This directory will contain all binaries for programs that can be executed by the system shell.

Users

This directory will contain a sub-directory for every user that is using the system. Each user directory will contain its own 'documents' and 'desktop' directories, to offer the users some built-in categorization should they choose to use it. While there is no graphical interface, the 'desktop' directory would be equivalent to the directory that is displayed on the main screen of the operating system.

System

This directory will contain all binaries and data for system programs, calls, and services. Respectively, the binaries will be contained within the 'bin' subdirectory and all data will be contained within the 'data' subdirectory. Within the 'data' subdirectory, there will be three further subdirectories that further categorize the data contained within. The 'users' subdirectory will contain a user registry file for every user registered with the system. The 'log' subdirectory will contain all system logs and information that the system may need to refer to or record. The 'locale' subdirectory will contain all system localization files needed for translations of system dialogue.



5.2 System Set-Up

When the system is initially installed, some basic actions will need to be performed in order to set up the system for the first time. This section will detail each step of the initial set up, and what each step impacts on the system itself.

5.2.1 System Language

First, the user will need to choose what language they wish to perform set-up in, as well as what language they wish for the operating system to be installed with. For the first release of this software, there is no guarantee that there will be translations aside from the primary English translation. If you would like to contribute a translation of your own to the project, please see the *System Localization* section for more information on how our localization files are formatted.

5.2.2 System Location

The user may choose to enter the country they are residing in, though this will not be a mandatory field. In future releases of the system, if the system has internet connectivity then this setting may be overridden by data provided through the internet connection.

5.2.3 System Keyboard Layout

This is included for the sake of completeness, though for the initial release of the system this will most likely not be customizable. Initially, the system will be using ASCII as the primary character set. For future releases of the system, we plan on supporting both ASCII and Unicode Standards. For more information on ASCII or Unicode, please follow the respective links:

<http://www.asciitable.com/>

<http://unicode.org/standard/standard.html>

This will ensure that the system can be used by a wide variety of users, despite a lack of international support in the early life of the system. We will do our best to accommodate all keyboards and character sets, however, this is secondary to making sure that the system functions on its initial release. For this reason, we will be guaranteeing that the system will work with ASCII by default, and at a bare minimum. As more character sets are supported by the system, these options will become

available to the user and the information here will be updated to reflect these changes. The first Unicode character set that will be supported will most likely be en_us.utf-8.

5.2.4 System Date and Time

Next, the user will need to choose their date and time zone to set their system clock appropriately. The system will automatically account for Daylight Savings Time, based on the date entered in this field. In the future, if the system has internet connectivity then these settings will be overridden by data provided through the internet connection. The correct date and time settings are important for logging information generated by system diagnostics and testing, as well as for additional information to include file creation, modification, and where necessary, deletion.

5.2.5 Master Password

Next, the user will need to provide a new password for the master user on the computer. By default, the master password is 'pass' for simplicity. Obviously, this is not a strong password so we recommend that the primary user of the system change the master password to something more secure. Please see our password guidelines if you would like tips on creating a strong password.

5.2.6 Create New User

Finally, the user installing the system will create a new user registry entry of their choosing. This user registry will ideally be for the primary user of the system. During the set-up of this registry, the user will be able to enable master permissions for that user account. For more information on this process, please read the *Log-In Menu* subsection in the *User Subsystem* section.

5.3 System Localization

While PegasOS is at its basic level a student operating systems project, there is still potential for the operating system to be used by a much wider audience than what is anticipated. For this reason, we will be including international localization support to the best of our ability. Localization is an inherently difficult task, as it requires careful future-proofing of the system so that it is not rewritten every time an additional language is added to the system's repertoire, as well as correct translations such that users understand what the system is asking of them.

5.3.1 International Localization

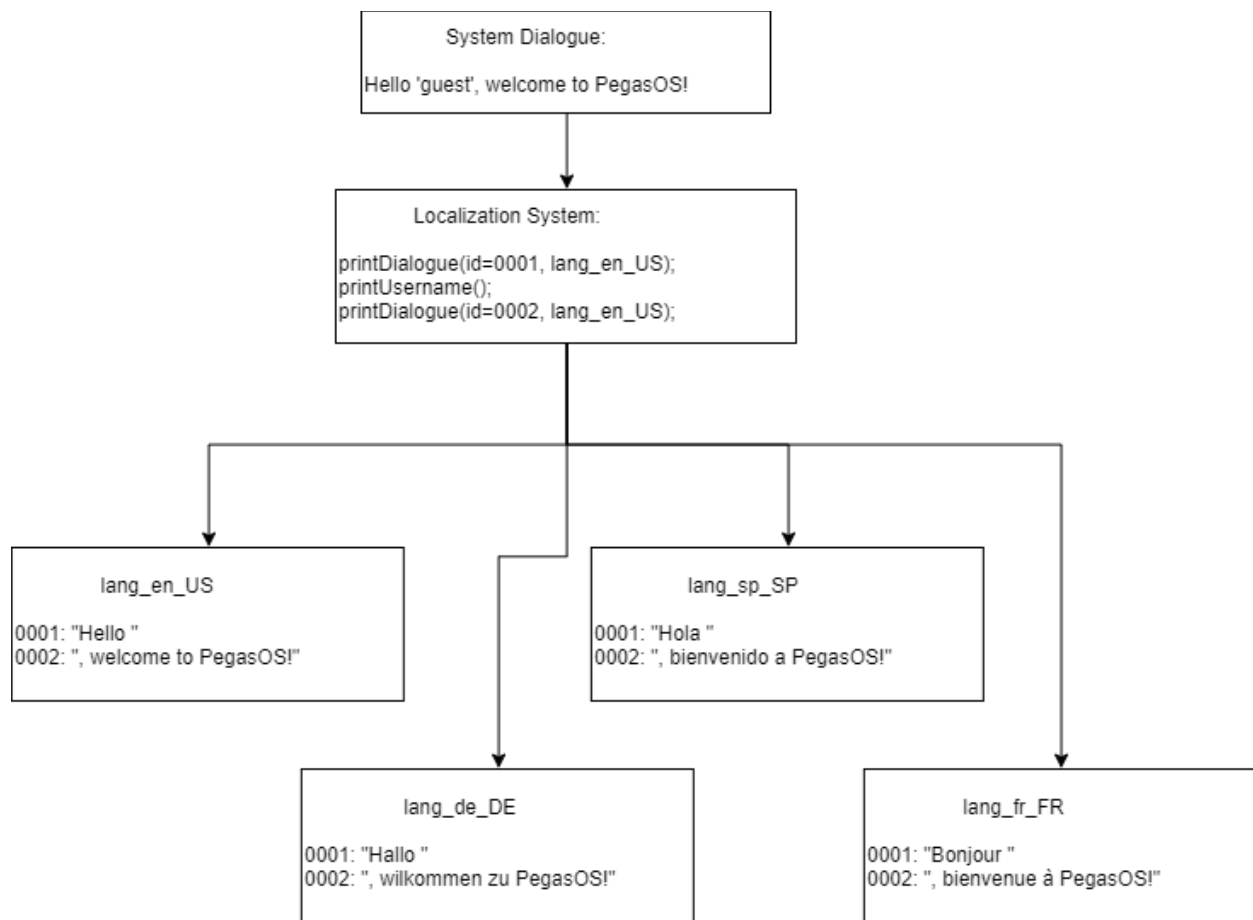
By localization, we, of course, mean that PegasOS will support the ability to be translated into many different languages in an effort to make the system as usable as possible for all users. These translations will be stored into various localization files in the system directory, and the appropriate translations will be displayed based on the user's language settings. We are not language experts, nor are we ignorant of the fact that the task of translating the system will get more difficult as the system expands and more system dialogue is added.

For this reason, we are not expecting to have fully-realized translations for the initial release of the system, apart from the default English translation. Despite this, we are optimistic that through future-proofing our localization system, future translations of the system will be relatively seamless in their installation. If you would like to contribute towards localization of PegasOS, you may apply to be a contributor to the project on GitHub, and submit your localization file there.

5.3.2 Localization File Standards

Localization files will be contained in a separate directory (as specified in the *System Directory* section), wherein the appropriate file will be read from when menu dialogue must be printed to the screen. To ensure that all localization files can be read by future versions of PegasOS, we will establish a file standard that all localization files must follow in order to be read successfully by the system.

The system will follow the guidelines as described in the following diagram. Please note that the code displayed is an approximation of what will be in the final system.



5.3.3 Loading Translated Text

With this file formatting, it will make loading the actual text for display in the system extremely easy. We will perform a simple lookup in the appropriate file based on the language the user has chosen and read in the string according to the dialogueID. After this string has been read in, we simply print it to the screen. The most complicated part will be ensuring that no dialogueIDs are reused between entries in the localization files, however, this should be simple if the main English translation has already been completed.

An important note about PegasOS and displaying text in other languages is that the system will be supporting ASCII initially, with plans to expand into Unicode at a later date. This may affect how localizations are displayed in the system until this problem can be addressed and Unicode is fully supported.

5.3.4 Localization Support

Through our localization system and future-proofing of the system, we hope to support as many translations of the system as possible. To do this, any translations that are submitted need to be checked to ensure that the translation is as accurate as possible to the original English that the system dialogue is written in. As a small team, it will take time to check the work of new localizations that are not made 'in-house', and as such we ask for your patience when reviewing 'out-of-house' localization files.

5.4 System Users

PegasOS will allow for multiple users to be registered at once, granting them unique directories to store their data for general use. Each user is given a 'desktop' directory and a 'documents' directory, as well as permission to create and remove subdirectories within their user-space. Users will be allowed to use most programs on the system, except those reserved for system-level use. The user will be able to execute these programs and commands through the terminal shell. The list of minimum user functions includes:

- Open and Close Files
- Create and Destroy Files
- Create and Destroy Directories
- Execute Shell Programs
- Read System Diagnostics

By default, the 'guest' user will be signed in if no users have set up the system. At a minimum, there will always be the 'master' user and the 'guest' user on the system.

5.4.1 User Registry File

User data will be contained within a registry file, which will be formatted as follows:

<struct> UserRegistry
unsigned char username[32] unsigned char pass[32] int usercolor int dircolor int superP

These files are read-only in most circumstances. If the user wishes to change their username, they must first confirm with their password that they wish to change their username, which will allow them to change the username hash by entering in their new desired username. The password is changed in the same way, however, the hash is salted. The values 'usercolor' and 'dircolor' can be changed through the user's Details menu, and do not require a password re-entry. The super-user flag, 'superP' can only be changed with the master's permission, which requires a master's password to grant.

5.4.2 Usernames

Usernames are tags that associate a registry file and specific info to a specific user. They are typically memorable names that a user has associated with themselves, and is used to log-in to a service or computer. PegasOS allows users to choose a username of their own choosing and is not tied to their real name unless they choose to. Usernames may be no longer than 24 characters. This limit may be increased at a later date and is mainly set for storage reasons. Apart from the length of the username, no other limits are placed upon the username of the user. This means that we do not filter the usernames by appropriate language, phrases, or symbols, and leave that up to the discretion of the user.

5.4.3 Username Storage

User's usernames are stored on the machine in the user registry file as a 256-bit hash, as an array of 32 chars with position 0 of the array representing the highest 8 bits of the hash. This is to ensure extra security for the user and to help prevent cracking passwords associated with specific usernames.

5.4.4 User Passwords

User passwords are what allow the user to identify themselves as the correct user of an account. Passwords may be no longer than 64 characters. This limit may be increased at a later date and is mainly set for storage reasons. Apart from the length of the password, no other limits are placed upon the password of the user. However, we do suggest that you use the following guidelines to create a secure password.

1. Use spaces in your password.

This is not always allowed for various reasons, however, it greatly increases the security of your password. For one, if you choose a password with a number of words, the spaces can clearly separate the words which make it easier to remember. Consider the following:

```
mysecurepassword
my secure password
```

This also increases the total length of your password, making it harder to crack. You may also choose to do something like the following:

my sec ure passwor d

By inserting random spaces into the password, it further jumbles it up, making it harder to crack. Be warned though, that making too many small sections within the password may introduce vulnerabilities in cracking algorithms that consider spaces. There is a fine balance between too many spaces and the right amount.

2. Use odd symbols in your password.

Perhaps the most commonly used symbols for most uses are '!', '@', '#', and '?'. While this will increase the security of your password by expanding it past basic numbers and letters, using these in place of normal letters is not ideal.

p@ssword

It is better to instead add less-frequently used symbols, or instead of swapping out letters for symbols to simply add the symbols to the word or phrase.

pa@ssword

It is *trivial* to make a program that simply swaps out letters in words for similar-looking symbols. At the very least, changing it to be a less-frequently used symbol will get past lesser programs, and take longer in others.

3. Don't only replace letters with similar-looking numbers.

Everyone has seen passwords that look like the following:

Passw0rd

These types of passwords are *trivial* to crack on their own. It is much more secure to simply insert numbers between letters rather than replacing letters with similar numbers. Consider the following:

passwor9d 12

This is much more secure than simply swapping letters with numbers.

4. Keep the length of your password to a minimum of 16 characters.

While PegasOS does support a password of any length up to 64 characters (you may even use no password for a user, such as with the guest user), to increase the security of your account we suggest using at least 16 characters in your password. This greatly increases the amount of time needed to crack the given password, at a minimal cost of remembering a longer password.

5. Do **NOT** use common words.

If nothing is taken away from these guidelines, please at least take this away.

If you use 'password123' as your password, your account has been compromised. Full stop. Do not use password in your password. Do not use your username in your password (at the very least, modify your username if you're using it as your password too). Do not use common phrases, such as 'iloveyou' or 'breakaleg'. Do not use the name of the program you are using, such as 'adobe123' or 'photoshop'.

6. Use multiple, unrelated words.

This is the easiest way to make your password more secure, without having to do anything special. Consider the following:

chicken trash blanket

These words are unrelated, yet make a memorable phrase that sticks out in the mind. In addition to this, it's 21 characters long! That's more than enough to make it secure in most brute-force cracking algorithms. And in a

dictionary attack or otherwise more sophisticated cracking algorithm, due to the words being unrelated it will take much longer than a phrase like the following:

chicken wing grill time

If you'd like to test out some sample passwords according to these guidelines, check out the following website:

<https://howsecureismypassword.net/>

Obviously, **don't enter your real passwords** into this site. While they do say that passwords entered into the website are not stored and sent out on the internet, you can never be too careful.

5.4.5 User Password Storage

Of course, a password is only as secure as how it is stored. For this reason, PegasOS will store user passwords on the machine as a salted and hashed password. For the sake of transparency, this process will be as follows:

1. Salt is applied to the password

This salt will come from the username, which will allow for users to choose the same password and result in different hashes for the password, keeping both users more secure and accounting for edge cases in which multiple users unknowingly pick the same password.

2. The password is sent into a hashing algorithm.

After the salt is applied to the password, it is bundled up and sent into a one-way hashing algorithm.

3. The resulting hash value is stored in the user's registry file.

The hash will be stored as a 256-bit hash, as an array of 32 chars, where position 0 in the char array represents the highest 8 bits of the hash.

5.4.6 User Permissions

Depending on the ‘type’ of user permissions that the user has, certain actions within PegasOS cannot be performed. User permissions can be of two types: system administrator (master) or standard. Master users may make any changes to the system as they see fit, much like a root user or sudoer can in Linux. Standard users cannot make these changes, but with the master account’s permission they may upgrade their permissions to that of a master user.

The main difference is that master users have full control over the system, and are able to create files and directories wherever they please. They are also allowed to modify, and remove files and directories wherever they please. **This means that great care must be taken when working in the root directory of the system, as it is possible to irreversibly damage the system.** For the average user, this will not be a concern, as most system files that the average user can access are read-only, or otherwise untouchable.

Full control over the system includes functions such as, but not limited to:

- Changing the system clock
- Terminating system processes
- Modifying the system directory (root)
- Modifying system binaries
- Changing user settings
- Deleting and creating users
- Viewing system logs

5.5 User Subsystem

The user subsystem is the system which handles user log-in, creation, deletion, and handles the display and changing of user info. The following describes the various menus within the PegasOS User subsystem, which is made up of two key systems. The first of these being the log-in system, which allows existing users to log-in as well as allow new users to create a new user account. This log-in system also allows the user to see a list of currently enrolled users on the system. The second being the user-menu system, which allows the logged-in user to adjust some of their account settings, preferences, and permissions. This section will also serve as a guide to how to use these systems, and detail all of the sub-menus within both services.

The system will have no hard limit on the number of users on that particular install, however if a user account is no longer needed, please be sure to remove that user account. As each user account is stored in a separate user registry file, and a new subdirectory is created for each user, this can quickly add up in the amount of space used on the SD Card, restricting space for other uses of the system.

Since user accounts are the primary method in which the system is interacted with, it is necessary for there to at least be one user account on the system at all times. While the master account can perform almost any operation on the system, they will not be allowed to remove themselves from the system. In addition to this, every system install will come with two premade user accounts, those being the 'guest' account and the aforementioned 'master' account. This will make logging in to the system easy for quick system setup and modification, and also allow for non-permanent users of the system to still use the system and be able to access all of the system's features.

While the majority of the user's functionality is used through the system's Shell component and the included programs - creating users, switching users, and logging users in and out is done through the aforementioned log-in and user menus.

5.5.1 Log-In Menu

To access the log-in subsystem, use the 'login' command in the shell. This will bring up the following menu.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: _
```

From here, the user presses the key indicated in the brackets to access the corresponding menu option. The selection does not need to be capitalized, only the key needs to be pressed. If an invalid selection is given, the following prompt will be displayed and the menu will return.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: l  
  
Invalid selection. Please choose a valid selection.  
  
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection:
```


Existing User:

To log-in as an existing user, press the 'E' key which will then bring up the following prompt.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: e  
  
Enter username:
```

If the username entered does not match a user account currently in the registry, the following prompt will appear. Otherwise, the system will continue and ask for that user's password.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: e  
  
Enter username: pegasOS  
User 'pegasOS' does not exist. Would you like to create this user? [Y/N]  
Selection: _
```

If 'Y' or 'yes' is selected, then the user will be directed through the same prompts as the New User command (see New User for more information). If 'N' or 'no' is selected, then the user will be returned to the Log-In Menu with the previous user being logged in.

```
=== Log-In Menu ===
```

```
[E]xisting User
```

```
[N]ew User
```

```
[R]egistry
```

```
[Q]uit
```

```
Selection: e
```

```
Enter username: pegasOS
```

```
User 'pegasOS' does not exist. Would you like to create this user? [Y/N]
```

```
Selection: n
```

```
User 'pegasOS' not created.
```

If the username entered matches a user account in the registry, then the following prompt will appear.

```
=== Log-In Menu ===
```

```
[E]xisting User
```

```
[N]ew User
```

```
[R]egistry
```

```
[Q]uit
```

```
Selection: e
```

```
Enter username: sample_user
```

```
Enter password: _
```

Upon the password being entered, it will be checked alongside the system's copy of the user's password. If there is a match, then the user will be logged in and the following confirmation will be displayed. Then the User Menu will be displayed.

```
=== Log-In Menu ===
[E]xisting User
[N]ew User
[R]egistry
[Q]uit

Selection: e

Enter username: sample_user
Enter password:
User 'sample_user' logged in successfully!

sample_user@root/home/desktop/:
```

If the password does not match, the following error will be displayed. After pressing any key, the user will return to the Log-In Menu.

```
=== Log-In Menu ===
[E]xisting User
[N]ew User
[R]egistry
[Q]uit

Selection: e

Enter username: sample_user
Enter password:
Password does not match. Log-in attempt failed.
```

New User:

To create a new user, press the 'N' key which will then bring up the following prompt.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: n  
  
Enter your username:
```

Upon entering the username, the following prompts will appear.

