Lightning Talk: Memory Model

(. . . i nie tylko)

Bartosz ’BaSz’ Szurgot

bartek.szurgot@baszerr.eu

2013-02-13

Plan

Wątki w C++11

- Sekcja 1.7: *The C ++ memory model*

Wątki w C++11

- Sekcja 1.7: *The C ++ memory model*
- The fundamental storage unit in the C ++ memory model is the byte. A byte is at least large enough to contain any member of the basic execution character set (2.3) and the eight-bit code units of the Unicode UTF-8 encoding form and is composed of a contiguous sequence of bits, the number of which is implementation- defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit. The memory available to a C ++ program consists of one or more sequences of contiguous bytes. Every byte has a unique address. [Note: The representation of types is described in 3.9. — end note] A memory location is either an object of scalar type or a maximal sequence of adjacent bit-fields all having non-zero width. [Note: Various features of the language, such as references and virtual functions, might involve additional memory locations that are not accessible to programs but are managed by the imple- mentation. — end note] Two or more threads of execution (1.10) can update and access separate memory locations without interfering with each other. [Note: Thus a bit-field and an adjacent non-bit-field are in separate memory locations, and therefore can be concurrently updated by two threads of execution without interference. The same applies to two bit-fields, if one is declared inside a nested struct declaration and the other is not, or if the two are separated by a zero-length bit-field declaration, or if they are separated by a non-bit-field declaration. It is not safe to concurrently update two bit-fields in the same struct if all fields between them are also bit-fields of non-zero width. — end note] [Example: A structure declared as struct char a; int b:5, c:11, :0, d:8; struct int ee:8; e; ; contains four separate memory locations: The field a and bit-fields d and e.ee are each separate memory locations, and can be modified concurrently without interfering with each other. The bit-fields b and c together constitute the fourth memory location. The bit-fields b and c cannot be concurrently modified, but b and a, for example, can be. — end example]

Wątki w C++11

- Sekcja 1.7: *The C ++ memory model*
- The fundamental storage unit in the C ++ memory model is the byte. A byte is at least large enough to contain any member of the basic execution character set (2.3) and the eight-bit code units of the Unicode UTF-8 encoding form and is composed of a contiguous sequence of bits, the number of which is implementation- defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit. The memory available to a C ++ program consists of one or more sequences of contiguous bytes. Every byte has a unique address. [Note: The representation of types is described in 3.9. — end note] A memory location is either an object of scalar type or a maximal sequence of adjacent bit-fields all having non-zero width. [Note: Various features of the language, such as references and virtual functions, might involve additional memory locations that are not accessible to programs but are managed by the imple- mentation. — end note] Two or more threads of execution (1.10) can update and access separate memory locations without interfering with each other. [Note: Thus a bit-field and an adjacent non-bit-field are in separate memory locations, and therefore can be concurrently updated by two threads of execution without interference. The same applies to two bit-fields, if one is declared inside a nested struct declaration and the other is not, or if the two are separated by a zero-length bit-field declaration, or if they are separated by a non-bit-field declaration. It is not safe to concurrently update two bit-fields in the same struct if all fields between them are also bit-fields of non-zero width. — end note] [Example: A structure declared as struct char a; int b:5, c:11, :0, d:8; struct int ee:8; e; ; contains four separate memory locations: The field a and bit-fields d and e.ee are each separate memory locations, and can be modified concurrently without interfering with each other. The bit-fields b and c together constitute the fourth memory location. The bit-fields b and c cannot be concurrently modified, but b and a, for example, can be. — end example]
- Sekcja 1.10: *Multi-threaded executions and data races*
- (niecałe 4 strony)

Najważniejsze elementy

- Jednostka pamięci:
  - Najmniejsza to 1 bajt
  - Bajt >= 8 bitów
  - Każdy bajt – unikalny adres

Najważniejsze elementy

- Jednostka pamięci:
  - Najmniejsza to 1 bajt
  - Bajt >= 8 bitów
  - Każdy bajt – unikalny adres
- „Memory location”:
  - Pojedyncza jednostka
  - Przylegające pola bitowe (niezerowej długości)

