Introduction to Systems Programming

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

September 16, 2024

General Information

- On iCorsi: ***INF.B.SA 2024-2025.390***
  also on ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***

General Information

- On iCorsi: ***INF.B.SA 2024-2025.390***
  also on ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***

- Announcements: through iCorsi

You are responsible for reading the course announcements

General Information

- On iCorsi: ***INF.B.SA 2024-2025.390***
  also on ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***

- Announcements: through iCorsi

You are responsible for reading the course announcements

- We are here to help you! (by appointment)
  - Antonio Carzaniga
  - Fabio Di Lauro
  - Pasquale Polverino
  - Christian Lämmle
  - Illia Zeller

Goals

- *Systems* programming in C, plus a bit of C++ (more on this later)

Goals

- *Systems* programming in C, plus a bit of C++ (more on this later)
- Concrete, practical programming skills
  - with basic, good software engineering practices

I want you to be able to ***write well-structured, correct programs (in C/C++)*** that satisfy non-trivial requirements

Goals

- *Systems* programming in C, plus a bit of C++ (more on this later)
- Concrete, practical programming skills
  - with basic, good software engineering practices

I want you to be able to ***write well-structured, correct programs (in C/C++)*** that satisfy non-trivial requirements

- Deep, clear understanding of program behavior
  - execution model
  - memory model

I want you to be able to ***read and thoroughly understand*** programs (in C/C++)

reading + *lectures* + *in-class exercises* + ***homework***

reading + ***lectures*** + ***in-class exercises*** + ***homework***

- *Lectures*
  - *preliminary reading* plus *interactive lectures*
  - in-class exercises
  - so, you should have your computer handy (and charged)

Course Structure

reading* + *lectures* + *in-class exercises* + *homework

- *Lectures*
  - *preliminary reading* plus *interactive lectures*
  - in-class exercises
  - so, you should have your computer handy (and charged)
- *Homework*
  - a programming assignment *every week*
  - a few assignments may be graded (we'll tell you which ones)
  - all assignments will be discussed in class

Evaluation and Other Policies

https://www.inf.usi.ch/carzaniga/edu/sysprog/policies.html

Evaluation

Evaluation

- +10% homework: programming assignments
  - grades added together, thus resulting in a weighted average
- +40% midterm exam
  - in-class programming using your computer
- +50% final exam
  - in-class programming using your computer
- ±10% instructor's discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - …

Evaluation

- +10% homework: programming assignments
  - grades added together, thus resulting in a weighted average
- +40% midterm exam
  - in-class programming using your computer
- +50% final exam
  - in-class programming using your computer
- ±10% instructor's discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - …
- −100% plagiarism penalties

Plagiarism

Plagiarism

A student must never take material, from any source, and present it as his or her own. Doing so means committing plagiarism.

Plagiarism

suggestions, corrections,
ideas, code, text, …

A student must never take material, from any source, and present it as his or her own.
Doing so means committing plagiarism.

Plagiarism

suggestions, corrections, ideas, code, text, …

people, books, course notes, stackoverflow, CheatGPT, …

A student must never take material, from any source, and present it as his or her own. Doing so means committing plagiarism.

Plagiarism

suggestions, corrections, ideas, code, text, ...

people, books, course notes, stackoverflow, CheatGPT, ...

A student must never take material, from any source, and present it as his or her own. Doing so means committing plagiarism.

- **Penalties:** committing plagiarism on an assignment or an exam will result in
  - failing that assignment or that exam
  - loosing one or more points *in the final note!*
- Penalties may be escalated...

Deadlines

Deadlines

Deadlines are firm.

Deadlines

Deadlines are firm.

- You know what I mean…
- Usual three-days-and-you're-out rule applies here…

Now on to ***Systems Programming!***

What is Systems Programming?

What is Systems Programming?

- Interfacing with a “system” (as opposed to a user)
  - rigid interfaces
  - complex interfaces

What is Systems Programming?

- Interfacing with a “system” (as opposed to a user)
  - rigid interfaces
  - complex interfaces
- Engineering for a non trivial platform
  - non-trivial performance profiles
  - going beyond algorithmic complexity

The Language(s) of Systems Programming

The Language(s) of Systems Programming

- Mostly C, and a bit of C++

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers
- System programming
  - “low-level” programming (e.g., a device driver)
  - “high-level” programming (e.g., the Firefox web browser)

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers
- System programming
  - "low-level" programming (e.g., a device driver)
  - "high-level" programming (e.g., the Firefox web browser)
- Relatively simple (C) but still powerful language
  - C++ is definitely not that simple
  - like any serious tool, C and C++ have hidden complexities…

How to Learn Systems Programming

How to Learn Systems Programming

reading + ***lectures*** + ***in-class exercises*** + ***homework***

How to Learn Systems Programming

reading + ***lectures*** + ***in-class exercises*** + ***homework***

- Read the lecture notes *before the relevant lectures!*

How to Learn Systems Programming

reading + ***lectures*** + ***in-class exercises*** + ***homework***

- Read the lecture notes *before the relevant lectures!*
- The notes contain many concrete examples and exercises: *try them as you read!*

How to Learn Systems Programming

reading + ***lectures*** + ***in-class exercises*** + ***homework***

- Read the lecture notes *before the relevant lectures!*
- The notes contain many concrete examples and exercises: *try them as you read!*
- ***Practice, Practice, Practice!***
  - solve as many exercises as you can
  - at least one exercise per week
  - ***regularly***
  - ***from the beginning!***

How to Practice with Systems Programming

How to Practice with Systems Programming

No magic pill, no silver bullet…

You must put in time and mental energy!

