Adding an Out-of-Tree Kernel Module in Yocto

Introduction

An **Out-of-Tree (OOT) Kernel Module** is a kernel module that is not part of the official Linux kernel source tree. This guide provides a step-by-step process to add an OOT kernel module in Yocto

---

Step 1: Set Up the Yocto Environment

```
# Set up the Yocto build environment
source oe-init-build-env
```

This creates the `build` directory where you configure your build.

Step 2: Create a Custom Layer for the Kernel Module

To keep the module organized, create a new layer:

```
cd $BUILDDIR
bitbake-layers create-layer ../meta-custom
bitbake-layers add-layer ../meta-custom
```

Ensure the layer is included in `bblayers.conf`.

Step 3: Add Kernel Module Source Code

Create a directory inside your custom layer for the module:

```
mkdir -p ../meta-custom/recipes-kernel/modules/my-module
cd ../meta-custom/recipes-kernel/modules/my-module
```

Copy your module source files (`hello.c` and `Makefile`) into this directory.

Step 4: Makefile for the Kernel Module

Use the following `Makefile` for building the kernel module:

```
obj-m := hello.o

SRC := $(shell pwd)

all:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC)

modules_install:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install

clean:
	rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
	rm -f Module.markers Module.symvers modules.order
	rm -rf .tmp_versions Modules.symvers
```

Importance of KERNEL_SRC Variable

The `KERNEL_SRC` variable is crucial because it tells the build system where the Linux kernel source code is located. This is necessary for compiling kernel modules outside the main kernel tree.

- When `$(MAKE) -C $(KERNEL_SRC)` is used, it changes the working directory to the kernel source directory and compiles the module in that context.
- `M=$(SRC)` specifies the directory where the module source files are located.
- This setup ensures that the external module is built using the kernel's build system, maintaining compatibility.

Step 5: Modify `local.conf` to Include the Module

Edit the `conf/local.conf` file and add the following line:

```
MACHINE_ESSENTIAL_EXTRA_RDEPENDS += "kernel-module-mymodule"
```

This ensures that the module is included in the root filesystem of the built image.

Step 6: Create a BitBake Recipe for the Kernel Module

Create a **BitBake recipe** file for the module:

```
touch mymodule_1.0.bb

Edit the mymodule_1.0.bb file:
SUMMARY = "Example of how to build an external Linux kernel module"
DESCRIPTION = "${SUMMARY}"
LICENSE = "GPL-2.0-only"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

SRC_URI = "file://Makefile \
           file://hello.c \
           file://COPYING \
          "

S = "${WORKDIR}"

RPROVIDES:${PN} += "kernel-module-hello"
```

Step 7: Build the Yocto Image

Run the following command to build the kernel and module:

```
bitbake core-image-minimal
```

Once built, check that `hello.ko` is in the `tmp/deploy` directories.

Step 8: Running the Kernel Module on the Target

After flashing the built image onto the target hardware, follow these steps:

1. **Boot the Target Device**

If using QEMU

```
runqemu qemuarm
```

Check if the Kernel Module is Installed

```
lsmod | grep hello
```

If the module is not listed, manually insert it:

```
modprobe hello
```

Manually Load the Module (If Not Auto-Loaded)

```
insmod /lib/modules/$(uname -r)/updates/hello.ko
```

Check Kernel Logs

```
dmesg | tail -n 20
```

This should confirm the module was loaded successfully.

Unload the Module

```
rmmod hello
```

Step 9: Understanding the `WORKDIR` Structure After Build

The `WORKDIR` is where Yocto stores temporary build files. You can navigate to it using:

```
cd tmp/work/<MACHINE>/<RECIPE_NAME>/<VERSION>/
```

Important Directories Inside `WORKDIR`

- **`temp/`** - Contains logs and execution scripts.
- **`build/`** - Where the module is compiled.
- **`image/`** - Contains the final packaged output.
- **`deploy/`** - Stores the final `.ko` file for installation.

To inspect build logs:

```
cat tmp/work/<MACHINE>/<RECIPE_NAME>/<VERSION>/temp/log.do_compile
```

This helps debug build issues effectively.
