



(<https://sel4.systems>)

[Getting Started \(/GettingStarted\)](#)

[Contributing \(/processes\)](#)

[Projects \(/projects\)](#)

[Tutorials \(/Tutorials\)](#)

**seL4 Docs (/)** / **Tutorials (/Tutorials/)** / Hello, World!

Current versions: **seL4-11.0.0** (/releases/sel4/11.0.0)

**camkes-3.8.0** (/releases/camkes/camkes-3.8.0)

**CapDL-0.1.0** (/releases/capdl/0.1.0)

**Announcing new releases:** [↗](#)

- Prerequisites
- Initialising
- Outcomes
- Building your first program
- Looking at the sources
  - CMakeLists.txt
  - main.c
- Making a change
- Getting help

# Hello, World!

This tutorial guides you through getting a simple “Hello World” program running as a user-level application on seL4. It allows you to test your local set up and make sure all the tools are working before moving onto more detailed tutorials.

## Prerequisites

1. Set up your machine (<https://docs.sel4.systems/HostDependencies>).

## Initialising

```
# For instructions about obtaining the tutorial sources see https://docs.sel4.systems/Tutorials/#get-the-code
#
# Follow these instructions to initialise the tutorial
# initialising the build directory with a tutorial exercise
./init --tut hello-world
# building the tutorial exercise
cd hello-world_build
ninja
```

## Outcomes

By the end of this tutorial, you should:

- Be familiar with the jargon *root task*.
- Be able to build and simulate seL4 projects.
- Have a basic understanding of the role of the `CMakeLists.txt` file in applications.

## Building your first program

seL4 is a microkernel, not an operating system, and as a result only provides very minimal services. After the kernel boots, an initial thread called the *root task* is started, which is then responsible for setting up the user-level system. When the root task starts there are no available drivers, however a minimal C library is provided.

The tutorial is already set up to print “Hello, world!”, so at this point all you need to do is build and run the tutorial:


```
# In build directory
ninja
```

If successful, you should see the final ninja rule passing:

```
[150/150] objcopy kernel into bootable elf
$
```

The final image can be run by:

```
# In build directory
./simulate
```

This will run the result on an instance of the QEMU  simulator. If everything has worked, you should see:

```
Booting all finished, dropped to user space

Hello, World!
```

## Looking at the sources

In your tutorial directory, you will find the following files:

- `CMakeLists.txt` - the file that incorporates the root task into the wider seL4 build system.
- `src/main.c` - the single source file for the initial task.
- `hello-world.md` - A generated README for the tutorial.

## CMakeLists.txt

Every application and library in an seL4 project requires a `CMakeLists.txt` file in order to be incorporated into the project build system.

```
# @TAG(DATA61_BSD)
include(${SEL4_TUTORIALS_DIR}/settings.cmake)
sel4_tutorials_regenerate_tutorial(${CMAKE_CURRENT_SOURCE_DIR})

cmake_minimum_required(VERSION 3.7.2)
# declare the hello-world CMake project and the languages it is written in (just C)
project(hello-world C ASM)

# In future tutorials, these setup steps will be replaced with
# sel4_tutorials_setup_roottask_tutorial_environment()
find_package(sel4 REQUIRED)
find_package(elfloader-tool REQUIRED)
find_package(musllibc REQUIRED)
find_package(util_libs REQUIRED)
find_package(sel4_libs REQUIRED)

sel4_import_kernel()
elfloader_import_project()

# This sets up environment build flags and imports musllibc and runtime libraries.
musllibc_setup_build_environment_with_sel4runtime()
sel4_import_libsel4()
util_libs_import_libraries()
sel4_libs_import_libraries()
sel4_tutorials_import_libsel4tutorials()

# Name the executable and list source files required to build it
add_executable(hello-world src/main.c)

# List of libraries to link with the application.
target_link_libraries(hello-world
    sel4runtime sel4
    muslc utils sel4tutorials
    sel4muslcsys sel4platsupport sel4utils sel4debug)

# Tell the build system that this application is the root task.
include(rootserver)
DeclareRootserver(hello-world)
```

## main.c

The main C is a very typical C file. For a basic root server application, the only requirement is that a `main` function is provided.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello, World!\n");

    return 0;
}
```

# Making a change

Test making a change to `main.c` by adding a second `printf` to output `"Second hello\n"` .

```
int main(int argc, char *argv[]) {  
    printf("Hello, World!\n");  
  
    printf("Second hello\n");  
    return 0;  
}
```

Once you have made your change, rerun `ninja` to rebuild the project:

```
# In build directory  
ninja
```

Then run the simulator again:

```
# In build directory  
./simulate
```

On success, you should see the following:

```
Second hello
```

---

## Getting help

Stuck? See the resources below.

- FAQ (<https://docs.sel4.systems/FrequentlyAskedQuestions>)
  - seL4 Manual [↗](#)
  - Debugging guide (<https://docs.sel4.systems/DebuggingGuide.html>)
  - IRC Channel (<https://docs.sel4.systems/IRCChannel>)
  - Developer's mailing list [↗](#)
- 

### seL4 docs



This site is for displaying seL4 related documentation. Pull requests are welcome.

Site last updated: Mon Aug 10 11:09:01 2020 +1000 84aefb8

Page last updated: Mon Oct 15 17:13:12 2018 +1100 83f68de

View page on GitHub [↗](#)

Edit page on GitHub [↗](#)

Sitemap (</sitemap>)