

# Embinux

Proceso de creación de un sistema Linux embebido

Sistemas Operativos

Grupo 6



Calderón Olalde Enrique Job  
Ugartechea Gonzalez Luis Antonio

**Facultad de Ingeniería**

16 de abril de 2025

# 1. Introducción

## 1.1. ¿Qué es Linux?

Linux es un kernel que en unión con GNU se vuelve un sistema operativo FOSS (Free and Open Source Software) basado en UNIX. El kernel fue creado por Linus Torvalds en 1991. Las herramientas GNU fueron creadas por Richard Stallman en 1983. GNU/Linux es el nombre correcto para referirse a la unión de ambos, aunque comúnmente se le llama Linux a todo el sistema operativo.

## 1.2. ¿Qué es un sistema embebido?

Un sistema embebido es un sistema informático compuesto por una combinación de hardware y software diseñado para realizar una función específica. Se utilizan actualmente para controlar funciones dentro de un sistema multifunción más grande, tienden a ser de bajo consumo de energía, de bajo costo y de tamaño reducido, facilitando la adaptación para otros dispositivos.

### 1.2.1. Componentes

A grandes rasgos, un sistema embebido se compone por lo siguiente:

- **Microprocesador:** Este componente actúa como el núcleo del sistema, diseñado para tareas específicas. Puede ejecutar instrucciones, controlar operaciones y gestionar los recursos del sistema.
- **Memoria:** Hay dos tipos de memoria en un sistema embebido, la memoria RAM (Random Access Memory) para almacenar datos temporales y variables de trabajo. La memoria flash, por otro lado, almacena los datos permanentes del sistema, como el firmware y el sistema operativo, incluyendo la configuración del sistema.
- **Software:** Es la parte “inteligente” del sistema, contiene las instrucciones que el sistema ejecuta, encargado del comportamiento y respuestas del mismo.
- **Periféricos:** Estos permiten la interacción del sistema embebido con el exterior. Pueden incluir sensores, actuadores, pantallas, teclados, entre otros.

## 1.3. Linux en sistemas embebidos

Linux se ha convertido en una opción muy popular para sistemas embebidos debido a su flexibilidad, escalabilidad, estabilidad y por ser open source. Su arquitectura modular permite adaptar el sistema operativo a las necesidades específicas del dispositivo, eliminando componentes innecesarios para ahorrar espacio y recursos.

## 1.4. Ventajas de Linux embebido

Una de las principales ventajas de utilizar Linux en sistemas embebidos es el acceso a una gran comunidad y a una amplia gama de herramientas y bibliotecas. Facilitando el desarrollo y

mantenimiento. Además, existen versiones específicas de Linux diseñadas para sistemas con recursos limitados, como *Yocto*, *Buildroot*, *OpenWrt* o distribuciones como *Raspbian* para Raspberry Pi.

En este contexto, el kernel Linux puede ser configurado para soportar únicamente el hardware del sistema embebido en cuestión, reduciendo el tiempo de arranque y el uso de memoria. También es posible integrar controladores específicos y definir mecanismos de comunicación eficientes con los periféricos del dispositivo.

## 1.5. Casos comunes de uso

El uso de Linux en sistemas embebidos abarca una gran variedad de industrias y dispositivos. Algunos de los casos más comunes:

- **Routers y dispositivos de red:** Linux es ampliamente utilizado en enrutadores domésticos y comerciales, switches y firewalls.
- **Televisores inteligentes y set-top boxes:** Muchos Smart TVs y dispositivos de streaming en el mercado utilizan Linux como sistema operativo base, debido a su capacidad de ejecutar interfaces gráficas, manejar decodificación multimedia y permitir actualizaciones OTA (Over-The-Air).
- **Automatización industrial:** En entornos industriales, Linux embebido se emplea en controladores lógicos programables (PLCs), sistemas SCADA, paneles HMI (Human-Machine Interface), etc.
- **Electrodomésticos inteligentes:** En refrigeradores, lavadoras, hornos, aspiradoras robot y otros dispositivos del hogar inteligente utilizan Linux embebido para conectividad, control remoto, programación de tareas, y comunicación con asistentes de voz.
- **Automóviles:** Muchas unidades de control electrónico (ECUs) y sistemas de infoentretenimiento en automóviles modernos utilizan Linux embebido.
- **Dispositivos portátiles y wearables:** Relojes inteligentes, rastreadores de actividad, cámaras deportivas y dispositivos GPS también hacen uso de Linux debido a su bajo consumo energético.

Estas aplicaciones son algunas de las más utilizadas aunque también se encuentra presente en otros dispositivos como cámaras de seguridad, drones, sistemas de control de acceso, sistemas de monitoreo ambiental, entre otros.

## 2. Componentes

### 2.1. Kernel

El **kernel** es el núcleo de un sistema operativo; actúa como intermediario entre el hardware y el software. Es responsable de gestionar el acceso al procesador, la memoria y los controladores (drivers) más importantes, permitiendo así la comunicación directa con el hardware.

