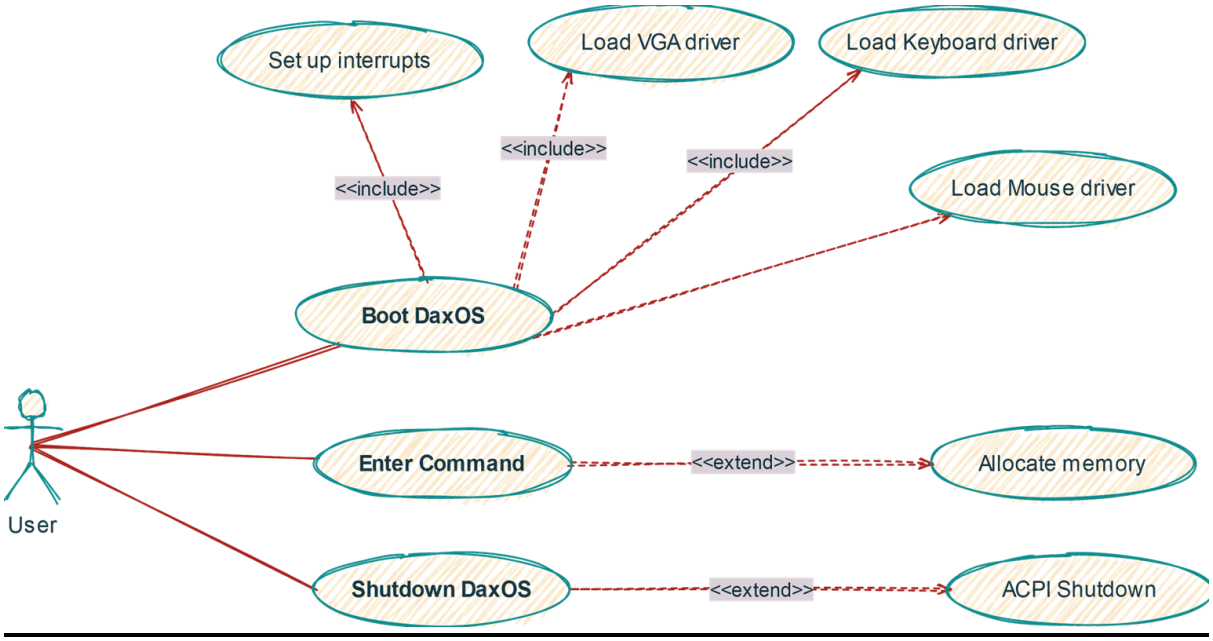
Design Diagrams

UML design diagrams are best suited to describe object oriented code. Our code and language is not object oriented. But we can still use UML concepts to describe some of the design of this project.

Use Case Diagram

This 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 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. In order to set up interrupts the following steps are performed at boot time :

1. **Create the IDT (Interrupt Descriptor Table)**

IDT is a table that instructs the Programmable Interrupt Controller where to find the ISR for each type of interrupt. This is implemented in C as an array of structs.

2. **Remapping the PIC**
After boot the user is in protected mode. Here some of the interrupts conflict with each other. In order to prevent this remapping of PIC is done by first sending Initialization Command Words (ICW) to the input port 0x20 and then sending the vector offsets to port 0x21.

3. **Filling the IDT**
The IDT table is filled with the address of each ISR.

2. Loading VGA driver

The two objectives here are:

1. **Write text to the screen**
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.
2. **Drawing on the screen**
This is accomplished by using VGA Graphics mode. This allows the user to plot pixels by writing to memory location 0xA0000. The maximum resolution supported is 640x480 pixels with 16 bit color support.

3. Load keyboard driver

There are two design choices for the implementation of the keyboard driver:
1. **Polling Driven**
In Polling, the CPU keeps checking the status of the PS/2 keyboard constantly to detect a keypress.
2. **Interrupt Driven**
When the user presses a key, the keyboard driver generates an interrupt which causes the CPU to run and ISR that handles the keypress.

