CS2850 operating system lab
week 1: introduction

nicolo colombo

nicolo.colombo@rhul.ac.uk

Office Bedford 2-21

outline

references

system programming

key features of C

compilers

standard library

control flow: `if`, `for`, `while` ...

examples

references

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

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

The GNU C Library Reference Manual

system programming

computing devices are equipped with a layer of software called the operating system

the operating system:

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

two popular operating systems are UNIX and Windows

the main features of an operating system are the subject of the CS2850 main lectures

why C-programming?

C is a general-purpose programming language

C is not tied to any operating system

C is a system programming language because it is useful for writing operating systems or compilers

UNIX is largely written in C

...

C is a relatively low-level language ...

C includes:

- V operators that deal with 'simple' objects: characters, addresses and numbers
- V single-thread control-flow statements: if, for, while, ...

C does not include:

- X operators acting on composite objects such as a string of characters, an array, or a lists
- X dynamical memory allocation facility
- X READ or WRITE statements
- X built-in file access method

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" *

*from Brian W. Kernighan, Dennis Ritchie: The C Programming Langauge

ANSI C

ANSI C is a standard version of C

a set of rules have been fixed for C-programs to be machine-independent and compatible

the same C code, e.g. `hello.c`, can work on computers with different operating systems

what changes is how `hello.c` is run, i.e. the corresponding executable (binary) program

the binary file, e.g. `a.out` obtained from `hello.c`, is produced by a *system-dependent* compiler

compilation under UNIX

compiling involves I/O operations and is performed by the OS

a standard way to communicate with the UNIX operating system is to use the *UNIX command interpreter* or shell

the shell is not part of the OS (but makes heavy use of many OS features)

the shell is a process that is started when a user logs in

the terminal (a text-based interface) is used as standard input and standard output

compilation process

example: how does the shell compile the C code `hello.c` into the executable?

- the user enters a command-line instruction, e.g.

```
compilerName -flag1 argument1 -flag2 argument2 ...  hello.c
```

- the shell reads the command from the terminal
- the shell creates a child process that runs the compiler
- when the compiler has finished the child process executes a system call to terminate itself

the compiler used in this course

`gcc` and `clang` are two popular UNIX C-compilers

for this course, we suggest you use `gcc` with the *"didactic"* flags `-Wall`, `-Werror`, and `-Wpedantic`, i.e. compile your programs using

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

if you do not use the additional option `-o yourExecutable` the exectuable of `yourProgram.c` will be `a.out`

execution under UNIX

the shell can also be used to execute the binary file **a.out** by typing

```
./a.out [args]
```

where **args** are possible *command-line arguments* of your program

to run a sanity check of your program, you should also try

```
valgrind ./a.out [args]
```

valgrind is a free system with a series of powerful debugging tools for Linux programs

standard library

ANSI C comes with a standard library of 'basic' functions for

- performing system calls (e.g. reading or writing files)
- handling input and output
- memory allocation
- manipulating strings

C codes always start with a series of headers that make the corresponding libraries accessible to the program

most functions in the library are written in C

except for some operating system details, e.g. system call syntax, the library is portable

headers

A few important parts of the C standard library are

- `<stdio.h>`: input and output
- `<stdlib.h>`: pseudo-random generation, memory allocation, process control
- `<unistd.h>`: access to the POSIX operating system
- `<string.h>`: string-handling
- `<errno.h>`: error reporting *
- `<math.h>`: common mathematical functions
- `<threads.h>`: managing multiple threads, mutexes, condition variables

*C functions usually report errors through numbers and the macros in `errno.h` convert them to text messages

control flow

the program's control flow is the order in which instructions are executed

C has few types of control-flow statements

- sequential: "`;`" (default line-by-line execution)
- grouping: { ... }
- selection: `if-else`, `switch`, ...
- repetition: `for`, `while`, ...

C statements mainly work as in other programming languages

example (1)

a C program that writes "`hello, world`" on the terminal

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

example (2)

a C program that produces exactly the same output

```
#include <stdio.h>
int main() {
  printf("hello, world\n");
  return 0;
}
```

example (3)

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++) {//for takes 3 arguments
    /*the following line prints ``hello, world'' once*/
    printf("%d) hello, world\n", i + 1);
    }
}
```

notes (1)

in C, all variables need to be declared before using it as in

```
int i;
```

the formatted-output function `printf` is defined in `stdio.h` and you can use it for printing

- 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);`

note (2)

macro definitions like

```
#define N 5
```

allow you to define shortcuts that are valid over the entire program

remember to include a new-line character `\n` at the end of the output string to avoid strange behaviours

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

exercise

copy, compile, and run the program in the last example to check that it writes

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

verify that the program executes without problems by reading carefully the output of

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