Eliminating implicit function declarations

Florian Weimer

GNU Tools @ LPC, 2021-09-23

Abstract

What should we do about GCC’s support for implicit function declarations? There were removed in C99, but GCC has yet to make the transition.

What are implicit function declarations?

```
main ()
{
  printf (”Hello, %s!\n”, getenv (”USER”));
  exit (0);
}
```

What are implicit function declarations?

```
implicit.c:1:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
    1 | main ()
      | ^~~~
implicit.c: In function ‘main’:
implicit.c:3:3: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-
declaration]
    3 |   printf ("Hello, %s!\n", getenv ("USER"));
      |   ^~~~~~
implicit.c:3:3: warning: incompatible implicit declaration of built-in function ‘printf’
implicit.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | main ()
implicit.c:3:27: warning: implicit declaration of function ‘getenv’ [-Wimplicit-function-
declaration]
    3 |   printf ("Hello, %s!\n", getenv ("USER"));
      |                           ^~~~~~
implicit.c:4:3: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-
declaration]
    4 |   exit (0);
      |   ^~~~
implicit.c:4:3: warning: incompatible implicit declaration of built-in function ‘exit’
implicit.c:1:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
  +++ |+#include <stdlib.h>
    1 | main ()
```

No compiler error!

Program links and runs[1].

[1] [illegible]

Implicit function declarations on 64-bit architectures

- No prototype, and the return type is `int`:
  - `extern int getenv ();`
- The x86-64 ABI *requires* that the upper 32 bits are zeroed when converted back to a pointer.
- On x86-64, `_Bool`-returning functions are peculiar.
- Weird corruptions are the result.

Red Hat

Implicit function declarations and shared objects

- binutils BFD ld defaults to `-z nodefs` with `-shared`.
- Undefined symbols are reported only (much) later.
  - When linking the shared object into a program (default: `-z defs`).
  - At run time, when loading (if linked with `-z now`).
  - At run time, when invoked (if linked with `-z lazy`).
- A typo on an error path might never be diagnosed properly due to lazy binding.

GCC's implicit function declarations are surprising to developers

- OpenSSL developers say this about implicit function declarations:

  We didn't anticipate a compiler with implicit declarations and lazy runtime binding. Given this is an unlikely setup, [. . .]

  `https://github.com/openssl/openssl/issues/16254#issuecomment-894624215`

- This results in silent miscompilation of the .Net Core run-time library during an OpenSSL 3.0 porting effort.

Just turn on more warnings as errors?

- It should be really easy to switch on `-Werror=implicit-function-declarations`.
- But only for new programs.
  - For example, libstdc++ lost futex support on Linux if built without implicit function declarations.
  - bash, gawk, gettext, gnulib, make, Perl, PHP, rsync, unzip, . . .
- If configure test fail, test suites bits are automatically disabled as well.
- *Enabling this at the distribution level is hard.*

GCC autoconf fragment

```
AC_LINK_IFELSE(
 [AC_LANG_PROGRAM(
  [#include <sys/syscall.h>
   int lk;],
  [syscall (SYS_gettid);
   syscall (SYS_futex, &lk, 0, 0, 0);
  ])],
  ...
```

`#include <unistd.h>` is missing for the `syscall` function.

Red Hat

Detecting packages that rely on implicit function declarations

- First attempt: `config.log`/`config.h` diffing
    - Build each package twice, with and without errors.
    - See if `configure` detects things differently.
    - Does not work with all crufty build systems.
- New idea: Patch GCC to write an error report file into a magic directory.
    - Fail the entire package build at the end if the directory is not empty.

Errors for implicit function declarations: Are they worth the effort?

- Sharing patches across distributions is not easy.
- This issue disproportionately affects old code with weird build systems and dormant upstreams.
- It’s mostly boring work, ideal for gloomy November days.
- Apple’s Xcode recently made the transition.
- I think: *The improvement in developer experience is real and worth the effort.*

Realistic target: GCC 13?

Thanks for listening.

Questions? Comments?