Aunque puede variar en complejidad y diseño, un kernel suele estar organizado en varias capas funcionales:

- **Interfaz con el hardware:** Es la capa más baja y se encarga de tareas como la gestión de controladores de red, controladores de buses como PCI Express, y otros elementos específicos del hardware.
- **Gestión de memoria:** Asigna y distribuye la memoria entre los procesos, controla el acceso a la memoria física y virtual, y maneja técnicas como la paginación o la segmentación.
- **Gestión de procesos:** Supervisa la creación, ejecución, suspensión y finalización de los procesos, además de encargarse de la planificación (scheduling) y sincronización entre ellos.
- **Gestión de dispositivos:** Controla el acceso a los dispositivos de entrada/salida (E/S), como discos duros, impresoras, teclados, entre otros, a través de sus respectivos controladores.
- **Gestión del sistema de archivos:** Regula el acceso a archivos y directorios, manteniendo la organización lógica de los datos almacenados y garantizando su integridad.

## 2.2. Sistema de archivos

Un sistema de archivos es el componente del sistema operativo encargado de organizar, almacenar y recuperar datos en un medio de almacenamiento, como un disco duro o una memoria flash. Define cómo se estructuran los datos y cómo se accede a ellos.

### 2.2.1. Componentes principales

- **Estructura de directorios:** Organización jerárquica que permite agrupar archivos en carpetas y subcarpetas.
- **I-nodos (o estructuras de control):** Contienen metadatos de cada archivo, como permisos, tamaño, fechas de creación/modificación y ubicación física en el disco.
- **Bloques de datos:** Espacios del medio de almacenamiento donde se guarda el contenido real de los archivos.
- **Tabla de asignación:** Registra cómo se distribuyen los bloques de datos entre los archivos. Ejemplos incluyen FAT (File Allocation Table) y estructuras similares en otros sistemas.
- **Gestor de permisos:** Controla el acceso a archivos y directorios, definiendo qué usuarios pueden leer, escribir o ejecutar.
- **Journaling:** Mecanismo que registra operaciones pendientes para mantener la integridad del sistema de archivos en caso de fallos o apagados inesperados.

### 3. Proceso de creación (buildroot)

Para crear este sistema embebido, se usará buildroot. Buildroot es una herramienta de construcción de sistemas Linux embebidos que permite crear un sistema Linux completo a partir de un conjunto de paquetes y configuraciones.

Buildroot utiliza un proceso con cross-compilation para generar un sistema Linux que se puede ejecutar en una arquitectura diferente a la de la computadora de desarrollo. Esto es posible gracias a que buildroot incluye su propio compilador y herramientas de construcción, lo que facilita la creación de un sistema embebido personalizado.

#### 3.1. Herramientas

Para compilar un sistema embebido con buildroot, se necesitan las siguientes herramientas:

- **Compilador:** Buildroot incluye su propio compilador, por lo que no es necesario instalar uno externo. Algunas de las opciones son `gcc` o `musl`.
- **Herramientas de construcción:** Buildroot incluye herramientas de construcción como `make`, `tar`, `gzip`, entre otras.
- **Sistema operativo:** Buildroot se puede ejecutar en cualquier sistema operativo Linux.
- **Conexión a internet:** Se necesita una conexión a internet para descargar los paquetes y herramientas necesarias para la construcción del sistema embebido. Esto pues buildroot compila de fuente todos los paquetes necesarios para el sistema embebido.
- **Hardware:** Se necesita una computadora con suficiente capacidad de procesamiento y memoria para compilar el sistema embebido. Esto pues la compilación puede tardar varias horas dependiendo del tamaño del sistema embebido y la velocidad de la computadora.

#### 3.2. Selección de hardware

Gracias a que buildroot usa su propio compilador, es posible compilar un sistema embebido para cualquier arquitectura. Esto permite usar buildroot para compilar un sistema embebido para una gran variedad de hardware, desde microcontroladores hasta computadoras de escritorio.

#### 3.3. Configuración de buildroot

Para compilar un sistema embebido con buildroot, es necesario configurar el sistema embebido. Esto incluye seleccionar la arquitectura del hardware, el sistema de archivos, las aplicaciones de usuario y las herramientas necesarias para el sistema embebido.

Buildroot incluye una interfaz gráfica llamada `make menuconfig` que permite seleccionar las opciones deseadas para el sistema embebido. Esta interfaz es similar a la de otras herramientas de construcción como la del mismo kernel Linux.

### 3.4. Compilación

Una vez que se ha configurado el sistema embebido, se puede iniciar la compilación. Esto se hace usando el comando `make`. Este comando iniciará la compilación del sistema embebido y generará una imagen de disco que se puede usar para instalar el sistema en el hardware seleccionado.

### 3.5. Instalación

Una vez que la compilación ha finalizado, se generará una imagen de disco que se puede usar para instalar el sistema embebido en el hardware seleccionado. La configuración de la imagen de disco dependerá del hardware seleccionado.

## 4. Ejemplo práctico

Para este ejemplo decidimos usar una Raspberry Pi 4 de 4GB, esto pues es una tarjeta común en el mercado además de que gracias a su tamaño puede ser usada en una gran variedad de proyectos.

El objetivo de este ejemplo práctico es mostrar el uso mínimo de recursos que se puede lograr al compilar un sistema embebido. Para esto, se usará `buildroot` para compilar un sistema Linux embebido para la Raspberry Pi 4. El sistema embebido tendrá únicamente un entorno de terminal y para efectos educativos contará con su instalación de `python` junto al editor de texto `vim`.

