

Trabajo Final de Grado

Desarrollo de un kernel académico para arquitecturas x86-64 en C++

Ernesto Martínez García ⁽¹⁾
me@ecomaikgolf.com

Tutor: Antonio Miguel Corbi Bellot ⁽¹⁾
Grado: Ingeniería Informática
Fecha: 13 de Junio del 2022
Modalidad: (A)

(1)  Universitat d'Alacant
Universidad de Alicante

 Escuela
Politécnica
Superior



Tabla de Contenidos

☰ Objetivos del Trabajo

✔ Trabajo Realizado

📦 Software Necesario

🧪 Máquina Virtual

⚙️ Sistema de Construcción

⦿ Bootloader

🐧 Kernel

🔗 Líneas Futuras

▶ Demostración

Introducción

⋮

Desarrollo

⋮

⋮

⋮

⋮

Conclusión

⋮



Escuela Politécnica Superior

Desarrollo de un kernel académico para arquitecturas x86-64 en C++

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Ernesto Martínez García
Tutor:
Antonio Miguel Corbi Bellot

Mayo 2022

Universitat d'Alacant
Universidad de Alicante

Objetivo del Trabajo

Desarrollar *alma*, un kernel capaz de hacer funcionar un ordenador x86 de 64 bits.

alma es

- Simple
- Un núcleo con fines de aprendizaje
- El único código en la CPU

alma **no** es

- Profesional
- Útil en entornos reales
- Una aplicación normal de C++

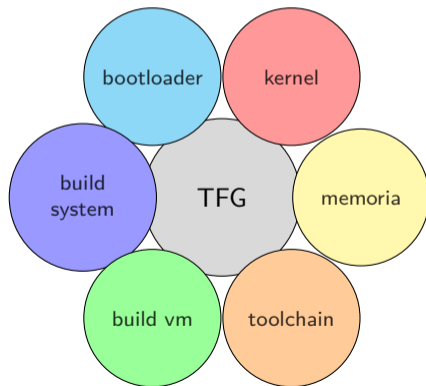
*“No se busca desarrollar un núcleo complejo ni funcional, se busca desarrollar los mecanismos que sostentan los sistemas operativos actuales. Se busca desarrollar el **alma** de los sistemas operativos.”*

Trabajo Realizado

No solo *alma*. Se ha trabajado en multitud de subproyectos necesarios para este.

✓ Necesidades encontradas

- Un *bootloader* que ejecute el kernel
- El kernel requiere modificar el compilador
- Mecanismo de construcción del kernel
 - Resistente a cambios
 - Elegante
- Entorno de desarrollo portable y estable
- Documentación



Software Necesario

13 paquetes, 6 submódulos *git* y construir: *posix-uefi*, *edk2*, *binutils* y *gcc*

alma requiere de un compilador construido con ciertas modificaciones:

- Target triplet: `x86_64-elf`
- *red-zone* desactivada completamente
- *libgcc* con `-mmodel=kernel`

Todas las construcciones se han automatizado en un script:

```
>_ make -C toolchain/  
  
git submodule update --init --recursive  
...
```

Máquina Virtual I

La “*alma build vm*”: un entorno de construcción y desarrollo portable y estable.

Se desarrolló para poder:

- Trabajar en *alma* desde cualquier máquina.
- Evitar el tiempo de construcción de la *toolchain*.
- Disponer de un entorno estable y controlado para el proyecto.

Mediante distintas técnicas que favorecen la compresión, el OVA ocupa tan solo 6.32GB

Permite trabajar con el proyecto mediante una interfaz visual:



Run Alma



Build Alma



Debug alma



Network alma



Tinker alma



Update Alma



Clean Alma



Browse Alma



Terminal alma



Demo Alma

Máquina Virtual II



The image shows a desktop environment for the Alma VM. The desktop background is black with the word "alma" in a large, white, cursive font. Below it, the text "Build VM" and "Xubuntu 20.04" is displayed in a white, monospace font. At the bottom, the URL "github.com/ecomaikgolf/alma" is shown in a white, monospace font. The desktop has a top panel with several application icons: Run Alma, Build Alma, Debug alma, Network alma, Tinker alma, Update Alma, Clean Alma, Browse Alma, Terminal alma, and Demo Alma. On the right side, there are three status indicators: "Building alma" (blue circle), "Correct build" (green circle), and "Running alma" (orange circle). On the left side, there are four desktop icons: sudo.bt, File System, Home, and Trash. At the bottom left, there is a Terminal icon.