Interrupt driven implementation is more efficient than polling driven keyboard drivers. Therefore in this project an interrupt driven keyboard driver is implemented.

4. Load Mouse driver
The mouse driver is similar to the keyboard driver.

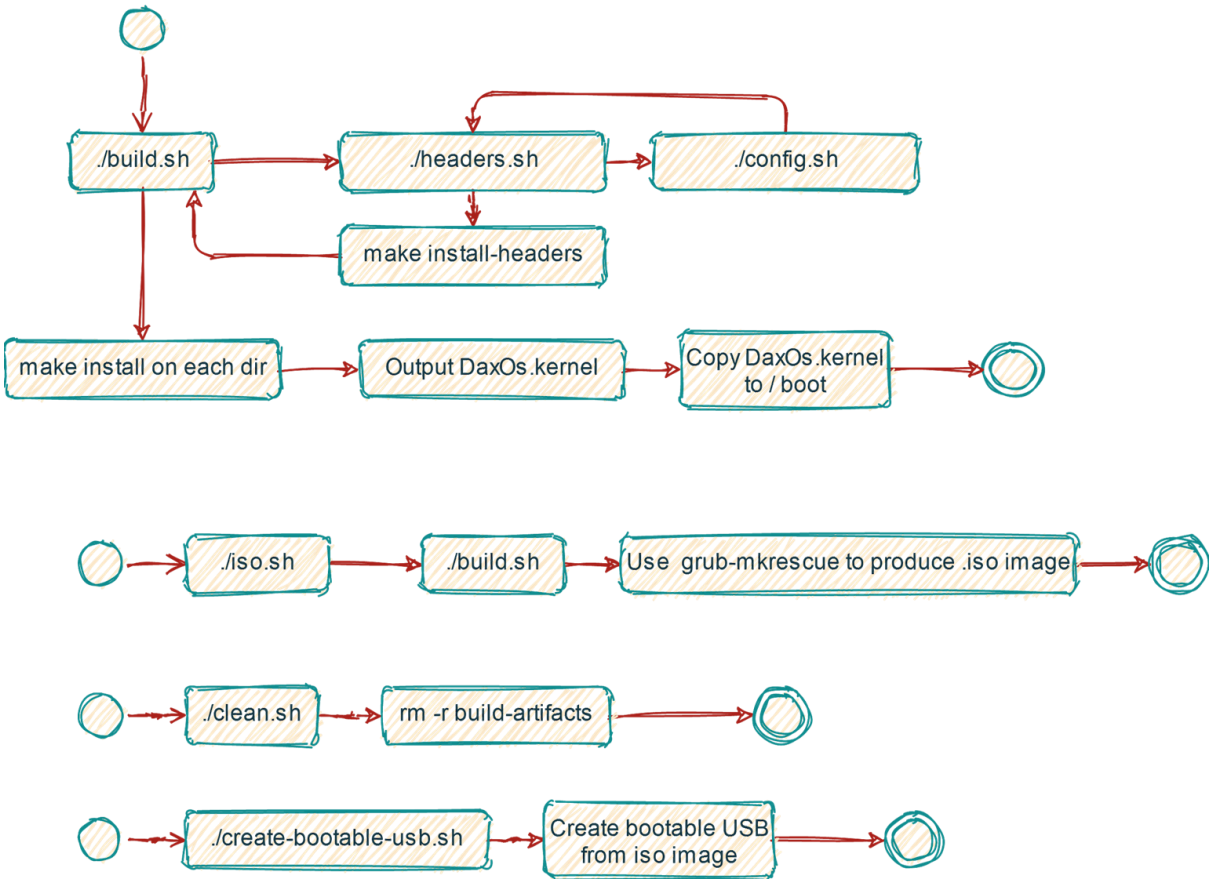
The second interaction involves the user issuing a command to the terminal. The command may require dynamic memory to be allocated. Dynamic memory is

allocated by implementing the malloc C function. The malloc function acts as a wrapper around the kernels memory manager. This needs paging and virtual memory to be enabled.