En esta ocasión nos comunicaremos con el sistema embebido utilizando la conexión serial de la Raspberry Pi 4. Para esto, se usará un adaptador USB a serial TTL. Con esto evitaremos usar un monitor y teclado además de conexión a red.

### 4.1. Hardware

La tarjeta es una computadora que incluye un procesador ARM Cortex-A72, 4GB de RAM y soporte para HDMI, USB y Ethernet. La Raspberry Pi 4 es una Single Board Computer (SBC) por lo que todo el hardware se contiene en una sola tarjeta.

### 4.2. Configuración de `buildroot`

Para la configuración de `buildroot` se inicia con un clon del repositorio de `buildroot`:

```
1 git clone https://github.com/buildroot/buildroot.git
2
```

Considerando que la Raspberry Pi 4 es una SBC muy común, podemos partir de una configuración ya hecha para esta tarjeta. Para esto, se puede usar el comando `make` seguido del nombre de la configuración deseada. En este caso, la configuración es `make raspberrypi4_64_defconfig`.

```
1 make raspberrypi4_64_defconfig
2
```

Esto generará un archivo de configuración en `.config` que contiene todas las opciones necesarias para compilar el sistema embebido.

### 4.3. Configuración de herramientas

Para configurar nuestro sistema más allá de lo que se encuentra en la configuración por defecto, se puede usar el comando `make menuconfig`. Este comando abrirá una interfaz gráfica en la terminal que permite seleccionar las opciones deseadas.

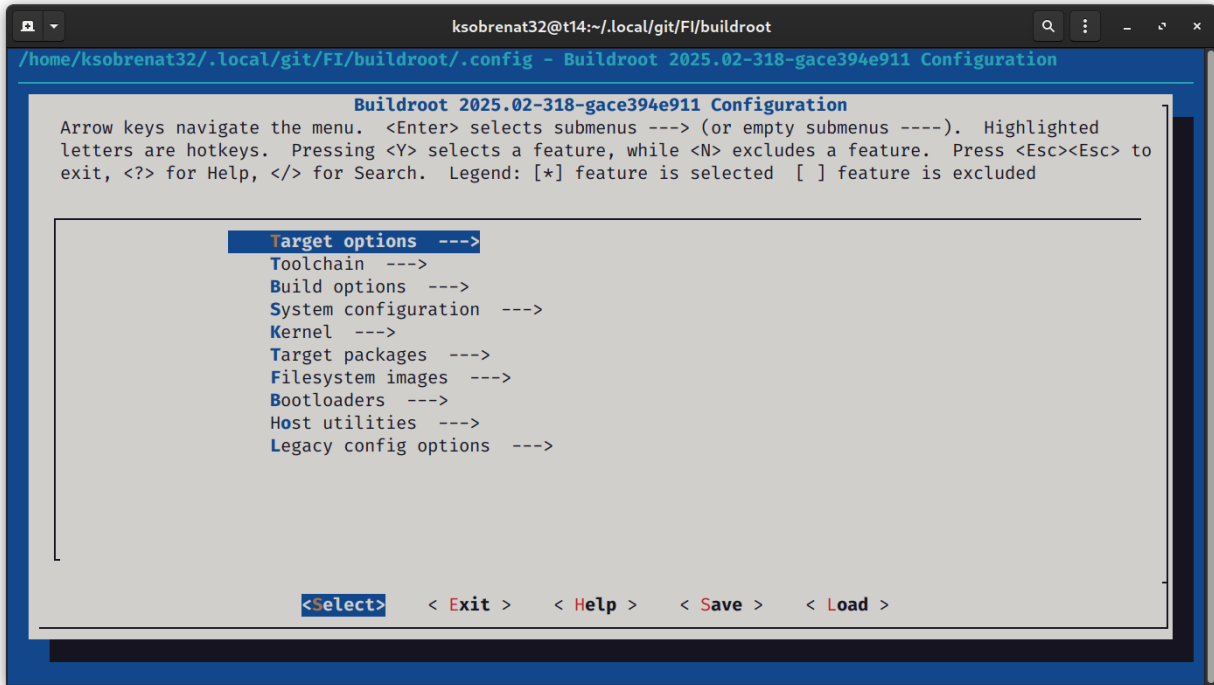


Figura 1: Interfaz gráfica de `make menuconfig`

Para reducir el tamaño del sistema embebido, Se seleccionará `musl` en lugar de `glibc` como la biblioteca estándar de C. Esto permitirá reducir aún el tamaño del sistema embebido.

