DaxOS

PROJECT REPORT

submitted by

By

Nihal Narayan (MBT17CS081)

Antony S. Chirayil (MBT17CS023)

Mathew Koshy (MBT17CS068)

R Midhun Suresh (MBT17CS095)

to

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

June, 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEERING

MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram 15

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MAR BASELIOS COLLEGE OF ENGINEERING & TECHNOLOGY

Nalanchira, Thiruvananthapuram.

CERTIFICATE

This is to certify that the report entitled ***DaxOS*** *submitted by* ***Nihal Narayan (MBT17CS081)***, ***Antony S. Chirayil (MBT17CS023)***, ***Mathew Koshy (MBT17CS068)***, ***R Midhun Suresh (MBT17CS095)*** *to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering and Technology is a bonafide record of the project work carried out by him/her under my/our guidance and supervision.This report in any form has not been submitted to any other University or Institute for any purpose*

Ms. Gayathri K. S.	**Mr. V. S. Shibu**	**Dr. Tessy Mathew**
Project Coordinator	Guide	Head of the Department

Place: Thiruvananthapuram

Date: 12/01/2021

Acknowledgement

We would like to take this opportunity to extend our sincere gratitude to **Mr. Shibu VS** for guiding us through the process of this project. We would also like to thank the seminar coordinator, **Mrs. Gayathri KS**, for providing us with this opportunity which has allowed us to explore a domain which would otherwise be beyond the confines of our academic course.

Nihal Narayan
Mathew Koshy
Antony S. Chirayil
R Midhun Suresh

ABSTRACT

Every computer science enthusiast is proficient in their operating system of choice. They may even know its underlying working from an old operating system course they took in college. However, it is usually the case that their knowledge and understanding is limited to theory and writing low-level system code is often considered an insurmountable challenge. This project hopes to change this attitude by developing a minimal yet functional 32-bit operating system that can be used in conjunction with theoretical teaching to promote and introduce systems programming. A minimal kernel guarantees easier to read source code (as opposed to the 27 million SLOC Linux kernel) and provides a gentler introduction to kernel development.

Contents

List of Figures

Nomenclature

GDT	Global Descriptor Table
IDT	Interrupt Descriptor Table
ISR	Interrupt Service Routine
VGA	Video Graphics Array
APM	Advanced Power Management

Chapter 1

Introduction

The Kernel is the fundamental interconnect between hardware and software of a computer system. Writing a kernel (kernel programming) is considered to be a difficult endeavor because development has to start from a bare metal state. DAX OS is a minimal 32-bit hobbyist operating system that can be used to provide a gentle introduction to students who wish to explore the domain of systems programming.

The project is open source and licensed under GNU General Public License v3.0 to ensure unrestricted access and complete transparency. DAX OS comes with a terminal driver, keyboard and mouse driver and basic memory management. The project also uses appropriate development practises such as unit-testing and version control.

Some key facts about this project are:

- Source code is hosted at https://github.com/DaxKernel/OS.
- About 2500 lines of code only
- 32-bit
- Low resource consumption

1.1 Objectives

We propose to build a 32-bit kernel that has the following functionality:

1. **Keyboard Driver**

   Dax-OS includes a fully functional PS/2 keyboard driver. The PS/2 keyboard driver will convert scan-codes generated when the user presses a key on the keyboard to an integer character code. It must be noted that all keyboards practically used in modern day utilize the USB standard. We stick with the PS/2 protocol because the USB protocol is massive and difficult to implement. However there are practically no disadvantages from such a decision because most motherboards will emulate USB keyboards as PS/2 keyboards.

2. **Terminal Display Driver**

   DaxOS is a terminal based operating system i.e it does not support windowing. The display support will be implemented using VGA text-mode and real-mode. It will later be re-implemented in Graphics.

3. **Interrupts**

   DaxOS makes heavy use of interrupts for device drivers. It also uses interrupts for handling CPU exceptions.

4. **Memory Management**

   DaxOS uses a flat memory model. Since it is an 32-bit operating system, it will support at most 4GB of addressable memory. A custom memory allocator has also be implemented.

5. **Graphics Support**

   DaxOS will provide graphics capability using VESA mode. The supported resolution is 1280x720.

6. **Font Rendering**

   With graphics support, the vga text-mode based terminal driver is depreciated in favor of the higher resolution VESA mode. Font rendering needed for this scenario is also available.