```
=== Log-In Menu ===  
[E]xisting User  
[N]ew User  
[R]egistry  
[Q]uit  
  
Selection: n  
  
Enter your username: sample_user  
Enter your password:  
Confirm your password:
```

If the password entered between the two password prompts does not match, the following error will be displayed.

```
Enter your username: sample_user  
Enter your password:  
Confirm your password:  
Passwords do not match. Retry password? [Y/N]  
Selection: _
```

If 'Y' or 'yes' is selected, then the user will be given an additional attempt to create their password. If this attempt is failed as well, the user will not be created and they will return to the log-in screen.

```
Passwords do not match. Retry password? [Y/N]
Selection: y

Enter your password:
Confirm your password:
Password does not match. Failed to create user 'sample_user'.
```

If 'N' or 'no' is selected, then the user will return to the Log-In Menu.

If the password entered between the two password prompts does match, the following confirmation will be displayed, the system's user registry file will be updated, and the user will be logged in.

```
=== Log-In Menu ===
[E]xisting User
[N]ew User
[R]egistry
[Q]uit

Selection: n

Enter your username: sample_user
Enter your password:
Confirm your password:
New user 'sample_user' created!

sample_user@root/home/desktop/:
```

Registry:

To view the list of currently registered users, press the 'R' key and the system will print out the list of all users in the system. At a minimum this will consist of the system master and a guest user.

```
=== Log-In Menu ===
[E]xisting User
[N]ew User
[R]egistry
[Q]uit

Selection: r

=== User Registry ===
master
guest

=== Log-In Menu ===
[E]xisting User
[N]ew User
[R]egistry
[Q]uit

Selection: _
```

Quit:

To return to the terminal shell, press the 'Q' key and the system will terminate the Log-In subsystem. If no user was logged in, then the system will power down.

5.5.2 User Menu

To access the user menu subsystem, enter “user” from the shell and the following menu will appear.

```
guest@root/home/desktop/: user

=== guest's Menu ===
[C]hange User Details
[L]ogout
[S]witch User
[Q]uit

Selection:
```

Change User Details:

To change the user's details, press the 'C' key and the following sub-menu will open.

```
=== guest's Menu ===
[C]hange User Details
[L]ogout
[S]witch User
[Q]uit

Selection: c

=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: █
```

Selecting 'P' or 'U' for either Password Change or Username Change respectively will require the user to re-enter their password to confirm it is a change they wish to make.

```
=== guest's Details ===  
[P]assword Change  
[U]sername Change  
[A]ccess Permissions  
[B]ack  
  
Selection: p  
Enter old password: █
```

Note: Passwords will NOT be shown as they are typed.

After their password has been confirmed, if the user selects to change their password, the following prompt will appear.

```
=== guest's Details ===  
[P]assword Change  
[U]sername Change  
[A]ccess Permissions  
[B]ack  
  
Selection: p  
Enter old password:  
Enter new password:  
Confirm new password:  
New password set!
```

Upon entering the new password and confirming the new password by re-entering it, the registry file will be updated to reflect the new password, and the following confirmation will be printed to the screen.

If the user selects to change their username, the following prompts will appear.

```
=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: u
Enter password:
Enter new username: newguest
Confirm new username: newguest
New username set!

=== newguest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: █
```

The user will be required to enter the new username and confirm it. Once the username has been confirmed, the user's username will be set to the new username, and a confirmation message will be printed to the screen.

If the username does not match between confirmations, the username will not be set and the user will be returned to the User Details menu.

```
=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: u
Enter password:
Enter new username: newguest
Confirm new username: new guest
Usernames do not match. New username not set.

=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: █
```

If the new password does not match between entries, an error will be reported to the user and they will be returned to the User Details menu.

```
=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: p
Enter old password:
Incorrect password entered. Please try again.

=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection:
```

If the 'A' key is pressed for Access Permissions, the user will be required to enter the system master's password.

```
=== guest's Details ===
[P]assword Change
[U]sername Change
[A]ccess Permissions
[B]ack

Selection: a
master Password:
```

If entered correctly, the user will be able to elevate their privilege access to system-level through the following sub-menu. A '*' will indicate which level of permissions the user currently has.

```
=== Permissions Menu ===
[U]ser Level Permissions*
[S]ystem Level Permissions
[B]ack

Selection: _
```

Upon pressing the 'S' key for System Level Permissions, the following prompt will appear. Upon confirmation that the user understands the risks, they will be granted the same privileges as that of the system master. They will then be returned to the Permissions Menu.

```
=== Permissions Menu ===
[U]ser Level Permissions*
[S]ystem Level Permissions
[B]ack

Selection: s
System Level Permissions allows the user full access over the system's files and commands.
Doing so may cause permanent system damage with negligent use of these commands, and as a result,
you may lose all of your system settings and data. This will require a full system reinstallation
and you will not get back the data that you lost.

Are you sure you want to continue? [Y/N]
Selection: _
```

```
=== Permissions Menu ===
[U]ser Level Permissions
[S]ystem Level Permissions*
[B]ack

Selection: █
```

Pressing the 'B' key will return the user to the User Details Menu.

Logout:

Pressing the 'L' key to select Logout will log the current user out of the system. This will display the following prompt, and return the user to the Shell as the system 'guest' user.

```
sample_guest@root/home/desktop/: user

=== sample_guest's Menu ===
[C]hange User Details
[L]ogout
[S]witch User
[Q]uit

Selection: 1
guest@root/home/desktop/: █
```

Switch User:

Pressing the 'S' key to select Switch User will allow the current user to log-out then immediately sign-in a new user. This process is the same as the 'Existing User' command for the Log-In Menu. For more information on this process, please view the guide on using the Log-In Menu.

```
=== guest's Menu ===  
[C]hange User Details  
[L]ogout  
[S]witch User  
[Q]uit  
  
Selection: s  
  
Enter username: █
```

Quit:

Pressing the 'Q' key to select Quit will return the user to the terminal shell. This will not logout the user.

```
=== guest's Menu ===  
[C]hange User Details  
[L]ogout  
[S]witch User  
[Q]uit  
  
Selection: q  
guest@root/home/desktop/:
```

5.6 System Calls

There are several system calls that PegasOS allows processes to use, which are compiled here for your convenience. These calls are split up into categories and organized alphabetically to make their function and usage easily identified, so that if modification of the system is necessary or a user wishes to make programs for the system, they understand what system calls will be best suited to the task. This list may be updated and revised as the need for additional system calls is met and dealt with.

5.6.1 System Diagnostics

*For the time being PegasOS will only support one clock, that being the hardware's clock.

int get_clocktime()

Returns the system's internal clock time in milliseconds.

int set_clocktime(int milliseconds)

Sets the system's internal clock time to start at the given number of milliseconds. Returns 0 if successful, -1 if permission was not granted to change the clock.

int set_clockzone()

Changes the system clock's time zone. Returns 0 if successful, -1 if permission was not granted to change the clock timezone.

5.6.2 Process Creation and Deletion

int p_clone(), p_clone(0), p_clone(1), p_clone(2)

It creates an identical copy to the current process as a child process. The child process uses the same data as the calling process.

If no argument is given or if '0' is passed as the argument, the calling process will resume execution upon the child process' termination.

If '1' is passed as the argument, the calling process is terminated and the child process takes over.

If '2' is passed as the argument, the calling process is terminated and all memory associated with the process is freed.

Returns 0 if the child process was created successfully and has terminated, -1 if the child process could not be created.

int p_destroy()

Immediately terminates the calling process, freeing all memory associated with it. Returns 0 if all memory is freed successfully, -1 if some of the memory could not be freed.

int p_wait(int m)

Suspends the current process for 'm' milliseconds, after which the process resumes execution. Returns 0 after the timer has ended.

5.6.3 Process Information

int get_p_class()

Returns the current process' class as an integer, as defined by the following scale.

- 0 - System Process
- 1 - Interactive Process
- 2 - Foreground Process
- 3 - Background Process

int get_p_id()

Returns the calling process' id, as kept in the Process Control Block for that instance.

char[] get_p_name()

Returns the process' string identifier in the Process Control Block. This is primarily used for displaying the current processes in the Scheduler. If no name or an empty string is kept in the Process Control Block, then the process id is returned preceded by "P#" will be returned instead. For example: "P#24601"

int get_p_priority()

Returns the process's priority level as an integer. The following scale determines its priority.

- 0 - Highest Priority, System Priority
- 1 - High Priority
- 2 - Medium-High Priority
- 3 - Medium Priority
- 4 - Medium-Low Priority
- 5 - Low Priority
- 6 - Lowest Priority

int get_p_uptime()

Returns the combined heuristic of the process' elapsed execution time and the process' elapsed wait time.

int set_p_id(int id)

Changes the process' id to that specified in the argument, granted the requested id is not being used by another process. Returns 0 if successful, -1 if unsuccessful.

int set_p_priority(int p)

Sets the current process' priority according to the argument 'p'. Priority '0' can only be requested by system services. Returns 0 if successful, -1 if unsuccessful.

5.6.4 File Manipulation

Process File Descriptor
<code>int fileId</code> <code>void *fileAddr</code> <code>int fileFlags</code>

int close(int fileId)

Closes the file associated with the file id in the calling process. This also frees the file descriptor associated with the given file.

int open(char const *path, int flags)

Opens or creates the file at the designated path. The flags specify what permissions the file will be given upon being opened or created. Upon success, a file descriptor is created for the process and the file descriptor id is returned. Returns -1 if the file could not be opened or created.

Flags Key

0 - Read Only

1 - Write Only

2 - Read, Write

int read(int fileId, void *buffer, int count)

Reads a number of bytes specified by 'count' into the data buffer 'buffer' from the file associated with the given file id and the calling process. The file must have been opened with read permissions. Returns 0 if successful, -1 if unsuccessful.

int write(int fileId, void *buffer, int count)

Writes a number of bytes specified by 'count' from the data buffer 'buffer' into the file associated with the given file id and the calling process. The file must have been opened with write permissions. Returns 0 if successful, -1 if unsuccessful.

int print(void *buffer, int count)

Writes a number of characters from the specified address to the screen.

5.7 Implementation of System Calls

Before going into full detail on how all the system calls will be executed, we will talk about a Table we'll be using that is similar to one used in the Linux operating system. In the Linux operating system when a system call is initiated and then sent off to be executed, the assembler will create the corresponding assembly code and will then have to reference that given command through an integer value. That value will represent the level of priority for the interrupt. We have looked at tables for reference and have narrowed down the list to put the information we'll need to use to implement the initial design of PegasOS. As this being the initial design for PegasOS, there is a high chance there will be more in the table in the final release for PegasOS.

Syscall Name	Value
get_clocktime	1
set_clocktime	2
set_clockzone	3
p_clone	4
p_destroy	5
p_wait	6
get_p_class	7
get_p_id	8
get_p_name	9
get_p_priority	10
get_p_uptime	11
set_p_id	12
set_p_priority	13
close	14

open	15
read	16
write	17
print	18

So with the table above, the assembler will use these values to distinguish the separate commands by storing the value into the register which represents the corresponding command being executed. As mentioned above, these are only for the syscall commands we have right now. There will be more commands added for needed functionality when we implement them into the operating system. Below will be an example of how we will execute system call functions. The examples below are samples of the Linux 0x80 systems, we will be implementing it in a similar fashion.

```
int main(int argc, char *argv[])
{
    write(1, "Hello World\n", 12); /* write "Hello World" to stdout */
    _exit(0);                      /* exit with error code 0 (no error) */
}
```

The program demonstrated above is a sample code of System Calls for a simple Hello World print to the screen program. The write command is given the proper parameters where the 1 signifies that it will be written to stdout. The string in the middle is what will be written to the screen, and 12 is the number of bytes to be written from the string given. The exit system call command simply is used to signify that the program has ended and the 0 is simply the return value.

```
.data
msg: .ascii "Hello World\n"

.text
.global _start
```

When the assembler begins to generate the code needed for the given system call code, it will create a string variable to hold the string given as a parameter. It also creates the starting variable name of “_start” where the assembly code begins.

```

_start:
    movl $4, %eax    ; use the write syscall
    movl $1, %ebx    ; write to stdout
    movl $msg, %ecx  ; use string "Hello World"
    movl $12, %edx   ; write 12 characters
    int $0x80        ; make syscall

    movl $1, %eax    ; use the _exit syscall
    movl $0, %ebx    ; error code 0
    int $0x80        ; make syscall

```

The last part of this example is the generated assembly code that will be executed by the CPU. One can see that we are storing the integer value corresponding to the command we are executing in the %eax register and the other parameters stored in their respective registers. Once these are done, it is then executed since the command 'int \$0x80' is encountered. When completed, it moves onto performing the assembly code for "_exit(0)".

We will be implementing our system calls very similar to this format and the same with the accompanying assembly code to perform the commands.

PegasOS

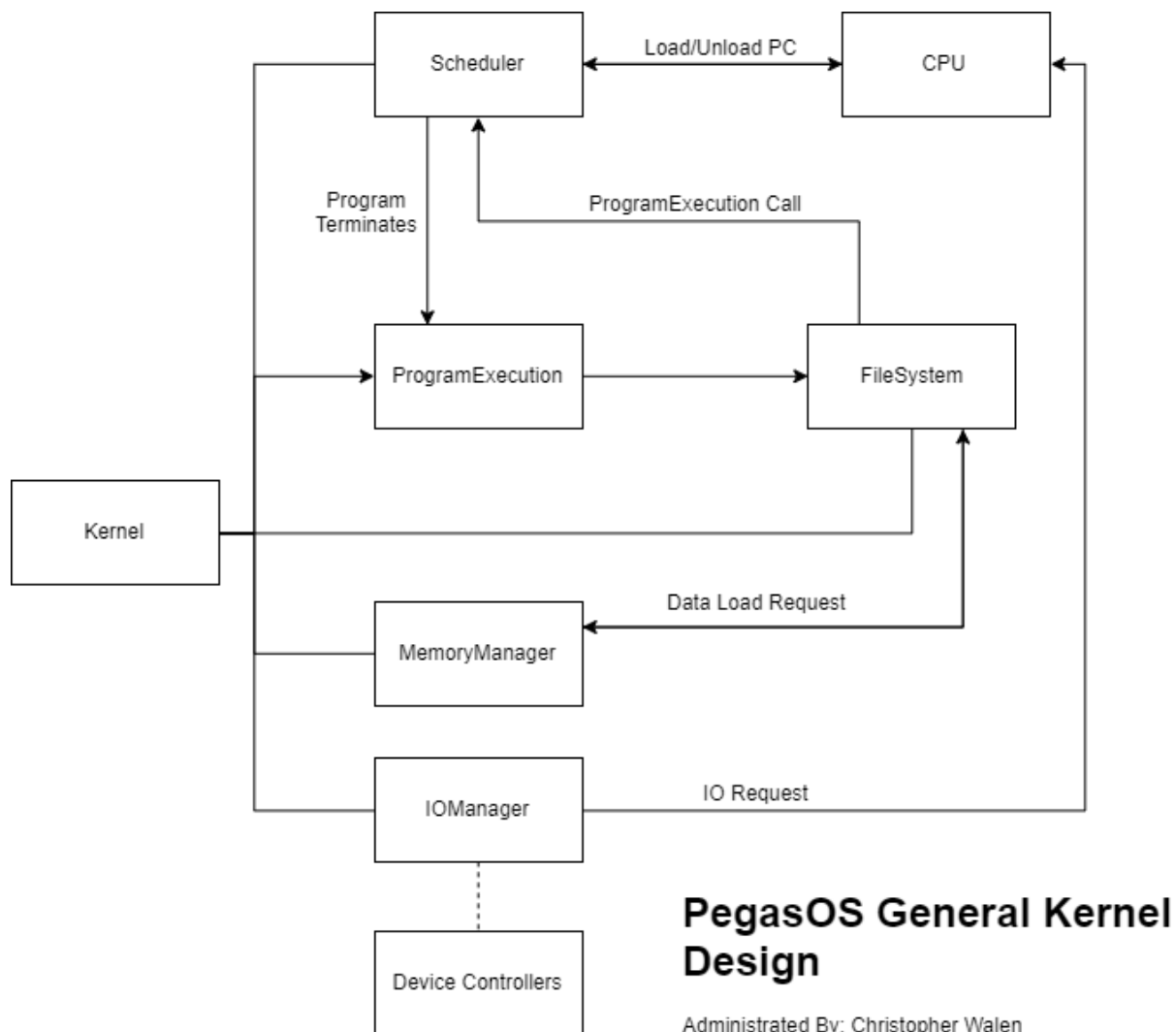
6. Kernel