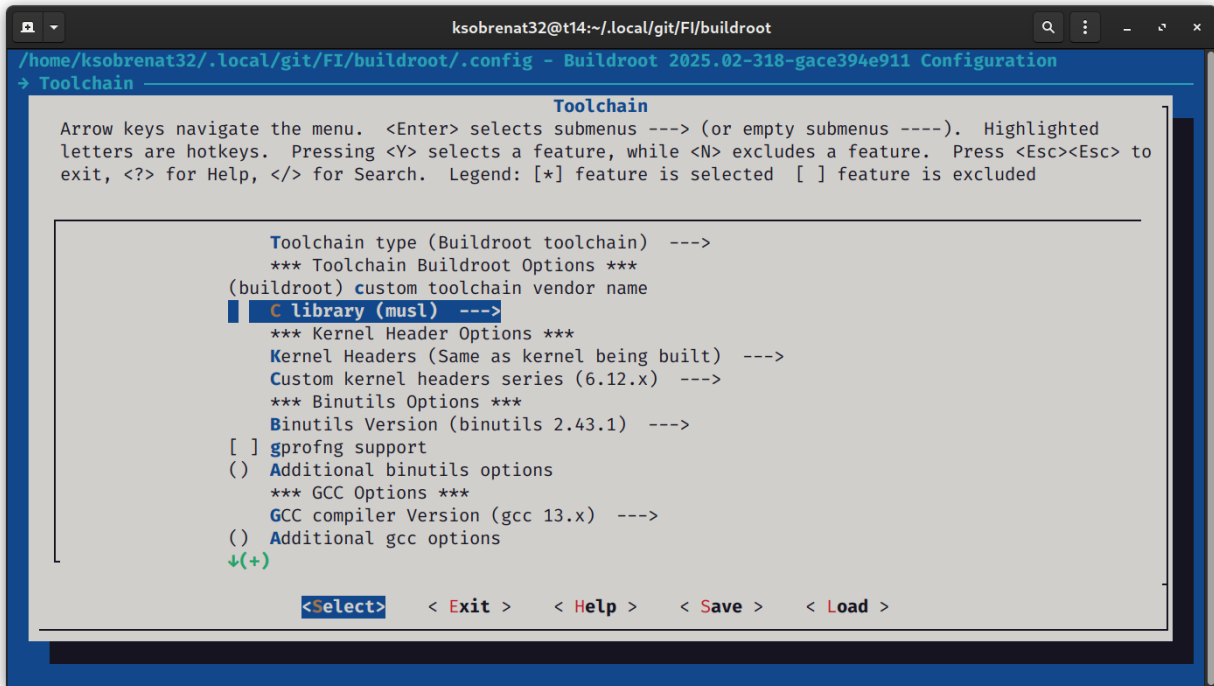


Figura 2: Configuración de musl como biblioteca estándar de C

En cuanto a las aplicaciones de usuario, se seleccionará `vim` como editor de texto y `python3` como intérprete de python. Esto permitirá tener un entorno de desarrollo básico en el sistema embebido.