7. **Image Rendering**

   DaxOs will natively support rendering images in TGA format.

1.2 Limitations

Some functionality DaxOS does not implement are:

- Process Management
- GUI
- Paging

1.3 Technology Stack

The bulk of the operating system is written in the C programming language. Certain functionality such as writing data to ports and loading tables (IDT, GDT) are implemented using either GCC inline assembly or using normal x86 assembly.

The project uses the GCC cross-compiler & binutils to target the generic i686 platform; which is a generic 32-bit Intel P6 architecture. The GNU Assembler and Linker are also used. The assembly syntax style used is AT&T.

The entire compilation process is driven using GNU Make which uses Makefiles to build the project. The compilation is initiated using BASH shell scripts. The kernel is tested using the qemu-i386 emulator and is developed on a stable Xubuntu distribution.

DAX OS uses git as its version control system of choice. The git repository is uploaded on Github and every new feature is developed on its own separate branch. Team communication and coordination are actualized using discord with Github integration enabled.

Chapter 2

Design Diagrams

2.1 Use-case diagram

The use case diagram illustrates how the user interacts with the operating system. There is a total of three interactions between user and the operating system in this case. The first interaction that the user initiates is the boot process.
This involves:

1. **Setting up interrupts**

   The concept behind interrupts is that when a piece of hardware or software wants to interrupt the CPU to do something important, an interrupt is raised. The CPU executes a program called the ISR (Interrupt Service Routine) and returns back to what is was doing before.

2. **Loading VGA driver**

   Using VGA to print text to the screen is quite simple and is achieved by using VGA text-mode. The user can directly write to the video memory located at address 0xB8000. Each character that is to be printed requires a two-byte representation: 1 byte, called the code-point is used to represent the character in ASCII. 1 byte, is used to set the background and foreground colors. There are 16 colors that can be used. In VGA text-mode a maximum of 80x25 characters can be printed on the screen at a time.

3. **Load keyboard driver**

   The keyboard driver uses the PS/2 interface to enable communication between the keyboard and the computer. When a key is pressed on the keyboard, the PIC raises IRQ1. This triggers an ISR which on execution will store the pressed key on a circular buffer. The data from the circular buffer is read by primitives from the C standard library such as scanf in stdio. The data from keyboard is read from port 0x60 using inline assembly function **inportb()** defined in io.c file.

4. **Setting up memory**

   This involves ensuring that multiboot is working as intended and confirming that the **multiboot_info_t** structure contains the fields neccessary for memory management. Other intermediate datastructures needed for memory management such as the stack and global variables are also initialized at this stage.

The second interaction involves the user issuing a command to the terminal.
The command may require dynamic memory to be allocated. Dynamic memory is allocated by implementing the malloc C function. The malloc function acts as a wrapper around the kernels memory manager.