```

:SS:
:SS/ /SS:
+SS:~ :: `:SS+-
.:OSS/. :OS//SS: ./SSO/.
`-/+000+-` -+SS/` `/SS+- `--+000+/-`
``.--:/+00000+/-` `-/OSO/. ./OSO/-` `--+00000+/:-.``
oy++/::-.`` `-/+0000:. :.0000+/-` `.-:/++ys
h: ::/+0000000+:-` `.:+0000000+/: :h
h: `N-.` `.-N` :d
h: `N N` :d
y/ M N` /h
so m. `N +s
/y h: :d y+
.m oo oo m.
m. -m d- .m
oo d- .m os
.m +y s+ m.
h/ m. .m` :h
-m /h y/ d-
y+ h/ /h +y
`m. `m. .m. `m`
:h :d` `d: h/
so /h h+ os
h/ +y yo /h
`d/ +h` `y+ :d`
`d/ /h` `h/ :d`
`h+ -d- -d- /h`
ss `yo +h` oy
+h. +h. .h+ `h+
.h/ .yo` oy. :h-
oy. :h/ /h: `yo
.yo +h: :h+ +h.
:h/ /y+` `+y+ /h:
/y+` -ss: :ss: /y/
:yo. `/ss/` `/ss/` .oy:
.oy/` -oss//sso- `/yo.
:ss/` .- `/ss:
-OSO- -OSO:
`/osso/`
```

6.1 Kernel Design

Overall, the kernel will have a straight-forward layout, made up of a few key components that will do the heavy lifting of the operating system, and the kernel will essentially be a 'super-manager' of these components. The main components are as follows: the Scheduler, Program Execution, File System, Memory Manager, and Input/Output Manager. The basic flow of connections between these components is demonstrated in the following diagram.



Implementing the kernel will be tricky, as it has to run on the CPU to start up the rest of the operating system, but the Scheduler is what's handling processes on the CPU. This means that the kernel will need to spin-up and get all of the initial system processes started and loaded onto the scheduler, then turn itself into a process (or at least copy itself into a process) then run that on the Scheduler.

Because the kernel is the first thing that will run on the Raspberry Pi, it will need to know the addresses of registers and where the Scheduler, Program Execution, Shell, File System, Memory Manager, and Input/Output Manager are residing in memory. One way that we could solve this is to include the shell in the binary for the kernel, then use the shell's program binary directory for the other managers (Scheduler, ProgramExecution, FileSystem, MemoryManager, and Input/Output Manager) so that the kernel can then spin those up as processes through the ProgramExecution. This means that we will have the following start-up order:

Kernel Start-Up Order

1. Kernel
2. Scheduler
3. FileSystem
4. MemoryManager
5. ProgramExecution
6. Shell

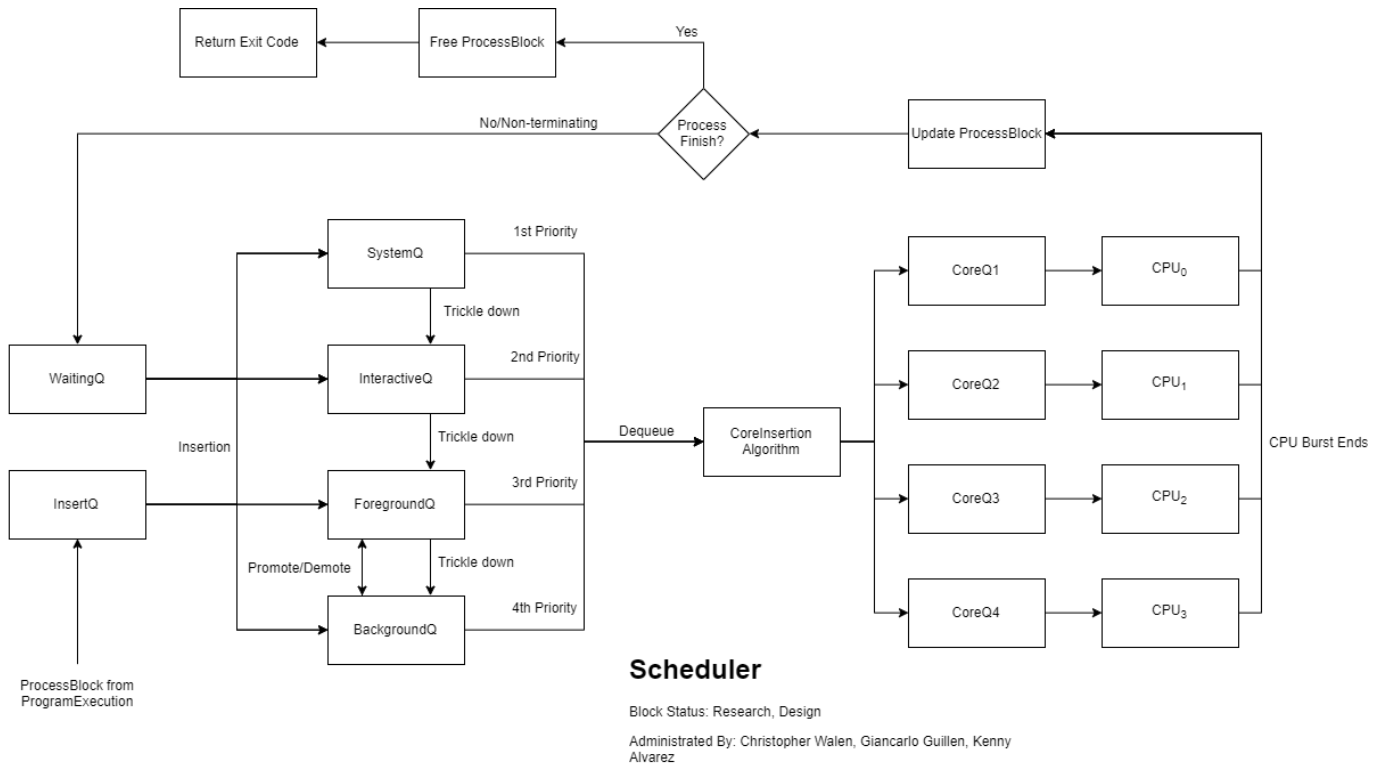
This way, the kernel can start the rest of the components in the order that we need them to start functioning. The scheduler has to be started as soon as possible because it is moving processes on and off of the CPU constantly, and apart from our boot-up sequence it is primarily responsible for process execution on the CPU. Which means that ProgramExecution will need to start up next, so that we can bundle those processes. However, ProgramExecution needs the Memory Manager to have already spun up so that it can load text into pages to grab those page addresses for the ProcessBlock, and the Memory Manager won't know exactly where to look and how to read the discs without the File System in place.

This means that the File System must spin-up immediately after the Scheduler has started, and from there the Memory Manager can spin-up and talk to the File System, and finally, the ProgramExecution can talk to the Memory Manager. The last component that the kernel will start is the Shell itself, which will need the rest of the system to be ready before we can allow the user to interface with it.

During step 1 of the start-up, any other initialization or component spin-up that the kernel needs for operation will happen during that step. If the component becomes too large or too important in function, then it will be moved into a separate component that ties into the kernel and receives a position in this start-up list.

6.2 Kernel Components

6.2.1 Scheduler



The scheduler uses a Multi-Level Feedback Queue to allow for processes of different categories. Using the multiple levels, we can create a hierarchy within the process 'types' themselves to allow for priority insertion to take place on the CPU. This will allow us to have system or kernel processes done in time before other processes need that information. For example, a file system request can be handled before a program that needs to access that file.

It is important to note that the initial implementation of this scheduler will not be making use of all of the cores on the processor unless time allows for it. However, the design of the scheduling algorithm is based around having multiple cores to better prepare for moving the operating system over to true multi-core processing.

To achieve the best optimization possible while observing priority among the processes, we've broken up the scheduler into four distinct queues that have precedence over the following queues; the SystemQ, InteractiveQ, ForegroundQ, and BackgroundQ. What this means is that the SystemQ has higher priority over the InteractiveQ, and if we are currently working in another queue when a system process is reinserted back into the SystemQ, we will suspend the current process and move back to the SystemQ.

The following contains a more detailed breakdown of each queue:

SystemQ

This queue is for kernel or system-level processes, such as page swapping, scheduling, etc. SystemQ uses a First-Come-First-Serve sort (or First In First Out if you prefer), to allow system processes to finish executing before moving on to the next one such that no system process is waiting on data from another system process, or tries to access the same piece of the system at the same time. Insertion into this queue will be very straight forward, and keep runtime for this queue very quick.

InteractiveQ

This queue is for processes that require the user's attention constantly for its operation, such as a game or text editor. The InteractiveQ uses a Bucket Sorting algorithm, wherein the buckets are filled based on priority, then sorted based on the Shortest Reaction Time of the processes. The InteractiveQ also allows for the aging of processes to change their priority within the InteractiveQ such that no process is starved. Due to the bucket sort implementation for priority within the queue, the runtime will suffer slightly but the programs that require a higher response time will be granted a higher priority, so that they get worked on more frequently.

ForegroundQ

This queue is for processes that are 'on the screen and active'. In this case where we do not have a Graphical User Interface and instead have a Command Line Interface, this will be the program(s) that are running on the shell. The shell itself will either be running in the ForegroundQ as well or as a System-level process. The ForegroundQ also uses a Bucket Sorting algorithm, wherein the buckets are filled based on priority, then sorted based on the Shortest Job First among the processes. The ForegroundQ also allows for processes to be

demoted to the BackgroundQ, as well as aging processes to change their priority within the ForegroundQ such that no process is starved.

BackgroundQ

This queue is for 'minimized' processes or non-System daemons that need to be running without the user's input. The BackgroundQ uses a Round-Robin algorithm, wherein each process gets a constant-quantum time slice of CPU time, before moving on to the next process. The BackgroundQ also allows for processes to be promoted to the ForegroundQ, in the case that it is 'maximized' and brought to the foreground of the user's interactive space, or if it is otherwise being starved in the BackgroundQ.

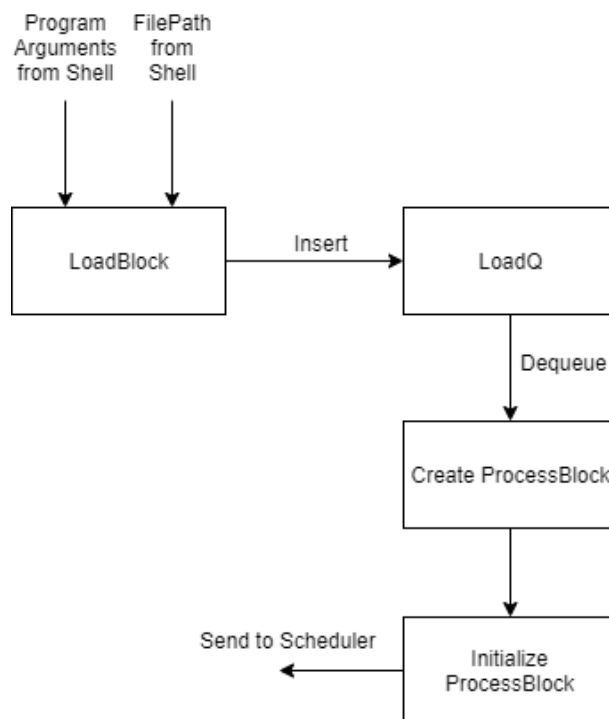
Inserting into these queues will be done in two ways, as shown in the Block Diagram for the Scheduler. Brand new processes that are coming from ProgramExecution will be placed into the InsertQ, while processes that have already passed through the Scheduler at least once will be placed in the WaitingQ. This ensures that there is a feed-in system for the Scheduler to receive new processes, and that the Scheduler can contain processes that it already has control of or access to. From here, the insertion algorithm will look at the process blocks in First In First Out order, and place them into the respective process class queue. The WaitingQ processes will only be reinserted if they are no longer waiting for an interrupt, or if they have asked to be placed back onto the CPU. In the case of the InteractiveQ or the ForegroundQ, the priority of the process will determine which bucket it is then placed into. After insertion from the InsertQ and WaitingQ are done, the SystemQ is looked at first for execution. Execution will be handled in one of two ways.

1. **For single-core usage**, the process will simply feed the CPU the Program Counter and instruction address contained within the ProcessBlock, and the CPU will begin execution until the burst ends. Once the burst ends, we will update the ProcessBlock's Program Counter and check the process's status. If it is waiting for Input/Output or is otherwise waiting or suspended, it will be placed on the WaitingQ for reinsertion. If the process has terminated, the Scheduler will broadcast the termination code to the ProgramExecution (in the case of a non-system process) such that it can report to the user or shell how the process ended.
2. **For multi-core usage**, each core on the CPU will be given a separate CoreQ that will allow processes to be queued for the given core. If a process is being put onto a core for the first time, we will update the ProcessBlock such that it has a 'preference' for being placed on a particular core. This is to prevent recaching

cores every time that we reload a process onto that core for execution. Once it has moved onto the CoreQ, the CoreQ will be cleared in First In First Out order, feeding the CPU the Program Counter and instruction address for that process. Once the burst ends, we will update the ProcessBlock's Program Counter and check the process' status. If it is waiting for Input/Output or is otherwise waiting or suspended, it will be placed on the WaitingQ for reinsertion. If the process has terminated, the Scheduler will broadcast the termination code to the ProgramExecution (in the case of a non-system process) such that it can report to the user or shell how the process ended.

6.2.2 Program Execution

The Program Execution loads programs into ProcessBlocks from memory, dealing with the Page Manager as needed, then sends the ProcessBlock off to the Scheduler for CPU execution. This manager will also be what is called through the Command Shell when the user wants to execute programs of their own choosing.



ProgramExecution

Block Status: Research, Design

Administrated By: Christopher Walen, Giancarlo Guillen

While the ProgramExecution component is simple in design, it will be communicating with the MemoryManager extensively. This is because it will need to request pages of text from memory, from the PageManager, and to know where the virtual addresses or page addresses of that text reside, to then place in the ProcessBlock. Once the data has been loaded by the MemoryManager and its components, the ProgramExecution will retrieve the text address in the virtual/page space and place that in the ProcessBlock. Then the rest of the ProcessBlock's information will be initialized and stored, after which the ProcessBlock can be sent off to the Scheduler. In addition to this, the ProgramExecution uses a LoadQ to keep track of which programs have been requested to be executed so that when the ProgramExecution gets CPU time it can remember which programs still need to be bundled up into processes.

The ProcessBlock will contain the following information (subject to modification as needed):

ProcessBlock
<pre>int processID char[] processName <<enum>> int processClass int processPriority int programCounter p* textAddress int textLength int corePreference int elapsedExecTime int elapsedWaitTime int estimatedRespTime int estimatedJobTime</pre>

- The process ID is unique to the process - no two processes may have the same ID (even if they are copies of the same program).
- The process name is not used in the algorithms but will be used when the user asks to display what processes are currently in the Scheduler or waiting for execution so that the user can quickly understand what programs are running on the operating system.
- The process class determines which queue in the Scheduler the process will be inserted into.
- The process priority determines which bucket the process will be inserted into if it is inserted into the InteractiveQ or ForegroundQ in the Scheduler.
- The program counter tracks which instruction we are currently working from in the process' text, and is updated at the end of a CPU burst.

- The text address is the location of the instructions in virtual memory, which will require a page look-up by the Scheduler. If this address is affected in some way by this look-up, it will be updated accordingly so that the code for this process is not lost.
- The text length is a line count of how many instructions make up the code for this program.
- The core preference is triggered when using the multi-core version of the Scheduler, which will tell the core insertion algorithm to insert this process on a particular core instead of the first available core to prevent reaching cores when it is not necessary.
- Elapsed execution time and elapsed waiting time are used to determine if a process needs to be aged, and to inform the user how long a process has been in the Scheduler/CPU track.
- The estimated response time and estimated job time are based on equations related to the length of code and general program execution analysis, to best guess what the needs of the program are for insertion into the InteractiveQ and ForegroundQ in the Scheduler.

6.2.3 Interrupt Handler

The Interrupt Handler is a major component of the kernel, as it redirects control from various operations in order to handle requests from various software and hardware.

In the Interrupt Handler there are 5 separate distinct types of interrupts:

1. Hardware: this is any input from a hardware component being used, ie. keyboard or mouse.
2. Non-maskable interrupt: these are interrupts that the processor can not ignore and have the highest precedence.
3. Software interrupts: these interrupts are also called maskable interrupts, these are occurred through interrupts brought from code being executed thus creating system calls to create interrupts.
4. Internal interrupt: this interrupt is for the processor runs into an error, ie. file not found or divided by zero.
5. Reset: this one will simply reset the state of the CPU.

After an interrupt has been received, the Handler must then determine what kind of action must be performed. The appropriate arguments will be set in the Interrupt Control Register, the state of the interrupt caller will be saved, and the Handler will resolve the request. Depending on the type of interrupt received - in this case the ARM exceptions (IRQ and FIQ, please see the *Exception Handling* section in *Hardware* for more information), we will need to resolve that request first.

6.2.4 Memory Manager

The Memory Manager is a more general manager that has two separate components, the Page Manager and the Virtual Memory Manager. In addition to those components, it will also talk to the Hard Disks/SD cards/Solid State Disks to load instructions from memory when called to do so.

The most hands-on task of the memory manager is cache maintenance. There are specific ARM commands to upkeep the cache, and it is the memory manager's job to make sure it stays working and clean.

6.2.5 Page Manager

The Page Manager swaps memory pages in and out of RAM when necessary, to give running programs the amount of space they require. The mechanics of this lie mostly in the page table design, which can be referenced in *Hardware*.

6.2.6 Virtual Memory Manager

The Virtual Memory Manager handles the allocation of virtual memory. Most of the virtual memory is processed by arm through the MMU, so much of our implementation will just be using ARM. For more information about this process, please refer to the *Arm Research* section in *Hardware*.

6.2.7 Cache Management

When the MMU is configured, the cache and page table is made. For a more in-depth view on how to set up the MMU, please refer to *Hardware*. For the cache itself, there are specific ARM commands created to manage the cache. This is in case it needs to be cleaned, or the MMU configuration is updated. In both cases, the cache must be updated as well. There are three different actions you can take with the cache: Invalidation, Cleaning, or Zero.

Invalidation: This clears a cache or a cache line of data. A cache must be invalidated after a reset.

Cleaning: This means writing the lines of the cache that are dirty into external memory and clearing the line.

Zero: This zeroes out the cache without reading the contents from an outer domain.

Most basic cache maintenance is done by the firmware and should not be edited too much. However, the cache can be used to move large amounts of data into a temporary space in which to be loaded. In this case, you would need special operations that a memory manager would handle. However, outside of special cases the firmware handles the rest, so there is no need for a large algorithm to handle this section.

6.3 File Systems

6.3.1 File System Research

File Systems organize data on the computer such that the user can read and write to it. File Systems act as a storage mechanism which is made for allocating space and organizing how files are saved on storage devices. Without a file system on a storage device, data is placed on it would be just one large block of data with no possible way to tell where a piece of data is, where it stops and where the next piece starts. With a file system separating the data into smaller pieces and naming each of them, it makes it easier to identify most pieces of data in a storage device.

A File System should be able to track available storage space, track which blocks of data belong to which file, create new files, read data from existing files into memory, update data in data, and delete existing files. Other functions for the file system that are practical are assigning names to files and renaming files, having some sort of structure to allow files to be divided into directories and/or folders providing a structure.

For the Raspberry Pi, there are several different file systems that can be used on the Pi and there are some pros and cons for each of them. There is also the option to create a new file system for the Raspberry Pi that we can use but that also has some advantages and disadvantages. There are two main file systems that we considered for PegasOS, those being FAT32 and Ext2.

6.4 FAT32

FAT is an acronym for File Allocation Table which was introduced with DOS v1.0. FAT is a file system architecture as well as a family of industry-standard file systems. There are different versions of the FAT system with each being made for different sizes of storage such as FAT12, FAT16, FAT32, and ExFAT. It cannot deliver the same performance as some other modern file systems but continues to be supported for its simplicity and compatibility. FAT32 was introduced with MS-DOS 7.1 / Windows 95 OSR2 in 1996. FAT32 is one of the most popular file systems in the world today, partly due to it being one of the most basic file systems.

FAT32 is compatible with a wide range of devices, including the Raspberry Pi. It is also compatible with almost all operating systems including macOS, Windows, and Linux. The Raspberry Pi's bootloader is built into the GPU, and only has support for the

reading of the FAT file system and using other file systems may require using drivers. One of the biggest limitations for FAT32 is that a single file cannot be greater than 4 GB in size due to its 32-bit nature, and does not include direct built-in support for long filenames. The FAT32 boot sector uses a 32-bit field for the sector count limiting the maximum volume size to about 2 TB with a sector size of 512 bytes. For a sector size of 4,096 bytes, the maximum size volume is about 17.6 TB.

The FAT file system views storage devices as a flat array of clusters. Microsoft designed FAT32 to support an increased number of possible clusters compared to the older versions of the FAT file systems and reuses a majority of the pre-existing code. The cluster values are represented by 32-bit numbers, 28 of those bits are used to hold cluster numbers. If the storage device does not address the data as a flat list of sectors, then the cluster numbers will need to be translated before being sent to the disk. The storage device is organized into three areas: **the boot record**, **the File Allocation Table (FAT)**, and **the directory and data area**.

6.4.1 Boot Record

The boot record is reserved on one sector, and it is arranged in sector zero of the partition. If the storage device hasn't been partitioned, then it is the beginning of the storage device's memory. This sector is the easiest to locate for the computer when it's loaded. If the media was already partitioned, then the MBR (x86) or other partition information are at the start of the media and the partition's first sector holds a Volume Boot Record.

6.4.2 BIOS Parameter Block (BPB)

The boot record contains both code and data that are mixed together and the data that isn't code is called the BIOS Parameter Block.

Offset (decimal)	Offset (hex)	Size (in Bytes)	Meaning
0	0x00	3	Jump Code
3	0x03	8	OEM identifier.
11	0x0B	2	The number of Bytes per sector (all numbers are in the little-endian format).
13	0x0D	1	Number of sectors per cluster
14	0x0E	2	Number of reserved sectors. The boot record sectors are included in this value.
16	0x10	1	Number of copies of FAT
17	0x11	2	Number of Root Directory Entries. (N/A for FAT32)
19	0x13	2	Number of Sectors in Partition Smaller than 32MB
21	0x15	1	the media descriptor type
22	0x16	2	Number of sectors per FAT. FAT12/FAT16 only.
24	0x18	2	Number of sectors per track.
26	0x1A	2	Number of heads or sides on the storage device.
28	0x1C	4	Number of hidden sectors.
32	0x20	4	Number of Sectors in Partition

Source: <https://wiki.osdev.org/FAT32>

<https://www.easeus.com/resource/fat32-disk-structure.htm>

6.4.3 Extended Boot Record

The extended boot record comes right after the BPB. The data at the beginning is called the extended BIOS Parameter Block (EBPB). The information it contains depends whether the partition is a FAT12, FAT16, or FAT32. Right after the EBPB is the boot code then the standard 0xAA55 boot signature to fill out the 512-byte boot sector.

