



Exercise sheet 5 – Processor architecture

Goals:

- Synchronisation commands
- Endianness

Exercise 5.1: Endianness (theoretical)

- (a) The given struct `meier` is transmitted serially (byte-by-byte) from a *little-endian* to a *big-endian* architecture. Assume both are 32-bit architectures.

```
1 struct employee {
2     char name[12];    //ASCII/UTF-8 (1 byte)
3     char birthday[12]; //ASCII/UTF-8 (1 byte)
4     int32_t id;
5 };
6
7 struct employee meier = {
8     .name      = "Josef Meier",
9     .birthday  = "01.02.1957",
10    .id         = 0x123456
11 };
```

Provide a solution similar to the scheme in the lecture. Which corrections may be necessary?

Proposal for solution:



Exercise 5.2: Endianness with integer (coding)

Given is a *big-endian* system program—the Java runtime environment—that transfers data via a file to a *little-endian* system C program. Investigate the behaviour: You may have to fix something for a correct transfer.

- Update the CA_exercises repository with `git pull`.
- Change into the directory CA_exercises/sheet_05_endianness/Endianness/Java_BE

Proposal for solution: `cd CA_exercises/sheet_05_endianness/Endianness/Java_BE`

- Inspect, build, and run the given Java program.

Proposal for solution:

```
1 make #build
2 java java_be_example #execute
```

- Inspect and interpret the result file `output.txt`. Hint: Use a HEX viewer, for example: `xxd`.


```
1 xxd -c 1 output.txt #to show the byte per byte view (memory/file view) \\
2 xxd -c 4 output.txt #to show the word view (register view in 32 bit/4 byte mode)
```
- Change into the directory CA_exercises/sheet_05_endianness/Endianness/C_LE

Proposal for solution: `cd CA_exercises/sheet_05_endianness/Endianness/C_LE`

- Inspect, build, and run the given C program.

Proposal for solution:

```
1 make                #build
2 ./c_le_example      #execute
```

- (g) Analyse the output of the C program. What happened? What could be the cause of this?

Proposal for solution: The C program is reading the content of *output.txt*, which was generated by the java-program. Because of the different endianness of Java (big endian) and C (little endian), the output of the C program is switched.

- (h) Fix the problem in the C program, following the *TODOs*.

Proposal for solution:

```
1 #include <stdio.h> //fopen, ...
2 #include <stdlib.h> //EXIT_SUCCESS
3 #include <stdint.h> //uint8_t, uint16_t
4
5 uint16_t swap_bytes(uint16_t value_to_swap); //prototype
6
7 int main(void) {
8
9     uint16_t value = 0;
10
11     FILE* file = NULL;
12     file = fopen("../output.txt", "rb");
13
14     if (file == NULL) {
15         printf("Error opening output.txt\n");
16         return EXIT_FAILURE;
17     }
18
19     fread(&value, sizeof(uint16_t), 1, file);
20     fclose(file);
21
22     printf("Read from output.txt -> : %2x\n", value);
23
24     //fix the byte order by calling the swap_bytes() function
25     value = swap_bytes(value);
26
27     //print the fixed value
28     printf("Converted to LE -> : %2x\n", value);
29
30     return EXIT_SUCCESS;
31 }
32
33
34 /*
35 We are using a union to access the uint16_t value via the byte-array (uint8_t[]).
36 This is working because you can only store one value at a
37 given time in a union
38 (a union is only as big as its biggest member)
39 and all members of that union access the same piece of memory,
40 but with their respective data-types.
41 */
42 uint16_t swap_bytes(uint16_t value_to_swap) {
43     union {
44         uint16_t value;
```

```
45     uint8_t byte[2];
46 } SWAP_TYPE;
47
48 SWAP_TYPE.value = value_to_swap;
49
50 uint8_t tmp = SWAP_TYPE.byte[0];
51 SWAP_TYPE.byte[0] = SWAP_TYPE.byte[1];
52 SWAP_TYPE.byte[1] = tmp;
53
54 //alternatively you can do bit shift operations
55 //SWAP_TYPE.value = ((0x00FF & SWAP_TYPE.value) << 8)
56 //                  | ((0xFF00 & SWAP_TYPE.value) >> 8);
57
58 return SWAP_TYPE.value;
59 }
```

- (i) Build and run the C program again. Is the problem now solved?