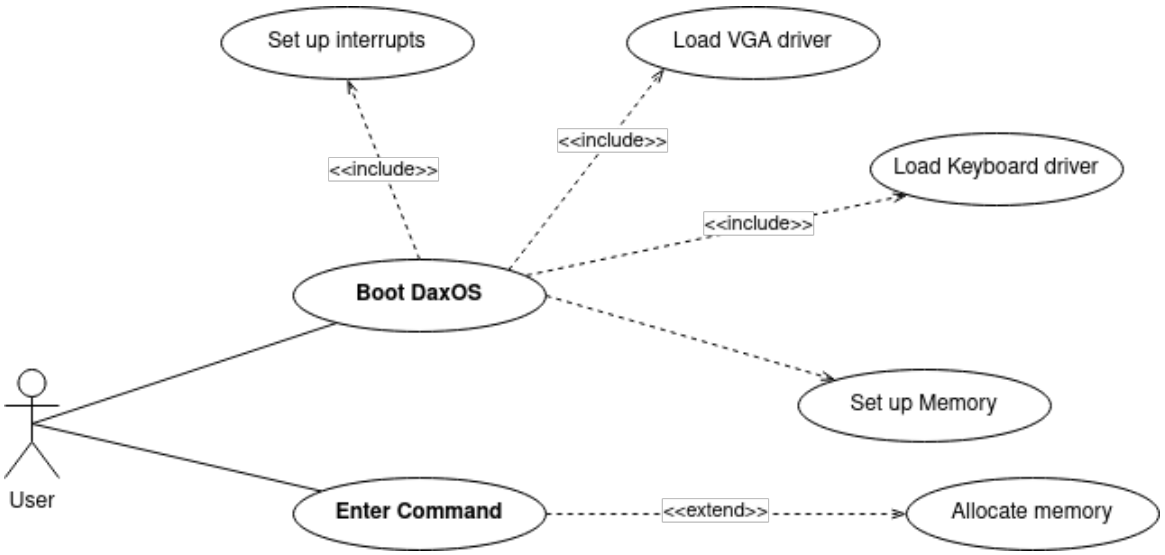

Figure 2.1: Use Case Diagram

2.2 Activity Diagram

The activity diagram illustrates the shell scripts that are used to build and compile the kernel.
The most important script here is the **build.sh** shell script which dives the make program to compile our source code. The build script relies further on two other shell scripts:

1. **headers.sh** script

   Since in this project a version of C standard library is implemented the compilation of kernel is done by instructing the GCC cross-compiler to look for system headers in the SYSROOT directory. This script compiles the custom C std-lib into an archive named libk.a and places it in the SYSROOT/usr/lib directory. Also it copies all the .h header files into SYSROOT/usr/include directory. Now compiling the kernel is done by passing the –sysroot=SYSROOT parameter.

2. **config.sh** script

   This script sets up all the environment variables used by GNU Make.This allows the user to easily change the core aspects and tooling used in the projects later. For example, we could change the compiler from GCC to

CLANG by simply changing the environment variable CC to CLANG. Likewise SYSROOT directory can be changed by changing a single variable. Once these two scripts are executed the build script runs the make-install command on each of the folder directory. It copies the output DaxOS.kernel file into the boot directory.

The **iso.sh** script builds a bootable .iso image file from our kernel. It first calls the previously described build script and then produced an iso file by using the grub-mkrescue program. This iso file can be loaded into any emulator to run the operating system. The iso files are built into the isodir directory.

The **clean.sh** script removes all the build artifacts from the compilation. Build artifacts are files that are produced by the compilation process that can always be reproduced by the compiler. Since it can be reproduced it is usually removed to maintain a clean project structure.
There are three kinds of build artifacts:

1. **Object files** - .o files
2. **Make dependencies** - .d files
3. **Unwanted directories** - sysroot, isodir

The clean script works by executing the make clean commands in each of the project directories.The make clean commands use the rm bash command to delete the build artifacts.

The last script is the **create-bootable-usb.sh** script. This script creates a bootable USB of the operating system. It needs sudo permission because it accesses the UNIX block file of the USB device. This of the form /dev/sda or something similar. This script formats the USB device and copies the kernel along with the GRUB bootloader into the USB device. Therefore there is a chance that the data in the USB device can be destroyed if the user is not careful. To prevent this a safety check that checks if the device has more than 10 GB capacity is done. If this is true we do not format the device and exits with error. If the capacity is less than 10GB, the USB is made bootable.