How to Practice with Systems Programming

No magic pill, no silver bullet…

You must put in time and mental energy!

1. Solve a programming problem
2. If you are stuck, ask somebody to help you out
   (your teacher is always happy to help you!)
   …But *do not simply copy code!*
3. When you're done—when your own solution is *complete*—analyze other solutions, such as the solutions presented in class
4. Go to step 1

Getting Started: One, Two, Three!

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
    print("Ciao!\n");
}
```

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
    print("Ciao!\n");
}
```

2. Compile the program (i.e., run the compiler)

```
$ cc ciao.c -o ciao
```

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
    print("Ciao!\n");
}
```

2. Compile the program (i.e., run the compiler)

```
$ cc ciao.c -o ciao
```

3. Run the program

```
$ ./ciao
```

Getting Started with C++

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

2. Compile the program (i.e., run the compiler)

```
$ c++ ciao2.cc -o ciao2
```

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

2. Compile the program (i.e., run the compiler)

```
$ c++ ciao2.cc -o ciao2
```

3. Run the program

```
$ ./ciao2
```

Getting Started with *Make*

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
    std::cout << "I said Ciao already!\n";
}
```

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
    std::cout << "I said Ciao already!\n";
}
```

2. Compile the program using *make*

```
$ make ciao3
```

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
    std::cout << "I said Ciao already!\n";
}
```

2. Compile the program using *make*

```
$ make ciao3
```

3. Run the program

```
$ ./ciao3
```

Errors

Try compiling the program:

```
#include <iostream>

int main() {
    cout << "I said Ciao already!\n";
}
```

Try compiling the program:

```
#include <iostream>

int main() {
    cout << "I said Ciao already!\n";
}
```

You should get some errors:

```
$ g++ errors.cc -o errors
errors.cc: In function ‘int main()’:
errors.cc:4:5: error: ‘cout’ was not declared in this scope
...
```

The function you will use to print data in C is `printf`:

```c
#include <stdio.h>

int main() {
    printf("My name is %s.\nI have been programming in C for %d years.\n",
           "Antonio", 2024 - 1989);
}
```

Printing

The function you will use to print data in C is `printf`:

```
#include <stdio.h>

int main() {
    printf("My name is %s.\nI have been programming in C for %d years.\n",
           "Antonio", 2024 - 1989);
}
```

The first argument is a ***format string*** that includes ***conversion specifications***, begining with a % sign, that tell printf how to interpret its other arguments:

%d prints an integer in decimal notation
%c prints an integer as a character
%g prints a `float` in decimal notation
... *see the documentation of printf()*

Printing in C++

Printing is quite different (simpler?) in C++:

```
#include <iostream>

int main() {
    std::cout << "My name is " << "Antonio.\n"
              << "I have been programming in C for "
              << 2000 - 1969
              << " years.\n";
}
```

Digression: How does this really work?

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the "standard input"
  - returns an `int` value
  - returns `EOF` at the end of file

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the "standard input"
  - returns an `int` value
  - returns `EOF` at the end of file

- **Exercise:** write a program that counts and prints the number of characters (bytes) in its standard input stream

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the "standard input"
  - returns an `int` value
  - returns `EOF` at the end of file

- **Exercise:** write a program that counts and prints the number of characters (bytes) in its standard input stream

```
#include <stdio.h>

int main() {
    int i = 0;
    while (getchar() != EOF)
        ++i;
    printf ("%d characters\n", i);
    return 0;
}
```

Examples...

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the “standard output”

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the "standard output"

- **Exercise:** what does this program do?

```
#include <stdio.h>
#include <limits.h>

int main() {
    int c;
    while ((c = getchar()) != EOF) {
        c += 3;
        if (c > UCHAR_MAX)
            c = UCHAR_MIN + (c - UCHAR_MAX);
        putchar(c);
    }
}
```

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the "standard output"

- **Exercise:** what does this program do?

```
#include <stdio.h>
#include <limits.h>

int main() {
    int c;
    while ((c = getchar()) != EOF) {
        c += 3;
        if (c > UCHAR_MAX)
            c = UCHAR_MIN + (c - UCHAR_MAX);
        putchar(c);
    }
}
```

- **Exercise:** write a program that inverts the transformation of the program above

Run this program:

```
#include <stdio.h>

int main () {
    putchar(67);
    putchar(105);
    putchar(97);
    putchar(111);
    putchar(33);
    putchar(10);
}
```

Now run this other program:

```
#include <stdio.h>

int main () {
    putchar(240);
    putchar(159);
    putchar(153);
    putchar(130);
    putchar(10);
}
```

Homework Assignment: wordcount

- Write a program called *wordcount* that counts the words in the standard input. A *word* is a sequence of one or more characters delimited by white space.
  - the output should be the same as the command:

```
$ wc -w
```