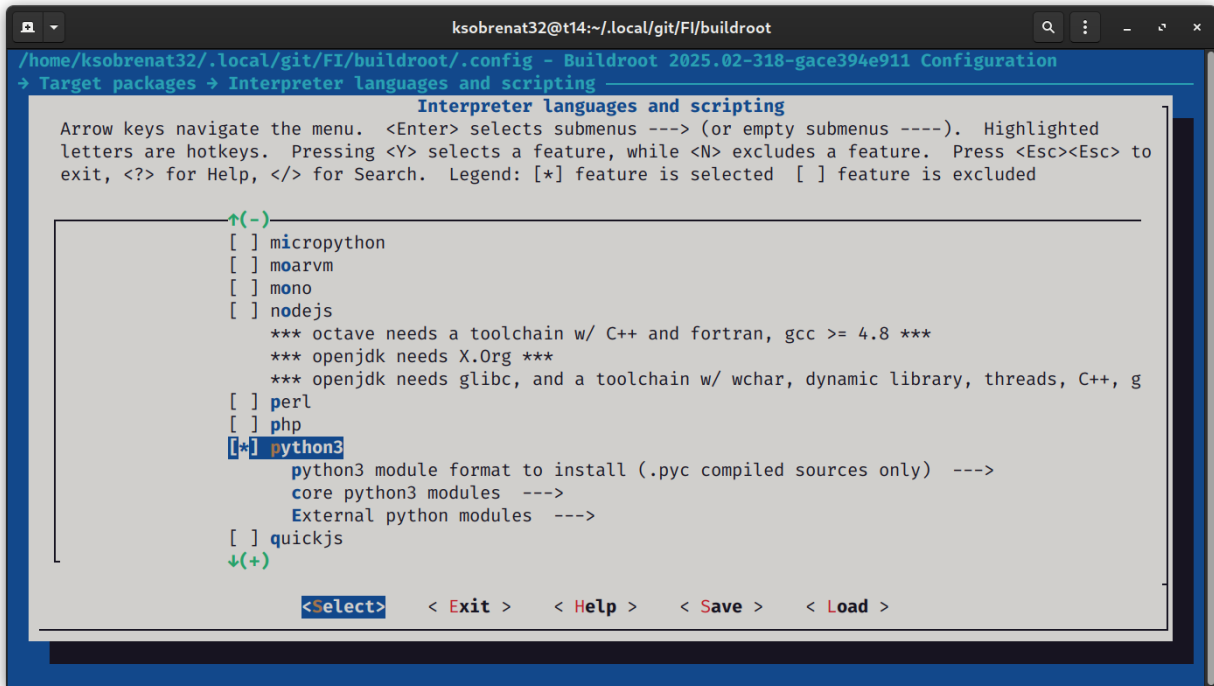


Figura 3: Instalación de python3

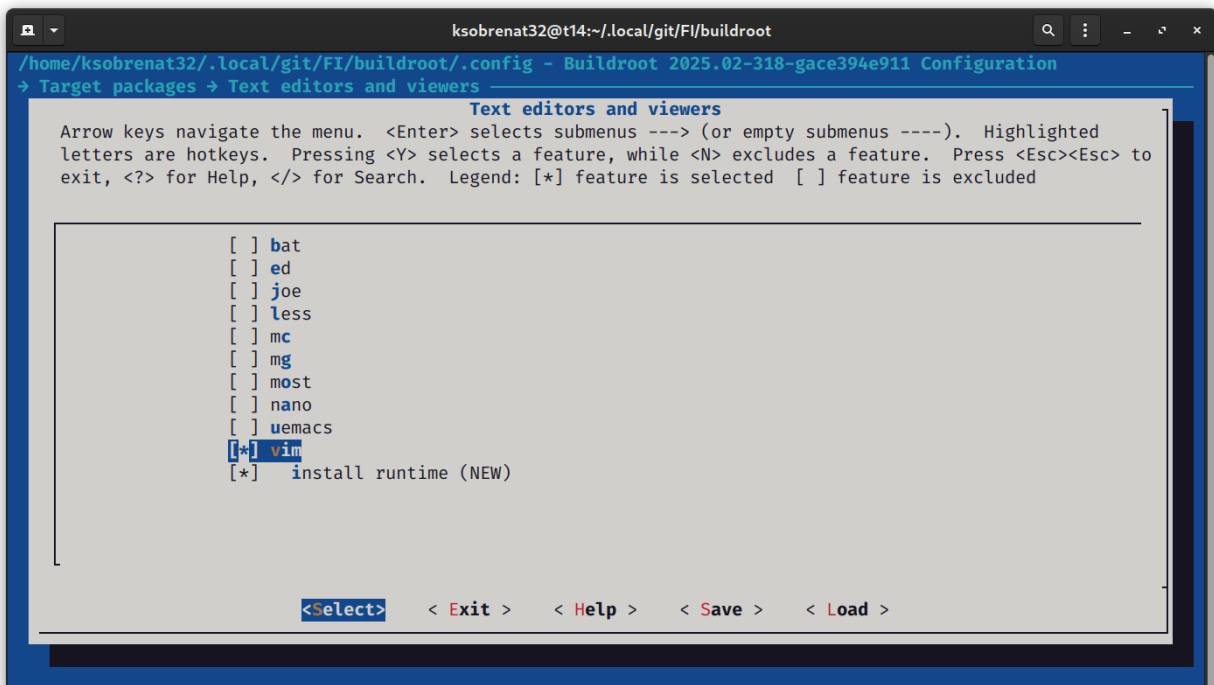


Figura 4: Instalación de vim

Añadiremos htop como herramienta de monitoreo del sistema. Esto permitirá ver el uso de CPU, memoria y procesos en ejecución.