The final and last interaction between the user and the system is the shutdown sequence. This can be implemented by supporting Intel's ACPI protocol or the now depreciated APM protocol. These protocols control the amount of power each device is given. Additionally shutting down specific emulators by writing data to specific ports can be done.

Activity Diagram

This diagram illustrates the shell scripts that is 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.

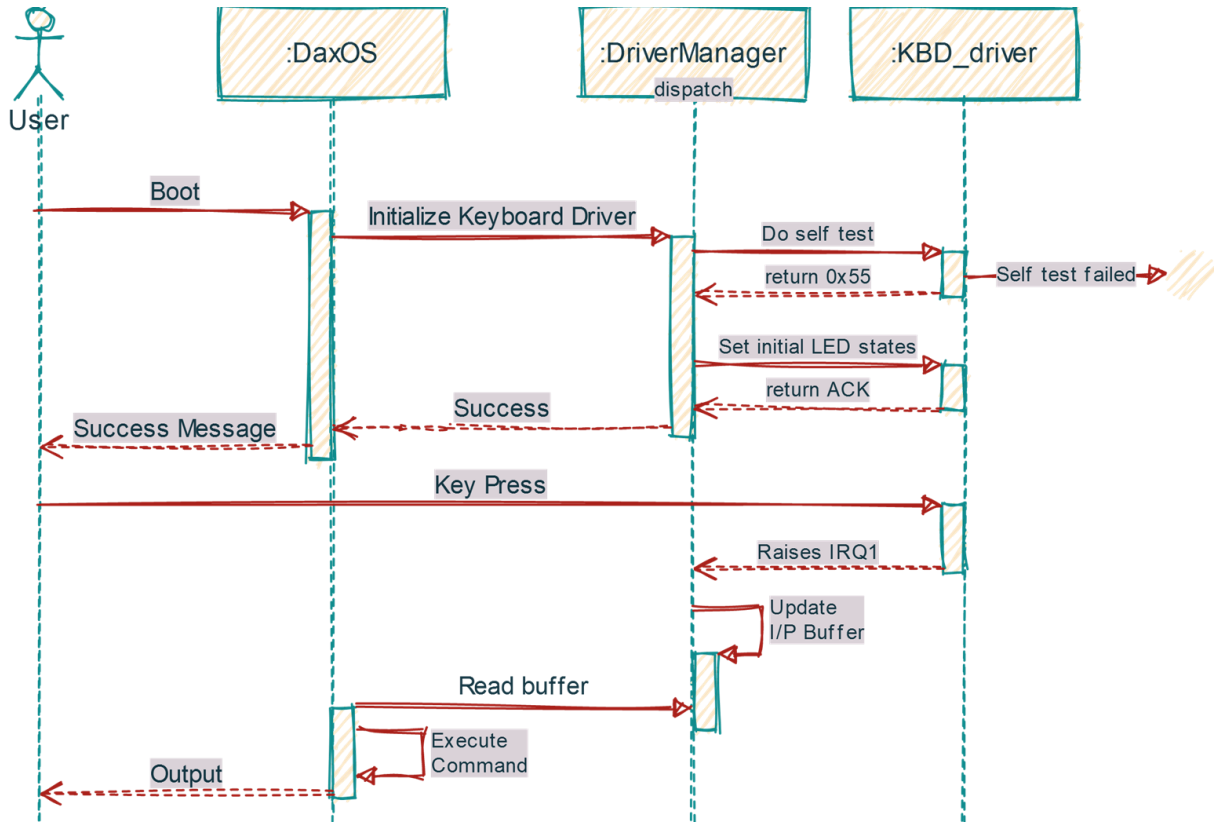
The clean.sh script removes all the build artifacts from the compilation. Build artifacts are files that are produced by the compilation process that can always be reproduced by the compiler. Since it can be reproduced it is usually removed to maintain a clean project structure.

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 and it is needed to pass 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 you are not careful. To prevent this a safety check which checks if the device has more than 10GB capacity is done. If this is true there is no need to format the drive and exit with error. If the capacity is less than 10GB, it is made bootable.

