Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores
Introducción a los Sistemas Embebidos
Prof. Ing. Jeferson González Gómez

Tutorial: Introducción a CMake

1. Introducción

Este tutorial presenta un enfoque introductorio al uso CMake, para la construcción de paquetes de sotware. En el desarrollo de este tutorial se describirá el funcionamiento de *CMake* tanto en modo usuario, como en modo desarrollador. Además, se aplicará *CMake* en la creación de un programa simple y una biblioteca dinámica. A continuación se presentan algunos conceptos importantes a considerar:

CMake

CMake [1] es conjunto de herramientas de código abierto de plataforma cruzada diseñado para construir, verificar y empaquetar software. Por medio de CMake se controla el proceso de compilación de software usando una plataforma simple y archivos de configuración independientes del compilador. Además, CMake permite la generación automática de Makefiles nativos y espacios de trabajo que se pueden utilizar en cualquier ambiente de compilación.

CMake utiliza una estructura de archivos similar a la de Linux para la construcción de software. En cada directorio de construcción (lib, src, etc) se requiere un archivo de entrada llamado *CMakeLists.txt*. Mediante este archivo CMake inicia el proceso de construcción de manera automática.

CMake soporta diferentes lenguajes como C, C++, Fortran, Ensamblador, entre otros.

En la siguiente sección se establecen los pasos para la configuración del ambiente y construcción de un programa simple.

2. Hello World con CMake

En esta sección se creará el ambiente y la construcción de un programa simple *Hello World*, utilizando CMake. Como punto de inicio, se parte del siguiente archivo de código fuente helloworld.c

```
#include <stdio.h>

int main(int argc, char const *argv[])
{
printf("Hello World!\n");
return 0;
}
```

Adicionalmente se creará un directorio cmake en el directorio *home* del usuario respectivo (p.e /home/juan). Dentro de cmake se creará otro directorio llamado helloworld. Dentro de helloworld se deberá copiar el código fuente *helloworld.c*.

1. Instalación CMake: Para la mayoría de distrubciones de Linux, la instalación se puede realizar con el comando

```
$ sudo apt-get install cmake
```

2. Archivo de Entrada CMakeLists.txt: Como ya se mencionó, CMake requiere un archivo CMakeLists.txt para cada directorio con códigos fuente. En este caso, el directorio de de código fuente será el mismo helloworld. De manera general, el archivo CMakeLists.txt debe contener las indicaciones adecuadas que describan el proceso de construcción (indicación de códigos fuente, ejecutables, bibliotecas, macros, etc). Para el caso de este programa simple *helloworld*, se debe utilizar el siguiente archivo CMakeLists.txt

```
#define minumum CMake version
cmake_minimum_required (VERSION 2.6)

#define the project name
project (hello)

#add executable to build environment
add_executable(hello helloworld.c)

#copy executable to bin directory
install (TARGETS hello DESTINATION bin)
```

   El archivo anterior se compone de 4 instrucciones. La primera establece la versión mínima de CMake necesaria para que el usuario pueda construir el paquete que se está desarrollando. La segunda instrucción define el nombre del proyecto como *hello*. La tercera instrucción agrega un ejecutable (binario) al proyecto; por medio del comando se liga el código fuente *(helloworld.c)* con el ejecutable recién agregado. Es por medio de esta instrucción que CMake reconoce automáticamente el lenguaje de programación y selecciona el compilador adecuado para el mismo, ubicado en la variable $PATH del sistema. El último comando creará un directorio bin, dentrol del cual copiará el binario resultante de la compilación.

3. Generación de Makefiles:Previo a realizar la construcción del programa, al igual que con Autotools, se creará un directorio llamado *build*, dentro del directorio *helloworld*.

```
$ mkdir build && cd build
```

   Adicionalmente, se definirá el directorio de instalación. Por defecto CMake asume el directorio de sistema */usr/* como base para instalación, pero para no interferir con el sistema se deberá definir otro directorio. Para esto, se debe crear un directorio *usr* dentro del directorio *build*.
   Para la generación del Makefile (dentro del directorio build) se debe ejecutar el comando

```
cmake ../ -DCMAKE_INSTALL_PREFIX:PATH=/home/USUARIO/cmake/helloworld/build/usr
```

   donde USUARIO debe reemplazar el usuario del sistema. La variable *-DCMAKE_INSTALL_PREFIX:PATH* establece el directorio de instalación para el programa.