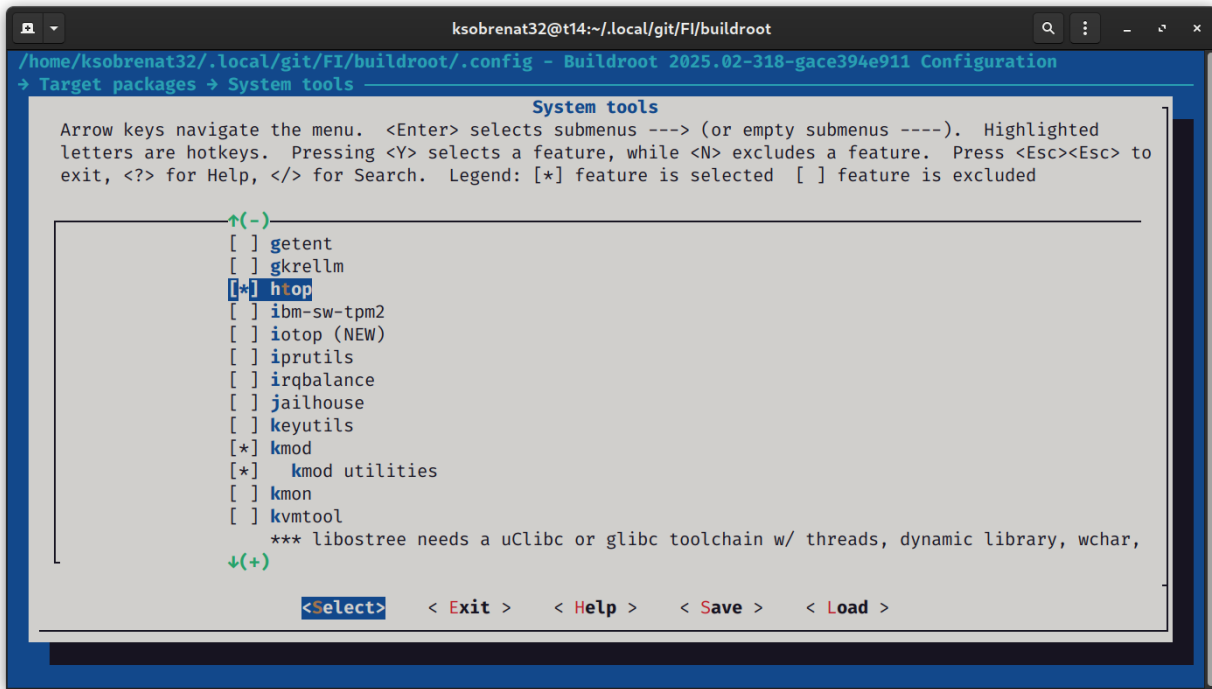


Figura 5: Instalación de htop

Con esto, se ha configurado el sistema embebido para que tenga un entorno de terminal básico con python y vim. Se puede agregar más software si se desea, pero esto aumentará el tamaño del sistema embebido.

#### 4.4. Compilación

Para empezar la compilación del sistema embebido, se usa el comando **make**. Este comando iniciará la compilación del sistema embebido y generará una imagen de disco que se puede usar para instalar el sistema en la tarjeta Raspberry Pi 4.

A diferencia de otras herramientas, buildroot no requiere de un compilador externo. Buildroot incluye su propio compilador y herramientas de construcción, lo que facilita la creación de un sistema embebido personalizado. Además, podemos olvidarnos de indicar el número de núcleos de la computadora, ya que buildroot se encarga de esto automáticamente.

```
ksobrenat32@t14:~/local/git/FI/buildroot
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/bcm2711-rpi-cm4.dtb' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'bcm2711-rpi-cm4s.dtb' as 'bcm2711-rpi-cm4s.dtb' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/bcm2711-rpi-cm4s.dtb' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'rpi-firmware/cmdline.txt' as 'rpi-firmware/cmdline.txt' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/rpi-firmware/cmdline.txt' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'rpi-firmware/config.txt' as 'rpi-firmware/config.txt' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/rpi-firmware/config.txt' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'rpi-firmware/fixup4.dat' as 'rpi-firmware/fixup4.dat' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/rpi-firmware/fixup4.dat' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'rpi-firmware/overlays' as 'rpi-firmware/overlays' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/rpi-firmware/overlays' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'rpi-firmware/start4.elf' as 'rpi-firmware/start4.elf' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/rpi-firmware/start4.elf' ':' (stderr):
INFO: vfat(boot.vfat): adding file 'Image' as 'Image' ...
INFO: vfat(boot.vfat): cmd: "MT00LS_SKIP_CHECK=1 mcopy -sp -i '/home/ksobrenat32/.local/git/FI/buildroot/output/im
ages/boot.vfat' '/home/ksobrenat32/.local/git/FI/buildroot/output/images/Image' ':' (stderr):
INFO: hdimage(sdcad.img): adding primary partition 'boot' (in MBR) from 'boot.vfat' ...
INFO: hdimage(sdcad.img): adding primary partition 'rootfs' (in MBR) from 'rootfs.ext4' ...
INFO: hdimage(sdcad.img): adding primary partition '[MBR]' ...
INFO: hdimage(sdcad.img): writing MBR
INFO: cmd: "rm -rf "/home/ksobrenat32/.local/git/FI/buildroot/output/build/genimage.tmp/" (stderr):
[ksobrenat32@t14]:buildroot $
```

Figura 6: Compilación finalizada con éxito

## 4.5. Instalación

Una vez que la compilación ha finalizado, se generará una imagen de disco en la carpeta `output/images`. Esta imagen se puede usar para instalar el sistema embebido en la tarjeta Raspberry Pi 4.

Para instalar el sistema embebido en la tarjeta, se puede usar el comando `dd` para copiar la imagen de disco a la tarjeta SD. Asegúrate de que la tarjeta SD esté conectada a la computadora y que sepas cuál es el dispositivo correspondiente (por ejemplo, `/dev/sdX`).

```
1 sudo dd if=output/images/sdcad.img of=/dev/sdX bs=4M status=progress
2 sync
3
```

## 4.6. Pruebas

Una vez que la imagen ha sido copiada a la tarjeta SD, se puede insertar la tarjeta en la Raspberry Pi 4 y encenderla. El sistema embebido debería arrancar y mostrar un entorno de terminal.

```
ksobrenat32@t14:~/local/git/FI/buildroot
[ 2.870505] usb 1-1: New USB device found, idVendor=2109, idProduct=3431, bcdDevice= 4.21
[ 2.878720] usb 1-1: New USB device strings: Mfr=0, Product=1, SerialNumber=0
[ 2.885865] usb 1-1: Product: USB2.0 Hub
[ 2.891102] hub 1-1:1.0: USB hub found
[ 2.895096] hub 1-1:1.0: 4 ports detected
[ 2.931439] EXT4-fs (mmcblk0p2): re-mounted c3fb2e13-c045-425c-9d59-b0ead3ae572a r/w. Quota mode: none.
Saving 256 bits of creditable seed for next boot
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Starting network: [ 3.012189] bcmgenet fd580000.ethernet: configuring instance for external RGMII (RX delay)
[ 3.021339] bcmgenet fd580000.ethernet eth0: Link is Down
udhcpd: started, v1.37.0
udhcpd: broadcasting discover
udhcpd: no lease, forking to background
OK
Starting crond: OK

Welcome to Buildroot
buildroot login: root
# cat /etc/os-release
NAME=Buildroot
VERSION=2025.02-318-gace394e911
ID=buildroot
VERSION_ID=2025.05-git
PRETTY_NAME="Buildroot 2025.05-git"
#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | ttyUSB0
```