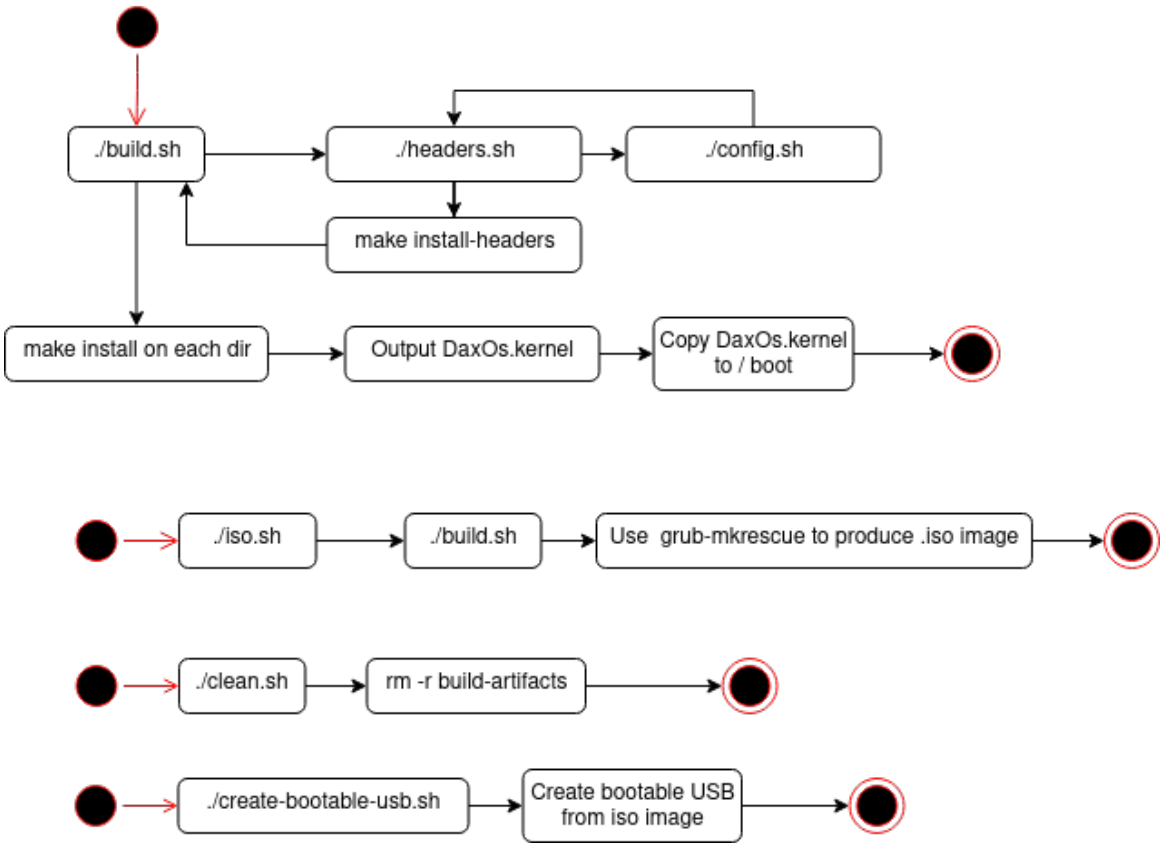

Figure 2.2: Activity Diagram

2.3 Sequence Diagram

This diagram illustrates the time dependent and sequential interaction between the user, the OS and the keyboard driver. During boot, the Operating System initializes the keyboard driver. This involves:

1. **Self Test**

   (a) The keyboard device is disabled by sending command 0xAD and 0xA7 to the PS/2 controller.

   (b) The PS/2 controller's output buffer is flushed by reading from port 0x60.

   (c) Initiate the PS/2 controller self test by sending command 0xAA to it. A response of 0x55 indicates success.

   (d) Enable the keyboard device by sending commands 0xAE and 0xA8 to the PS/2 controller.

   (e) Reset the device by sending 0xFF to the keyboard.

   (f) Set the LED states by sending appropriate commands.

2. **Handling Keypress**

   When the user presses a key the keyboard device initiates an interrupt. This done by activating IRQ1 which is the standard interrupt line used by keyboards. In response to this interrupt, the CPU executes an ISR which updates a buffer with the read characters. The I/O functions in stdio.h like scanf will read from this buffer to perform the necessary computation and show the results back to the user.

