# Práctica I

# Desarrollo de módulos de kernel

**Implementación de Manejadores de Dispositivos**

**Maestría en Sistemas Embebidos**

## Año 2022

**Autor**

## Mg. Ing. Gonzalo Sanchez

# Tabla de contenido

# Registro de cambios

| Revisión | Cambios realizados | Fecha |
|---|---|---|
| 1.0 | Creación del documento | 17/10/2021 |
| 1.1 | Agregado de subsección “Siguiendo lineamientos de escritura para el kernel (coding standards)” | 18/10/2021 |
| 1.2 | Corrección de errores de tipeo | 26/10/2022 |

# Escribiendo un módulo de kernel básico

## Objetivos de la práctica

Luego de esta guía de práctica, el lector debería estar en capacidad de crear módulos de kernel básicos, cuyo código está fuera de las fuentes del kernel, compilarlos y hacer test de ellos. Asimismo, el lector estará en capacidad de acceder a las definiciones internas del kernel y de crear un patch para el mismo.

## Primeros pasos

Para poder comenzar a construir un módulo de kernel, se recomienda crear un directorio nuevo, llamado *hello_world* en el directorio */root* que se está sirviendo a la SBC por medio del servidor NFS. Este directorio en realidad podría estar en cualquier sitio, solo es a fines de organizar el código.

```
$ mkdir -p -v $HOME/IMD/nfsroot/root/hello_world
```

## Escritura del módulo

Una de las grandes ventajas de utilizar un servidor NFS para el desarrollo es que se puede trabajar en la workstation y ver reflejado los cambios instantáneamente por la SBC. Esto acelera en gran manera el desarrollo y es deseable para aumentar la productividad y evitar errores, optimizando el tiempo a la hora de desarrollar módulos.

Dentro del directorio *hello_world* debe crearse un archivo **hello_version.c** que contenga el código del módulo propiamente dicho. Se deja a criterio del lector el editor de texto a utilizar.

Utilizando el ejemplo visto en la teoría, se desea implementar el código necesario para que el módulo **hello_version** muestre en pantalla un mensaje de la índole "*Hola Mundo. Estas utilizando Linux <version>*", donde el tag <version> muestre la versión del kernel que se está utilizando. Asimismo, el módulo debe mostrar un mensaje de despedida cuando es quitado de RAM.

Se sugiere al lector hacer una búsqueda entre los archivos fuente del kernel que contienen la palabra *versión* en sus nombres y ver su implementación, utilizándolos a modo de guía para la resolución del requerimiento planteado.

Otra sugerencia es que se inicie con un módulo que muestre un mensaje al inicio y al ser quitado de RAM para verificar que el esqueleto del mismo está funcionando correctamente, y luego agregar lo necesario para mostrar la versión.

Existen macros que proveen la versión del kernel que fue utilizado para compilar el módulo, pero no es lo que se busca en este momento. Se insta a que busque una variable o función que proporcione la información deseada de forma "*dinámica*" si así desea llamarlo.

## Compilación del módulo

Para compilar el módulo y obtener el archivo **hello_version.ko** se debe hacer uso de la herramienta make junto con el código fuente del módulo y un archivo make mínimo. Este archivo *make* indica los headers que son utilizados para la compilación del módulo, a fin de que pueda ser compilado de manera out-of-tree.

Es muy importante notar que esta compilación es en realidad una **compilación cruzada**: el módulo que se está compilando en la workstation será cargado y ejecutado en la SBC. El makefile mínimo presentado hace uso del makefile del kernel, por lo que las variables *ARCH* y *CROSS_COMPILE* son válidas y deben ser especificadas como cuando se compila el mismo kernel.

En el caso de querer utilizar un makefile que automatice las definiciones de estas variables, se propone otro código:

```
obj-m += hello_world.o
KERNEL_DIR ?= $(HOME)/<KERNEL_TOPDIR>
all:
     make -C $(KERNEL_DIR) \
```

```
    ARCH=arm CROSS_COMPILE=arm-linux- \
    SUBDIRS=$(PWD) modules
clean:
    make -C $(KERNEL_DIR) \
    ARCH=arm CROSS_COMPILE=arm-linux- \
    SUBDIRS=$(PWD) clean
deploy:
   scp *.ko root@<IP_SBC>:
```

Como se ve en el makefile propuesto, solo hay que modificar el nombre del módulo para cada proyecto. El tag <KERNEL_TOPDIR> hace referencia al path a la carpeta raíz de las fuentes de linux. En el caso que el sistema haya sido construido con soporte para SSH y este utilizando una memoria SD, el comando *make deploy* copiara mediante scp a la carpeta /root (que es el “home” del usuario root) cualquier archivo generado con extensión **.ko**. Para completar esta acción es necesaria la contraseña para root.