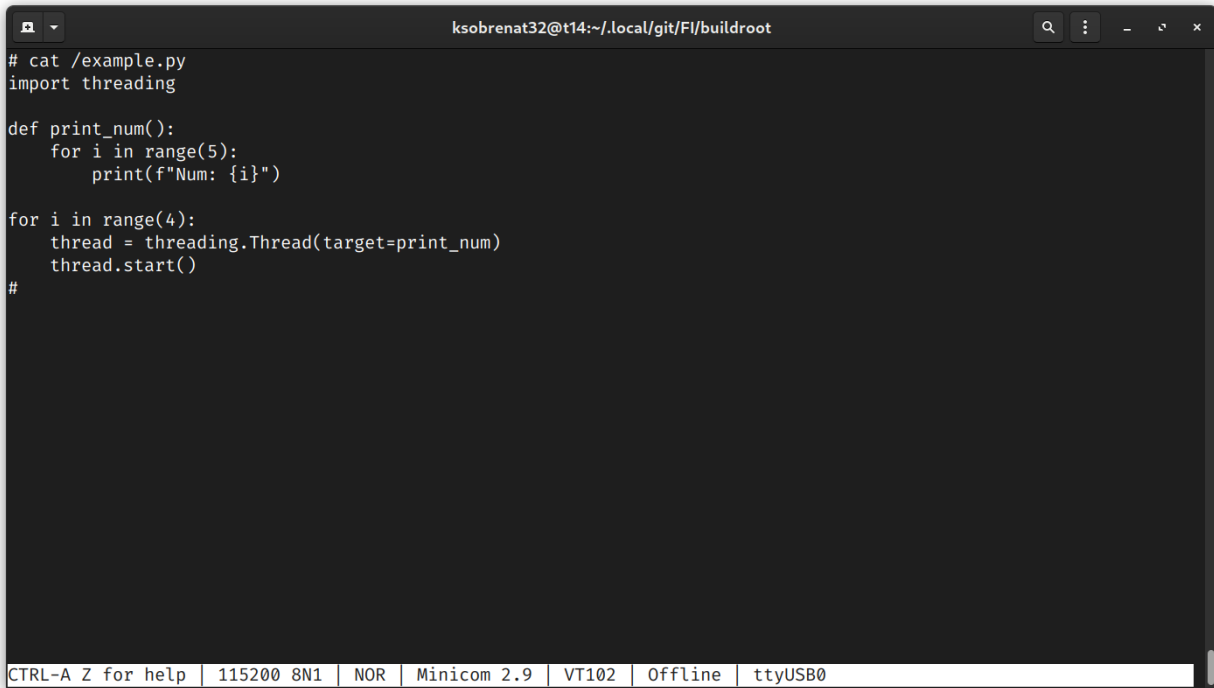
Figura 7: Arranque del sistema embebido

```
ksobrenat32@t14:~/local/git/FI/buildroot
#
# df -hT


| Filesystem | Type     | Size   | Used  | Available | Use% | Mounted on |
|------------|----------|--------|-------|-----------|------|------------|
| /dev/root  | ext4     | 107.0M | 63.3M | 35.3M     | 64%  | /          |
| devtmpfs   | devtmpfs | 1.8G   | 0     | 1.8G      | 0%   | /dev       |
| tmpfs      | tmpfs    | 1.8G   | 0     | 1.8G      | 0%   | /dev/shm   |
| tmpfs      | tmpfs    | 1.8G   | 36.0K | 1.8G      | 0%   | /tmp       |
| tmpfs      | tmpfs    | 1.8G   | 20.0K | 1.8G      | 0%   | /run       |


#
```