Sistema de Construcción

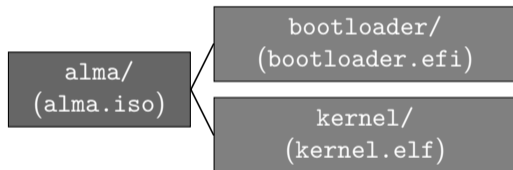
CMake multinivel completamente automático, sólido y resistente a cambios



Características remarcables

- *Out source builds*
- Construcciones con `make` o `ninja`
- `ccache` automático

Esquema multinivel



Se ha simplificado al máximo el complejo proceso de construcción del proyecto:

```
>_ cmake -B build; cd build
```



```
make
```

```
| make run
```

```
| make debug
```

```
| make doc
```


Bootloader

Un fichero ejecutable con formato *PE* capaz de cargar *alma* en entornos UEFI

⚡ Funcionalidades

- Acceso a servicios EFI con `posix-uefi`
- Carga y verificar el fichero ELF de *alma*
- Llama a los constructores globales
- Proporciona información del hardware al kernel
- Da control de la CPU a *alma*

```
(I) [startup.nsh] bootloader.efi finding started
(I) [startup.nsh] bootloader.efi found in fs0:\bootloader.efi
(I) [bootloader] started bootloader main function from /home/ecomaikgolf/Projects/os-
dev-cmake/bootloader/bootloader.c
(I) [bootloader] C environment is freestanding
(I) [bootloader] Compilation datetime Oct 30 2021 02:46:29
(I) [bootloader] opening "kernel.elf" file
(I) [bootloader] kernel.elf file opened
(I) [bootloader] allocated 26952 bytes for kernel.elf contents starting in 0x00000000
0e881018
(I) [bootloader] copied kernel.elf contents to memory 0x00000000e881018 (26952 bytes)
(I) [bootloader] closed kernel.elf
(I) [bootloader] ELF magic number is correct
(I) [bootloader] ELF is a executable object
(I) [bootloader] ELF target arch is x86_64
(I) [bootloader] ELF target is 64 bits
(I) [bootloader] ELF program header counter is non zero
(I) [bootloader] loading program header 0 at: 0x0000000000001000
(I) [bootloader] loading program header 1 at: 0x0000000000006000
(I) [bootloader] Window width: 800
(I) [bootloader] Window height: 600
(I) [bootloader] opening "zap-light16.psf" file
(I) [bootloader] zap-light16.psf file opened
(I) [bootloader] allocated 5312 bytes for zap-light16.psf contents starting in 0x0000
0000e853018
(I) [bootloader] copied zap-light16.psf contents to memory 0x00000000e853018 (5312 b
ytes)
(I) [bootloader] closed zap-light16.psf
(I) [bootloader] PSF1 font correct magic header
(I) [bootloader] jumping to kernel code at address: 0x000000000001c00
(I) [bootloader] Exiting UEFI Boot Services before the jump
```

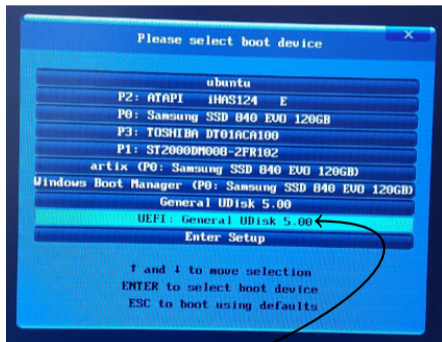
“Actualmente no se utiliza debido a una mejora que permite cargar *alma* con cualquier *bootloader* que siga la especificación *stivale2*”

Kernel I

alma, núcleo encapsulado en un *iso* arrancable desde la BIOS/UEFI

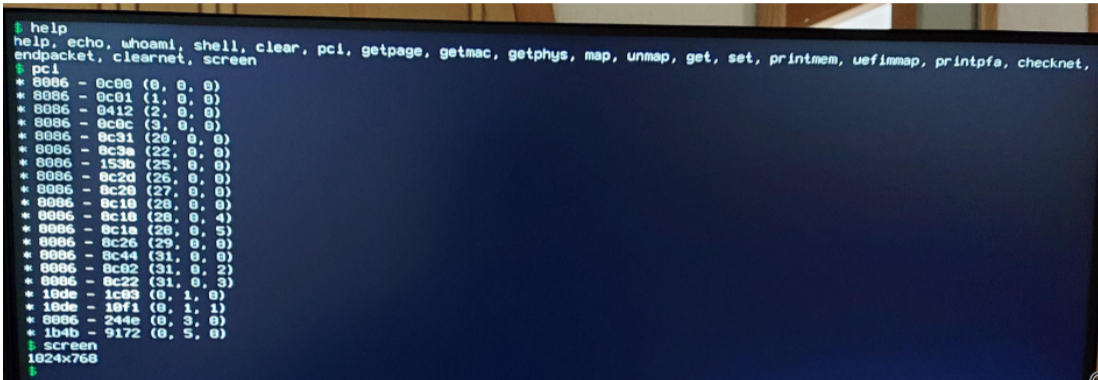
⚗ Funcionamiento

1. Construimos el archivo *iso*
2. Cargamos el *iso* en un USB
3. Entramos al menú de arranque
4. Arrancamos desde el USB con *alma*
5. Se ejecuta el bootloader
6. El bootloader carga y ejecuta *alma*