Sequence Diagram (Keyboard Driver)

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**

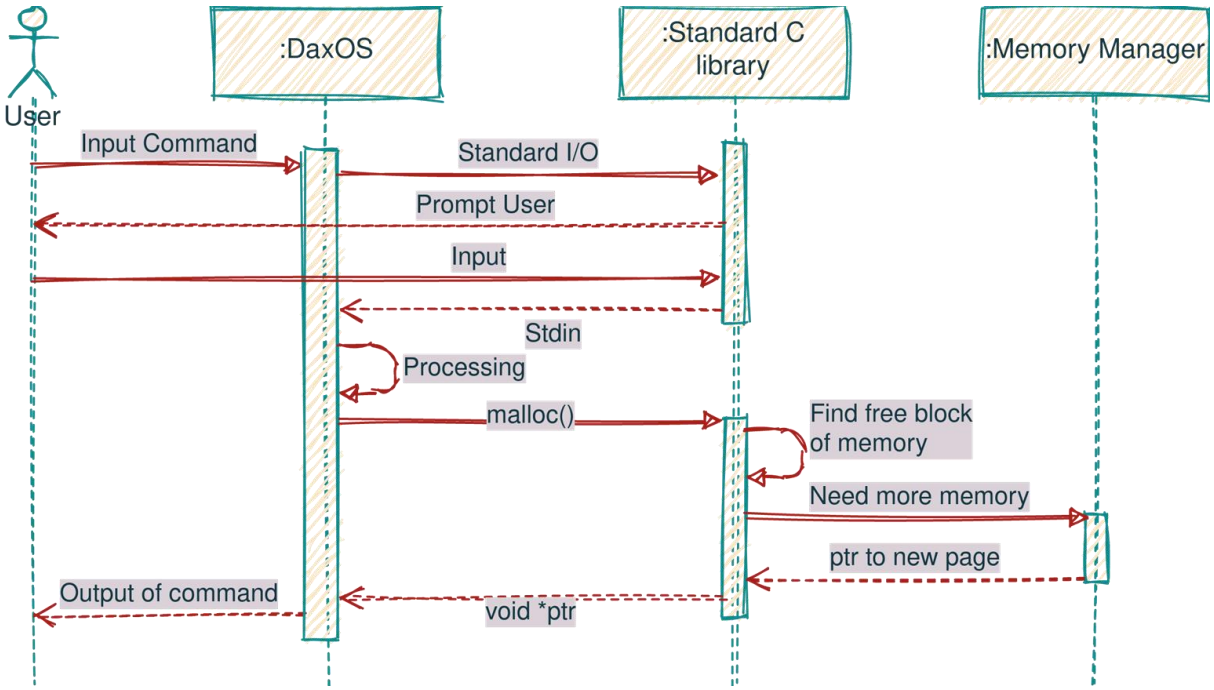
    1. The keyboard device is disabled by sending command 0xAD and 0xA7 to the PS/2 controller.
    2. The PS/2 controller's output buffer is flushed by reading from port 0x60.
    3. Initiate the PS/2 controller self test by sending command 0xAA to it. A response of 0x55 indicates success.
    4. Enable the keyboard device by sending commands 0xAE and 0xA8 to the PS/2 controller.
    5. Reset the device by sending 0xFF to the keyboard.

6. 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.