Proposal for solution:

```
1 make                #build
2 ./c_le_example      #execute
```

Exercise 5.3: Endianness with an UTF16 character (coding)

Given is a *big-endian* UTF16 encoded character saved in a file. A C/C++ program (*little-endian*) wants to read and print the encoded character to the terminal.

- (a) Change into the directory

CA_exercises/sheet_05_endianness/Endianness_UTF16/read_UTF16_character

Proposal for solution: `cd CA_exercises/sheet_05_endianness/Endianness_UTF16/read_UTF16_c`

- (b) Inspect the UTF16 character within the given file

CA_exercises/sheet_05_endianness/Endianness_UTF16/utf16_character_be.txt.

Hint: Use a HEX viewer, for example: xxd.

```
1 xxd -c 1 utf16_character_be.txt #to show the byte per byte view (memory/file view) \\
2 xxd -c 4 utf16_character_be.txt #to show the word view (register view in 32 bit/4 byte
```

- (c) Inspect, build, and run the given C/C++ program.

Proposal for solution:

```
1 make                #build
2 ./read_utf16_character #execute
```

- (d) Analyse the output of the C/C++ program. What happened? What could be the cause of this?

Proposal for solution: The C/C++ program is reading the content of *utf16_character_be.txt*, which contains an UTF16 character with *big-endian* encoding. That causes the output to be wrong.

- (e) Fix the problem in the C/C++ program, following the *TODOs*.

Proposal for solution:

```
1 #include <stdlib.h> //EXIT_SUCCESS
2 #include <string>    //std::string, char16t
```



```
3 #include <iostream> //std::cout
4
5 #include <locale>    // wstring_convert
6 #include <codecvt>   // codecvt_utf8
7
8 //prototypes
9 void print_utf16_string(const std::u16string& message,
10                        std::u16string utf16_character);
11 char16_t swap(const char16_t character);
12
13 int main(void) {
14     char16_t message[] = u"A nuclear power plant is ";
15     char16_t utf16_character = u' ';
16
17     //read utf16 character from file
18     FILE* file = NULL;
19     file = fopen("../utf16_character_be.txt", "rb");
20
21     if (file == NULL) {
22         std::cerr << "Error opening: " << "utf16_character_be.txt" << std::endl;
23         return EXIT_FAILURE;
24     }
25
26     fread(&utf16_character, sizeof(uint16_t), 1, file);
27     fclose(file);
28
29     //fix the byte order by calling the swap() function
30     utf16_character = swap(utf16_character);
31
32     //print the message with the utf16 character
33     print_utf16_string(message, std::u16string{utf16_character});
34
35     return EXIT_SUCCESS;
36 }
37
38 void print_utf16_string(const std::u16string& message,
39                        std::u16string utf16_character) {
40     std::wstring_convert<std::codecvt_utf8<char16_t>, char16_t> convert;
41     std::cout << convert.to_bytes(message)
42               << convert.to_bytes(utf16_character) << std::endl;
43 }
44
45 char16_t swap(const char16_t character) {
46     union {
47         char16_t c;
48         char bytes[2];
49     } SWAP_TYPE;
50
51     SWAP_TYPE.c = character;
52
53     char tmp = SWAP_TYPE.bytes[0];
54     SWAP_TYPE.bytes[0] = SWAP_TYPE.bytes[1];
55     SWAP_TYPE.bytes[1] = tmp;
56
57     return SWAP_TYPE.c;
58 }
```

(f) Build and run the C program again. Is the problem now solved?



Proposal for solution:

```
1 make #build
2 ./read_utf16_character #execute
```