“Arrancamos *alma* igual que al instalar Linux o Windows”

Kernel II



```
$ help
help, echo, whoami, shell, clear, pci, getpage, getmac, getphys, map, unmap, get, set, printmem, uefimap, printpfa, checknet,
endpacket, clearnet, screen
$ pci
* 8086 - 0c00 (0, 0, 0)
* 8086 - 0c01 (1, 0, 0)
* 8086 - 0412 (2, 0, 0)
* 8086 - 0c0c (3, 0, 0)
* 8086 - 0c31 (20, 0, 0)
* 8086 - 0c3a (22, 0, 0)
* 8086 - 153b (25, 0, 0)
* 8086 - 0c2d (26, 0, 0)
* 8086 - 0c20 (27, 0, 0)
* 8086 - 0c10 (28, 0, 0)
* 8086 - 0c10 (28, 0, 4)
* 8086 - 0c1a (28, 0, 5)
* 8086 - 8c26 (29, 0, 0)
* 8086 - 8c44 (31, 0, 0)
* 8086 - 8c02 (31, 0, 2)
* 8086 - 8c22 (31, 0, 3)
* 10de - 1c03 (0, 1, 0)
* 10de - 10f1 (0, 1, 1)
* 8086 - 244e (0, 3, 0)
* 1b4b - 9172 (0, 5, 0)
$ screen
1024x768
$
```

Ejecución de *alma* en hardware real: comando help, listado de dispositivos pci y obtención de la resolución de la pantalla.

“No existe printf() ni malloc() ni scanf(). Cada píxel, cada pistón del teclado, cada reserva de memoria, todo está gestionado manualmente por alma”

Kernel III

¿Puedo ejecutar *alma* en mi ordenador? ¿Qué puede hacer?

⚡ Funcionalidades

- Renderizado de fuentes PSF1
- Memoria virtual de 4 niveles
- Reserva de páginas de memoria
- Teclado PS/2 con mayúsculas
- Ethernet con RTL8139
- Tablas ACPI
- PCIe
- Interrupciones
- Memoria Dinámica

☑ Requisitos

- CPU x86 de 64 bits
- Preferiblemente UEFI
- PCI Express
- ACPI 2.0+
- Tarjeta de red RTL8139
- Teclado PS/2

“No se puede garantizar el funcionamiento en cualquier hardware.”

Líneas Futuras

Aplicación de Chat

Dificultad: **Baja**

Objetivo: Intercomunicación básica

Sistema de Ficheros

Dificultad: **Alta**

Objetivo: Datos permanentes

Ratón PS/2

Dificultad: **Moderada**

Objetivo: Puntero controlable

Programmable Interval Timer

Dificultad: **Baja**

Objetivo: Noción del tiempo

Demostración



https://drive.google.com/file/d/1NG-6Ttt1gUkZ_8zPws8nobBq3rYJAgmz

Fin de la Presentación

Autor:	Ernesto Martínez García
Tutor:	Antonio Miguel Corbi Bellot
Grado:	Ingeniería Informática
Institución:	Universidad de Alicante
Facultad:	Escuela Politécnica Superior
Curso Académico:	2021/22 (C3)

Métricas del Proyecto

- 446 commits en el repositorio
- 1 año y 1 mes desde 1er commit
- 12.831 líneas de código
- 0.41 segundos en compilar *alma*
- *alma* ocupa 178K
- 45.18s en generar el TFG (\LaTeX)



Descarga el trabajo

Disponible en <https://github.com/ecomaikgolf/alma/tree/master/docs>

Anexo I: Proceso de Arranque

¿Cómo se arranca *alma* internamente?

🔌 BIOS

- CPU + 💡 → IP = BIOS ROM
- BIOS realiza POST
- BIOS enumera dispositivos
- BIOS busca dispositivo arrancable
 - Carga 512 bytes en 0x0:0x7c000
 - IP = Bootloader
- Bootloader inicializa
- Bootloader carga *alma*
- *alma* apagará el ordenador

🔌 (U)EFI

- (SEC) *Root of trust* del sistema
- (PEI) Memoria permanente
- (DXE) Inicialización general
- (BDS) Selección de arranque
 - Inicializa pantalla y drivers
 - Arranca bootloader
- (TSL) Bootloader carga *alma*
- (RT) *alma*
- (AL) Drivers UEFI persistentes