Offset (decimal)	Offset (hexadecimal)	Size (in bytes)	Meaning
36	0x024	4	Number of Sectors Per FAT
40	0x028	2	Flags
42	0x02A	2	Version of FAT32 Drive (Highbyte means Major Verison and Lowbyte means minor
44	0x02A	4	The cluster number of the root directory. Often this field is set to 2.
48	0x030	2	The sector number of the FSInfo structure.
50	0x034	2	The sector number of the backup boot sector. (Referenced from the Start of the Partition)
52	0x034	12	Reserved
64	0x040	1	Drive Number of Partition
65	0x041	1	Flags in Windows NT. Reserved otherwise.
66	0X042	1	Signature (must be 0x28 or 0x29).
67	0X043	4	Serial Number of Partition
71	0X047	11	Volume Name of Partition
82	0X052	8	FAT Name
90	0X05A	420	Executable Boot code.
510	0x1FE	2	Boot Record Signature

Source: <https://wiki.osdev.org/FAT32>
<https://www.easeus.com/resource/fat32-disk-structure.htm>

6.4.4 File System Info Structure

This is in the second sector of the partition:

Offset (decimal)	Offset (hexadecimal)	Size (in Bytes)	Meaning
0	0x0	4	Lead signature (must be 0x41615252 to indicate a valid FSInfo structure)
4	0x4	480	Reserved, these bytes should never be used
484	0x1E4	4	Signature of FSInfo Sector
488	0x1E8	4	Number of Free Clusters
492	0x1EC	4	Cluster Number of Cluster that was Most Recently Allocated
496	0x1F0	12	Reserved
508	0X1FC	4	Boot Record Signature

Source: <https://wiki.osdev.org/FAT32>
<https://www.easeus.com/resource/fat32-disk-structure.htm>

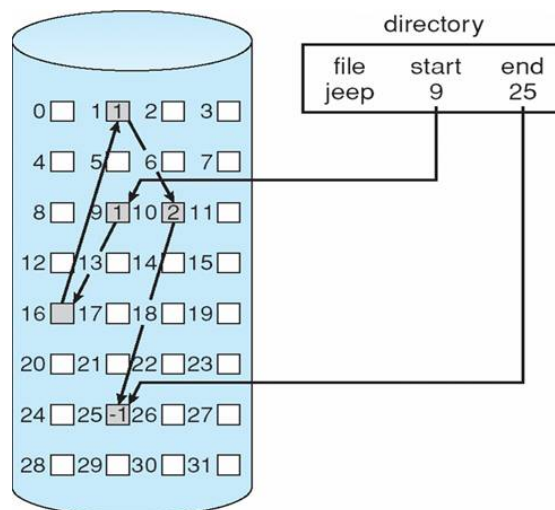
A cluster is a Group of Sectors on the disk that has information in them. Each Cluster is reserved at the FAT Table and when looking at an entry in the FAT, it tells you if it has data, If it is toward the end of the data, or if another cluster is right after it. if the entry is 0 then there isn't any data in the current cluster. If the entry is 0FFFFFFF then it's the last entry in the chain.

6.4.5 File Allocation Table

The File Allocation Table is a table stored on the storage device that shows the status and location of all the data clusters that reside on the disk. The cluster may be available for use and the operating system can reserve it. It can also be unavailable because of a bad sector that may be in the storage device or it is likely that it could currently be in use by a different file. Clusters don't have to be near each other, and it would just be distributed all over the disk. The operating system can follow chains of clusters in a file.

FAT32 uses 28 bits to address the cluster on the storage device and the highest 4 bits are reserved. They need to be ignored when reading and unchanged when writing.

This is an example of how a file allocation table works and how they are structured:



Source:

<https://www.cs.odu.edu/~cs471w/spring10/lectures/FileSystemImplementation.htm>

6.4.6 Directory

A directory entry stores information about where a file's data or a folder's children are stored on the disk. It also has information like the entry's name, size, time of creation, extension, address of the first cluster of the directory's data and the attributes (such as directory, hidden, read-only, system and volume). There are two types of directories for the FAT file system, the standard 8.3 directory entries which appear on all the FAT file systems.

6.4.7 Standard 8.3 Format

8.3 filenames are limited to eight characters followed by an extension which is limited to three characters. If the system only supports 8.3 filenames the characters after the limited eight characters are ignored and if there is no extension in the file name, the period if present has no significance in the filename. The root directory region and in subdirectories are in the following format.

Offset (in Bytes)	Length (in Bytes)	Meaning								
0	11	8.3 file name. The first 8 characters are the name and are padded with spaces. The last 3 bits are the extension								
11	1	Attributes of the file.								
12	1	Reserved for use by Windows NT.								
13	1	Creation time in tenths of a second. Range 0-199 inclusive. Based on simple tests, Ubuntu16.10 stores either 0 or 100 while Windows7 stores 0-199 in this field.								
14	2	The time that the file was created. Multiply Seconds by 2.								
		<table><tr><th>Bits</th><th>Meaning</th></tr><tr><td>5</td><td>Hour</td></tr><tr><td>6</td><td>Minutes</td></tr><tr><td>5</td><td>Seconds</td></tr></table>	Bits	Meaning	5	Hour	6	Minutes	5	Seconds
		Bits	Meaning							
		5	Hour							
		6	Minutes							
5	Seconds									

16	2	<div>The date on which the file was created.</div> <table><tr><td>Bits</td><td>Meaning</td></tr><tr><td>7</td><td>Year</td></tr><tr><td>4</td><td>Month</td></tr><tr><td>5</td><td>Day</td></tr></table>	Bits	Meaning	7	Year	4	Month	5	Day
Bits	Meaning									
7	Year									
4	Month									
5	Day									
18	2	Last accessed date. Same format as the creation date.								
20	2	The high 16 bits of this entry's first cluster number. For FAT 12 and FAT 16 this is always zero.								
22	2	Last modification time. Same format as the creation time.								
24	2	Last modification date. Same format as the creation date								
26	2	The low 16 bits of this entry's first cluster number. Use this number to find the first cluster for this entry.								
28	4	The size of the file in bytes.								

Source: <https://wiki.osdev.org/FAT32>

6.4.8 Long File Names

Long file names entries are always placed immediately before 8.3 entry. Long Name directory is optional and allows FAT to have longer file names.

Offset (in Bytes)	Length (in Bytes)	Meaning
0	1	The order of this entry in the sequence of long file name entries.
1	10	the first 5, 2-byte characters of this entry.
11	1	Attribute. Always equals 0x0F. (the long file name attribute)
12	1	Long entry type. Zero for name entries.
13	1	Checksum generated from the short file name when the file was created. The short filename can change without changing the long filename in cases where the partition is mounted on a system which does not support long filenames.
14	12	The next 6, 2-byte characters of this entry.
26	2	Always zero.
28	4	The final 2, 2-byte characters of this entry.

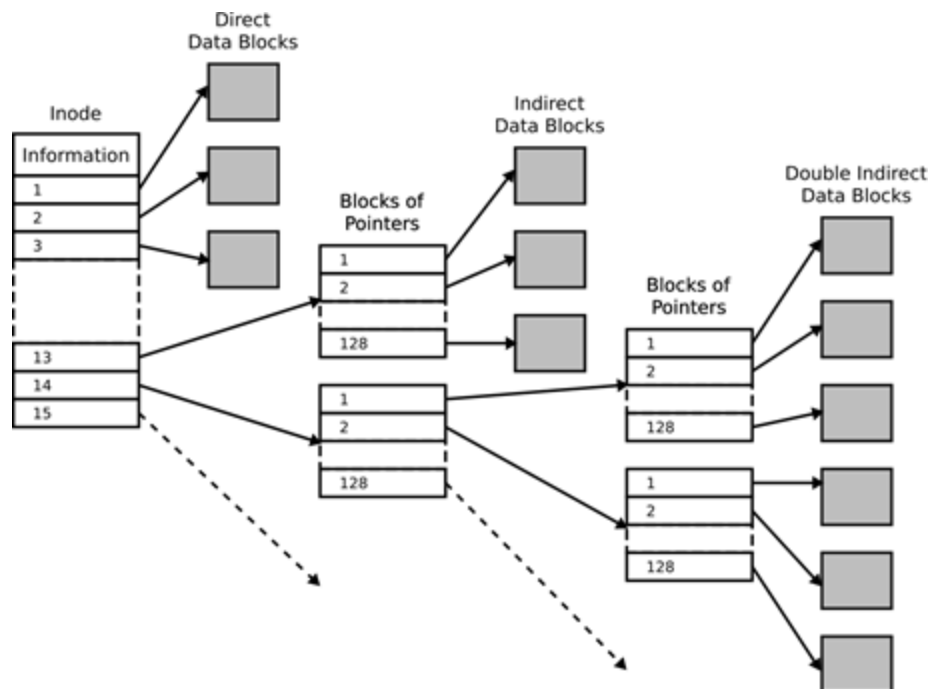
Source: <https://wiki.osdev.org/FAT32>

6.5 Ext2

Ext2 stands for Second Extended File System and it is the rewrite from the Extended File System based around “inodes”. Ext2 was the filesystem for Linux for about a decade from the early 1990s to the early 2000s when it was succeeded by Ext3 and later Ext4.

An inode is an essential concept in the Ext2 file system. Every object that is in the file system is defined by an inode. They don't contain the actual data that they represent but they contain pointers to blocks that contain the data. With this, the inodes have a defined size which helps them be arranged in indexed arrays. Every block group has an array of inodes and every inode belongs to one block group.

Example of Ext2 inode data structure:



Source: <https://en.wikipedia.org/wiki/File:Ext2-inode.svg>

The file system divides the disk into groups called “block groups”. These groups result in the distribution of data across the disk which helps to minimize head movement

and impact fragmentation. Some groups are required to contain backups of data that can be used to rebuild the file system in case of a system failure.

Ext2 divides the disk space into blocks of adjoining space. The blocks don't need to be the same size as the sector size of the disk Ext2 resides on. The size of the blocks can be determined by reading the field at byte 24 in the superblock.

Blocks are divided into block groups and they are contiguous groups of blocks. Block groups reserve a few of their blocks for purposes such as a bitmap of free and allocated blocks within the group, a bitmap of allocated inodes with the group, a table of inode structures that belong to the group, and some or all block groups may contain a backup copy of the superblock and the Block Group Descriptor Table.

A Superblock contains all the information about the layout of the file system and the configuration of the filesystem. The information the Superblock contains is the number of inodes and blocks being used and how much is free, and the number of inodes and blocks in each block group. It also shows when the file system was mounted, modified, and shows the current version and what operating system it was created on. Implementing an Ext2 driver is to find, extract, and parse the superblock. It is always located at byte 1024 at the beginning of the volume and is 1024 bytes in length.

To determine the number of block groups from the Superblock we must check the size of each block, the total number of inodes, the number of blocks, and the number of inodes in each block group. With this, the number of block groups can be determined by rounding up the total number of blocks divided by the number of blocks per block group or the number of inodes divided by the number of inodes per block group. For added security, we can perform both and check them against each other.

6.5.1 Base Superblock Fields

Starting Byte	Ending Byte	Size (in Bytes)	Meaning
0	3	4	Total number of inodes in file system
4	7	4	Total number of blocks in file system
8	11	4	Number of blocks reserved for superuser
12	15	4	Total number of unallocated blocks
16	19	4	Total number of unallocated inodes
20	23	4	Block number of the block containing the superblock
24	27	4	\log_2 (block size) - 10.
28	31	4	\log_2 (fragment size) - 10.
32	35	4	Number of blocks in each block group
36	39	4	Number of fragments in each block group
40	43	4	Number of inodes in each block group
44	47	4	Last mount time
48	51	4	Last written time
52	53	2	Number of times the volume has been mounted since its last consistency check
54	55	2	Number of mounts allowed before a consistency check
56	57	2	Ext2 signature (0xef53), used to help confirm the presence of Ext2 on a volume
58	59	2	File system state
60	61	2	What to do when an error is detected
62	63	2	Minor portion of version

64	67	4	last consistency check
68	71	4	Interval between forced consistency checks
72	75	4	Operating system ID from which the filesystem on this volume was created
76	79	4	Major portion of version
80	81	2	User ID that can use reserved blocks
82	83	2	Group ID that can use reserved blocks

Source: <https://wiki.osdev.org/Ext2>

6.5.2 Extended Superblock Fields

Present in Major version if the version is 1 or greater.

Starting Byte	Ending Byte	Size (in Bytes)	Meaning
84	87	4	First non-reserved inode in the file system
88	89	2	Size of each inode structure
90	91	2	Block group that this superblock is part of (if backup copy)
92	95	4	Optional features present
96	99	4	Required features present
100	103	4	Feature that if not supported, the volume must be mounted read-only
104	119	16	File system ID
120	135	16	Volume name
136	199	64	Path volume was last mounted

200	203	4	Compression algorithms used
204	204	1	Number of blocks to pre-allocate for files
205	205	1	Number of blocks to pre-allocate for directories
206	207	2	Unused
208	223	16	Journal ID
224	227	4	Journal inode
228	231	4	Journal device
232	235	4	Head of orphan inode
236	1023	x	Unused

Source: <https://wiki.osdev.org/Ext2>

6.5.3 Block Group Descriptor Table

The Block Group Descriptor Table is an array of block group descriptors that are used to define parameters of all the block groups and it also contains a descriptor for each of them in the file system. The descriptor provides the location of the inode bitmap, inode table, and other important data. The block group descriptor table starts right after the superblock. Depending on how many block groups are defined, the table may require multiple blocks of storage.

6.5.4 Block Group Descriptor Table Structure

Starting Byte	Ending Byte	Size (in Bytes)	Meaning
0	3	4	Block address of block usage bitmap
4	7	4	Block address of inode usage bitmap
8	11	4	Starting block address of inode table
12	13	2	Number of unallocated blocks in group
14	15	2	Number of unallocated inodes in group
16	17	2	Number of directories in group
18	31	x	Unused

Source: <https://wiki.osdev.org/Ext2>

6.5.5 Inodes

Every inode contains a numerical address and it always starts at one. They have a fixed size as defined by the field in the superblock. All inodes are located in inode tables that are in the block group and The size of the inode table is fixed at the time of formatting. Locating an inode is determining what block group it belongs to and indexing that block group inode table.

Each inode contains 12 direct pointers, one singly indirect pointer, one doubly indirect block pointer, and one triply indirect pointer. Space overflows to singly indirect space, then into doubly indirect space, then triply indirect space. If valid, direct block pointers value is non-zero and every pointer is the address for the block containing the data. When a file needs more than 12 blocks, a block is allocated then it stores the rest of the data block's addresses to store the content. The allocated block is called the indirect block as there is an indirection between the inode and its data. The addresses that are stored in the block are 32-bit and the addresses of the indirect block are called Singly Indirect Block Pointer. If the file needs more blocks that it can't fit in 12 direct pointers and the indirect block, then a double indirect block is needed. The double indirect block is the extension of a single indirect block with there being two blocks

between the inode and data blocks. If more space is still needed, the file will use the triple indirect block, which is, similar to the double indirect block, it's another extension. The triple indirect block has the addresses from the double indirect blocks, which has addresses of the single indirect blocks and that has the addresses of the data block.

6.5.6 Inode Data Structure

Starting Byte	Ending Byte	Size (in Byte)	Meaning
0	1	2	Type and permission
2	3	2	User ID
4	7	4	Lower 32 bits of size in bytes
8	11	4	Last access time
12	15	4	Creation time
16	19	4	Last modification time
20	23	4	Deletion time
24	25	2	Group ID
26	27	2	Count of hard links to this inode
28	31	4	Count of disk sectors in use by this inode
32	35	4	Flags
36	39	4	Operating system specific value
40	87	48	Direct block pointers 0-11, each being the size of 4 bytes
88	91	4	Singly indirect block pointer
92	95	4	Doubly indirect block pointer
96	99	4	Triply indirect block pointer

100	103	4	Generation number
104	107	4	Reserved
108	111	4	Reserved
112	115	4	Block address of fragment
116	127	12	Operating system specific value

Source: <https://wiki.osdev.org/Ext2>

6.5.7 Directories

Inodes are directories and they contain entries as their contents, and they are name/inodes pairs. Directories are stored as blocks of data and are referenced by inodes. A directory inode stores the entries as a linked-list in the contents block and the directory entry size may be longer than the length the name would suggest, and the record must be lined up to 4-byte boundaries. The second entry in the inode table contains the inode pointing to the data that belongs to the root directory. Entries can't span over several blocks in the file system so there might be space between entries. Space is not allowed between entries, any space found will be used for preceding record and increasing record length will include the space. Space may be marked by a different directory entry with inode number zero, implying that the entry should be skipped.

6.5.8 Directory Entry Structure

Starting Byte	Ending Byte	Size (in bytes)	Meaning
0	3	4	Inode
4	5	2	Total size of this (include all subfields)
6	6	1	Name length least-significant 8 bits
7	7	1	Type indicator
8	8+N-1	N	Name characters

Source: <https://wiki.osdev.org/Ext2>

6.5.9 Creating a Customized File System

When creating a file system, it must be considered what exactly it will be used for and what functions it will have. The file system is one of the core components of any operating system. While the main functions of a file system are to read a file, write/update the data in a file, and delete files, there are other practical features that go into a file system. Such practical features like:

- Assigning readable names to files and being able to rename them after they have been creating. While this does not make the file system any more efficient or change on how it functions, it is a practical and very common function for any file system that will be used by any person that will be using this operating system. This will make using the file system easier to use and manage data for the user.
- Allowing files to be divided into directories/folders. This is to have some sort of structure for the file system, this is the function that helps keep all the data in a storage device organized, not for the computer but for the user that will be using the operating system.
- Allowing a file to be read-only to prevent corruption of important data within the system.
- Buffering reading and writing to reduce the number of operations on the storage device.
- Providing some sort of security for the file system in order to prevent unauthorized access to the data within the files.

Other features that are found in file systems but are not required for a practical file system:

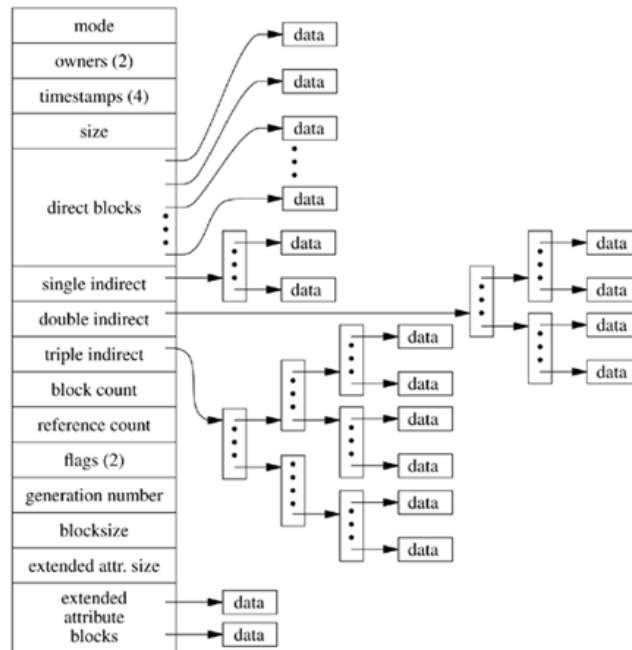
- Automatic encryption - This is to allow the files to be encrypted to protect important data from attackers that have physical access to the computer.
- Journaling of read and write activity - Journaling is when a file system stores every change that was made within the file system. With this feature, it is faster to recover after a force system shutdown and it is less likely that the system or the files will become corrupted. Some of the file systems that have this feature are ext3, ext4, and ReiserFS. There are two methods of writing journals for the file system:
 - Physical journals – These are journals that put every change that was ever made into the file system into the journal. This provides protection when a forced shutdown happens because the log can be read and rewrite data based on that and nothing will be lost. This can cause more

time to write to storage but it is a small price when having data protected from data loss or corruption.

- Logical journals – These kinds of journals write metadata every time the journal gets written to. This makes it faster to use than using the physical journaling method but there is an increased likelihood of the journal being corrupted. The system can still recover but cause the metadata and non-journaled data to be out of sync and this can cause the block to be corrupted.

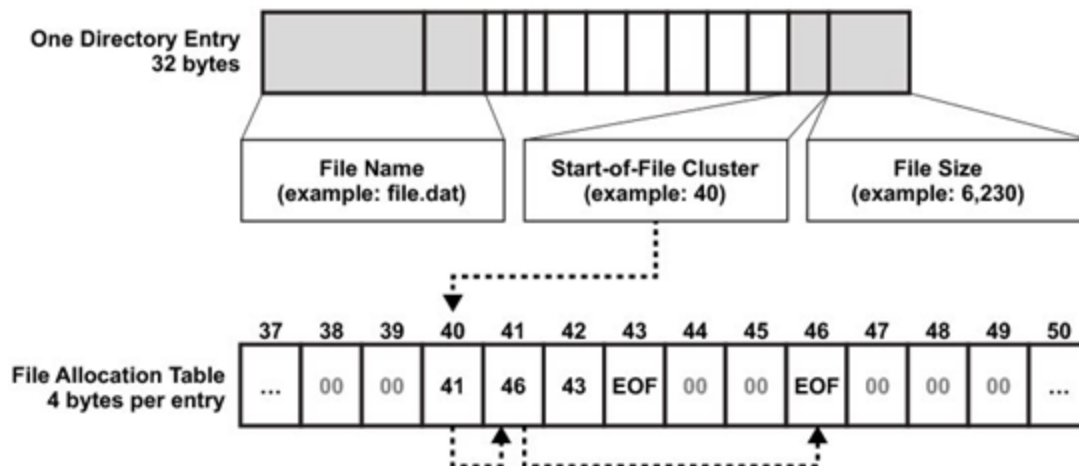
When organizing data with a file system there are different methods to it. With FAT32 and ext2 file systems, they both use different methods when indexing contents of files which makes them different on how they structure data within the storage. Such methods that are commonly used:

- Inodes – Ext2 file system uses this method for indexing their data. They are also called information nodes and are design elements in most Unix file systems. Each file is made of blocks of data which are sectors that contain raw data, index blocks that contain pointers to data blocks, and one inode block. The inode is the root of the index blocks and can be the only index block if the file is small enough. Inodes are good for storing metadata such as file size, creation time, access time, modifications, etc.



Source: <https://flylib.com/books/en/2.849.1.103/1/>

- File Allocation Table – FAT32 uses this indexing method for their data. The general design for the file allocation table is to have a table that holds a list of blocks and map the capacity of the disk.



Source:

<https://www.cs.odu.edu/~cs471w/spring10/lectures/FileSystemImplementation.htm>

6.6 PegasOS File System Design

We have decided to use the Ext2 file system for our operating system because it is a file system already made and can be used for the Raspberry Pi 4. The main reason for choosing Ext2 over FAT32 is because FAT32 has many limitations when compared to Ext2 such as file sizes are limited to 4GB.

With every storage device such as a hard drive, flash drive, etc. the space that those drives have gets formatted and divided up into small blocks. The size of those blocks depends on the page size that the processor can support. For the Raspberry Pi 4, it uses the ARM processor and the smallest page size it can support is 4KB and the largest is 64KB. For our operating system, we are going to format the SD card with Ext2 and divide up space with 4KB blocks of space as that is the smallest it can go.

The main reason the blocks must be the smallest available is that the project will be installed on a 16GB SD card and using smaller blocks can help use up more space. With blocks in disk space, when a computer stores files it uses the blocks in the disk and the data in the file is stored in those blocks. When the data is too much for one block, the file system goes to the next available block and stores the data there, when it finishes storing the data the block is then marked that it is the end of the file.

The main problem that happens with storing data like this is that the rest of the block is unusable for other files to use regardless of how much free space is available in that block and using bigger blocks can result in a large amount of wasted free space that can't be used. For example, if a file that is going to be stored is 9KB in size and the file system is using 8KB blocks to store data, the first free block will be filled and stores the remaining data to the next free block but only a small amount is going to be used and the remaining space in the block can't use. So, with smaller blocks, there will be less wasted space on the disk but the downside is that with smaller blocks it limits the size a file can be, that won't be a problem with 4KB blocks as the file size limit is 2TB with the max volume size being 16 TB.

Each inode has a numerical address similar to blocks and their addresses start with 1. Inodes have two fixed sizes, 128 bytes and 256 bytes, depending on if the extra features are implemented or not. For our project, it will have the standard 128-byte structure which contains all the necessary information for files. All inodes are in the inode tables and they are in block groups. To find an inode is to find the block group it belongs to. With the inode address, which is numerical, the group can be determined by using the formula below:

