C++-Threads

Hvorfor introdusere trådstøtte?

- Portabilitet
  - Posix Threads, Windows Threads
  - Boost.Thread?
- Effektivitet
  - Unødvendig oppdatering av cache i multi-core/cpu-systemer.
- Korrekthet
  - Nåværende standard kan ikke bidra.

Korrekthet

- Abstract machine og sequence points – hva er problemet?
- Eksempel: DCLP

1.9p5

A conforming implementation executing a well-formed program shall produce the same observable behavior as one of the possible execution sequences of the corresponding instance of the abstract machine with the same program and the same input. However, if any such execution sequence contains an undefined operation, this International Standard places no requirement on the implementation executing that program

1.9p6

The observable behavior of the abstract machine is its sequence of reads and writes to volatile data and calls to library I/O functions

```
1. Singleton* Singleton::instance()
2. {
3.   if (pInstance == 0) // 1st test
4.   {
5.     Lock lock;
6.     if (pInstance == 0) // 2nd test
7.     {
8.       pInstance = new Singleton;
9.     }
10.  }
11.  return pInstance;
12.}
```

Double Checked Locking Pattern

```
1.  Singleton* Singleton::instance()
2.  {
3.    if (pInstance == 0)
4.    {
5.      Lock lock;
6.      if (pInstance == 0)
7.      {
8.        pInstance = operator new(sizeof(Singleton));
9.        new (pInstance) Singleton;
10.     }
11.   }
12.   return pInstance;
13. }
```

Double Checked Locking Pattern

```
1.  Singleton* Singleton::instance()
2.  {
3.    if (pInstance == 0)
4.    {
5.      Lock lock;
6.      if (pInstance == 0)
7.      {
8.        Singleton* tmp = new Singleton();
9.        pInstance = tmp;
10.     }
11.   }
12.   return pInstance;
13. }
```

Double Checked Locking Pattern

```
1.  Singleton* Singleton::instance()
2.  {
3.    if (pInstance == 0)
4.    {
5.      Lock lock; // implisitt barrier
6.      if (pInstance == 0)
7.      {
8.        Singleton* tmp = new Singleton();
9.        // … release barrier
10.       pInstance = tmp;
11.     }
12.   }
13.   return pInstance;
14. }
```

C++0x: Oversikt

- Enkelt trådbibliotek
- Atomics
- Revidering av programflytspesifkasjon
  - 1.9 ”Program Execution”, endres
  - 1.10 "Multi-threaded executions and data races”, legges til

C++0x: Trådbibliotek

- Utvidelse av standardbibliotek
  - std::thread

```
1.  #include <thread>
2.  void f();
3.  void bar()
4.  {
5.    std::thread t(f); // f() executes in separate thread
6.    t.join(); // wait for t to end
7.  }
```

  - std::mutex
  - std::lock
    - scoped_lock, timed_lock, etc.
- Koblet til OS-api, med tilgang til tråd-id osv.

C++0x: Atomics

- Et sett enkle typer, samt en template, for atomiske operasjoner
- Målsetning: kunne brukes fra C++ *og* C
  - Makroer i C
  - Members i C++
- Feature-queries

C++0x: Atomics – Memory Order

- Eks. memory_order_relaxed, memory_order_rel og memory_order_seq_cst.
- Angir i hvilken grad minneoperasjoner må gjøres i streng rekkefølge, samt synlighet for andre tråder.

C++0x: Atomics – Operasjoner

- store
- load
- swap
- compare-and-swap
- fence
- fetch-and-{add, sub, and, or, xor}

C++0x: Atomics – DCLP v2

```
1.  class Singleton
2.  {
3.  // ... public stuff
4.  private:
5.    static atomic<Singleton*> pInstance;
6.    static std::mutex singleMutex;
7.  };
8.  // ... static init

9.  Singleton* Singleton::instance()
10. {
11.   if (pInstance.load(memory_order_acq) == 0)
12.   {
13.     std::scoped_lock<std::mutex> _(singleMutex);
14.     if (pInstance.load(memory_order_acq) == 0)
15.     {
16.       pInstance.store(memory_order_rel, new Singleton());
17.     }
18.   }
19.   return pInstance;
20. }
```

C++0x: ”Program Execution” revideres

- 1.9 får nå kjennskap til tråder
- ”Sequence points” bort, ”sequenced before” inn.
- Fremdeles åpne spørsmål og revideringer.

C++0x: ”Multi-threaded executions and data races”

- Bygger på revidert 1.9, og definerer gyldig hendelsesforløp mellom tråder, samt atomics
- Definerer ”data races”

1.10p8

An evaluation A happens before an evaluation B if:

- *A is sequenced before B or*
- *A synchronizes with B; or*
- *for some evaluation X, A happens before X and X happens before B.*

C++0x: Andre tillegg

- Dependency chains
- Thread-local storage
- Mulige biprodukter: save / rethrow av exceptions

Kilder

- Multi-threading Library for Standard C++
- Atomic Types and Operations
- A finer-grained alternative to sequence points (1.9)
- Concurrency Memory Model (1.10)
- C++ Data Dependency
- ISO/IEC 14882:2003 (Hovedsakelig 1.9)
- http://www.aristeia.com/Papers/DDJ_Jul_Aug_2004_revised.pdf

Takk for meg!

- http://www.ocppug.org
- http://einaros.blogspot.com