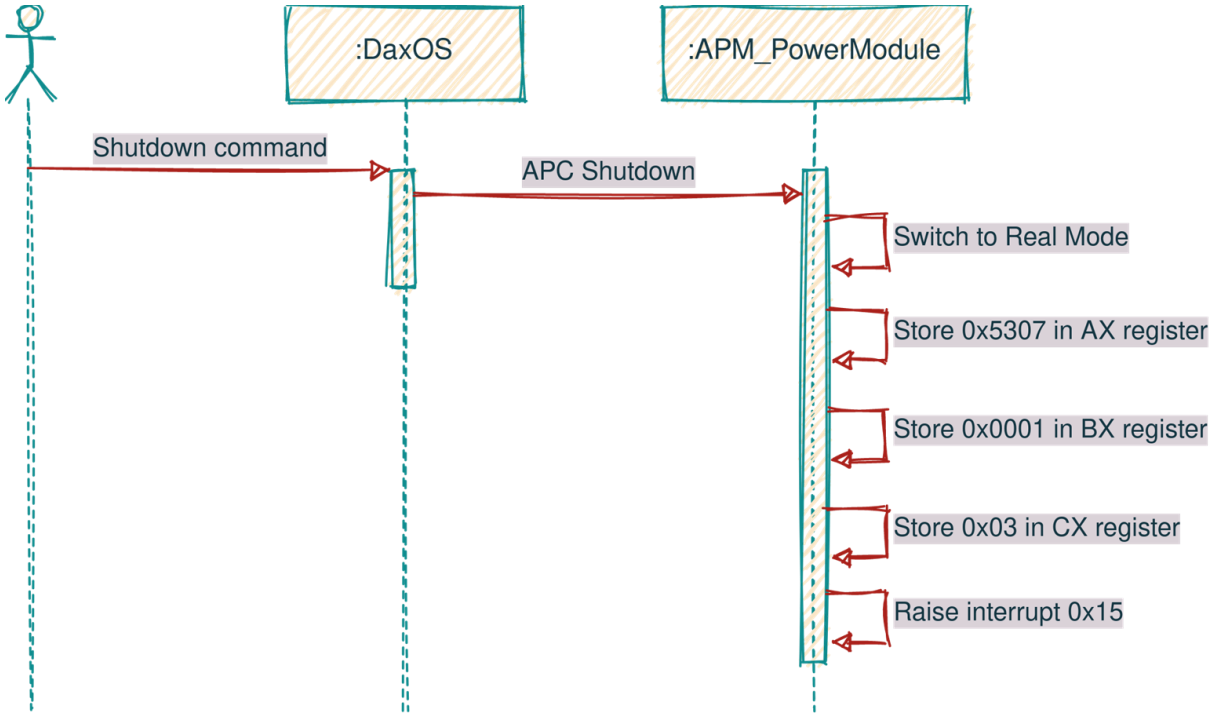
Sequence Diagram (User command)



After boot the user interacts with DaxOS through a command prompt like windows. It is here that the user enters commands which will be executed by the kernel. These are not processes.
Here the following commands are implemented:
1. **gfx_demo**
This demo program will demonstrate some of the graphics and drawing capabilities.
2. **dcalc**
This is a calculator application that can evaluate mathematical expressions.
3. **term_art**
This will demonstrate some ASCII terminal art using VGA driver.

The sequence diagram illustrates the process of running the commands. All the commands use operations from standard C library.To take input and output text to the screen the stdio functionality will be used.Then the command proceeds to do its computation. If dynamic memory is needed, the command again uses the malloc function from standard C library. The malloc function maintains a list of free stores - that is free blocks of memory. If the requested chunk of memory is already in the free store then the malloc function marks it as being used and returns a pointer to it. On the other hand if the memory in the list is not enough, the malloc function asks the kernel to provide more memory. In this case the kernel provides a new page to the malloc function and the malloc function promptly adds the block of memory into its free store and returns a pointer to it.

Sequence Diagram (Shutdown)



Shutdown is implemented using Advanced Power Management (APM). The user initiates the shutdown command. From that point onwards there is no more interaction between the user and the OS. The OS in return asks the power management module to perform APC shutdown.This involves calling a BIOS interrupt. In order to do that switching from protected mode to real mode is to be done. Then the BIOS is informed about what kind of operation is to be performed. The BIOS is informed that it is required to perform APC operation by setting the register AX with value 0x5307. The next information given to the BIOS is that the current operation is to be done on all connected devices by setting register BX with value 0x0001. Lastly the value 0x03 is put in CX register to indicate shutdown operation. Now the BIOS function is activated by raising software interrupt 0x15. Now the computer is powered off.