CS2850 Operating System Lab

Week 1: Introduction

nicolo colombo

nicolo.colombo@rhul.ac.uk

Office Bedford 2-21

Outline

References

Brian W. Kernighan, Dennis Ritchie: The C Programming Langauge Prentice-Hall 1978 ISBN 0-13-110163-3

Randal Bryant, David O'Hallaron: Computer Systems: A Programmer's Perspective C Pearson Education Limited, 3rd edition, 2016 ISBN-13: 9781292101767.

The GNU C Library Reference Manual

The Operating System (OS)

The OS is a layer of software that

- provides a better, simpler, cleaner, model of the computer and
- helps the user handle resources: processors, disks, printers, keyboard, display, ...

Two popular OS are UNIX and Windows.

Why C-programming?

C is a general-purpose programming language. You can write almost anything in C.

C is not tied to any OS. Your programs will work on any machine.

UNIX is largely written in C.

A relatively low-level language

C does not include

- x operators acting on composite objects, e.g. strings of characters, array,s or lists,
- x Dynamical memory allocation facilities,
- x READ or WRITE statements (you need to *call* dedicated functions),

C is ‘easy’

“... keeping the language down to modest size has real benefits. Since C is relatively small, it can be described in a small place and learned quickly. A programmer can reasonably expect to know and understand and indeed regularly use the entire language” [1]

[1] from Brian W. Kernighan, Dennis Ritchie: The C Programming Langauge

ANSI C

C is machine-independent.

The program below works on computers with different OS.

```
#include <stdio.h>
int main() {
  printf("hello, world\n");
}
```

To run it, you need the *system-dependent* executable, `a.out`,

```
00000000: 01111111 01000101 01001100 01000110 00000010 00000001   .ELF..
00000006: 00000001 00000000 00000000 00000000 00000000 00000000   ......
0000000c: 00000000 00000000 00000000 00000000 00000011 00000000   ......
00000012: 00111110 00000000 00000001 00000000 00000000 00000000   >.....
....
```

Compilation under UNIX

The OS produces `a.out` from the provided C code[2].

To compile `hello.c`, use the shell command

```
gcc -Wall -Werror -Wpedantic hello.c
```

A useful sanity check of your program is run by entering[3]

```
valgrind ./a.out
```

[2]Use the additional option `-o yourExecutable` to change the executable name.

[3]`valgrind` is a powerful debugging tool for Linux programs.

The Standard Library

ANSI C is based on a *established library* of functions.

You need standard library functions to

- read or write files,
- allocate memory,
- handle strings,
- ....

They are mostly written in C and may contain *a few* non-portable OS details, e.g. system call syntax.

Headers

Write `# include <...>` to make a piece of the library accessible to your program.

`<stdio.h>`: input and output.

`<stdlib.h>`: memory allocation, process control.

`<unistd.h>`: system calls.

`<string.h>`: string-handling.

`<errno.h>`: error reporting.

`<math.h>`: common mathematical functions

Control flow

The control flow fixes the order in which instructions are executed

The most used control-flow statements are

- sequential instructions: “;” (default line-by-line execution)
- grouping symbols: { `...` }
- selection commands: `if-else`, `switch`, ...
- repetition tools: `for`, `while`, ...

Example

A C program that prints “`hello, world`” several times

```
#include <stdio.h>
#define N 5
int main() {
   int i;
   for (i = 0; i < N; i++) {
      printf("%d) hello, world\n", i + 1);
    }
}
```

The output is

```
1) hello, world
2) hello, world
3) hello, world
4) hello, world
5) hello, world
```

Notes

All variables need to be declared before using them.

`printf` [4] can print

- simple strings: `printf("hello, world\n");`
- variable values: `printf("%d \n", i);` where `%d` specifies that `i` should be printed as an `int`
- a mix of string and values:
  `printf("%d) hello, world \n", i);`

Add `/* .... */` and `// ...` to comment out multiple or single lines.

[4] Defined in `stdio.h`