```
block_group = (inode - 1) / INODES_PER_GROUP
```

INODES_PER_GROUP is a field in the superblock. When the block group the inode resides in is determined, to look up the inode within the block group the starting address of the group must be determined. The location of the inode in the block group can be found by using the formula below:

```
index = (inode - 1) % INODES_PER_GROUP
```

Then once the index is found, the block the inode is in can be determined with using this formula below:

```
containing_block = (index * INODE_SIZE) / BLOCK_SIZE
```

Inside each inode has all the information that is needed about a file such as a file type, size, number of blocks allocated to it, time information such as when it was created, modified, or last accessed, and also has the information about where the blocks of data reside on the disk storage. This is referred to as the metadata and any information that isn't user data is called that.

One of the most important things about the inode is the use of pointers. Pointers refer to one block in the disk and the pointer itself is about 4 bytes and does not contain any data. Each inode has 12 pointers that directly point to the block that contains the data and in the case of the file being too large for those 12 blocks, the inode will use indirect pointers to get more space. Indirect pointers are used if a file is too big, they point to a block but instead of it containing data they contain more pointers with each pointing to a free block that will contain the data.

The entire block the indirect pointer is pointing at will be full of pointers, with 4KB blocks and each pointer being 4 bytes which means the block will contain 1,024 pointers. This adds to the file size limit to $(12 + 1024) * 4KB$ which equals 4,144KB or just over 4MB. If the file size is still too large, then a double indirect pointer is needed. It works similarly to an indirect pointer except that those pointers in that block point to blocks that contain pointers to blocks that contain the data. This drastically increases the file size limit to $(12 + 1024 + (1024 * 1024)) * 4KB$ which equals about 1 million 4KB or just over 4GB. The file size limit can be increased even further with triple indirect pointers which work similarly to a double indirect pointer but another degree of pointers. This adds the file size limit to $(12 + 1024 + (1024 * 1024) + (1024 * 1024 * 1024)) * 4KB$

which equals to well over 4TB but in Linux the file size is capped at 2TB. With Ext2, there are no worries about reaching the file size limit or any other size limitations.

With file systems, there is a flow of operations when reading and writing a file that even with small files it can be complicated. For example, within Ext2 there is a file that is 12KB in size so it will take up three blocks on the disk. The “open(“/folder/file”)” must be called but the file system must find the inode that contains the file to get some basic information about it. For the file system, it must traverse the pathname and locate the needed inode. All traversals begin at the root of the file system, the root directory or root inode. The inode number, or its numerical address, for the root inode is always 2 and from there the process of reading the file begins.

Once the root inode is read in, the file system can look to find pointers to blocks. It will read through the directory to find the entry of the folder. By reading in more directory data blocks it will find the entry of the folder then get the inode number for it. After that, the file system will read the block containing the inode and then its directory data to finally find the inode number of the desired file. The final step of “open()” is to read the file’s inode to memory. When the file is open, the program can issue a “read()” system call to read the file. The first read will read the first block then update the last accessed time and the in-memory open file table for the descriptor. The read updates the file offset so the next read can read the second block, etc. Each of these systems calls like “read()” and “write()” generate I/O.

Writing to a file follows a similar process, the file opens then “write()” system calls are issued to update the file. Unlike reading a file, writing to a file might allocate blocks. When writing a new file, each write call not only has to write data to the disk but has to decide which block to allocate for the file and update other structures of the disk-like the inode and the data bitmap. For this, it will generate more I/Os than “read()” calls as it has to read and write the data-bitmaps along with reading and writing the inode and the data blocks.

The number of system calls will increase even further with common operations like file creation. To create a file, the file system must allocate an inode for the file and allocate space within the directory containing the new file. The I/O traffic will increase as it has to read to the inode bitmap, write to the inode bitmap, write to the new inode for the file, update the data of the directory, and to read and write to the directory inode to update it.

If there are problems with using Ext2 on the Raspberry Pi 4 then the backup plan is to create our file system similar to Ext2 on how it works and how it is structured, using inodes. Ext2 is more efficient than FAT32 and there is already supporting software that can help mount Ext2 on the Pi 4.

PegasOS

7. Shell

[illegible]

7.1 Shell Background Research

7.1.1 What is a Shell?

A Shell is a program that interprets commands from the user and turns them into functions that the operating system can understand. Every operating system today has some variation of a shell program that while not used as often today, still grants the user a degree of control over the system that the new GUI-based operations don't always allow. And in the olden days of operating systems, this was the only way to interact with the computer itself which was already a step up from what there was previously - which was nothing.

Shells are text-based interfaces, which use the keyboard for input and display text to the screen. Typically they use ASCII and Unicode characters, and occasionally have support for various text fonts. Modern shell programs will also include various hotkeys and controls involving the mouse but are still completely operable with only the keyboard.

Since shell programs are an interpreter between the user and the operating system, it occupies a space somewhere between user programs and the kernel itself. Due to the nature of the shell's translation, it can also be implemented into the kernel directly and act as the user's access port into the operating system and by extension, the kernel itself (given the appropriate permissions).

To ensure that PegasOS is an operating system that the user can interact with, we will be using a shell program to take in user input.

7.1.2 Graphical User Interface (GUI)

From Windows to macOS and Linux all three of these dominant operating systems are built and optimized to where the user, new or old, can pick-up and use due to the great Graphical User Interface made for user interaction. The GUI is such an important aspect of a successful OS since it's how the user interacts with the computer itself, and it is much less complex and daunting than the traditional command line. With GUI-based operating systems the user primarily uses the keyboard and the mouse to interact with different elements, typically called windows, displayed to them. The operating system, in this case, is in an infinite loop waiting for the next input from the user to do a new action. Now creating a working and well-polished GUI is no easy feat, as not only must it work programmable, but it must be visually appealing as well as visually intuitive to the user. Creating a working GUI is almost as much work as making the operating system that the GUI interacts with.

7.1.3 Command Line Interface (CLI)

Before the common GUI was created, the standard platform for navigating through a computer's operating system was the Command Line Interface. The CLI was simply a window where the user would input commands and certain arguments for the command in which the operating system would search through its library for the command given. This approach had a very steep learning curve since the user had to memorize all the commands and the certain arguments for said commands to change how the command will be executed. Although the steep learning curve was a hassle for the users, it was greatly liked for how direct the user could be with the machine. The main usage for this method was not for the ease of the user, but more for handling and executing more programs in a shorter time than previously designed computers. So this is why the CLI was very bare compared to a common day GUI.

7.1.4 PegasOS Interface

Because creating a GUI for an operating system is almost as monumental a task as writing the operating system itself, we are focusing on making the core of the operating system first and foremost. And in order to interact with that core, we will be using the traditional Command Line Interface, which is where our shell program comes into play.

Here is an example of what the interface will look like:



We will be using a Linux-like format for displaying the username and directory information to the user (nicknamed the userline), as follows:

```
john_smith@root/home/desktop/: _
```

Commands are then entered, such as in the following example:

```
john_smith@root/home/desktop/: hello  
Hello there, john_smith! Welcome to PegasOS!  
john_smith@root/home/desktop/:
```

We will also be supporting the usage of ANSI escape codes for colors, and allow the user to change the colors of their userline:

```
john_smith@root/home/desktop/: usercolor  
Enter the user color code: 10  
john_smith@root/home/desktop/: dircolor  
Enter the user color code: 25  
john_smith@root/home/desktop/:
```

7.1.5 Common Shell Components

When operating through an Operating system, it is imperative that it feels smooth and with no errors. So allowing the user different ways to maneuver through it is also important. One key component that allows a user to do so is the Shell or Command-Line. The Shell represents the operating system in the most basic format so it is very straight-forward, yet it has a learning curve to it. The Shell displays a GUI like command prompt which allows the user to enter commands and manipulate the system and run programs within the operating system. Also, it typically displays the current directory the user is in. This is an important piece of information to display to the user, as most operating systems allow users to maneuver through the different levels of the file system, so the user can access specific files and directories. Such cases as when the user is in the current directory and is trying to reach a file in the parent or child directory. Users should be able to enter keywords that map to certain pre-built commands which allows the user to do various operations. Through these operations, the user is able to create, delete, and perform operations on files and other elements within the Userspace.

There are some key features that the shell is able to perform to allow the user to perform operations faster and hassle-free.

Auto Completion:

Users can input a certain amount of a given command and be able to hit 'TAB' in which should complete the rest of the command so long there is only one choice.

Character Insertion:

If a user has something typed out already, they are able to move through what they have written via arrow keys and either delete or insert more characters.

Shell History:

Users can use arrow keys to maneuver through previously entered commands and input them again or edit what they entered.

Piping:

Linux and other Unix-like operating systems allow for multiple commands to be inputted into the prompt by separating each one via '|'. The results of the previous command's becomes a parameter for the next command.

Scrolling:

When the output of the commands are larger than the screen size, the outputs should be saved in a buffer so that the user can scroll up or down and see the previous outputs.

7.1.6 Common Shell Commands

Assoc (Windows)

This command will show which files are associated with the program at hand. This is handy when the user isn't sure which files can be used for the program being used. One can also extend the command to change file associations. For example, "assoc .txt=" will change the file association for text.

Cat (Linux/Unix)

This command will allow the user to concatenate multiple if needed, but also this can be used to create new files, few files, and redirect the output for certain files.

Cd (Windows/Linux/Unix)

Allows the user to change directory from the child to the parent directory or vice versa.

Cls (Windows)

This command clears the command prompt screen clean of all the previous inputs and outputs. Yet this does not remove the previous entered commands from memory.

Copy (Windows)

This command will copy a file from one place to another, or create a new copy of a file in the same location with a different name. Say if a file is named "file" running the command -> copy file file.bak, this creates the back-up file for the main 'file' file..

Del (Windows)

This command simply deletes one or more files depending on the number of files entered in the parameter.

Dd (Linux/Unix)

This command will allow the user to change the file type of a certain file. One neat thing one could do is change a .txt file to.* to whichever language one is using.

df (Linux/Unix)

This command allows the user to see the disk space for the user's filesystem.

Dir (Windows)

This command displays all the files and folders in the current directory.

Echo (Windows)

This command simply prints out the parameters given to it.

Exit (Windows)

This command simply closes the command prompt completely.

Find (Windows)

This command is given a string as a first parameter and a file for the second. It will search every string that contains the string given in the file given.

Driverquery (Windows)

This command will show the drivers that are installed and running on the user's pc. This will help since some drivers can be missing or improperly configured.

Fc (Windows)

This command lets the user compare two different files and show the differences between the two. The user can follow 'fc' with either '/c', '/b', or '/c' for the testing of binary output, disregard the case of text, or just compare ASCII text. This command is very useful for programmers especially since they are trying to differentiate they're output with the desired output

Head (Linux/Unix)

This command will allow the user to view the first few lines of a file. This file is most useful for when trying going through multiple files and the user is trying to find a certain file.

Lsblk (Linux/Unix)-

This command will print out a tree structure of the block devices of the user.

Mkdir (Linux/Unix)

This command will make a new folder from the directory in which the user is in.

Mount (Linux/Unix)

This command will basically mount any device, i.e. USB or SD card, to the User's existing filesystem.

Rmdir (Linux/Unix)

This command simply deletes a folder in a directory in which the user is in.

Shutdown (Windows)

This command is pretty trivial since it does in fact shutdown the pc the user is using. The neat trick about this command is that the user can combine it with "shutdown /r /o" to restart the pc and will launch the Advanced Start Options. This is most useful for troubleshooting purposes.

Systeminfo (Windows)

This command will give you a detailed config. The total and usable memory for the system, initial Windows download date, all hardware plugged into the system, the user's BIOS version, and much more.

Tail (Linux/Unix)

This command will allow the user to view the last few lines of a file. This is useful in conjunction with commands like echo and cat.

Tasklist (Windows)

This command will show all the running programs and the memory used per program. This command may seem redundant since we do not have a task manager which will show the same thing but more user friendly to understand, but the main advantage about this is when the OS is Command Line Interface (CLI) based this is how the user will see it. Also another main advantage of actually using the command over task manager is that there are things running that the manager won't really show and this could be vital for when issues with the system don't seem apparent.

Taskkill (Windows)

This command simply terminates a running program by giving the PID of the program that the user wants to terminate. This command works off the call from the tasklist since the user needs to find the program they want to terminate and the corresponding PID.

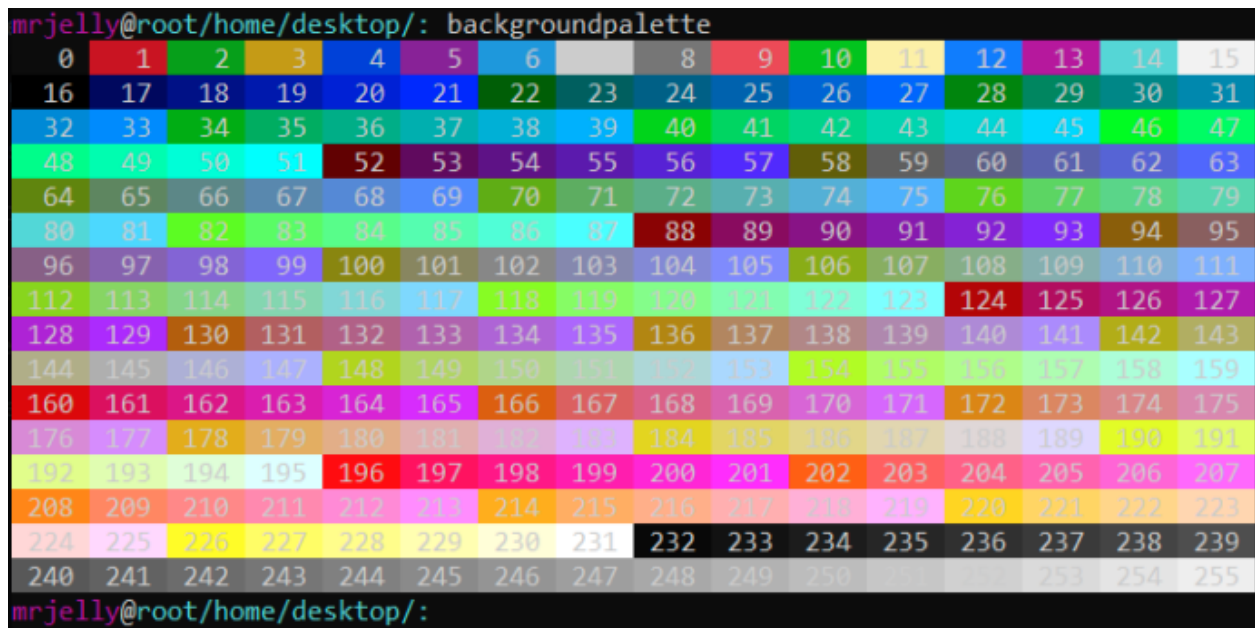
Touch (Linux/Unix)

This command will allow the user to create a new empty file. This is most useful for programmers especially since they need to create new files a myriad of times for developing or debugging purposes.

7.2 Shell Commands for PegasOS

backgroundpalette

This command will print out every possible background color that the shell terminal supports. Background colors fill the space around text in that character's space.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

cd

This command will simply change the directory in which the user is in. The user can move up more than one directory so long as the directory exists.

Implementation: For cd we have similar structure to how Linux handles the changing of directory. If the user simply just put “cd” then it will put them in the root directory. If the User were to input “cd ..” then this will send the User to the parent directory.

clear

This command will simply clear the screen of all the previous commands and outputs. This will basically clear the buffer in which is keeping the outputs of the previous successful commands.


