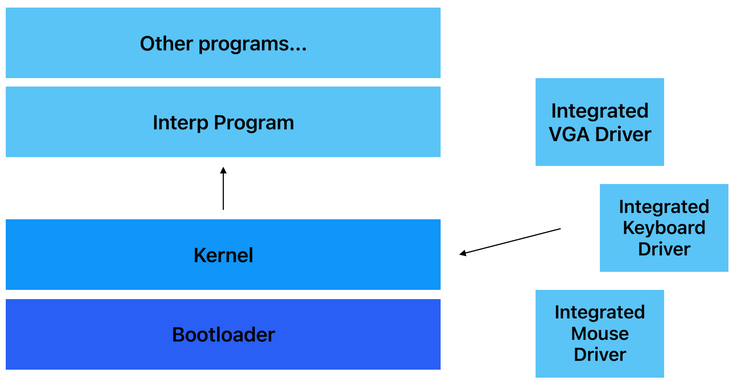
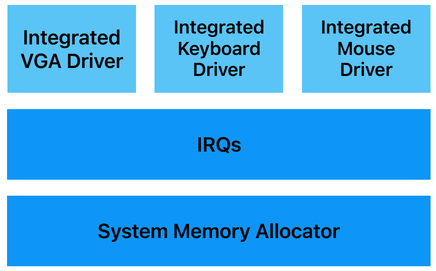
OS Overview



Kernel



IRQs, memory layouts, multi-threading...
For later :)

Interp Program

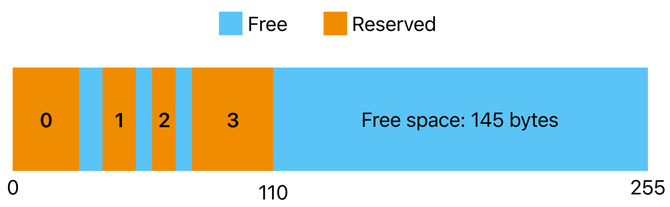
This is the shell of the OS. It runs right after the kernel is initialised and it is responsible for launching other programs the user specifies. The programs will be very limited as there is yet no way to include programs from outside the *VM*, and nowhere to install them, as the kernel lacks hard drive functionality.

System Memory Allocator

Memory is reserved with *MemoryBlocks*. They segment memory into chunks that can be expanded (triggering a reallocation if it cannot expand further into the free space trailing). Once a block is deallocated, its memory is marked as freed.

Allocation Algorithm

A call for memory allocation has been dispatched.
Size: 50 bytes, alignment: 16
Given the following memory layout:



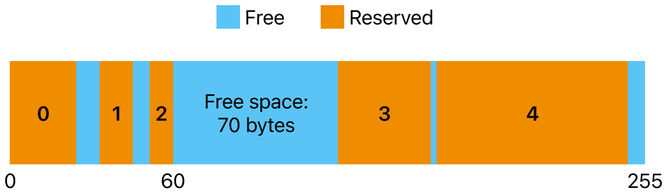
First, the algorithm will add a padding to the requested allocation size to accommodate for the alignment. It is calculated as such:
size + *alignment* - 1 = 50 + 16 - 1 = 65

The algorithm will then find the last *MemoryBlock* and calculate the trailing contiguous free space (145 bytes). If it is larger or equal than the padded size (145 > 65) a new *MemoryBlock* is created and added after the last *MemoryBlock*. The start of this *MemoryBlock* must be a multiple of the alignment, and so the following is calculated:
start + (*alignment* - (*start* % *alignment*)) =
110 + (16 - (110 % 16)) = 112

The *MemoryBlock* will then be created starting at *112* and with a size of *50*.

Another example

A call for memory allocation has been dispatched.
Size: 50 bytes, alignment: 16
Given the following memory layout:



First, the size will be padded the same as the above:
size + *alignment* - 1 = 50 + 16 - 1 = 65

The algorithm will then find the last *MemoryBlock* and calculate the trailing contiguous space (17 bytes). In this example, 17 < 65, so the algorithm will need to use another way to allocate this *MemoryBlock*. It will iterate over all *MemoryBlocks* and calculate the space between them. If it finds a space between blocks that fits the padded size calculated above, it stops and calculates the start of the new *MemoryBlock* the same as the previous example:
start + (*alignment* - (*start* % *alignment*)) =
60 + (16 - (60 % 16)) = 64

Then, the *MemoryBlock* will be inserted in between the two other *MemoryBlocks,* causing a reindexation.

Integrated VGA driver

The kernel will have an integrated VGA driver, as it doesn't yet have driver support. The driver has two modes: ***direct screen painting*** and ***text printing***. The driver can switch between the both, and the contents of each mode will be the same when the driver switches back to it. Also, if you try to access functionality from one mode while in another (eg. printing a character while in *direct screen painting* mode), the screen will not update, however, the changes will be conserved when the driver switches to the other mode.

In ***text printing mode***, the default characters per line is 80, and the default number of lines that fit in the screen is 25, however, unlike other shell-only OSes, the user has the ability to scroll up and down the text buffer. The buffer will be able to store up to 30,000 characters (15 screens worth of text).

In ***direct screen painting mode***, the user will be able to modify the RGB values of any pixel on the screen.

Integrated Keyboard Driver

This driver will support PS/2 keyboards.

This driver is based on events. Anyone can attach itself to a specific event by calling **on_key_down** or **on_key_up**.

It works by listening to CPU Interrupt Requests, then it interprets the PS/2 keyboard commands, and triggers a specific event.

Methods:

on_key_down(func: FnMut(KeyboardEvent))
This method registers a function to be called when a key is pressed.

on_key_up(func: FnMut(KeyboardEvent))
This method registers a function to be called when a key is released.

Integrated Mouse Driver

This driver will support PS/2 keyboards.

This driver is based on events. Anyone can attach itself to a specific event by calling **on_mouse_down**, **on_mouse_up** or **on_mouse_move**.

It works the same way as the *Integrated Keyboard Driver*.

Methods:

on_mouse_down(func: FnMut(KeyboardEvent))
This method registers a function to be called when the mouse click is pressed.

on_mouse_up(func: FnMut(KeyboardEvent))
This method registers a function to be called when the mouse click is released.

on_mouse_move(func: FnMut(KeyboardEvent))
This method registers a function to be called when the mouse moves.