Para el caso de estudio, se puede modificar el comando deploy para copiar el módulo a la ubicación que se está sirviendo mediante NFS.

```
deploy:
   cp *.ko <.../nfsroot/DEST_DIR>
```

Este comando debe ser escrito según el directorio dentro del Root Filesystem donde se desee que esté alojado el módulo, reemplazando el tag simbólico por un path real.

## Testeo del módulo

Una vez compilado y copiado a la SBC, se puede testear el módulo cargando y verificando su funcionamiento. En el caso de que no funcione como lo esperado, debe

quitarse de RAM, reescribir su código, re-compilar y cargar nuevamente. Este ciclo es extremadamente sencillo mediante NFS. Para hacer un chequeo completo, una vez que el módulo fue cargado, se utiliza el comando **lsmod** para mostrar la lista de módulos cargados.

## Agregando parámetros

Para agregar parámetros al módulo que se está compilando se deben declarar variables y utilizar las macros **module_param** y **MODULE_PARM_DESC**.

Intente agregar información de usuario a su módulo, mostrándolo a través de la función *pr_info()*. Asimismo, intente agregar la cantidad de segundos que el módulo estuvo cargado, mostrando esta información al momento de quitar el módulo, mediante el uso de la función *ktime_get_seconds()*. Busque en las fuentes del kernel que drivers utilizan esta función y como lo hacen.

## Siguiendo lineamientos de escritura para el kernel (coding standards)

El código de los módulos debe apegarse a lineamientos de escritura muy estrictos si es deseo del desarrollador que este módulo sea algún día insertado en el mainline.

Una de las razones principales de estas reglas es la legibilidad del código. Si cualquiera utilizase su propio estilo, dada la cantidad de contribuidores, leer el código del kernel sería una tarea no grata.

Afortunadamente, la comunidad del kernel de linux provee una utilidad para encontrar violaciones a estas prácticas de escritura estándar. Lo primero necesario es poseer los paquetes **python3-ply** y **python3-git**. Una vez instalados, se puede correr el comando **scripts/checkpatch.pl -h** en las fuentes del kernel para ver las opciones disponibles.

Ejecutando el siguiente comando puede verse un reporte a modo de lista de la cantidad de violaciones al coding standard (si hubiese) en el archivo fuente del módulo escrito.

```
$ ~/linux/scripts/checkpatch.pl --file --no-tree hello_version.c
```

Se debe ejecutar este script cuantas veces sea necesario luego de corregir el código hasta tanto no haya ninguna violación. Es importante tener en cuenta que si existen muchos errores relacionados a indentación, puede que la fuente siendo verificada esté mal configurada, mayormente tabuladores en vez de espacios, y cantidad de espacios por indentación.

## Agregando el modulo hello_version a las fuentes de kernel

Lo primero que debe hacerse es crear una rama a partir del commit del cual se está trabajando:

```
$ git checkout linux-5.10.y

$ git pull

$ git checkout -b hello
```

Habiendo creado una nueva rama, se debe agregar el código fuente del módulo al directorio drivers/misc/ en las fuentes del kernel. Por supuesto, es necesario modificar la configuración del kernel y los archivos de construcción de forma acorde, de manera que el módulo sea seleccionable a través de menuconfig y se compile mediante el comando make.

Debe agregarse dentro del archivo Kconfig presente el en directorio donde se encuentran las fuentes del módulo el siguiente texto:

```
config MSE_HELLOWORLD

    tristate "Modulo hola mundo para IMD - MSE"

    default m

    help

        Utilice la opcion Y para compilar de manera built-in.
```

```
Por defecto se compila como modulo in-tree
```

Dentro del makefile que se encuentra en el mismo directorio debe agregarse la siguiente línea:

```
obj-$(CONFIG_MSE_HELLOWORLD) += hello_world.o
```

De esa manera el módulo está listo para compilar *in-tree* o de manera *built-in*.

Se debe verificar el funcionamiento y la compilación mediante la ejecución del comando make. Puede utilizarse el target *modules.*

```
$ make modules
```

Habiendo compilado, se pueden agregar los archivos fuente al indice de versionado mediante git add, y luego hacer un commit. Cabe destacar que todos los commits del kernel deben estar firmados, por lo que es obligatorio utilizar la opción -s al momento de hacer un commit.

```
$ git add <files>
```

```
$ git commit -as
```

A partir de este nuevo commit, generar un patch es muy sencillo, solamente se debe indicar la rama de la cual se partió para implementar la nueva funcionalidad mediante el comando format-patch.

```
$ git format-patch linux-5.10.y
```

Se invita al lector a investigar el archivo de salida.