Handout

Series S04:
- Memory Model II
- Functions
- Macros & C Preprocessing
- Make

Reading

a) Study the tutorial “Memory Model II: Process and Thread". Note that no exercise is associated with this tutorial in this series, but we will need the explained concepts and model later; and it can also contribute to better understand the scope and lifetime of C variables and functions, cf. Exercise 1.

b) Study the lecture notes and browse/read [KR 88], chap. 4.

c) Study the tutorial "AST and CS – section: function calls" (visit 3) [TS02]

d) Study the tutorial “Macros and C Preprocessor : a first contact” [TC04]

e) Study the tutorial "Make: a first contact" [TU04]

1. Scope and Lifetime of C variables

Study the programs 'Scope&Lifetime' available on Moodle [C01] (/extras/c/Scope&Lifetime_c). Then, do the exercise described in the file 'exercises.txt'.

2. Flow of Function Calls + AST + gdb (visit 3)

Let be the program of Example 1 of the gdb Tutorial.

a) Draw the flow for the function calls `main()` and `swap1(i,j)` and the associated AST, including the simplified control stack just before `swap1` returns to its caller function `main` – ignore the function call `swap2(&i, &j)`.

b) Use gdb to observe the values of `swap1()`’s arguments, and also the local variables `i` and `j` in `main()`:

   1) just after `swap1()`'s frame has been built,

   2) just before `swap1()` is left.

   After that, explain what you observed.

Hint: you can use the following gdb commands:

```
(gdb) bt                 // print backtrace of stack frames for all
                         // active subroutines
(gdb) frame <frame #>    // select and print a stack frame
(gdb) info args          // print arg. variables of current stack frame
(gdb) info locals        // print local variables of current stack frame
```

For (a) and (b), check specifically if the values of i, j were swapped in main() and report your result.

3. Macros and C Preprocessor : a first contact

Do the exercises 1-3 of the tutorial "Macros and C Preprocessor: a first contact" [TC04].

4. Make: a first contact

Answer the questions 1-4 of the tutorial "Make: a first contact" [TU04], and optionally questions 5-6.

5. Git & GitLab

Create a new directory with the *wcount.c* example from Series 02.

Write a *Makefile* with the following functions:

1. Compiles the *wcount.c* source file to a *wcount* binary.
2. Runs *wcount* with counting the *wcount.c* source file.

After everything compiles and runs accordingly:

1. Initialize a GIT repository in the directory with the above files:

   ```
   git init
   ```

2. Add the *Makefile* and the *wcount.c* source. (Do not include the binary!)

   ```
   git add Makefile wcount.c
   git commit -m "Initial commit"
   ```

3. Create on GitLab (https://*diuf-gitlab.unifr.ch*) a Project called “SOP_S04”.
4. Connect the local GIT repository to the new Project:

   ```
   git remote add origin https://diuf-gitlab.unifr.ch/<user>/SOP_S04.git
   ```

5. Upload your files to GitLab.

   ```
   git push -u origin master
   ```

Finally share your GitLab Project with the Group “SOP_supervisor” with *Reporter* access level:
→ Settings → Members → Share with group

Hand in.

Upload your answers on Moodle.

References

[KR88] B. Kernighan, D. Ritchie, The C Programming Language, 2nd Ed., Prentice Hall, 1988.

[C01] Moodle > Tutorial “C01_KR88”

[TC04] Moodle > Tutorial “TC04 Macros and C Preprocessor”

[TU04] Moodle > Tutorial “TU04 Make”