   Al ejecutar el comando, CMake se encargará de generar automáticamente los Makefiles necesarios para la construcción del proyecto. En este caso se genera únicamente un Makefile.

4. Construcción de programa: Para la construcción del programa (modo usuario) únicamente se debe ejecutar el comando

```
$ make
```

5. Instalación: Para la instalación del programa se ejecuta

```
$ make install
```

   El comando anterior instala el binario en el directorio *usr/bin*.

6. Verificación: Para verificar el programa se requiere la ejecución del mismo.

```
$ cd usr/bin && ./hello
```

3. Biblioteca con CMake

En esta sección se configurará el ambiente y se construirá una biblioteca dinámica simple *libsayhello.so* y una aplicación que prueba la biblioteca, con el fin de analizar el funcionamiento de CMake, así como aplicar el proceso de construcción la biblioteca como tal, así como el mecanismo para enlazar aplicaciones con sus bibliotecas respectivas.

Como punto de partida se debe crear un directorio llamado *libsayhello* dentro del directorio cmake, creado en la sección anterior. Adicionalmente se deben crear los archivos fuente y cabecera tanto de la aplicación como de la biblioteca según se muestra:

****libsayhello/lib/say.c****

```
#include <say.h>
void say_hello(void){
printf("Hello World!!\n");
}
```

****libsayhello/include/say.h****

```
#include <stdio.h>

void say_hello(void);
```

****libsayhello/helloworld.c****

```
#include <say.h>

int main(int argc, char const *argv[])
{
say_hello();
return 0;
}
```

1. Archivo de Entrada CMakeLists.txt de la biblioteca (libsayhello/lib/CMakeLists.txt): CMake funciona de manera recursiva dentro de los directorios en los que se invoca. De esta manera deberá existir un archivo CMakeLists.txt en cada directorio que posea códigos fuente. Para el caso de la biblioteca, el código del archivo CMakeLists.txt del subdirectorio *lib/* es

```
#add include to directories
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/../include)

#set the proper macros
set(LIBRARY_NAME    sayhello)
set(SRC_FILES say.c)
set(INCLUDE_FILES ${CMAKE_CURRENT_SOURCE_DIR}/../include/say.h)
```

```
#add the library
add_library(${LIBRARY_NAME} SHARED ${SRC_FILES} ${INCLUDE_FILES})


#installing the library
install (TARGETS ${LIBRARY_NAME}  DESTINATION lib)
install (FILES ${INCLUDE_FILES} DESTINATION include)
```

En el archivo, el primero comando agrega el directorio que posee los archivos de cabecera para la biblioteca (*../include*) a la lista de directorios del proyecto. Los siguientes tres comandos crean macros (en mayúscula) para hacer referencia de una mejor manera a la biblioteca, código fuente y archivo de cabecera. El siguiente comando agrega la biblioteca a la lista de objetivos de construcción. Acá se establece la biblioteca como compartida/-dinámica (SHARED) y se especifica el macro del código fuente y el archivo de cabecera. Los últimos dos comandos copian la biblioteca al directorio *lib/* y el archivo de cabecera al directorio *include/*, en la instalación.

2. Archivo CMakeLists.txt principal: El archivo CMakeLists.txt principal define las generalidades del proyecto e incluye el subdirectorio *lib* para su construcción recursivamente. Adicionalemente, este archivo genera el Makefile principal que compila la aplicación que utiliza la aplicación. A continuación se presenta el contenido del Archivo CMakeLists.txt principal

```
cmake_minimum_required (VERSION 2.6)

#define the project
project (libhello)

#add lib subdirectory to build
add_subdirectory (lib)

#include directories to building path
include_directories(${PROJECT_SOURCE_DIR}/include)
include_directories(${PROJECT_SOURCE_DIR}/lib)


# add the executable
add_executable (hello helloworld.c)


#link against required libraries
target_link_libraries(hello sayhello)

#install the binary
install (TARGETS hello DESTINATION bin)

#package source generation
set(MAJOR "0")
set(MINOR "1")
set(PATCH "1")
set(CPACK_SOURCE_GENERATOR "TGZ")
set(CPACK_SOURCE_PACKAGE_FILE_NAME "${CMAKE_PROJECT_NAME}-${MAJOR}.${MINOR}.${PATCH}")
set(CPACK_SOURCE_IGNORE_FILES "/build/;${CPACK_SOURCE_IGNORE_FILES}")
include(CPack)
```