Figura 8: Uso del sistema de archivos



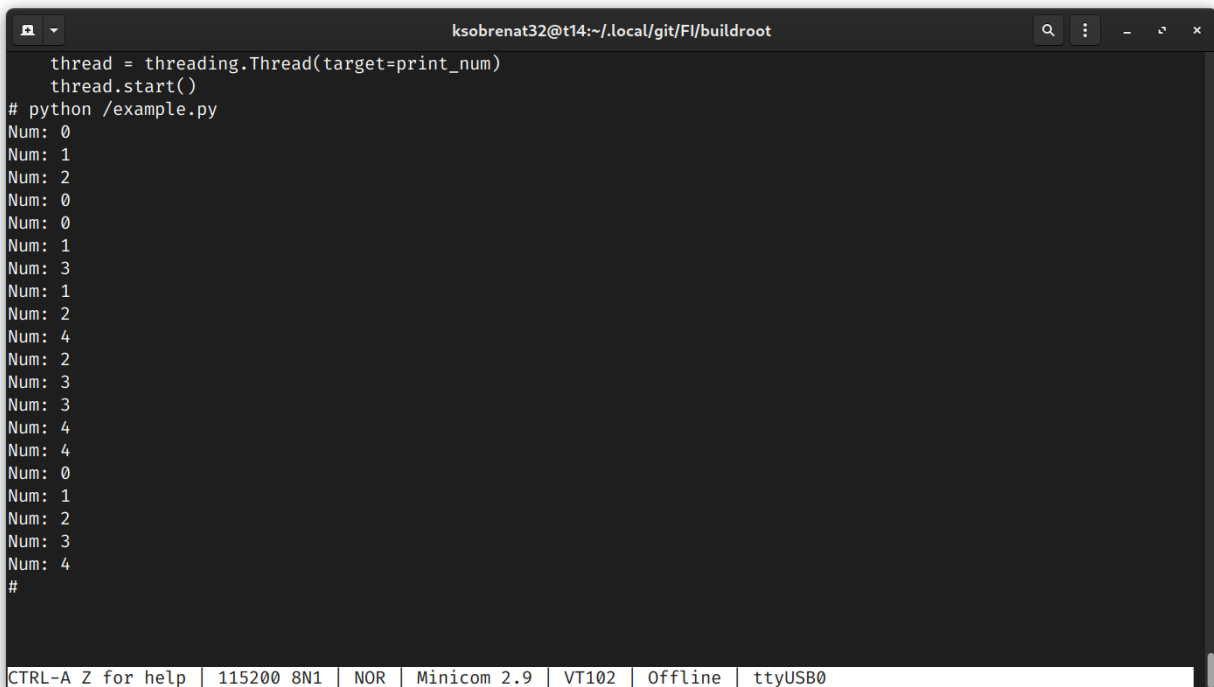
```
ksobrenat32@t14:~/local/git/FI/buildroot
# cat /example.py
import threading

def print_num():
    for i in range(5):
        print(f"Num: {i}")

for i in range(4):
    thread = threading.Thread(target=print_num)
    thread.start()
#
```

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | ttyUSB0

Figura 9: Ejemplo de código en python



```
ksobrenat32@t14:~/local/git/FI/buildroot
    thread = threading.Thread(target=print_num)
    thread.start()
# python /example.py
Num: 0
Num: 1
Num: 2
Num: 0
Num: 0
Num: 1
Num: 3
Num: 1
Num: 2
Num: 4
Num: 2
Num: 3
Num: 3
Num: 4
Num: 4
Num: 0
Num: 1
Num: 2
Num: 3
Num: 4
#
```

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.9 | VT102 | Offline | ttyUSB0

Figura 10: Ejecución de un script en python

```

ksobrenat32@t14:~/local/git/FI/buildroot

0[          0.0%] Tasks: 8, 0 thr, 97 kthr; 0 running
1[          0.0%] Load average: 0.00 0.00 0.00
2[          0.0%] Uptime: 00:04:20
3[          0.0%]
Mem[[] #0$      32.6M/3.68G]
Swp[[]          0K/0K]

[Main] [I/O]
PID USER      PRI  NI  VIRT   RES   SHR  S  CPU%-MEM%  TIME+  Command
  1 root        20   0  1676    768    768  S   0.0  0.0  0:00.14  init
117 root        20   0  1676    928    928  S   0.0  0.0  0:00.00  /sbin/syslogd -
121 root        20   0  1676    836    836  S   0.0  0.0  0:00.00  /sbin/klogd -n
155 root        20   0  1676    764    764  S   0.0  0.0  0:00.02  udhcpc -t1 -A3
160 root        20   0  1676    880    880  S   0.0  0.0  0:00.00  /usr/sbin/crond
161 root        20   0  1692   1076   1076  S   0.0  0.0  0:00.01  -sh
162 root        20   0  1676    828    828  S   0.0  0.0  0:00.00  /sbin/getty -L
225 root        20   0  1760   1464   1208  R   0.6  0.0  0:00.10  htop

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

CTRL-A Z for help | 115200 8N1 | APP | Minicom 2.9 | VT102 | Offline | ttyUSB0

```

Figura 11: Uso de **htop** para monitorear el sistema

## 5. Bibliografía

- [1] Administrador CEUPE. (s. f.). *Sistema embebido: Qué es, características y componentes*. Recuperado de: <https://www.ceupe.com/blog/sistema-embebido.html>
- [2] Rebound Electronics. (s. f.). *Explicación de los sistemas embebidos*. Recuperado de: <https://reboundeu.com/es/insights/blog/embedded-systems-explained-15/>
- [3] Red Hat. (2023, enero 3), *Definición de Linux*. Recuperado de: <https://www.redhat.com/es/topics/linux/what-is-linux>
- [4] TRBL Services. (2021, julio 27), *LINUX Embebido / Qué es, cómo funciona y para qué se usa*. Recuperado de: <https://trbl-services.eu/blog-linux-embebido-que-es/>
- [5] Buildroot. (s. f.). *Buildroot manual*. Recuperado de: <https://buildroot.org/downloads/manual/manual.html>
- [6] Equipo editorial de IONOS. (2021, julio 5). Kernel - El núcleo del sistema operativo. Recuperado de: <https://www.ionos.mx/digitalguide/servidores/know-how/que-es-el-kernel/>