

C - compiling code

Johan Montelius

KTH

VT26

The big difference

- Java: source code is compiled and then executed by a program - the JRE.
- C: source code is compiled and then executed as *native code* on the computer.

WTF?

The big difference

- Java: source code is compiled and then executed by a program - the JRE.
- C: source code is compiled and then executed as *native code* on the computer.

WTF?

The big difference

- Java: source code is compiled and then executed by a program - the JRE.
- C: source code is compiled and then executed as *native code* on the computer.

WTF?

The big difference

- Java: source code is compiled and then executed by a program - the JRE.
- C: source code is compiled and then executed as *native code* on the computer.

WTF?

The big difference

- Java: source code is compiled and then executed by a program - the JRE.
- C: source code is compiled and then executed as *native code* on the computer.

WTF?

The big picture

hardware - CPU, memory, ...

The big picture

operating system (OS)

hardware - CPU, memory, ...

The big picture

program executing

operating system (OS)

hardware - CPU, memory, ...

The big picture

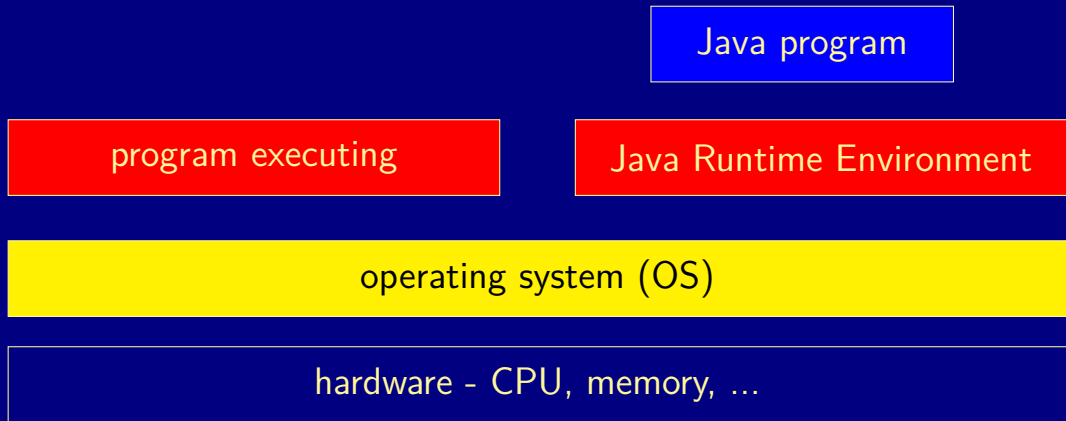
program executing

Java Runtime Environment

operating system (OS)

hardware - CPU, memory, ...

The big picture



The steps of running a program

- The compiler (gcc/clang/...) takes source code and creates an *executable* file.
- The operating system creates a process and allocates memory for:
 - the code
 - global data structures,
 - a heap
 - and a stack.
- The operating system *loads* code and *links* it to system calls and shared libraries (dynamic linking)
- The operating system schedules the process for execution.

The steps of running a program

- The compiler (gcc/clang/...) takes source code and creates an *executable* file.
- The operating system creates a process and allocates memory for:
 - the code
 - global data structures,
 - a heap
 - and a stack.
- The operating system *loads* code and *links* it to system calls and shared libraries (dynamic linking)
- The operating system schedules the process for execution.

The steps of running a program

- The compiler (gcc/clang/...) takes source code and creates an *executable* file.
- The operating system creates a process and allocates memory for:
 - the code
 - global data structures,
 - a heap
 - and a stack.
- The operating system *loads* code and *links* it to system calls and shared libraries (dynamic linking)
- The operating system schedules the process for execution.

The steps of running a program

- The compiler (gcc/clang/...) takes source code and creates an *executable* file.
- The operating system creates a process and allocates memory for:
 - the code
 - global data structures,
 - a heap
 - and a stack.
- The operating system *loads* code and *links* it to system calls and shared libraries (dynamic linking)
- The operating system schedules the process for execution.

compiling a C program

- Preprocessing, expand all *macro directives* (`#include`, `#define`, ...).
- Compile code to assembly for the target architecture.
- Link calls of procedures to the code for the procedures.
- Create an executable file with directives to the operating system.

compiling a C program

- Preprocessing, expand all *macro directives* (`#include`, `#define`, ...).
- Compile code to assembly for the target architecture.
- Link calls of procedures to the code for the procedures.
- Create an executable file with directives to the operating system.