En el archivo, el comando *target_link_libraries(hello sayhello)* se encarga de realizar en enlance entre la biblioteca dinámica creada *sayhello.so* y la aplicación. De requerir otras bibliotecas, se deben agregar utilizando este comando de la forma *target_link_libraries(application lib1 lib2 lib3)*. Finalmente los últimos 6 comandos se encargan de configurar CPack, una de las herramientas de CMake para la generación del empaquetado final con el código fuente de la biblioteca y aplicación. El macro *Major, Minor y Patch* definen la versión del paquete de software. El comando *set(CPACK_SOURCE_GENERATOR "TGZ")* establece el formato de salida del paquete como *.tar.gz*. El comando *set(CPACK_SOURCE_PACKAGE_FILE_NAME...* establece el nombre del paquete con los macros descritos

anteriormente, el siguiente comando excluye el directorio *build* en el empaquetado y, finalmente, el comando *include(CPack)* permite la futura construcción del paquete de código abierto del proyecto.

3. Generación de Makefiles:Previo a realizar la construcción del programa, al igual que en la sección anterior, se creará un directorio llamado *build*, dentro del directorio *sayhello*.

```
$ mkdir build && cd build
```

Adicionalmente, se definirá el directorio de instalación, para esto, se debe crear un directorio *usr* dentro del directorio *build*.
Para generar los Makefiles del paquete (dentro del directorio build) se debe ejecutar el comando

```
cmake ../ -DCMAKE_INSTALL_PREFIX:PATH=/home/USUARIO/cmake/sayhello/build/usr
```

donde USUARIO debe reemplazar el usuario del sistema. La variable *-DCMAKE_INSTALL_PREFIX:PATH* establece el directorio de instalación para el programa.

Al ejecutar el comando, CMake se encargará de generar automáticamente los Makefiles necesarios para la construcción del proyecto. En este caso se genera únicamente un Makefile.

4. Construcción de programa: Para la construcción del programa (modo usuario) únicamente se debe ejecutar el comando

```
$ make
```

5. Instalación: Para la instalación del programa se ejecuta

```
$ make install
```

El comando anterior instala el binario en el directorio *usr/bin*.

6. Verificación: Para verificar el programa se requiere la ejecución del mismo. Antes que eso se debe exportar la dirección de la biblioteca dinámica. Esto es:

```
$ export LD_LIBRARY_PATH=/home/USUARIO/cmake/sayhello/build/usr/lib
```

Para verificar el programa, se ejecuta

```
$ cd usr/bin && ./hello
```

7. Empaquetado: Para la generación del paquete de código abierto con CMake, dentro del directorio **build** se debe ejecutar:

```
make package_source
```

El comando anterior genera el empaquetado para el código fuente de la biblioteca y la aplicación. En este caso es: *libsayhello-0.1.1.tar.gz*

4. Evaluación

4.1. Descripción

Debe crear una biblioteca, en lenguaje C, la cuál ofrecerá cinco funciones matemáticas: suma, resta, multiplicación, división y raíz cuadrada. Para este caso, debe utilizar CMake tanto para la generación de las biblioteca, como de las aplicaciones que las verifican. La estructura de la solución de este Ejercicio es la siguiente:

- Archivo empaquetado de la biblioteca creada con CMake (ejemplo *libhello-1.1.0.tar.gz*), siguiendo el formato presentado en el tutorial. La estructura de archivos deberá ser la estandar (bin, lib, include, etc).

4.2. Entregable

(Subir al tecDigital)

- Único archivo .tar (incluya su nombre como parte nombre del archivo) con el archivo empaquetado de la biblioteca descrito anteriormente.