

## Laboratorio: Compilación Cruzada

Autor: Víctor Hugo García Ortega

La compilación cruzada consiste en compilar, en una arquitectura, un programa para generar el archivo ejecutable para otra arquitectura diferente a la del procesador donde se compila.

Por ejemplo: Podemos utilizar una computadora de escritorio o portátil con una arquitectura x86 (con procesador INTEL o AMD) y compilar un programa en lenguaje C para un procesador con una arquitectura ARM. Esta es la arquitectura, ARM, en la que se basan los sistemas SoC de Raspberry.

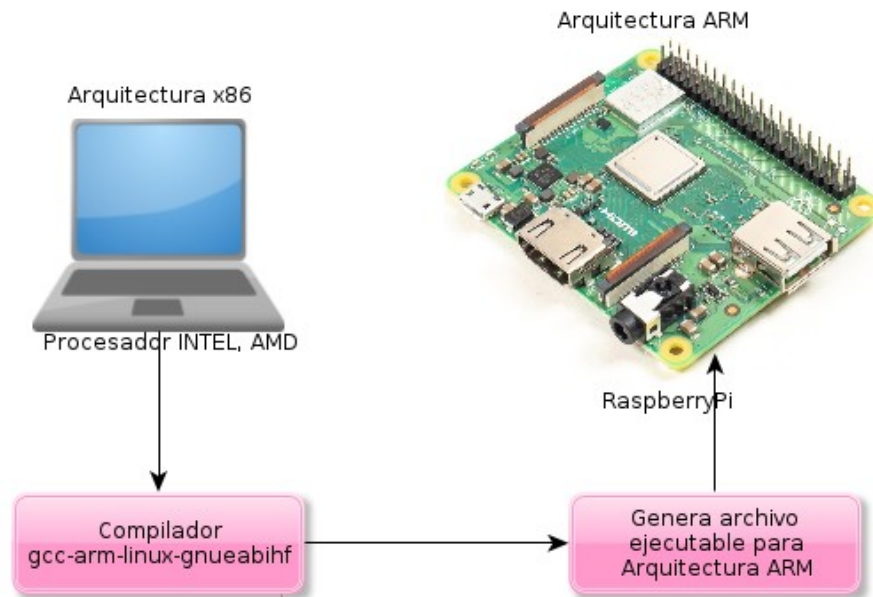


Figura 1: Compilación cruzada

### 1. Instalación de paquetes

Primero se deben instalar todos los paquetes necesarios para la compilación cruzada.

```
$ sudo apt-get install git bc bison flex libssl-dev make lib32z1
```

### 2. Instalación de la herramienta de compilación

Se debe instalar el compilador de lenguaje C para la arquitectura ARM, llamado "gcc-arm-linux-gnueabi", y sus herramientas, a todo eso se le conoce como "Toolchain". Esto se hace clonando el repositorio de herramientas de RaspberryPi en nuestro "home".

```
$ git clone https://github.com/raspberrypi/tools ~/tools
```

Se debe actualizar de la variable de entorno \$PATH, esto hace que el sistema conozca las ubicaciones de archivos necesarias para la compilación cruzada.

```
$ echo PATH=$PATH:~/tools/arm-bcm2708/arm-linux-gnueabi/bin >> ~/.bashrc  
$ source ~/.bashrc
```

### 3. Probando la herramienta de compilación.

En este momento podemos compilar un programa en lenguaje C en una computadora con arquitectura x86 y generar el archivo ejecutable para una arquitectura ARM.

Escribir el programa:

```
hola.c x
/** @brief: Este programa imprime "Hola mundo"
 */

#include <stdio.h>

int main()
{
    printf("Hola mundo \n");

    return 0;
}
```

Compilar usando el “toolchain” instalado:

```
$ arm-linux-gnueabi-gcc hola.c -o hola
```

Si se quiere correr el programa ejecutable obtenido después de la compilación cruzada en una arquitectura x86 se obtiene el siguiente error:

```
victorhgo@victor-pc:~/kernelProg$ arm-linux-gnueabi-gcc hola.c -o hola
victorhgo@victor-pc:~/kernelProg$ ./hola
bash: ./hola: no se puede ejecutar fichero binario: Formato de ejecutable incorrecto
```