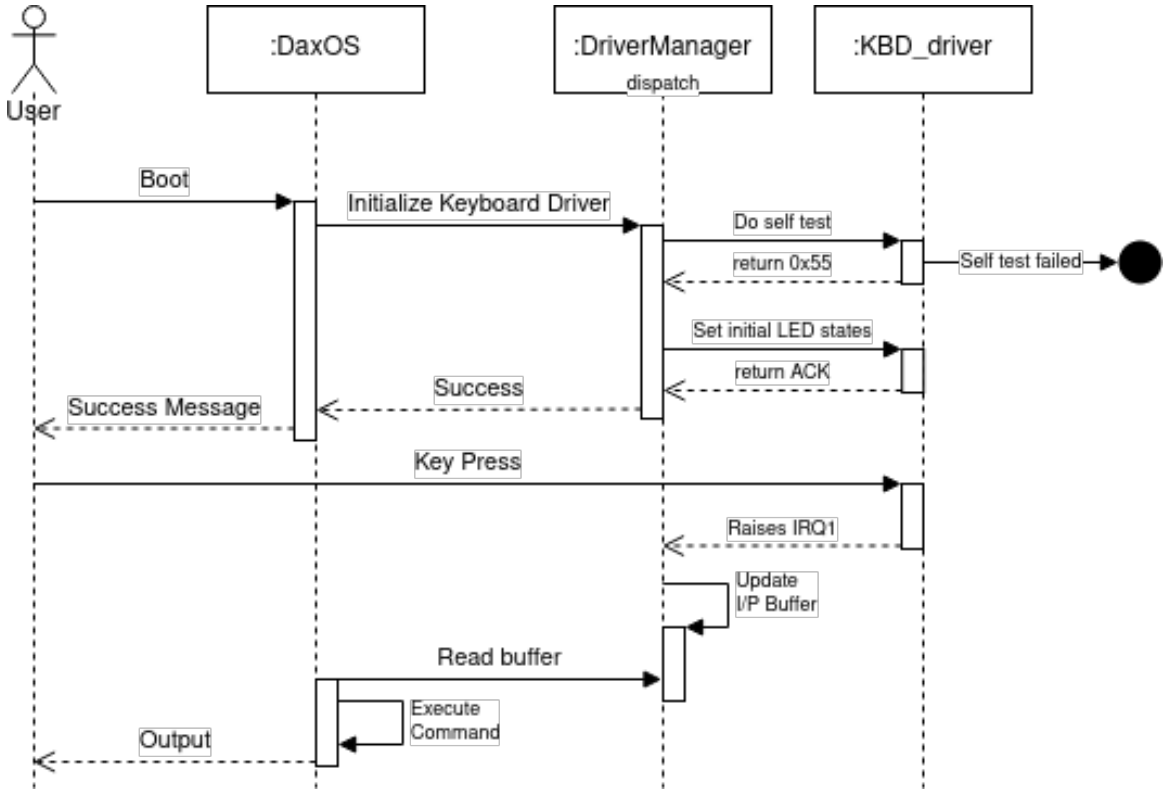

Figure 2.3: Sequence Diagram for keyboard driver

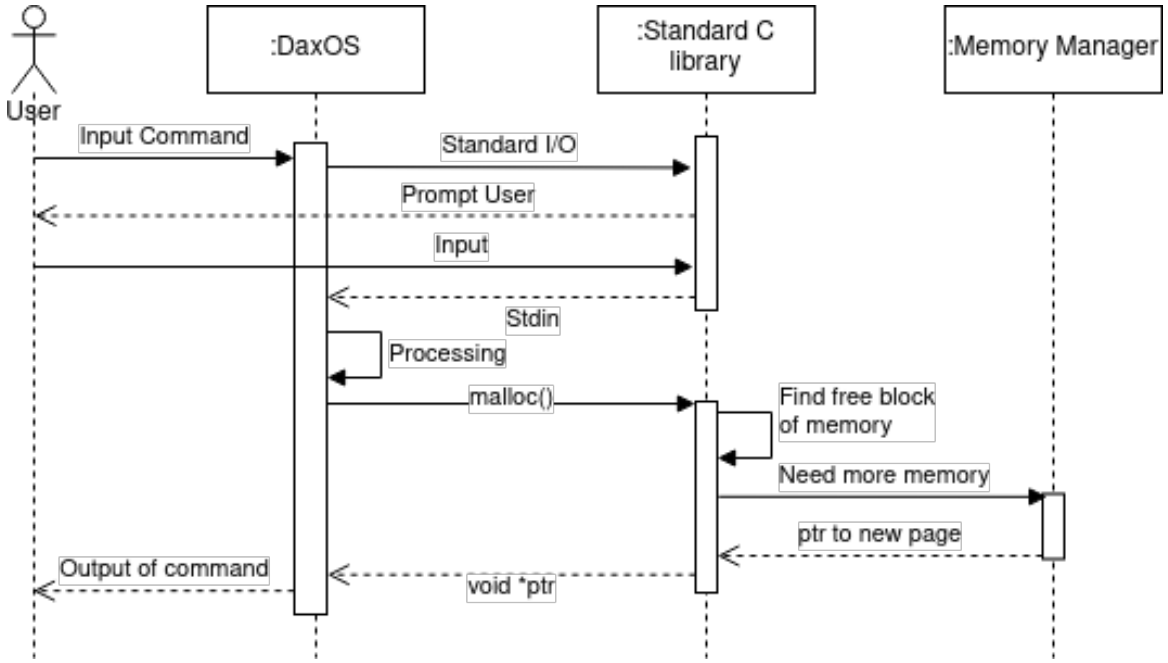

Figure 2.4: Sequence Diagram for user command

Chapter 3

Build Process

3.1 Overview

DaxOS requires a certain number of tools for the build process:

- **Cross-compiler**:
  This is a special kind of compiler that can output binaries for a different kind of computer architecture than the one in which it resides.

