Thread-Local Storage

Lawrence Crowl, 2006-02-23, N1966=06-0036

Revision of 2005-08-25, N1874=05-0134

Introduction

In multi-threaded applications, there often arises the need to maintain data that is unique to a thread. We call this thread-local storage.

Several techniques have been used to accomplish this task. Notable among them is the POSIX `getthreadspecific` and `setthreadspecific` facility. Unfortunately, this facility is clumsy and slow. In addition, the facility is not particularly helpful when converting a single-threaded application to a multi-threaded application.

Several vendors have provided a language extension for a new storage class that indicates that a variable has thread duration. Use of thread variables is relatively easy and access to thread variables is relatively fast. In addition, the conversion of a single-threaded application using static-duration variables to a multi-threaded application using thread-duration variables requires less wholesale program restructuring.

Roughly equivalent extensions are available from

GNU	Thread-Local Storage
Hewlett-Packard	Using Thread Local Storage
Hewlett-Packard	Tru64 UNIX to HP-UX STK: critical Impact: TLS - feature differences (CrCh320)
Intel	Thread-local Storage
Microsoft	Thread Local Storage
Sun Microsystems	Thread-Local Storage

The C++ standard should adopt existing practice for thread-local storage. In addition, the C++ standard should extend existing practice to enable broader use.

Proposal

The specification outline is as follows. We defer detailed changes to the text of the standard pending adoption of the proposal in principle.

Thread Storage Duration

Add a new storage duration called thread duration. Objects with thread duration are unique to each thread.

Those objects which may have static duration may have thread duration instead. These objects include file global variables, file static variables, function local static variables, and class static member variables.

Storage Class `__thread`

Add `__thread`, a new keyword and storage class specifier. The `__thread` specifier indicates that the variable has thread duration.

Variables declared with the `__thread` specifier are bound as they would be without the `__thread` specifier.

Addresses of Thread Variable

The address-of operator (`&`), when applied to a thread variable, is evaluated at run time and returns the address of the current thread's variable. Therefore, the address of a thread variable is not a constant.

Thread-local storage defines lifetime and scope, not accessibility. That is, one may take the address of a thread-local variable and pass it to other threads.

The address of a thread variable is stable for the lifetime of the corresponding thread. The address of a thread variable may be freely used during the variable's lifetime by any thread in the program. When a thread terminates, all addresses of that thread's variables are invalid and may not be used.

Thread Variable Initialization

A thread variable may be statically initialized as would any other static-duration variable.

At present, all implementations of thread-local storage do not support dynamic initalization (and presumably non-trivial destructors). There was mild consensus at the Mont Treblant meeting to support dynamic initialization of function-local, thread-local variables. The intialization of such variables is already guarded and synchronous, so new technology is not required. On the other hand, the implementation for dynamic initialization of namespace-scope variables is much more difficult, and may require additional linker and operating system support. There was no consensus to support dynamic initialization of namespace-scope variables at this time.

Other Issues

There are some other issues that deserve mention even though they are not properly part of the C++ standard because they affect real programs.

Dynamic Libraries

The allocation of thread-local storage for the full product of threads and dynamic libraries could result in very large storage requirements. The Sun Microsystems implementation only allocates thread-local storage for a dynamic library when the thread uses a variable from that library. That is, the Sun implementation allocates memory lazily for each thread and dynamic library pair. To avoid bloated programs, the language definition must permit this optimization.

The system may immediately deallocate the storage associated with a thread and dynamic library pair when either the thread terminates or the library is closed. The system is not required to deallocate immediately. However, the system is required to not leak storage. Thread-local storage for a thread must be reclaimed no later than a subsequent thread creation. Thread-local storage for a library within a thread must be reclaimed no later than a subsequent open of that library. (Opening another library does not require storage reclamation, though doing so would ceratinly reduce storage consumption.)

System Interface

When `dlsym()` is used on a thread variable, the address returned will be the address of the currently executing thread's variable.