### 4. Obteniendo el código fuente de la imagen del kernel de RaspberryPi.

Para realizar este paso clonamos el repositorio:

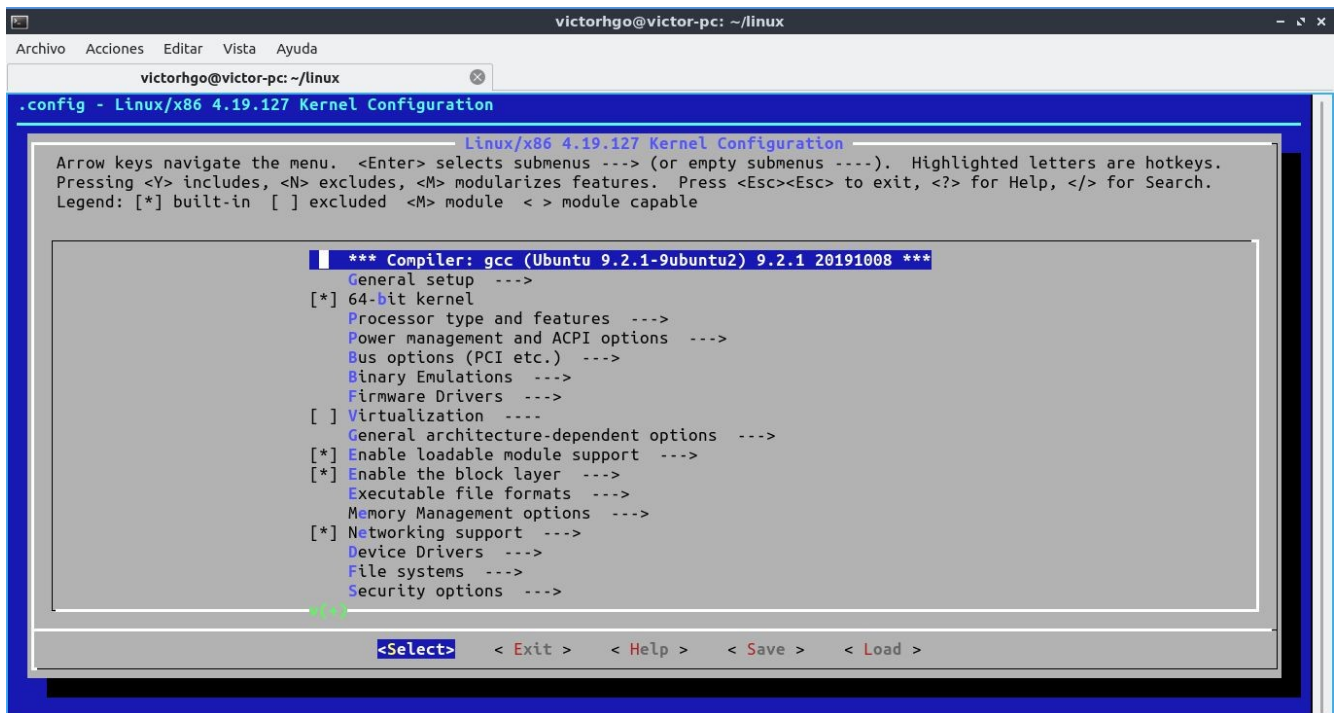
```
$ git clone --depth=1 https://github.com/raspberrypi/linux
```

Si se omite `--depth=1` se descargará todo el repositorio, incluido el historial completo de todas las ramas, esto llevará mucho más tiempo para descargar el repositorio y ocupará más espacio de almacenamiento.

La forma tradicional de configurar el kernel de linux es usando su menu de configuración el cual se obtiene con su archivo `makefile`.

```
$ make menuconfig
```

Con este menú podemos configurar cual opción que se desee para optimizar el kernel de linux.



