Files, Structs, and Typedef

Eike Ritter

Modified: October 4, 2012

Lecture 5: Operating Systems with C/C++
School of Computer Science, University of Birmingham, UK

Outline

Files

- We will explore file systems in a later lecture, but for today we must be aware of the following:
  - A file is an abstraction that allows programmers to read and write data to some arbitrary (usually permanent/non-volatile) storage device (*e.g.* a hard disk, USB stick, *etc.*).
    - Without such an abstraction, we would have to use specific low-level operations to control each different device, saying the exact physical location to read and write data — a nightmare!
  - We saw in an earlier lecture how the OS keeps track of some state for each process, and part of this state is a list of open files and the process' current read or write position within those files.

Structs

- Simple types are useful up to a point, but then we need some way of modelling more complex information in our C programs (*e.g.* for describing people, vehicles, genome sequences, *etc.*).
- The C `struct` allows us to group several types into a single, composite type, then we let the compiler worry about how that gets represented as a chunk of memory.
  - Usually the chunk of memory is occupied by each constituent type in source-specified order
  - Though, in order to optimise CPU performance or by necessity of the CPU, the compiler may, through padding, align certain types of a struct to address boundaries.

Typedefs

- To save typing long-winded type declarations, we can use `typedef` to define our own types.
- This is often useful for `struct` definitions, but you will also see many weird and wonderful types that are defined with `typedef` by the core C libraries, such as:
  - `size_t`
  - `FILE`
  - `int8_t`, `int16_t`, `int32_t`, `int64_t`, *etc.* — so defined in `stdint.h` to give the programmer exact-width integer types on any CPU architecture, which are useful when we care exactly how many bits are in our data types (*e.g.* for constructing network protocols, manipulating devices, *etc.*)

Summary

We looked at:

1. Files
2. Structs
3. Typedefs