```
guest@root/home/desktop/: copy shell.c
Where do you want to copy shell.c to?
Destination:root/desktop/
So you want to copy shell.c to root/desktop/?
[Yes]
[N]o
Response:y
shell.c was successfully copied to root/desktop/
guest@root/home/desktop/: guest@root/home/desktop/: copy test.txt
Where do you want to copy test.txt to?
Destination:root/downloads/
So you want to copy test.txt to root/downloads/?
[Yes]
[N]o
Response:n
test.txt was not copied
guest@root/home/desktop/: guest@root/home/desktop/: copy test.c
Oof, the file you want to copy does not exist!
guest@root/home/desktop/:
```

createdir

This command will create a new subsection directory within the directory in which the user is in.

Implementation: As stated this will create a sub-directory in the current directory or in the path given as a parameter. Error messages will be given to the User if they are trying to create a directory to a location not permitted, or if they are trying to create a directory that already exists in that path.

createfile

This will create a new file. By default it will be a .txt file if the type of the file isn't stated.

Implementation: There will be a verification question asking the user to make sure they want to create the file. Also, if the file trying to be created already exists, then the User will be prompted if they want to overwrite the current file or cancel the entire process all together.

```

guest@root/home/desktop/: createfile
Please enter the filename and type you want for the file: test.c
Is test.c the file you want to create?
[Y]es
[N]o
Response:y
The test.c file has been created!
guest@root/home/desktop/: guest@root/home/desktop/: createfile
Please enter the filename and type you want for the file: TEST.cpp
Is TEST.cpp the file you want to create?
[Y]es
[N]o
Response:n
The TEST.cpp file was not created!
guest@root/home/desktop/: guest@root/home/desktop/: createfile
Please enter the filename and type you want for the file: real.txt
Oof, the file you are trying to create already exists
Would you like to replace that file with this new one or no?
[Y]es
[N]o
Response:y
The real.txt file has been replaced!
guest@root/home/desktop/: guest@root/home/desktop/:

```

currentdir

This command will show which directory the user is currently in.

Implementation: This command is very helpful for maneuvering through the directories of the OS. If the User does get lost then they will be able to see the contents in the directory and also which one they are currently in. The example below is what a user would see when inputting the 'currentdir' command. For the beginning version of PegasOS, we probably won't include the amount of bytes in total there are in the current directory.

```

Giancarlo@root/home/desktop/: currentdir

Current contents of your directory are:
04/18/2020 03:17 PM          412 ascii_printer.c
04/18/2020 03:17 PM        3,425 cog_ascii
04/18/2020 03:17 PM        3,425 cog_ascii.txt
04/18/2020 03:17 PM        4,448 hashTest.c
04/18/2020 03:17 PM        6,130 pegasus_ascii
04/18/2020 03:17 PM        3,466 pegasus_ascii2
04/18/2020 01:37 PM          49 README.md
04/18/2020 06:29 PM       21,233 shell.c
04/18/2020 06:29 PM       57,995 shell.exe
04/18/2020 03:17 PM        3,880 tableofcontents_notes.txt

Giancarlo@root/home/desktop/:

```

delete

This command will delete the file given as the parameter.

Implementation: There will be a verification question asking the user to make sure they want to delete the file.

```
guest@root/home/desktop/: deletetest.c
Are you certain you want to delete test.c?
[Y]es
[N]o
Response:y
test.c was successfully deleted!
guest@root/home/desktop/: guest@root/home/desktop/: deletetest.c
Are you certain you want to delete test.c?
[Y]es
[N]o
Response:n
test.c was not deleted
```

deletedir

This command will simply delete a directory passed through as a parameter.

Implementation: Like described above, this command will simply delete a directory. If the target directory does not exist then an error message will be displayed to the User. Also during the process of deleting the directory, the User will be prompted to verify if they want to continue with deleting it. This is done to help the User to make sure they are deleting the right directory in case they put the wrong path in the input.

echo

This will simply print out to the screen the string given as a parameter.

Implementation: Very basic and simple command, but was added for the User to see whether they are inputting commands correctly. Also, it's a common command across different operating systems so it is only fitting that we include this one.


```
guest@root/home/desktop/: echo well, hello there
well, hello there
guest@root/home/desktop/: echo It is yet anohter day in paradise
it is yet anohter day in paradise
guest@root/home/desktop/:
```

df

This command will allow the user to see the remaining disk space for the Pi (SD card).

Implementation: This command is very helpful since it will allow the User to see the current available space that can be used. I can also show all the downloaded contents that are in the disk space and their file size.

find

This command will display in which sub-directory the file is, given by the parameter.

Implementation: This command will give an error message if the User is trying to find a file that does not exist.

```
guest@root/home/desktop/: find test.c
test.c does not exist in the system
guest@root/home/desktop/: find shell.c
shell.c is located in root/home/desktop/
guest@root/home/desktop/:
```

head

This command will allow the user to see the first few lines of the file given file from the parameter.

Implementation: While the overall usage of this command is pretty straight forward, we are adding a little kick to it. The user will have the capability to add another parameter after giving the filename in which it will be an integer value that can be {1-(length of file)}. This parameter will be used to select n number of lines to be displayed to the screen. A warning will be given when the parameter given has exceeded the length of the file, but it will display the entire contents of the file in the end.

```
guest@root/home/desktop/: head myshell.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <windows.h>

#define MAX_INPUT_LENGTH 256
#define MAX_DIRECTORY_LENGTH 1024

#define CLEAR_SIZE 128
#define DEBUG_DEFAULT 0
#define TOTAL_USERS 10
guest@root/home/desktop/:
```

hello

This command is pretty straight forward for inputting this command simply prints to the screen “Hello there, guest! Welcome to PegasOS!”. As one can see, it has guest as the default but upon logging in it will put the user’s name there.

```
john_smith@root/home/desktop/: hello
Hello there, john_smith! Welcome to PegasOS!
john_smith@root/home/desktop/:
```

login

This command will allow the User to login into their account in the OS or create a new account if they have no account. Once logged in, the User will then be prompted.

Implementation: This command was discussed in full detail in the “User Subsystem” section in which operations can be done and to what extent the User can do. As stated above, this command is one of the most important ones in the matter that it allows the User to login or create an account and then in turn be granted certain permissions to do a variety of operations.

mount

This command will mount any I/O in the Pi to verify the connection is good for reads and writes.

Implementation: Overall this command is quite important for the aspect that it will verify that our drivers are working to the specification needed to maximize the reads and writes to devices.

move

This command will move a certain file from one directory to another.

Implementation: This command is pretty straightforward in its use, there is some clarification needed. Like described above this command will simply move an existing file into the designated destination of choice. There will be an error given to the User if they are trying to move a file that doesn't exist. Later we may implement the choice of creating that new file if none exist. Also there will be an error message given if the path they are trying to move the file to doesn't exist.

```
guest@root/home/desktop/: move shell.c
Where do you want to move shell.c?
Destination:root/desktop/
So you want to move shell.c to root/desktop/?
[Yes]
[N]o
Response:y
shell.c was successfully moved to root/desktop/
guest@root/home/desktop/: guest@root/home/desktop/: move test.txt
Where do you want to move test.txt?
Destination:root/downloads/
So you want to move test.txt to root/downloads/?
[Yes]
[N]o
Response:n
test.txt was not moved
guest@root/home/desktop/: guest@root/home/desktop/: move test.c
Oof, the file you want to move does not exist!
```

power

This command will enable the user to simply power off the Pi.

Implementation: Even though the most optimal way to power off the Pi is by simply disconnecting it from its power source, we believe by having an integrated power off command it will help negate any chances of errors when powering off by the plug.

systeminfo

This command will for the time being will display to the user all the usable memory, certain downloads, and version number.

Implementation: This command works kinda similar to TaskList and FileSpace where this command will display to the User the amount of available memory(RAM), the downloads that have been done since the last time being powered off, and the version number for PegasOS. There will be other features to be added in the future once more components are added to the OS. In later versions, this will be helpful in terms that it will display to the User if they are on the latest version of the OS.

tail

This command will display the last few contents of a given file.

Implementation: This command will be very similar to that of Head. There will be an extra feature to Tail in that the User will be able to input an extra parameter which will be an integer value. This integer value will be used to grab n number of lines starting from the end of the file. The integer will be verified that it is between {1-(length of the file)}. If the integer given is greater than the file length then a warning will be given to the user yet it will display all the contents of the file. An error message will be given if the User inputs a value less than 1.

```
guest@root/home/desktop/: tail myshell.c
// Clear out stdin for next round
if (feof(stdin))
{
    while (!feof(stdin))
        throwaway = fgetc(stdin);
}
while (!exitFlag);

return 0;
}
```

tasklist

This command will display all the running tasks, their PID, and their memory consumption.

Implementation: This command is quite important since we will be running a CLI compared to a GUI for PegasOS. Like described briefly above this will display all the running programs and executables in the background. The need for this is for when a certain program isn't responding anymore as planned then there is the need to terminate it via PID or name. Another use for this command is to see the amount of memory that is being used

per program. Since the Pi is only running on 2-4GB of RAM it is imperative that we conserve the amount each program uses.

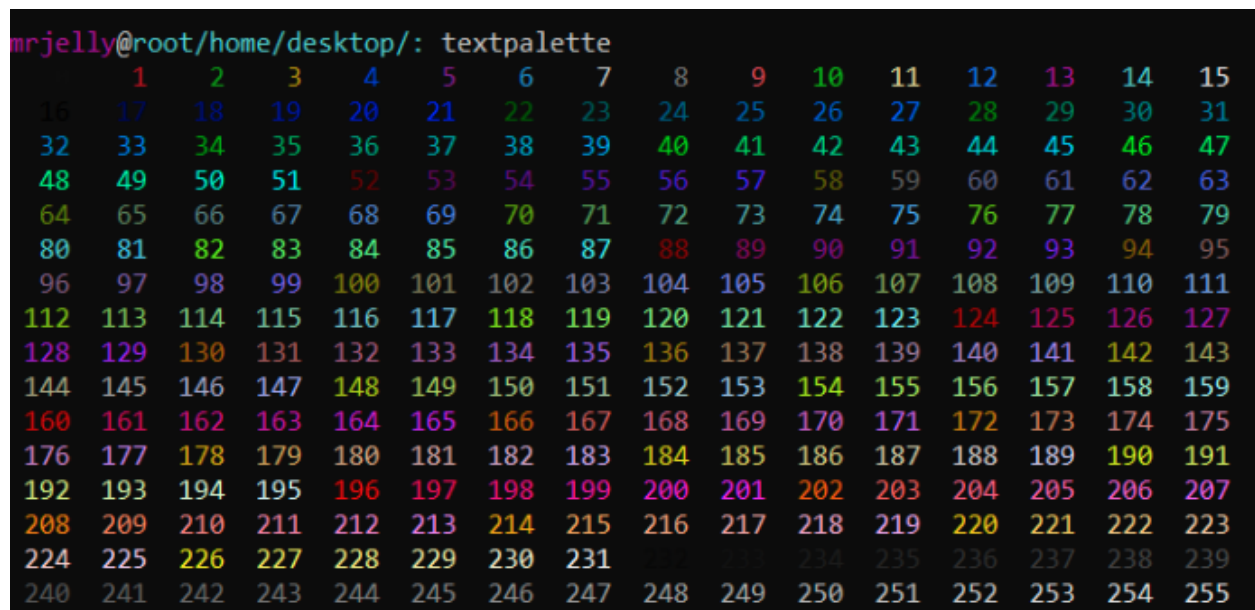
terminatetask

This command will simply end the task via PID given as a parameter. This command works in conjunction with Tasklist since the user will need the PID from there for the parameter.

Implementation: This command works in conjunction with TaskList since we need to find the name and PID for which the program needs to be terminated. Once calling this program with the corresponding PID/Name there will be a prompt asking if the User is sure about ending the program. A success message will be given when the program has been successfully terminated.

textpalette

This command prints out all of the available colors for text in the shell terminal.



uninstall

This command will allow the user to delete any downloaded stuff to the Disk Space.

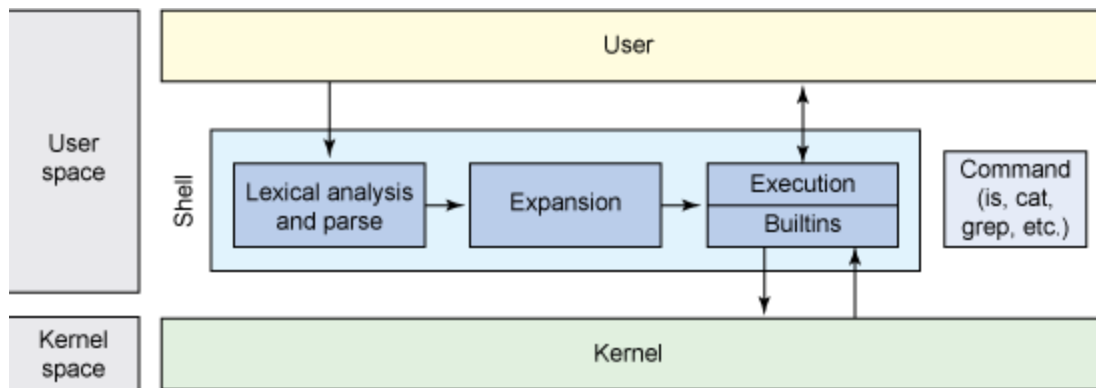
Implementation: This command is pretty important for the reasons that PegasOS is at the moment CLI based. The command does what the

name entails, it simply uninstalls any downloaded content to the User's PC so long as the contents aren't key components to the OS.

These commands were selected for the final cut for the initial design for the necessity for maneuvering through PegasOS and being able to use the Pi to its fullest potential. There will be more added after later versions of this Operating system when more applications are able to be done by the Pi. Also since the Pi keeps getting updates and newer versions, there will be more updates for PegasOS as well.

7.3 Basic Shell Architecture

Here we have the overall layout in which a shell will take in input(s) from the user and in which will then be parsed and bundled up with other executables, and once that is finished it is sent to the Program Execution cycle in the Kernel.



7.3.1 Lexical Analysis

The Lexer will simply take in a input stream from some source ie. the user or some program and then will break up the input into a sequence of tokens. The lexer will look at each character and classify them to some token or a group of characters which will be a lexeme and then assign that lexeme its token. Such a token can be “add” for ‘+’ or “print” for ‘printf’ about C. A token is an object for which it represents the lexeme it was mapped to and these tokens can in turn have their classification: Number, Identifier, or Operator. Some tokens can have other attributes such as which row and column in which the token was seen in the code.

The method in which the lexer can recognize the tokens from the input stream is by running based on some sort of predefined grammar. A lexical grammar is simply the rules in which a group of characters can be categorized and have more distinct character cases. One example, the “identifier” token can be represented by ‘(letter | _)(letter| digit| _)*’ So for the identifier token it is expecting a letter first followed by more characters or numbers until it sees a space which ends that instance for the token. If the lexer doesn't get the following characters to fulfill the following regular expression, then it will give an error message.

A regular expression is the small component of a grammar that holds the rules for one set of token aspects. Like a “Letter” regular expression can be in the form of [a-z,A-Z]. The overlying main concept for this to work is thanks to a concept of finite state automata/machines. These finite state machines are a graph/ control flow for

characters to be interpreted by the lexer. It will have a set starting state where the first character would “enter” and then would continue along the path of accepting states so long as the next coming character is valid in the state machine. The characters will continue to flow in and make their way through the accepting states until they hit the terminating state which will give a success to the lexer and then create the token. If the current string never hits the terminating state then it will terminate it on its own and issue an error. When all of the input from the stream has been lexed and then tokenized it will be sent to the parser for further inspection.

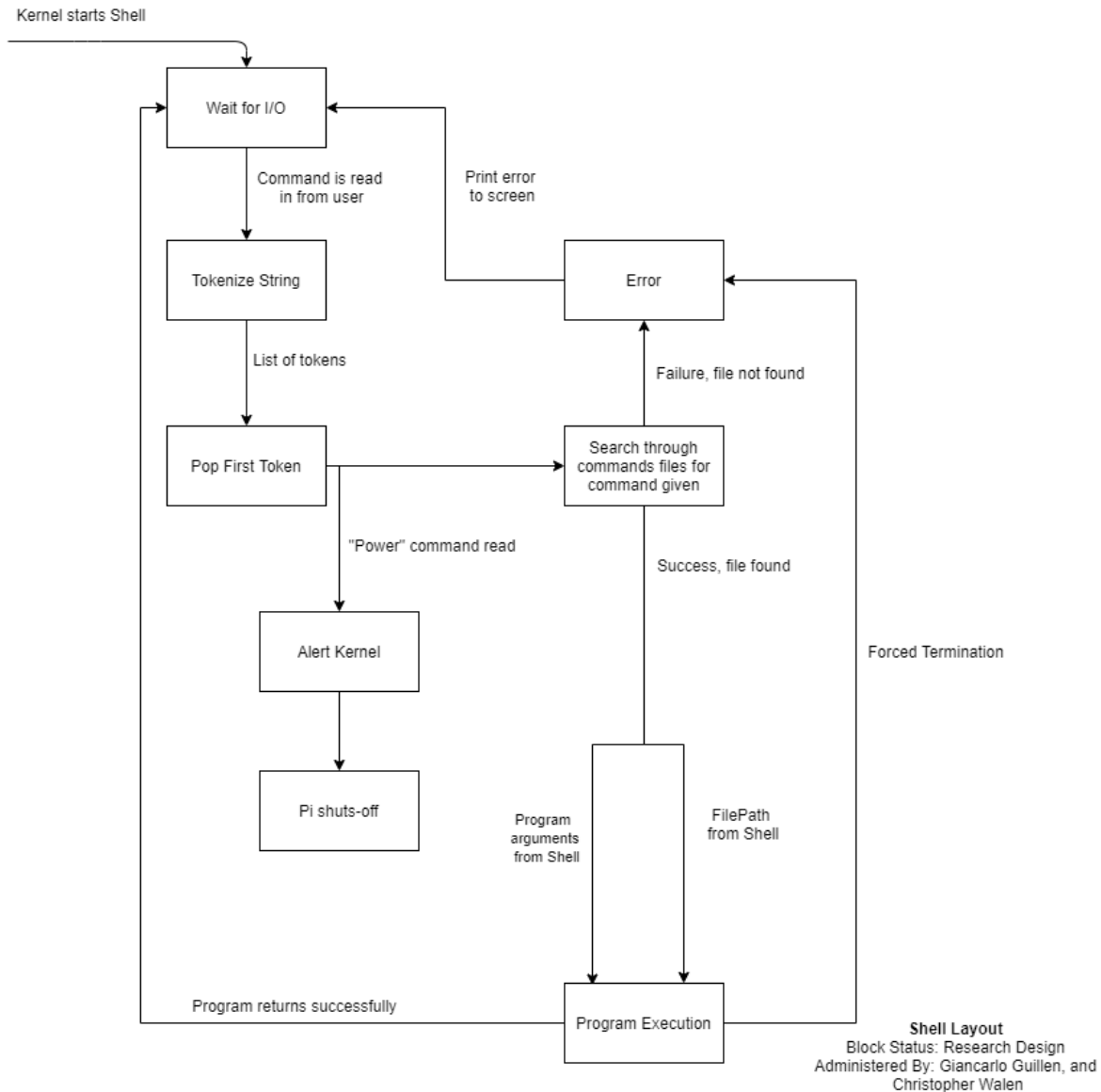
7.3.2 Parser

The Parser works very closely to the Lexer but not enough to be the same component overall. While the Lexer simply took the input stream and created tokens representing all the characters/strings read in, the job of the Parser is to look at each token and make sure they follow the Grammar set so that the token makes a working error-free program. The overall concept for the Parser is that it reads a token and it will verify that the following tokens are valid and fit for the current section of the Grammar. The following Grammar representation below is something similar to what grammar we will be following when Parsing the incoming inputs.