- **Binutils**:
  This contains other tools such as the assembler and linker.
  It also contains an archiver used to produce libraries.

- **Emulator**:
  Since an OS cannot be run as easily as other types of software, an emulator is used to accelerate the code-build-debug cycle.

- **Linux**:
  Although this is not a strict requirement, most of the tools specified above are easily available on Linux.
  Other OS such as Window can still be used although it is more time-consuming and non-intuitive.

3.2 Building cross-compiler and binutils

These tools cannot be downloaded as as binary package. Instead they must be compiled from source. The buld process for these tools is somewhat complicated. However in summary, the steps to be followed are:

1. Download respective source files from the GNU website.
2. Install the necessary dependencies using the package manager. For example on debian based distributions, the *apt* package manager is used as follows:

```
sudo apt install build-essentials, bison, flex, libgmp3-dev ...
```

3. Add path variables:

```
export PREFIX="$HOME/opt/cross"
export TARGET=i686-elf
export PATH="$PREFIX/bin:$PATH"
```

4. Run configure scripts
5. Issue make-install commands as follows:

```
make all-gcc
make all-target-libgcc
make install-gcc
make install-target-libgcc
```

6. Add newly compiled compiler binaries to $PATH variable.

3.3 Setting up the emulator

DaxOS uses *qemu* as its default and preferred emulator.
Qemu is fast, open-source and can be easily run from the terminal. It can be easily installed using:

```
sudo apt install qemu-system-i386
```

3.4 Overview of shell-scripts

Building DaxOS is a complicated process that involves running many different commands with a large number of optional parameters and flags. Therefore to make the build process easier, DaxOS comes with a set of shellscript wrappers to automate the heavy build process. With the shellscript, compiling the OS is as easy as invoking:

```
.\build.sh
```

Other available shellscripts and their functionalities were introduced in the Design Diagrams chapter.

3.5 Makefile

A Makefile is a particular type of file that specifies the steps to be taken to build a particular piece of software.
DaxOS uses Makefiles to drive the compilation process.
Makefiles are written by specifying a target and the steps needed to produce that target. Additionally dependencies needed to build a target can also be specified, in which case, the steps needed to produce the dependencies also need to be written.

Consider the following snippet of a Makefile that is used in DaxOS:

```
DaxOS.kernel: $(OBJS) $(ARCHDIR)/linker.ld
$(CC) -T $(ARCHDIR)/linker.ld -o $@ $(CFLAGS) $(LINK_LIST)
grub-file --is-x86-multiboot DaxOS.kernel

.c.o:
    $(CC) -MD -c $< -o $@ -std=gnu11 $(CFLAGS) $(CPPFLAGS)

.S.o:
    $(CC) -MD -c $< -o $@ $(CFLAGS) $(CPPFLAGS)
```

This makefile can be interpreted as follows:

- To produce **DaxOS.kernel**, the files in *OBJS* and *linker.ld* in *ARCHDIR* needs to be produced first.
- Object files(*.o*) are produced from C source files (*.c*) files by executing $(CC) command.
- Object files(*.o*) are produced from Assembly source files (*.S*) files by executing $(CC) command.
- $(foo) indicates a variable named *foo*.

Note: $(CC) here is a variable that is set to the GCC cross-compiler by the shellscript.

Chapter 4

Conclusion

This report has attempted to list the many design decisions that were available to the authors of this project and the justification for each such decision. UML diagrams for various interactions were also provided. This document and the decisions stated here are in no means final. DaxOS, like any other software project will undergo design changes as more functionality is implemented. However, the base of the design will more or less be static.

Interested readers can find the source code of this project at
https://github.com/DaxKernel/OS.
You can also check out our website at https://daxkernel.github.io/.

REFERENCES

[1] https://wiki.osdev.org/Bare_Bones

[2] http://www.osdever.net/tutorials/