CIC, FES, DM, JRB

Sistemas Embebidos

– XXXX –

10 de octubre de 2012

Springer

Capítulo 1
Herramientas libres para construir proyectos Hardware-Software

Resumen En los anteriores capítulos se ha tratado el uso de herramientas libres generar tanto los archivos finales a implementar en tanto para diseños software como para diseños Hardware. En este capítulo se pretende mostrar las diferentes etapas de la construcción de los diseños Hardware/Software a partir de la herramienta Make y la cadena de compilación basada en GCC.

1.1. Introduccion

Introduccion ....

1.2. Makefile

El desarrollo de aplicaciones a partir de herramientas libres se apoya en la herramienta de construcción *make*, la cual automatiza la creación los archivos finales de implementación Hardware/Software. El constructor *make* generalmente utiliza el archivo **Makefile** para tomar la especificación del proyecto que se desee procesa, basado en este se ejecutan las reglas que en él se especifiquen. En general el objetivo final del *make* es construir una aplicación, pero dada la flexibilidad que tiene se utiliza en la construcción de casi cualquier tipo de proyecto donde se requieran de muchas etapas en la construcción de un proyecto.

1.2.1. Target, prerequisitos y reglas asociadas

La herramienta *make* a través de las reglas descritas en el archivo *Makefile* determina cual y cuando se debe reconstruir un archivo para completar la construcción

del proyecto completo, todo esto se realiza tomando la marca de tiempo del archivo de salida y la marca de los fuentes, si los fuentes que son prerequisitos de una regla tiene una marca de tiempo mas actual, el constructor determina que este archivo se debe reconstruir. Las reglas del constructor se componen de un *target* o archivo de salida (generalmente un archivo de salida por regla) y una serie de pre-requisitos para cada regla, estos pre-requisitos son los archivos que deben existir dentro de las rutas de búsqueda del constructor. A continuación se muestra un ejemplo de la sintaxis de una regla para compilar *foo.c* en *foo.o*

```
target1.o:  source1.c header1.h
      gcc -c source1.c
```

En las reglas del constructor, el nombre del archivo ubicado antes de los dos puntos (**:**) se le denomina *target* en este caso *target1.o***:**. La lista de archivos que suceden a los dos puntos (**:**) se le llama la lista de prerequisitos.

Los prerequisitos son los archivos que el *target* necesita para su construcción, en este caso *source1.c* y *header1.h*. El tipo de archivo del *target* y de los prerequisitos no tiene importancia dado que únicamente se verifica su existencía o marca de tiempo. La linea que sigue a la del *target* se le conoce como la primer linea de ejecución de la regla *recipe*. Es necesario identar con un caracter de tabulación (→) las lineas de comandos para que el constructor las ejecute correctamente, esta ejecución se realiza en una consola diferente a la en que se invoca el comando *make*. Cuando se tienen lineas muy extensas se puede usar el carácter *backslash*

1.2.1.1. Tipos de reglas del constructor

Existen diferentes tipos de reglas que el constructor puede tomar como referencia para ejecutar comandos externos que conllevan a la construcción de un *target*. Estas reglas pueden ser reglas explícitas, reglas basadas en comodines, reglas imperativas o reglas vacias.

Reglas explicitas:

Las reglas explícitas son las que constructor toma cuando un *target* no se encuentra actualizado. El anterior ejemplo muestra como una regla explicita asociada al *target1.o*. En algunas ocaciones dos o más *targets* tienen los mismos prerequisitos para estos casos se pueden definir todos los *targets* en una lista y asociarlos a una sola regla:

```
target1.o target2.o:  source1.c header1.h
```

Reglas basadas en comodines *Wildcards*:

Cuando los proyectos contienen una gran cantidad de objetos por construir la cración de un *Makefile* puede tornarse en una tarea repititiva y tomar un tiempo considerable. Para evitar esto el constructor tiene la herramienta de comodines (*wildcards*), los comodines permiten agilizar la creación de un *Makefile* mediante la reducción de la cantidad de códigos que deban construir. Dada la relación estrecha entre el constructor y el *shell* los comodines allí utilizados son heredados por el constructor, los comodines más utilizados son:

. ~ : Este símbolo se usa para representar la ruta de la carpeta *home* del usuario que ejecuta el constructor.
. : El asterísco es usado para reemplazar una cadena de caracteres completa por ejemplo: Los siguientes archivos *target1.o*, *target2.o* y *target3.o* se pueden representar como **.o*
. **?** : El funcionamiento de este carácter es similar al pero en este caso no se remplaza una cadena completa sino un solo carácter.

Reglas de ejecución imperativa (.PHONY):

Generalmente las reglas incluidas en los *Makefile* son reglas de tipo explicitas, como ya se mencionó estas reglas revisan la relación entre el *target* y sus prerequisitos. Si se requiere que una regla se ejecute sin tomar en cuenta ninguna relación de los prerequisitos y el *target* se puede usar las reglas de tipo imperativa (.PHONY). Un ejemplo claro de cuando se requiere utilizar las reglas imperativas en el caso de tener el siguiente *Makefile*. Este archivo tiene la regla *clean* la cual no tiene prerequisitos y ejecuta el comando **rm -f *.o** para eleminar todos los archivos con extensión *o*. La ejecución de esta regla se puede realizar mediante **$ make clean**.

```
clean:
    rm -f *.o
```

La anterior regla se ejecuta sin ningún inconveniente cuando no existe un archivo llamado *clean* dentro de las rutas de búsqueda del cosntructor, dado que no tiene ningún pre-requisito esta regla siempre se ejecuta al ser invocada. Pero si en la ruta del constructor existe un archivo de nombre *clean* el constructor determina que el *target* de dicha regla se encuentra al día y no se debe ejecutar sus comandos de construcción terminando su ejecución con la siguiente salida:

```
\$ make clean
make: 'clean' is upt to date.
```

Para prevenir este comportamiento indeseado se puede declarar la regla de la siguiente manera:

```
.PHONY: clean
clean:
    rm -f *.o
```