```
letter      = [a-zA-Z]
digit       = [0-9]
digits      = digit digit*
identifier  = letter | (letter | digit | _)*
fraction    = . digits | ε
exponent    = ((E | e) (+ | - | ε)) digits) | ε
number      = digits fraction exponent
operator     = + | - | * | / | > | >= | < | <= | = | ==
parenthesis = ( | )
```

One thing to clarify and explain is the idea and usage of a Grammar and how it is represented. The image above this represents a very similar grammar that was used in the Lexer to create the tokens. The only difference is that we only used a fraction of the presented grammar since all we are looking for are the commands and the files and other associated parameters for that given command. So in cases where we read in the tail command, we would be expecting a file as a parameter rather than say a path to a directory.

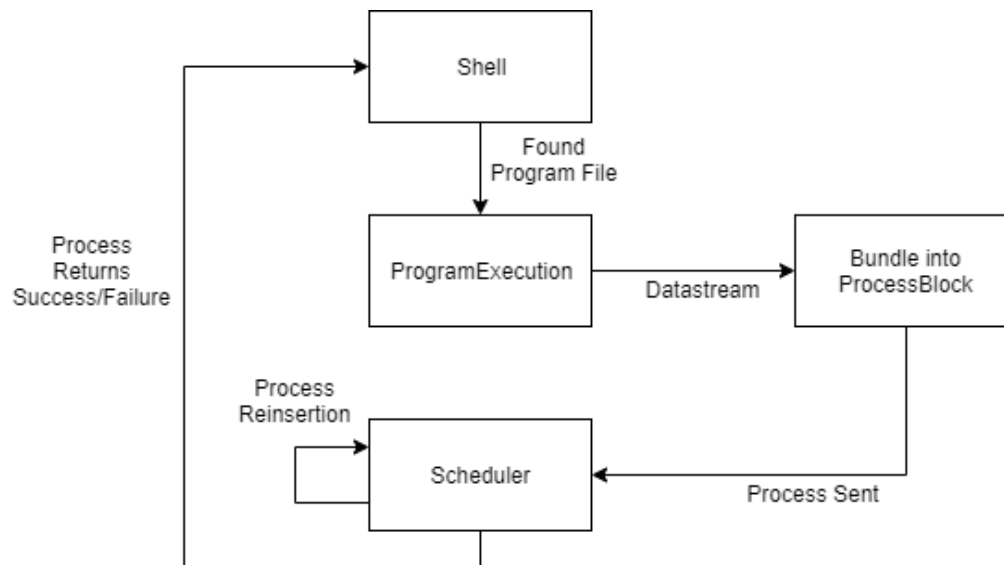
7.3.3 Shell Architecture for PegasOS



Once all the booting has been completed and the kernel has done the system check, the kernel will then call the shell to be executed. The shell will be running until the user inputs a command with parameters if needed. Once the input has been read in it will be sent to be tokenized via Lexer/Parser. It will tokenize the command and then the following parameters for the given command. Once the tokenizing has been completed, we add them to a stack or queue like data structure and pop the first token which is the command token.

We first compare it to the command keyword “Power” and if the token matches the keyword it will then send an alert to the Kernel which will then shut the Pi off. The other case is that the keyword isn't the same then we send the token to be compared to the command file names in our directory. If there is a file that matches our command, then we progress to send the parameters and the file path to the ProgramExecution Block which will be bundled up and sent to the Scheduler to be executed by the CPU. If a file cannot be found for the command given or it fails when the CPU tries to execute the command then it will display an error to the screen and then wait for another input from the user. The same happens when the CPU runs the command and is successful in doing so since it will display the result to the screen and then wait for the next input from the user.

7.3.4 Process Flow



Process Flow

Block Status: Research, Design

Administrated By: Christopher Walen, Giancarlo Guillen

Here we have the Process Flow block diagram that will come from the Shell execution path and will then enter the ProgramExecution once we find the associated command file for the command entered through by the user. From here we send those components to the “Bundle into ProcessBlock” block. In this block, we create a struct that will contain the command file, the parameters, the pc clock in which the CPU will return to once completed, and other necessary information for which the CPU will need

for execution. Once all this is complete we send it to the scheduler which will do its job and send it to the CPU to be executed. It will continue to send it to the CPU until it completes it and once the task is done it will send back either a success or failure to the user.

7.3.5 Process Flow for Shell

Before going into full detail of how the shell will be operated and completed, there is one step that will be explained first. In the Linux operating system, the format in which each command is done is quite unique and actually intuitive for its purpose. When a user inputs a command and its parameters, the shell of Linux takes that command inputted and then goes into a sub-directory folder where the shell file is and looks for a file that matches the command given. Once the file has been found, it then sends the parameters given by the user as inputs for the matched file.

The reason for explaining this is the fact that we are using this format for executing each and every command that a user can use in PegasOS. The main reason we wanted to implement this is that it will allow for easier updates for the OS when adding new commands since the shell should work for every command once launched. Of course, there would be instances that we will need to change things in the main shell file.

Once the kernel has completed all the systems checks for each component and every driver, the kernel will then launch the shell to start to run. The program file running the shell will begin, first it will create all the structs and macro variables needed to perform all the necessary actions. Such macro variables will be:

```
#define INPUT_BUFFER_SIZE=256;  
#define PAST_COMMANDS_BUFFER=100;
```

The macro variable INPUT_BUFFER_SIZE has the value 256 for it holds the input string that the User inputted for a command to be completed. The value of 256 is used for the length of the char array. We used 256 for it is the standard length for input strings.

The macro variable PAST_COMMANDS_BUFFER is used to set the max length for the Struct List which is a Linked list. This list is used to hold all the previously inputted commands the User has done.

Structs that will be needed are:

```
Typedef struct token{  
    Char input [INPUT_BUFFER_SIZE];  
    Struct Token *parent;  
    Struct Token* next;
```

```

    } Token;

    Typedef struct List{
        Token* head;
        Token* tail;
    } List;

```

The usage of the struct Token is very important. The struct itself holds a character array of length 256 and a pointer to another Token called next. The character array simply holds the inputted string from the User. We hold the entirety of the string here and then later “tokenize” the string so we are able to separate the commands and the parameters. The pointer to the Token called Next will simply point to the next token in line, in this case, it would point to the token that is created after it. The pointer to the Token called parent is very important for being able to traverse back up the list.

The main reason we needed to add this component to the struct is that when the User is scrolling back up to view the previous inputted commands it would be easier and more time-efficient for the child node to be able to “look” back up to the parent pointer. Like mentioned, having this pointer is way more time-efficient since we would only have to do one line of code, while if we didn't have that pointer we would have to go to the head of the list and then traverse down till we find the parent Token. Doing that would be completely time inefficient since the length of the List could be arbitrarily long at some instances which would then slow down the usage of the OS when simply doing a simple scroll up operation.

The usage of the struct List is pretty straightforward. It has the same structure of a linked list that has a pointer to the beginning of the list, head, and also a pointer to the end of the list, tail. The majority of the time we will be inserting token pointers to the end of the list so that way they are in order from first to last. Having a tail token pointer is very useful because it will make the insertion to the tail a lot easier and faster since we won't have to start from the head and traverse our way to the end. The head Token pointer is needed for instances in which we need to maneuver through the list from the beginning.

Once these essential variables and structs have been defined, we move on to defining vital functions that will be needed to do certain operations to prep the incoming strings that will be sent off to a different file to which matches the corresponding command given by the User. As described earlier through the Shell Layout Diagram, if the shell program is unable to find any corresponding file that matches the command given, then an error message will be displayed to the User thus asking them to enter their previous input but to put the correct command.

Such functions that we'll be using in the main shell program are:

```

Int getCommand();

```

```

Void sendToCommandFile();
Void scroll();
List* createList();
Token* createToken();

```

For the function, `getCommand()`, will take a Pointer to a token and return an integer value. Once entering the function, we will copy the character array to a local array in the function to keep the integrity of the original array so no changes can be made to it by mistake. Once copied successfully, we will run through a giant string compare if-else statements to find a matching if statement to the command found in the array. Once an equivalent string compare was found, we enter the corresponding if statement which will then return a certain integer which will correspond to that command. If no string compare equivalent is found, then the function will default and return a 0.

The following table will show all the corresponding integer return values to each of their commands:

Command Name	Integer Value
backgroundpalette	1
cd	2
clear	3
concat	4
copy	5
createdir	6
createfile	7
currentdir	8
delete	9
deletedir	10
echo	11
filespace	12
find	13

head	14
login	15
mount	16
move	17
power	18
textpalette	19
systeminfo	20
tail	21
tasklist	22
terminatetask	23
UnInstall	24

For the function `sendToCommandFile()`, we will take a pointer to a Token and an integer value. The integer value should be equal to the corresponding command value set though the `getCommand` function. Once entering the function, it filters through the character array so that way it can separate the parameters from the command name and then store it in a local character array, and then it is taken straight into a giant switch statement which will be comparing the integer value passed as a parameter. When the appropriate integer is found, it is taken into its case statement. In the case statement, it will be sent off with the character array to where the process flow will find the corresponding file for the command.

For the function `createList()`, it will simply allocate memory and create a new Linked List with both the head and tail Token pointers set to NULL. Once that is done, it will return the address to the newly created Linked List.

For the function `createToken()`, it will simply allocate memory and create a new Token with the character array, and both Token pointers set to NULL. Once that is done, it will return the address to the newly created Token.

Once all of these functions are set and defined, we enter the `int main()` function. In this, we will create variables and structs then enter a while loop that will only ever end when either the power is turned off or the command power is given by the User. In the loop, the previously mentioned algorithm will be running by getting input from the User and then sending it off to the corresponding command name file.

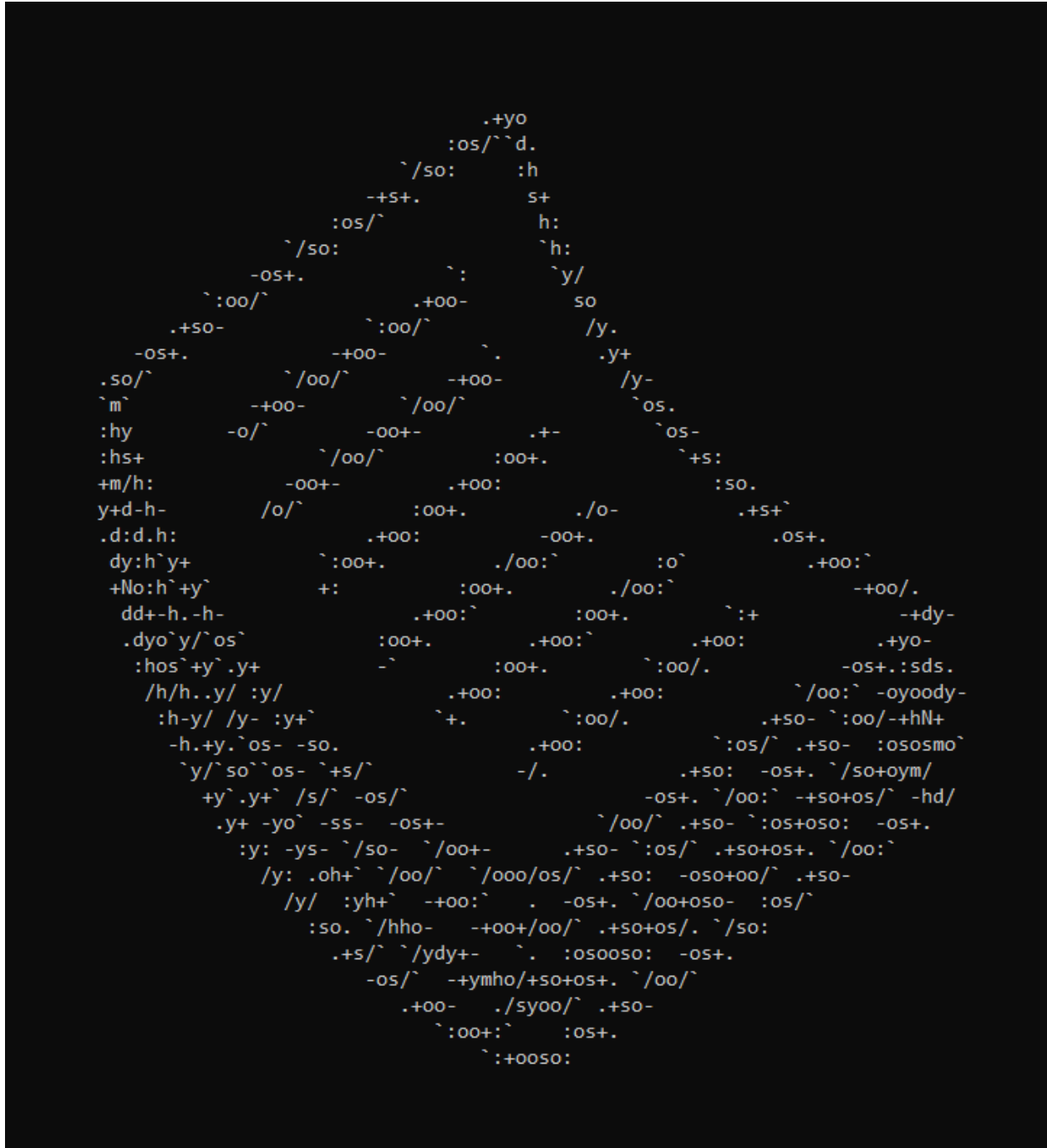
7.4 Shell Interactions with Kernel

The way the Shell interacts with the Kernel is mostly through system calls and software interrupts that have been clarified previously in the System Calls section in this document. In this case, the system calls will be the majority of the commands stored in the file that each command represents. There will be calls to check if certain files, directories, or drivers, etc. are present before the rest of the code in the file can execute.

Due to the nature of how closely the Shell is working to the Kernel, it is essentially a component of the Kernel. The number of system calls and kernel subsystems that the Shell must interact with means that it must be developed alongside the Kernel's development, as the Shell is doubling as our interface and as a user-operated extension of Program Execution. A primary concern is the communication between the completion or termination of a process and the resulting error or termination codes and the passing of control back to the Shell. This can be more easily solved if Shell is a component of the Kernel with direct connections to the Scheduler and Program Execution.

PegasOS

8. Administrative Content



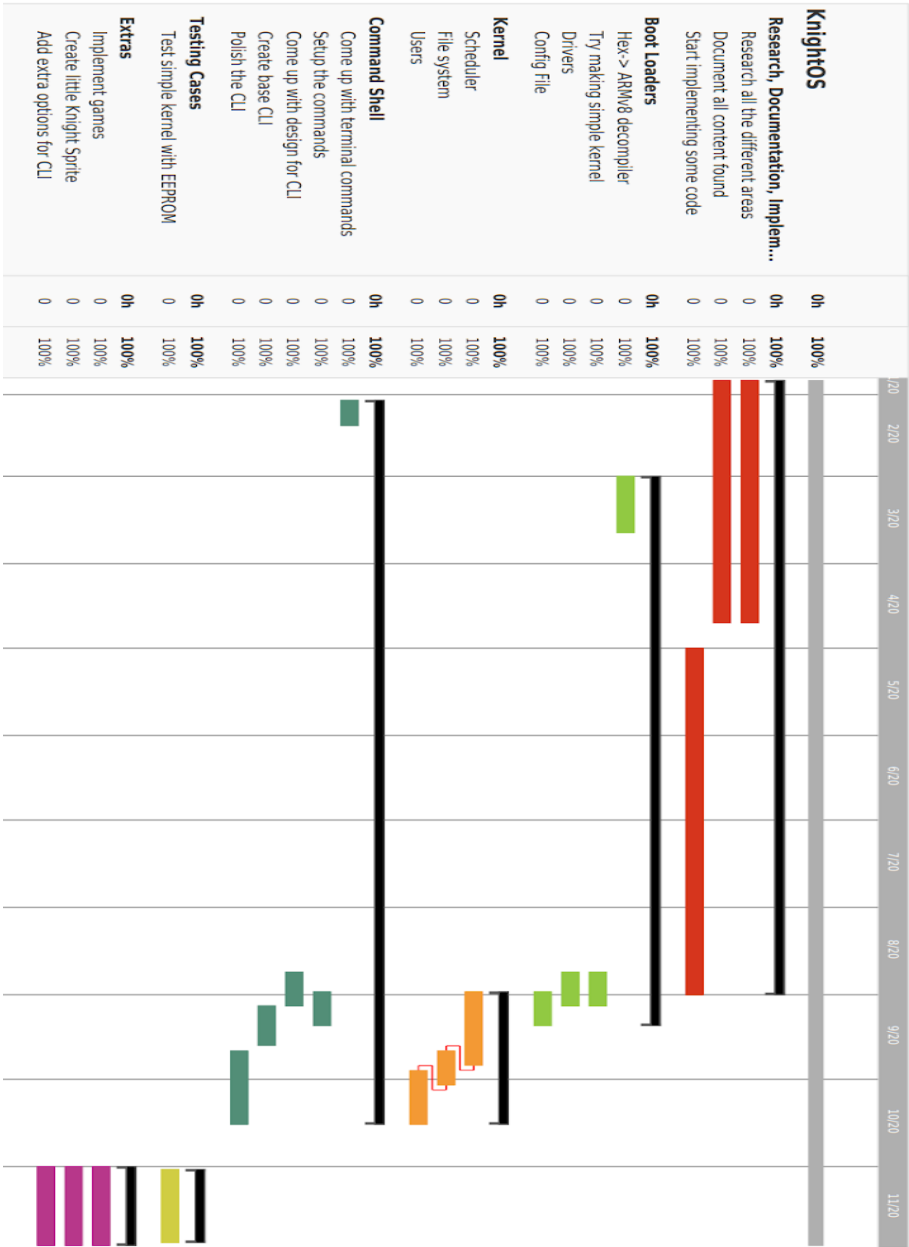
8.1 Budgeting and Finance

Since our group doesn't have any sponsors for our project, the team fully funded this project by themselves. This somewhat limited our budget since our group is made up of college students that either work part-time or don't have a job.

We had no predetermined budget for our group, so we want to keep the budget as low as possible. Each team member either bought themselves only the necessary components like the Raspberry Pi, micro HDMI, an SD card or they bought the Raspberry Pi Kit that contains the necessary components plus more like heatsinks and a case. It should be noted that this budget is just estimated since some of our teammates already had a Raspberry Pi before picking out this project. This table shown below contains the estimated finance for this project.

Project Budget			
Item	Price	Quantity	Total Price
Raspberry Pi			
Micro HDMI cable	\$7	1	\$7
Micro SD card	\$9	1	\$9
Raspberry Pi Kit	\$75	3	\$225
Total			\$241

8.1.1 Initial Project Gantt Chart



8.2 Project Roadmap and Milestones

8.2.1 Spring Semester

January - The team is selected, and initial meetings have taken place. Weekly meetings have been scheduled, and the majority of project hardware has been acquired.

February - Project-specific teams are assigned, and specific research areas for each team are assigned. Research Collection and Design begins. Prototypes for smaller designs as needed are created. All hardware has been acquired for the project. Development environments have been set up for testing, prototyping, and implementation purposes.

Important Dates in February:

Feb. 3, 10, 17, 24 - Weekly Team Meetings for Progress, Research, and Documentation

Feb. 7, 2020 - Scheduled Check-In with TA Eric

Feb. 12, 2020 - The Initial Design Document is due

Feb. 24, 2020 - Project Status Presentation with Dr. Heinrich or Dr. Leinecker

March - Detailed designs for components in the operating system are nearing completion. 60% of the predicted research and design for the project has been completed and recorded by the end of the month. Implementation may begin alongside further prototypes.

Important Dates in March:

March 2, 9, 16, 23, 30 - Weekly Team Meetings for Progress, Research, and Documentation

March 2, 2020 - Individual Turn-In of 15 page contributions to Final Design Document

March 30, 2020 - Weekly Team Meeting plus Contribution Check-up

Week of March 30, 2020 - Scheduled Check-In with TA

April - The design of the project is 80-90% complete, with flexibility to reflect sudden changes over the summer months.

Important Dates in April:

April 3, 2020 - Designs are Due for Components

April 6, 2020 - Final Weekly Team Meeting for Progress, Research, and Documentation of the Spring Semester

April 13, 2020 - Tentative Team Meeting slot if necessary

April 17, 2020 - Final Design Document is printed and bound

April 21, 2020 - Final Design Document Due

8.2.2 Summer Semester (Original Schedule)

May/June/July - Full speed ahead on implementation as time allows. Iron out kinks in designs that become apparent through implementation. Modify designs if need be, and update Design Document/GitHub to reflect any changes made and why. Ideally, at least 20% of the operating system will be done by the end of the summer.

8.2.3 Fall Semester (Original Schedule)

***Note: Specific due dates for Senior Design 2 are unknown at time of writing

August - Continue work on implementation, with adjustments made for Senior Design 2.

September - At least 40% of the operating system is done by the end of the month.

October - At least 70% of the operating system is done by the end of the month.

November - Final work on the operating system is completed, as well as any final adjustments made to the Final Design Document.

December - The operating system is bundled up as a release, the Final Design Document is printed out and bound. The presentation of the operating system is practiced and given.

8.3 COVID-19

No one could have predicted the impact that COVID-19 would have on our school and our day to day schedules, and as such, plans have changed dramatically as the team and school have adjusted to the new circumstances. As such, we have changed some aspects of the Project Schedule, which we will list below. As updates about our situations unfolded, and as the stay-at-home order was made, work on the project was temporarily postponed as we all adjusted to the new circumstances. While we do not know when our routines will return to normal, we can do the best that we can and keep ongoing.

Please keep in mind that the document will undergo more changes as more features and developments arise from the implementation of the project into source code. On the following two pages we have detailed the adjusted project schedule, and details about the new schedule.

8.3.1 Remainder of Spring Semester

Finish all sections of the document that have been addressed through our current waves of design and brainstorming. Address key components and issues that may have been missed in previous meetings, and fill in as much information as possible that is necessary to include. Final formatting will be completed at the beginning of the week that the final copy of this document is due, wherein the Table of Contents will be updated to reflect the current state of the document.

8.3.2 Summer (May - Mid August)

After the remainder of the primary version of the document has finished, we will hold a 'mid-term' review of the current state of PegasOS's design. Due to the size of the project, this will be necessary to ensure that everyone on the team has a grasp of where we need to go from here as a whole. At the end of this review, we will assign the first wave of tasks to start being implemented. Documentation will be copied and moved to a sister GitHub repository that will link to the source code.

While we are hopeful that normal day-to-day routines will resume over the Summer, we must assume that this will not be the case. Reports are inconsistent about a date for a 'return to work', however since online education will be continuing through

the entire summer semester, we must also assume that normal routines will not resume until the earliest August. For this reason, we are moving to completely online interactions for meetings until the end of August.

Every week we will have a voice meeting through our Discord channel, screen-sharing and video-sharing where necessary. In these meetings, we will be discussing the current state of the project's design and implementation. Updates on current work will be presented, and tasks will be delegated and assigned based on completion of previous tasks. As changes are made to the design through issues arising in the implementation, the documentation will be updated to reflect these changes. During these updates, new information may be introduced into the document.

The milestones that we hope to achieve during this timeframe are:

- Boot into the simple kernel
- Display text through GPIO
- Display text through HDMI
- Implement File System
- Implement Memory Manager
- Implement Device Drivers

8.3.3 Fall (Late August - Early December)

Once we reach the beginning of the Fall Semester, there will be changes to the project schedule to accommodate the team's school schedule, as well as the schedule of due dates (if any) as needed by the Senior Design II class. We are hopeful that the number of due dates will remain minimal so that we can continue to focus full-speed on the implementation of PegasOS, as this timeframe will essentially be the second half of the programming required.

An important thing to note about the Fall Semester is that we do not know for sure if the social distancing and the stay-home order will still be in effect, and so we must also assume the 'worst' for the Fall Semester. Unless otherwise specified, all meetings will continue to be online through our Discord channel. If we can meet in person on campus, we will have face-to-face meetings as needed.

After the semester has started and we have moved into September, all schedule changes will be done. Based on these changes, we will schedule a day to meet either

face-to-face or through Discord. This scheduled day will be a day that every team member is consistently available each week, so that we may have weekly meetings to discuss progress. If progress is advancing rapidly, we may push these meetings back to be bi-weekly if time allows.

The milestones we hope to achieve over this time include:

- Booting into kernel from power-off
- Kernel can schedule and execute processes
- Users can log in to the system
- Users can make system calls to the kernel
- Files stay persistent between boot sessions

8.4 Project Summary and Conclusions

8.4.1 Summary

PegasOS is a 64-bit operating system designed for the Raspberry Pi 4, and as such it has required a great deal of consideration for its design and an even greater amount of background research for each component in the system. Operating Systems are no easy feat, which has required that each member of the team become the master of their domain, and tackle the various aspects of the system in such a way that every member has a key function in the design and eventual implementation of the system.

At its core, it is made up of several key components that drive the major systems within the kernel. The Input/Output Manager to handle input from keyboards, displays, and other devices. The Memory Manager to handle the swapping of pages in virtual memory, and translating virtual to physical space on disk and in RAM. The Scheduler to send processes to the CPU core(s) for execution. The File System for the storing, reading, writing, and moving of files. The Interrupt Handler for events from software or hardware to allow processes to continue their execution. The Shell to allow users to interact with the system through program execution as well as file and system modification.

PegasOS will have specific future-proofing systems in place to ensure that we are not re-writing the system every time that we wish to include more features or new programs to run on the system. Two examples of this are the Localization Files and their standards, as well as the directory for shell program binaries that allow for easy searching and execution of those programs without having to manually inform the Shell of their existence. Design decisions like these ensure that the system remains extendable and modern, without having to modify system code after the fact.

As a whole, it is an incredible opportunity for us to be able to create this project and this system, and we have all learned a great deal about how these systems function and what design decisions go into making the best possible operating systems. The next step is to begin our implementation of these systems, which will be a long road for sure, but at the end the destination as well as the journey will have been well worth the effort.

8.4.2 Conclusions

While optimization is a big aspect of our goals for the system, realistically speaking the operating system must be functional before it can be optimized. First, the system must be operational in 64-bit mode as opposed to the typical 32-bit mode of most Raspberry Pi operating systems, and this will require careful considerations when programming the operating system in the coming months that the design cannot fully accommodate or prepare for until those problems are encountered in implementation.

A problem that we quickly encountered in the design of the system is that many of the systems rely on each other to operate in full capacity, or even to start other necessary systems. This realization has allowed us to attempt to counter this problem early, and section the various components of the operating system into core 'packs' that can be completed at once and used to create the next pack of components.

The first of these will be that of the system's general input and output, the file system, and the memory management. With these systems in place, we can load files from memory for program execution and scheduling, manage processes within memory and the files that the processes need in the page table, and begin development of the shell to interact with the system. All of these systems are built into the kernel as it is developed, so that it can run the system as one monolithic piece of software.

With all of our planning and research in one document that is sure to grow as we discover more about the operating system through its development, we can use it as our master blueprint to create the system that we've set out to build. As we find more considerations that we haven't made, and more designs that must be planned and drawn, we will round out the system and transform it into a functional and educational software and documentation that will guide countless others into the fascinating realm of systems software.

PegasOS

9. Appendices

