Deferred Initialization of Static Objects

X3J16/91-0143, WG21/N0076

John Wilkinson, jfw@sgi.com

Silicon Graphics

See Stephen Kearns's paper (X3J16/91-0137, WG21/N0070) and my earlier paper (X3J16/91-0053) for context.

In most (perhaps all) current implementations of C++, static objects are initialized before control enters main(). The Working Group at the Dallas meeting seemed to agree that the Standard should sanction, but not mandate this behavior: that is, an implementation should be permitted to defer the initialization of the static objects in a translation unit. In this paper I offer a justification for deferred initialization, a suggestion for the wording in the standard, and a few remarks on implementation.

The obvious reason for permitting deferred initialization is to decrease startup time. Consider, for example, a large interactive application which implements many commands. Suppose each command is implemented in a translation unit which requires some static initialization. If every translation unit must be initialized before control enters main(), the user may experience an annoying delay at startup while initializations are performed which could just as well be done later, and which perhaps would not have to be done at all. Large startup times are a well-known bane of interactive applications: we should try not to make them compulsory.

Also dynamic linking

It is not entirely trivial to find the right thing to say about deferred initialization. The current draft says "the initialization of the nonlocal static objects in a translation unit may be deferred until the first use of any function or object defined in that translation unit," but this language runs afoul of the example

```
  int x = f();
  ....
  int f() {...}
```

Here the initialization of x can clearly not take place before the first reference to f(), since that very initialization contains a reference to f().

It has been suggested that this problem could be solved by adding the words "in a thread starting from main()." This, however, is inadequate, as the following example shows:

```
main.C:    X.C:            Y.C:

main()     int x = g();    int y = ...
{          ...             ...
  ...      void f() {...}  int g() {...}
  f();
}
```

If the initialization of y can be deferred until the first use of a function or object defined in Y.C, then it can be deferred forever, since there is no use of g() in a thread starting from main().

I suggest instead language equivalent to the following:

"The nonlocal static objects in a translation unit must be initialized either before control enters main() or before any function or object defined in that translation unit is used by any other translation unit."

Note that if the initialization takes place before control enters main(), then the initialization-before-use requirement is waived, so objections on the grounds of implementation cost do not arise. An application need not implement deferred initialization.

Implementation of the rule for deferred initialization requires catching the first reference into a translation unit from outside. In most virtual memory systems this need not be expensive. At startup, the code and data addresses of those translation units for which initialization is to be deferred can be invalidated; then an exception handler can revalidate the addresses and invoke the initialization code.