

# BuildRoot

Mg. Ing. Gonzalo E. Sanchez  
MSE - 2022

**Implementación de Sistemas Operativos II**

# BuildRoot

- Introducción

# Introducción

# BuildRoot

- BuildRoot es una herramienta que simplifica y automatiza el proceso de construcción de un sistema Linux completo.
- Pensado para sistemas embebidos, se prevé compilación cruzada.
- BuildRoot puede generar:
  - Toolchain (cross-compiler).
  - Root FileSystem.
  - Imagen de Kernel.
  - Bootloader.

# BuildRoot

- No necesariamente deben utilizarse todas las configuraciones.
- Puede usarse una combinación de cualquiera de ellas.
- Para esta materia, utilizamos el toolchain ya generado.
- Soporta varios procesadores y arquitecturas.
- Existen archivos defconfig para varias SBC en el mercado.
- Existen proyectos que basan su BSP en BuildRoot.
- Existen proyectos que basan su SDK en BuildRoot.

# BuildRoot

- **IMPORTANTE:** BuildRoot tiene una política de que todo se construya sin privilegios root.
- Todos los comandos necesarios corren en modo usuario.
- Esto permite una protección del sistema HOST por cualquier configuración equivocada o comportamiento inesperado.

# BuildRoot

- No es necesario hacer ningún export para compilar buildroot.
- Esto es porque el toolchain se especifica en la configuración junto con los paths correspondientes.
- BuildRoot no soporta parallel-builds, por lo que ejecutar **make -jN** no es necesario.
- Sin embargo se pueden especificar cantidad de trabajos paralelos en la configuración.

# BuildRoot

- El comando make ejecutará los siguientes pasos:
  - Descarga de archivos fuentes (a requerimiento).
  - Configuración, construcción e instalación de toolchain para cross compilación (si no se especifica un toolchain externo).
  - Configuración, construcción e instalación de los paquetes seleccionados para el TARGET.
  - Construcción de una imagen de Kernel (si es seleccionado).
  - Construcción de un bootloader (si es seleccionado).
  - Creación de root filesystem en los formatos seleccionados.



# BuildRoot

- La salida se guarda en un solo directorio **output/** que contiene:
  - **images/** donde se guardan el kernel, bootloader y root filesystem.
  - **build/** donde se construyen todos los componentes. Incluye herramientas necesarias para BuildRoot en el HOST.
  - **host/** contiene tanto las herramientas construidas para el HOST como el sysroot para el toolchain.
  - **staging/** es un link simbólico al sysroot del toolchain. Solo para compatibilidad hacia atrás.
  - **target/** contiene el root filesystem casi completo. Todo excepto el contenido de /dev/ (recordar todo se corre como usuario regular).

# BuildRoot

- La opción de un toolchain externo permite la utilización de un cross compiler ya construido (nuestro caso).
- Se da la posibilidad de descargar el toolchain de forma automática (Sourcery CodeBench y ex-Linaro para ARM).
- El soporte para toolchains externos esta testeado para juegos de herramientas generados por crosstool-ng.

# BuildRoot

- La administración de dispositivos dentro de /dev puede configurarse de cuatro maneras distintas.
- **Primer método:** Estático utilizando una tabla de dispositivos.
- Con este método, los dispositivos son persistentes almacenados en el root filesystem.
- No existe forma de crear o remover estos dispositivos dinámicamente.
- La tabla de dispositivos se crea en **system/device table dev.txt** en el código fuente.

# BuildRoot

- **Segundo método:** Dinámico utilizando solo **devtmpfs**.
- Recordar que este pseudo filesystem se monta en **/dev**.
- Dispositivos se agregan o son removidos de forma automática.
- No son persistentes entre reinicios del sistema.

# BuildRoot

- **Tercer método:** Dinámico utilizando solo **devtmpfs + mdev**.
- También depende del pseudo filesystem devtmpfs.
- Agrega la aplicación espacio usuario **mdev** sobre devtmpfs.
- **mdev** termina siendo una aplicación de busybox que es llamada cada vez que un dispositivo se agrega/remueve.
- Utilizando **/etc/mdev.conf**, mdev puede configurarse para especificar permisos o pertenencia de un device file.
- También puede llamarse scripts al agregar ciertos dispositivos.

# BuildRoot

- En general, **mdev** permite al espacio usuario reaccionar a eventos de conexión o desconexión de dispositivos.
- mdev también puede ser utilizado para cargar módulos automáticamente al conectarse un dispositivo.
- También es útil cuando un dispositivo necesita firmware, porque carga este firmware en el kernel.
- **mdev** es una implementación lightweight de **udev** (menos funcionalidades).

# BuildRoot

- **Cuarto método:** Dinámico utilizando solo **devtmpfs** + **eudev**
- Muy similar al método anterior, utiliza el daemon **eudev** en vez de **mdev**.
- **eudev** corre en background, contrario a **mdev** que es llamado en cada cambio de devtmpfs (daemon vs app).
- Es más flexible que mdev, pero más pesado.

# BuildRoot

- **eudev** es en realidad una version stand alone de **udev** (**udev** es parte de **Systemd** en linux de escritorio).
- **NOTA:** En el caso de utilizar **systemd** como sistema **init**, entonces **udev** es utilizado sobre **devtmpfs**.



# BuildRoot

## HANDS ON

1. Descargar BuildRoot
2. Generar un sistema mínimo con BuildRoot



Gracias.