```

/-o/:
/ //-
`y50
`y+0
.s+0
-s+0
-o+5
.:::+++o:::-`
.::::/:-+:::~
`:-` /` /`:-
+. -./
/`+.,
/ /-.
/ /-.
/ /-.
/ /-.
/ /-.
/ /-.
/ /-.
/ /-.
/ /--
/ /--
/ /--
/ /.-
/ /.-
/ /.-
/ /.-
/ /.-
/ /.-
/ /.-
/ /--
/ /:-
-./ /
/5-
-
```

9.1 Copyright

9.1.1 KnightOS

The original name for this operating system was KnightOS, and as such the name 'KnightOS' may still appear in some aspects of the code and documentation. This name was chosen without the knowledge of the existing KnightOS, which is designed for Texas Instruments calculators. While we have done our best to remove the name KnightOS and all associations of that name from our project, code, and documentation, it may still appear in the code and documentation. If you find a reference to 'KnightOS', please notify us and we will remove it promptly.

For more information regarding the pre-existing operating system, KnightOS, please follow the following link.

<https://knightos.org/>

9.1.2 PegasOS

The new name for this operating system is PegasOS, which bears some resemblance to the name of the mythical creature, the pegasus (also known as pegasos), as well as two other entities. Namely, the Pegasos MicroATX Motherboard which had two versions - the Pegasos I and Pegasos II, as well as the Pegasos Swiss Foundation. PegasOS has no affiliation with either the motherboards designed and produced by *bplan GmbH*. and sold by *Genesi USA Inc.*, nor the *Pegasos Swiss Foundation* founded in 2019.

For more information on the Pegasos MicroATX motherboards, please follow the following link(s).

<https://en.wikipedia.org/wiki/Pegasos>
<https://genesi.company/products/pegasos>

For more information on the Pegasos Swiss Foundation, please follow the following link.

<https://pegasos-association.com/welcome/>

9.1.3 GNU General Public License

This operating system is licensed under the GNU General Public License, versions 3 and later. Version 3 of the license has been copied here for your convenience.

Copyright © 2007 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not

part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and non commercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those

permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying”

means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to simultaneously satisfy your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work,

but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA

BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

9.2 Software

9.2.1 Diagrams

All diagrams presented in this document that have not been accredited to an outside source have been made by the members of the PegasOS team with the free software, draw.io. Draw.io is licensed under the Apache 2.0 license, which can be found here at the draw.io GitHub page.

<https://github.com/jgraph/drawio/blob/master/LICENSE>

9.2.2 Source Code and Documentation Repositories

All source code is hosted on GitHub under public repositories. These repositories are free to clone from, download from, and modify private copies of under the GNU General Public License, versions 3 and later. If you would like to contribute to the repository itself and we have opened the repository to public contribution, please make your contribution request on the GitHub page for the source code, which can be found here:

<https://github.com/MrJellimann/PegasOS>

A copy of the documentation is also hosted on GitHub, with cross-links to the source code for PegasOS. If you would like to view this documentation on GitHub, please follow this link:

<https://github.com/MrJellimann/PegasOSDocumentation>

9.2.3 Discord

For meetings and discussions that were not held in person at UCF, we have used a Discord channel created specifically for our team. Discord is a free messaging software, with an optional paid service, primarily intended for gaming groups to use text and voice chat to communicate. It has since expanded to include features for more than just gaming groups, and includes many features typically seen in a forum. Recently, Discord has released fully-fledged screen-sharing and video chat, which allows us to use the software to host meetings amongst the team when we cannot otherwise meet in person. For more information on Discord and its services, please follow this link:

<https://discordapp.com/>

9.3 Miscellaneous

9.3.1 Hashing Algorithm

For our 256-bit hashes, we will be using an implementation of SHA-256 based on the information and source code provided at the following link:

<https://www.movable-type.co.uk/scripts/sha256.html>

This is for two main reasons:

- SHA-256 is still reliable yet secure, and it is not too long in computation to delay hashing and rehashing usernames and passwords for log-ins and user registry creation. Of course on modern hardware, the speed of such a hashing algorithm should not be noticeable to the average user, even with a longer hash.
- Creating a new hashing algorithm that is reliable and more secure than existing options requires an amount of focus and knowledge that is outside the scope of this project.

PegasOS

10. References



10.1 References

A

Allen, Jay. "New User Tutorial: Basic Shell Commands." New User Tutorial: Basic Shell Commands, Liquid Web, 6 Sept. 2018, www.liquidweb.com/kb/new-user-tutorial-basic-shell-commands/.

Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. Operating Systems Three Easy Pieces. Arpaci-Dusseau Books, 2018.

B

"BCM2711." BCM2711 - Raspberry Pi Documentation, Raspberry Pi, www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/README.md.

"BCM2837." BCM2837 - Raspberry Pi Documentation, Raspberry Pi, www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md.

"BCM2837B0." BCM2837B0 - Raspberry Pi Documentation, Raspberry Pi, www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837b0/README.md.

"Boot Sequence." Boot Sequence - Raspberry Pi Documentation, Raspberry Pi, www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/bootflow.md.

C

Castro, Kristi. "What Are System Calls in Operating System?" Tutorialspoint, 9 Sept. 2018, www.tutorialspoint.com/what-are-system-calls-in-operating-system.

Cawley. "9 Raspberry Pi Operating Systems That Aren't Linux." MakeUseOf, 3 Sept. 2019, www.makeuseof.com/tag/raspberry-pi-operating-systems-not-linux/.

Curren, Josh. "What Does 'Int 0x80' Mean in Assembly Code?" Stack Overflow, Jan. 2010, stackoverflow.com/questions/1817577/what-does-int-0x80-mean-in-assembly-code.

E

“Ext2.” OSDev Wiki, 1 Oct. 2018, wiki.osdev.org/Ext2.

F

“FAT.” OSDev Wiki, 9 Oct. 2019, wiki.osdev.org/FAT32.

“File Systems.” OSDev Wiki, 2 Feb. 2020, wiki.osdev.org/File_Systems.

Freenergi, Antonius. “Old Knowledge of x86 Architecture : ‘8086 Interrupt Mechanism.” Ringlayer Robotacist, 9 Sept. 2014, sw0rdm4n.wordpress.com/category/interrupt-vector-table/.

G

“GCC Cross-Compiler.” OSDev Wiki, 5 Apr. 2020, wiki.osdev.org/GCC_Cross-Compiler.

“GPIO.” GPIO - Raspberry Pi Documentation, Raspberry Pi, www.raspberrypi.org/documentation/usage/gpio/.

H

Hasan, Mehedi. “The 50 Most Useful Linux Commands To Run in the Terminal.” UbuntuPIT, 29 Feb. 2020, www.ubuntupit.com/best-linux-commands-to-run-in-the-terminal/.

“How the Linux Kernel Runs a Program.” LinuxInside, 0xax.gitbooks.io/linux-insides/SysCall/linux-syscall-4.html.

Hughes, James. “Raspberrypi/Firmware.” Raspberry Pi Firmware Wiki, Raspberry Pi, 15 Nov. 2019, github.com/raspberrypi/firmware/wiki.

J

Jones, M. “Evolution of Shells in Linux.” IBM Developer, 9 Dec. 2011, developer.ibm.com/tutorials/l-linux-shells/.

“Journaling.” OSDev Wiki, 1 July 2011, wiki.osdev.org/Journaling.

K

Kerr, Dave. "Effective Shell Interlude: Understanding the Shell." Dwmkerr.com, 21 May 2019, dwmkerr.com/effective-shell-part-5-understanding-the-shell/.

King, Tracy. "FAT32 Structure Information - MBR, FAT32 Boot Sector Introduction." EaseUS, 5 Aug. 2016, www.easeus.com/resource/fat32-disk-structure.htm.

M

Mika J. Jarvenpaa, "Linux Interrupts: The Basic Concepts." University of Helsinki, Department of Computer Science.
https://www.cs.montana.edu/courses/spring2005/518/Hypertextbook/jim/media/interrupts_on_linux.pdf.

O

"Operating System - File System." Tutorialspoint,
www.tutorialspoint.com/operating_system/os_file_system.htm.

"OSDev Wiki." Expanded Main Page - OSDev Wiki, 16 Dec. 2011,
wiki.osdev.org/Main_Page.

P

"Pi4 Bootflow." Pi4 Bootflow - Raspberry Pi Documentation, Raspberry Pi,
www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/bootflow_2711.md.

Poirier, Dave. "The Second Extended File System." NonGNU, 2001,
www.nongnu.org/ext2-doc/ext2.html.

R

Raspbian, www.raspbian.org/FrontPage.

"Raspberry Pi 4 Boot EEPROM." Raspberry Pi 4 Boot EEPROM - Raspberry Pi Documentation, Raspberry Pi,
www.raspberrypi.org/documentation/hardware/raspberrypi/booteprom.md.

S

Sieber, Tina. "15 Windows Command Prompt (CMD) Commands You Must Know." 15 Windows Command Prompt (CMD) Commands You Must Know, MakeUseOf, 12 Nov. 2018,
www.makeuseof.com/tag/15-cmd-commands-every-windows-user-know/.

Silberschatz, Abraham, et al. Operating System Concepts. Wiley, 2019.

T

Tchirou, Faiçal. "Lexical Analysis." Hacker Noon, 10 Mar. 2020,
hackernoon.com/lexical-analysis-861b8bfe4cb0.

Thorén, Oskar. "What Is a Shell and How Does It Work?" What Is a Shell and How Does It Work? · OSKAR, 2019, oskarth.com/unix01/.

W

"What is the difference between user space and the kernel space?" Quora, Aug 17, 2017,
<https://www.quora.com/What-is-the-difference-between-user-space-and-the-kernel-space>.

"What is the relation between the shell and system calls in Linux?" Quora, July 29, 2018,
<https://www.quora.com/What-is-the-relation-between-the-shell-and-system-calls-in-Linux>.

"Windows IoT Core Downloads." Download Windows 10 IoT Core,
www.microsoft.com/en-us/software-download/windows10IoTCore.

10.2 Uncategorized References

https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/host/i686-linux-glibc2.7-4.6/+refs/heads/jb-mr1-dev/sysroot/usr/include/asm/unistd_32.h

https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/host/x86_64-linux-glibc2.7-4.6/+refs/heads/jb-dev/sysroot/usr/include/asm/unistd_64.h