Linux kernel and C programming

BLOCK 2: CMake and Pointers

Gergely Korcsák

Óbuda University
korcsak.gergely@nik.uni-obuda.hu

March 10, 2025

Presentation Overview

CMake

CMake variables

CMake variables also have type and scope. We can define them from the command line, and also in the CMake files.

Command line example

```
$ cmake -S . -B __build \
    -DMY_STRING:STRING="hello"
```

Or they can be defined in a CMake files.

CMakeLists.txt

```
set(<variable> <value>
    [[CACHE <type> <docstring> [FORCE]] | PARENT_SCOPE])
```

CMake variable types and scopes

Types

Type	Description
FILEPATH	Exact file path
PATH	Directory path
STRING	Arbitrary string
BOOL	Boolean ON/OFF
INTERNAL	Persistent

CMake variable types and scopes

Scopes

By default every CMake variable is accessible from their local file, and in every other included file, from the same CMake file. We can chage the scopes by eahter setting it as CACHE-ed, or to PARENT_SCOPE

Scope	Description
CACHE	Make the variable accessible globally, after its file was included
PARENT_SCOPE	Make the variable accessible to the file, which it is included from

```
1 #// <variable>  <value>       <type> <description>
2 set(MY_STRING   "hello" CACHE STRING "Hello string")
3
4 #// <variable>  <value> <scope>
5 set(MY_STRING   "hello" PARENT_SCOPE)
```

CMake conditionals

```
1 if(<condition>)
2   <commands>
3 elseif(<condition>) # optional block, can be repeated
4   <commands>
5 else()              # optional block
6   <commands>
7 endif()
```

CMake conditionals

```
1  if(NOT <condition>)
2  if(<cond1> AND <cond2>)
3  if(<cond1> OR <cond2>)
4
5  if((condition) AND (condition OR (condition)))
6
7  if(DEFINED <name>|CACHE{<name>}|ENV{<name>})
8  if(EXISTS <path-to-file-or-directory>)
9
10 if(<variable|string> MATCHES <regex>)
11 if(<variable|string> EQUAL <variable|string>)   # double
12 if(<variable|string> STREQUAL <variable|string>)
13
14 if ("/a//b/c" PATH_EQUAL "/a/b/c")
```

CMake include

We can eather add a subdirectory containing a `CMakeLists.txt` file, or include a .cmake file like the following:

```
.
|-- CMakeLists.txt <this file>
|-- my_config.cmake
`-- terminal
    |-- terminal.h
    |-- terminal.c
    `-- CMakeLists.txt

include(my_config.cmake)

add_subdirectory(terminal)
```

Extendxed C

Unions

Unions allow to store different data types in the same memory location.

```
1  struct http_request {
2      uint32_t version;
3      uint32_t protocol;
4      uint32_t sender;
5      uint32_t randomdata;
6      uint8_t  data[0x1000];
7  };
8
9  union http {
10     struct http_request header;
11     uint8_t payload[0x2000];
12 };
```

Unions

```
1  int handle_payload(...) {
2      union http req;
3
4      req.header.version    = 1;
5      req.header.protocol   = 0x34;
6      req.header.sender     = IP_TO_INT32(192,168,1,12);
7      req.header.randomdata = random();
8
9      sprintf(req.header.data,
10             "Hello from http payload...!!");
11
12     eht_phy->send(req.payload,
13                   4 * sizeof(uint32_t)
14                   + strlen(req.header.data));
15 }
```

Function pointers

Where the fun begins...

```
1 // <return type> (*<name>)   (args...);
2    int             (*sum)     (int, int);
```

Function pointers

We can define function pointers as follows:

```
1  // First lets use the function
2  int calculate_sum(int a, int b) { return a + b; }
3
4  // And create a pointer
5  int (*sum)(int a, int b);
6
7  // Assigne it
8  sum = calculate_sum;
9
10 // Call it
11 int sum_of_ab = sum(12, 13);
```

Function pointers in structs

```
1 struct command {
2     char* name;
3     void (*cmd)(int argc, char** argv);
4 };
```

Argc and argv

Command line arguments will follown the program separated by a space character.

```
$ terminal help
```

In this example the terminal is the program, its arguments will be [terminal, help]. The first argument will always be the name of the application called.

```
int main(int argc, char** argv) {
    int i;

    for (i = 0; i < argc; i++) {
        printf("arg[%i]: %s", i, argv[i]);
    }

    return 0;
}
```

Excercises

Read input arguments

Lets try out the following:

```
1 int main(int argc, char** argv) {
2     int i;
3
4     for (i = 0; i < argc; i++) {
5         printf("arg[%i]: %s", i, argv[i]);
6     }
7
8     return 0;
9 }
```

Write a simple console application

```
.
|-- CMakeLists.txt
|-- config.cmake
`-- userspace
    |-- functions
    |   |-- help.h
    |   |-- help.c
    |   |-- echo.h
    |   `-- echo.c
    |-- terminal.h
    |-- terminal.c
    `-- CMakeLists.txt
```

The End

Questions? Comments?