compiling a C program

- Preprocessing, expand all *macro directives* (`#include`, `#define`, ...).
- Compile code to assembly for the target architecture.
- Link calls of procedures to the code for the procedures.
- Create an executable file with directives to the operating system.

compiling a C program

- Preprocessing, expand all *macro directives* (`#include`, `#define`, ...).
- Compile code to assembly for the target architecture.
- Link calls of procedures to the code for the procedures.
- Create an executable file with directives to the operating system.

preprocessing, macro expansion

- `#include <stdlib.h>`
- `#define MYNAME "Johan"`
- `#ifdef DEBUG ...#endif`
-

preprocessing, macro expansion

- `#include <stdlib.h>`
- `#define MYNAME "Johan"`
- `#ifdef DEBUG ...#endif`
-

preprocessing, macro expansion

- `#include <stdlib.h>`
- `#define MYNAME "Johan"`
- `#ifdef DEBUG ...#endif`
-

preprocessing, macro expansion

- `#include <stdlib.h>`
- `#define MYNAME "Johan"`
- `#ifdef DEBUG ...#endif`
-

A procedure needs to be *declared* before it is used in a *definition*.

header files

A *header file* holds the declarations of procedures that other files may use.

object files

An object file contains the compiled code of a source file.

linking

Object files are *linked* to resolve procedure calls from one object file to its definition.

from source to executable



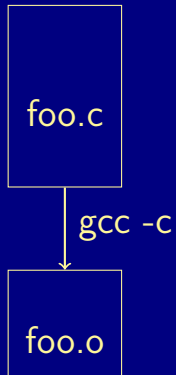
foo.c

from source to executable

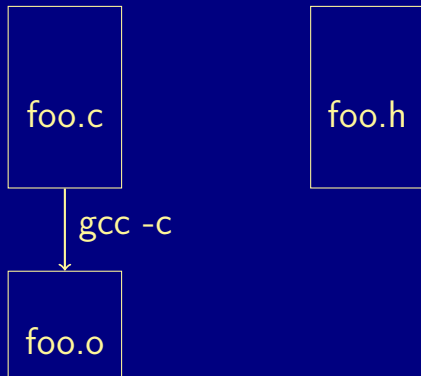
foo.c

foo.o

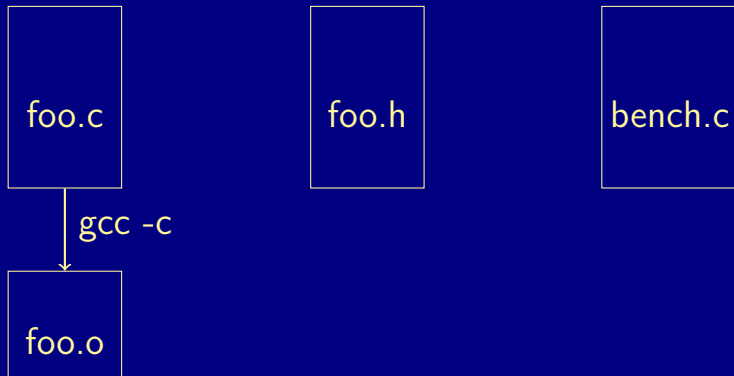
from source to executable



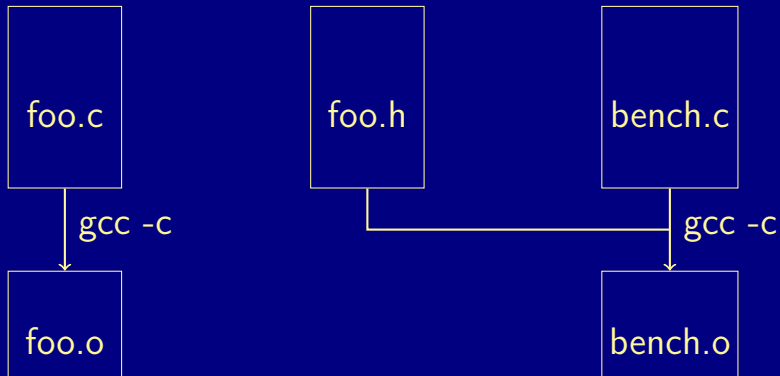
from source to executable



from source to executable



from source to executable



from source to executable

