

Tutorial: Yocto Project Parte II

1. Introducción

Este tutorial presenta la continuación a la herramienta del proyecto Yocto, para la generación de imágenes a la medida de Linux para sistemas embebidos. A continuación se describen los pasos para la generación de una imagen de Linux a la medida para la tarjeta Raspberry pi 2, así como los pasos para la construcción de su cross-toolchain respectivo y un ejemplo básico de compilación cruzada sobre la tarjeta utilizando la imagen y el toolchain creado.

2. Pasos para construcción de imagen

1. Dentro del directorio poky-sumo-19.0.1/ descargar el paquete de soporte de tarjeta (BSP) para la plataforma Raspberry pi 2:

```
git clone -b sumo http://git.yoctoproject.org/cgit.cgi/meta-raspberrypi
```

2. *(dentro del TEC) En el directorio raíz de yocto poky-sumo-19.0.1/, ejecutar script para corregir problemas de puertos bloqueados con git:

```
./git_fix meta-raspberrypi
```

3. *(dentro del TEC) Descargar manualmente kernel de Raspberry pi: Este es uno de los pasos automáticos en la generación de la imagen para Raspberry pi. En algunas redes (como el caso del TEC) la descarga se congela, lo que causa que el proceso de generación de la imagen no termine nunca. Para solucionarlo:

- a) Crear un directorio llamado *files*, dentro de directorio meta-raspberrypi/recipes-bsp/bootfiles/

- b) Dentro de meta-raspberrypi/recipes-bsp/bootfiles/files, descargar kernel de Raspberry pi:

```
git clone http://github.com/raspberrypi/firmware.git
```

- c) Una vez finalizado (proceso puede tardar +2h), crear archivamiento para kernel:

```
tar -cvf git2_github.com.raspberrypi.firmware.git.tar firmware
```

- d) Editar archivo meta-raspberrypi/recipes-bsp/common/firmware.inc . Las líneas correspondientes deberán verse como sigue:

```
RPIFW_SRC_URI = "file://git2_github.com.raspberrypi.firmware.git.tar"
RPIFW_S ?= "${WORKDIR}/firmware"
```

4. *(dentro del TEC) Modificar el archivo meta-raspberrypi/recipes-kernel/linux/linux-raspberrypi_4.9.bb, línea 6. La línea correspondiente verse como:

```
SRC_URI = "git://github.com/raspberrypi/linux.git;branch=rpi-4.9.y;protocol=http"
```

5. Crear directorio para construcción de imagen:

```
mkdir rpi2
```

6. Configurar el ambiente de Yocto para el directorio de construcción:

```
source oe-init-build-env rpi2
```

7. Editar archivo de configuración local para construcción (conf/local.conf). Agregar al final del archivo:

```
MACHINE ?= "raspberrypi2"
INHERIT += "rm_work"
DL_DIR ?= "/home/usuario/poky-sumo-19.0.1/downloads"
```

Esto corresponde respectivamente a: cambiar la máquina objetivo de construcción a Raspberry pi 2, directiva para eliminar archivos intermedios en construcción, cambiar directorio de descargas a utilizado anteriormente, habilitar paquetes para conexión wifi y habilitar openssl, necesario para funcionalidad con comando *wget*, que será utilizado posteriormente.

8. Agregar capa de soporte de tarjeta (BSP) para Raspberry pi 2 (meta-raspberry) a archivo de configuración de capas de Yocto (conf/bblayers.conf). El archivo final debe verse como:

```
BBLAYERS ?= " \
    /home/usuario/poky-pyro-17.0.0/meta \
    /home/usuario/poky-pyro-17.0.0/meta-poky \
    /home/usuario/poky-pyro-17.0.0/meta-yocto-bsp \
    /home/usuario/poky-pyro-17.0.0/meta-raspberrypi \
"
```

Esto permite que se habilite la construcción de todas las recetas (.bb) y paquetes, en general, necesarios para la construcción de imágenes para la tarjeta Raspberry pi 2.

9. Construir la imagen:

```
bitbake rpi-basic-image
```

10. Al finalizar, podrá copiar la imagen a una tarjeta uSD de la forma tradicional:

a) Ingresar a directorio de imagen creada:

```
cd tmp/deploy/images/raspberrypi2
```

b) Copiar imagen a uSD (of=/dev/sdb o of=/dev/mmcblk0 según corresponda).

```
sudo dd if=rpi-basic-image-raspberrypi2.rpi-sdimg of=/dev/mmcblk0 bs=1M  
sync
```

Puede usar de igual manera el método con ddrescue. Al finalizar, la imagen está lista ser utilizada en la Raspberry pi. El usuario para inicio de sesión de *root*

3. Pasos para construcción de cross-toolchain

En esta sección se describen los pasos para la construcción y uso del *toolchain* cruzado para desarrollo de aplicaciones para la imagen de Raspberry pi 2, creada en la sección anterior.

