Threads Cannot Be Implemented As a Library

Paper by Hans-J. Boehm

Presented by Håkan Lundvall

Outline

- Memory models
- The library approach
- Three correctness issues
- Performance

- Some languages knows about threads
  - Java, C#, Ada…
- Some do not…
  - C/C++
- Libraries must be used to handle concurrency
  - Pthreads

- The context of this discussion
  - C
  - Pthreads

  Widely used and reasonable well specified

Pthread's approach and why it appears to work

- To support concurrency a memory model must be specified
  - Sequential consistency
    - x = 1; r1 = y; // thread 1
    - y = 1; r2 = x; // thread 2
    - Either r1 or r2 must be 1
  - In practice
    - Both can be 0

Why?

- Memory operations in one thread may be reordered by the compiler
- Memory operations may be reordered by the hardware
- Java as well as the Pthread standard both allow r1 = r2 = 0 as a result

Pthread

"Formal definitions of the memory model were rejected as unreadable by the vast majority of programmers. In addition, most of the formal work in the literature has concentrated on the memory as provided by the hardware as opposed to the application programmer through the compiler and runtime system. It was believed that a simple statement intuitive to most programmers would be most effective".

"Formal definitions of the memory model were rejected as unreadable by the vast majority of programmers. In addition, most of the formal work in the literature has concentrated on the memory as provided by the hardware as opposed to the application programmer through the compiler and runtime system. It was believed that a simple statement intuitive to most programmers would be most effective".

..., pthread mutex lock(), ...,

..., pthread mutex unlock(), ...

Why it appears to work

Functions to synchronize memory

pthread_mutex_lock()
pthread_mutex_unlock()

The compiler knows nothing of the functions.

Any global variable may be read or written as far as the compiler can tell.

So all variables must be written to memory before the call.

Three examples

- Concurrent modification
  - A bit far fetched, no actual case reported
- Rewriting of adjacent data
  - There are anecdotes about failures due to this
- Register promotion
  - Personal experience of the author

Concurrent modification

```
if (x == 1) ++y; // thread 1
if (y == 1) ++x; // thread 2
```

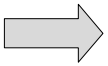
```
++y; if (x != 1); --y; // thread 1
++x; if (y != 1); --x; // thread 2
```

Rewriting of adjacent data

```
struct {
  int a:17;
  int b:15;
} x;

…

x.a = 42;
```



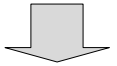
```
{
    tmp = x;
    tmp &= ~0x1ffff;
    tmp |= 42;
    x = tmp;
}
```

Rewriting of adjacent data

```
struct {
 char a; char b; char c; char d;
 char e; char f; char g; char h;
} x;

…

x.b = ´b´; x.c = ´c´; x.d = ´d´;
x.e = ´e´; x.f = ´f´; x.g = ´g´; x.h = ´h´;
```



```
x = *(long*)(”hgfedcb”) | x.a;
```

Assuming 64 bit architecture

Register promotion

```
for (…) {
  …
  if (mt) pthread_mutex_lock(…);
  x = … x …
  if (mt) pthread_mutex_unlock(…);
}
```

```
r = x;
for (…) {
  …
  if (mt) {
    x = r; pthread_mutex_lock(…); r = x;
  }
  r = … r …
  if (mt) {
    x = r; pthread_mutex_unlock(…) r = x;
  }
}
x =r;
```

Performance

- Pthread standard says concurrent modifications of shared variables shall be protected by mutual exclusion primitives
  - Atomic hardware memory update instruction
    - Prevents hardware reordering of memory references or separate memory barrier instruction
  - Library calling overhead

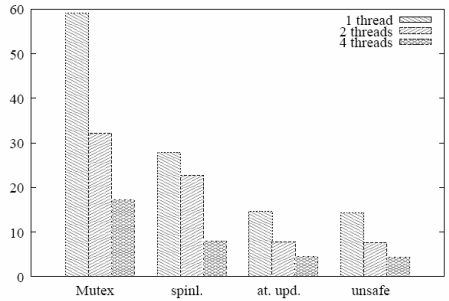
Example of expensive synchronization

- Sieve of Eratosthenes
  - Calculate primes between 10,000 and 100,000,000
- Garbage collection
  - Mark-sweep on a heap containing 200MB of 24-byte objects

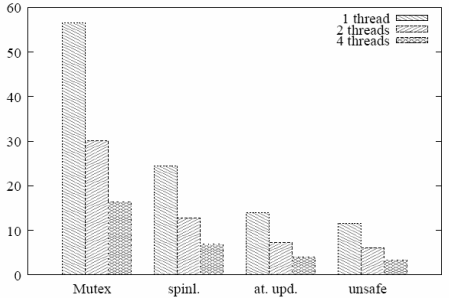
Sieve of Eratosthenes

```
for (my_prime = start; my_prime < 10000; ++my_prime)
  if (!get(my_prime)) {
    for (multiple = my_prime;
         multiple < 100000000;
         multiple += my_prime)
      if (!get(multiple)) set(multiple);
  }
```

Sieve using bytearray



Sieve using bitarray



Garbage collection