Otra forma para realizar la configuración del kernel para la arquitectura ARM es ejecutar el archivo de configuración ubicado en:

```
$~/linux/arch/arm/configs/bcm2709_defconfig
```

Para las versiones Pi 1, Pi Zero, Pi Zero W, o Módulo de Computo:

```
$ cd linux
```

```
$ KERNEL=kernel
```

```
$make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig
```

Para las versiones Pi 2, Pi 3, Pi 3A+, Pi 3B+ o Módulo de Computo 3:

```
$ cd linux
```

```
$ KERNEL=kernel7
```

```
$make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
```

Para la versión Pi 4:

```
$ cd linux
```

```
$ KERNEL=kernel7
```

```
$make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2711_defconfig
```

## 5. Reconstruyendo el kernel.

Para reconstruir el kernel del Soc RaspberryPi y actualizarlo a su última versión con todos sus drivers disponibles ejecutamos:

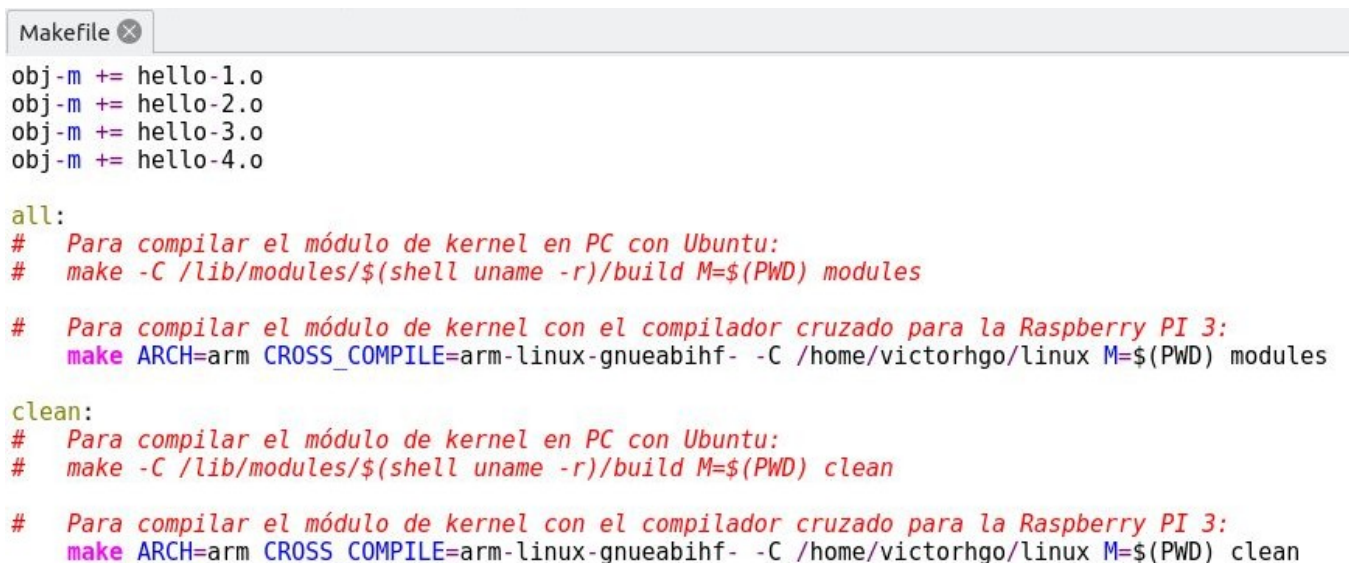
```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

Podemos agregar -jn, donde “n” es el número de procesadores. Con esto la compilación se paralelizará y se realizará mas rápido.

```
$ make -j4 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

## 6. Realizando compilación cruzada de módulos de kernel.

Para hacer compilación cruzada de módulos de kernel debemos usar el compilador para la arquitectura ARM, “gcc-arm-linux-gnueabihf”, y la herramienta de construcción kbuild del código fuente del kernel de linux para ARM. En los pasos anteriores ya se hizo eso, por lo que procedemos a modificar el archivo Makefile.



```
Makefile
obj-m += hello-1.o
obj-m += hello-2.o
obj-m += hello-3.o
obj-m += hello-4.o

all:
# Para compilar el módulo de kernel en PC con Ubuntu:
# make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

# Para compilar el módulo de kernel con el compilador cruzado para la Raspberry PI 3:
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C /home/victorhgo/linux M=$(PWD) modules

clean:
# Para compilar el módulo de kernel en PC con Ubuntu:
# make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

# Para compilar el módulo de kernel con el compilador cruzado para la Raspberry PI 3:
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C /home/victorhgo/linux M=$(PWD) clean
```

Figura 2: Archivo Makefile para compilación cruzada de módulos de kernel