1. Dentro de directorio raíz de yocto (poky-sumo-19.0.1). Ejecutar script para configuración de ambiente sobre el directorio de construcción de imagen para Raspberry pi 2:

```
source oe-init-build-env rpi2
```

2. Crear toolchain. El paquete para el toolchain se construye de la misma manera que se construye una imagen para la plataforma, como sigue:

```
bitbake rpi-basic-image -c populate_sdk
```

3. Una vez finalizado el proceso, debe instalar el toolchain. Se recomienda utilizar el directorio estandar sugerido por el script. Para esto debe ejecutarse:

```
. tmp/deploy/sdk/poky-glibc-i686-meta-toolchain-cortexa7hf-vfp-vfpv4-neon-toolchain-2.0.sh
```

Nota: puede que el script se nombre diferente (p.e poky-glibc-i386...). Verifique el archivo (o su equivalente) existe al realizar la construcción del toolchain en el directorio (tmp/deploy/sdk)

Prueba de toolchain

Una vez construido e instalado el toolchain, se realizará una prueba simple. Para esto deberá seguir los siguientes pasos:

1. Crear un directorio test_toolchain en /home/usuario/
2. Dentro de directorio test_toolchain, genere un archivo helloworld.c con el siguiente contenido:

```
// 'Hello World!' program
```

```
#include <stdio.h>
```

```
int main()
```

```
{
    printf("Hello World\n");
    return 0;
}
```

3. Ejecutar script para configuración de ambiente de compilación, utilizando el toolchain instalado:

```
. /opt/poky/2.0/environment-setup-cortexa7hf-vfp-vfpv4-neon-poky-linux-gnueabi
```

4. Probar ambiente de compilación: antes de realizar la compilación cruzada, verifique que las variables de ambiente estén configuradas para la compilación. Para esto, ejecute:

```
echo $CC
```

La salida del comando debería ser algo como:

```
arm-poky-linux-gnueabi-g++ -march=armv7-a -marm -mthumb-interwork
-mfloat-abi=hard -mfpu=neon-vfpv4 -mtune=cortex-a7
--sysroot=/opt/poky/2.0/sysroots/cortexa7hf-vfp-vfpv4-neon-poky-linux-gnueabi
```

De no ser el caso, debe verificar que el toolchain se ha instalado de la manera correcta. Si es necesario puede volver al paso 3 de la construcción del toolchain.

5. Compilación. Ejecute el siguiente comando para la compilación cruzada de la aplicación *helloworld.c*

```
$CC -o sayhello helloworld.c
```

6. *Ejecución.* Ejecute la aplicación *sayhello*

```
./sayhello
```

¿Por qué no debería verse *'Hello World!'* ? ¿Qué quiere decir la salida de la aplicación?

7. Inserte la uSD con la imagen de Linux creada para Raspberry pi 2 y copie la aplicación *sayhello* en la raíz del sistema de archivos:

```
sudo cp sayhello /media/rootfs/
```

8. Extraiga la tarjeta SD y conéctela a la Raspberry pi 2. Encienda la tarjeta, inicie sesión como root e ingrese al directorio raíz del sistema de archivos (dentro de la Raspberry pi):

```
cd /
```

9. Ejecute finalmente la aplicación *sayhello* dentro de la Raspberry pi:

```
./sayhello
```

¿Cuál es la salida en este caso? ¿Qué quiere decir?

4. Evaluación

Investigación

1. ¿Qué pasos debe seguir antes de escribir o leer de un puerto de entrada/salida general (GPIO)?
2. ¿Qué comando podría utilizar, bajo Linux, para escribir a un GPIO específico?

Ejercicio práctico:

Descripción

Debe crear una biblioteca dinámica para el manejo de dispositivos de entrada y salida digitales, por medio de `gpio`, para la plataforma Raspberry pi 2, en lenguaje C, la cuál ofrecerá las siguientes funciones:

- `pinMode (pin, MODE)`: Permite establecer el modo (entrada/salida) de un pin específico (número de pin/`gpio`).
- `digitalWrite(pin, value)`: Permite escribir un valor de 0 o 1 en el pin específico configurado como salida.
- `digitalRead(pin)`: Permite leer el estado (0,1) de un pin digital.
- `blink(pin, freq, duration)`: Permite generar un *blink* (establecer y desestablecer un valor binario) en un pin a una frecuencia determinada, por un tiempo de duración determinado.

Para esta biblioteca, debe utilizar las herramientas generadas con el proyecto Yocto (meta-toolchain) para la construcción de software bajo la metodología de desarrollo cruzado.

Para verificar la biblioteca deberá crear una aplicación que configure dos `gpios` de la tarjeta como salida y uno como entrada. La aplicación debe escribir valores binarios (0,1) a uno de los `gpio` configurados como salidas y en el otro establecer un *blink* a una frecuencia determinada por una duración de 5 s. Adicionalmente, se debe leer el valor del pin configurado como entrada y presentarlo en pantalla.

4.1. Entregable

(Subir al `tecDigital`)

- Archivamiento, con nombre: `Estudiante_Eval_Tut_7.tar.gz` que incluya los códigos fuente de aplicación y biblioteca, `makefiles`, `scripts`, y otros archivos necesarios para la construcción cruzada, siguiendo la estructura estándar vista en clase (directorios `src`, `include`, `lib`, `bin`,). Debe incluir además un archivo `README` que contenga nombre y carné de los estudiantes, así como la respuesta las preguntas y una guía sobre cómo construir y utilizar su aplicación.