Najważniejsze elementy

- Jednostka pamięci:
  - Najmniejsza to 1 bajt
  - Bajt >= 8 bitów
  - Każdy bajt – unikalny adres
- „Memory location”:
  - Pojedyncza jednostka
  - Przylegające pola bitowe (niezerowej długości)
- Wątki mogą operować na rozłącznych obszarach pamięci bez wpływu na siebie

Najważniejsze elementy

- Jednostka pamięci:
  - Najmniejsza to 1 bajt
  - Bajt >= 8 bitów
  - Każdy bajt – unikalny adres
- „Memory location”:
  - Pojedyncza jednostka
  - Przylegające pola bitowe (niezerowej długości)
- Wątki mogą operować na rozłącznych obszarach pamięci bez wpływu na siebie
- „Data race”...

Data race

- „Conflicting access”:
  - Ta sama lokacja w pamięci
  - Min. 1 wątek czyta
  - Min. 1 (inny) wątek pisze

Data race

- „Conflicting access”:
  - Ta sama lokacja w pamięci
  - Min. 1 wątek czyta
  - Min. 1 (inny) wątek pisze
- „Data race”:
  - „Conflicting access”
  - Jednocześnie z min. 2 wątków

Data race

- „Conflicting access”:
  - Ta sama lokacja w pamięci
  - Min. 1 wątek czyta
  - Min. 1 (inny) wątek pisze
- „Data race”:
  - „Conflicting access”
  - Jednocześnie z min. 2 wątków
- „Data race” != „race condition”

Data race

- „Conflicting access”:
  - Ta sama lokacja w pamięci
  - Min. 1 wątek czyta
  - Min. 1 (inny) wątek pisze
- „Data race”:
  - „Conflicting access”
  - Jednocześnie z min. 2 wątków
- „Data race” != „race condition”
- Standard:
  - Wymaga – „data race free” (DRF)

Data race

- „Conflicting access”:
  - Ta sama lokacja w pamięci
  - Min. 1 wątek czyta
  - Min. 1 (inny) wątek pisze
- „Data race”:
  - „Conflicting access”
  - Jednocześnie z min. 2 wątków
- „Data race” != „race condition”
- Standard:
  - Wymaga – „data race free” (DRF)
  - Zapewnia – „sequential consistency” (SC)

Sequential consistency

- Początkowo $x = y = a = b = 0$
- Jaki będzie wynik?

Sequential consistency

- Początkowo $x = y = a = b = 0$
- Jaki będzie wynik?

```
1 void thread1(void)
2 {
3   x = 1;
4   a = y;
5 }
```

```
1 void thread2(void)
2 {
3   y = 1;
4   b = x;
5 }
```

Sequential consistency

- Początkowo $x = y = a = b = 0$
- Jaki będzie wynik?

```
1 void thread1(void)
2 {
3   x = 1;
4   a = y;
5 }
```

```
1 void thread2(void)
2 {
3   y = 1;
4   b = x;
5 }
```

- Możliwe wyniki:
  - $a = 0, b = 1$
  - $a = 1, b = 0$
  - $a = 1, b = 1$

Sequential consistency

- Początkowo $x = y = a = b = 0$
- Jaki będzie wynik?

```
1 void thread1(void)
2 {
3   x = 1;
4   a = y;
5 }
```

```
1 void thread2(void)
2 {
3   y = 1;
4   b = x;
5 }
```

- Możliwe wyniki:
  - $a = 0, b = 1$
  - $a = 1, b = 0$
  - $a = 1, b = 1$
- Byle nie: $a = 0, b = 0$!

Sequential consistency

- Początkowo $x = y = a = b = 0$
- Jaki będzie wynik?

```
1 void thread1(void)
2 {
3   x = 1;
4   a = y;
5 }
```

```
1 void thread2(void)
2 {
3   y = 1;
4   b = x;
5 }
```

- Możliwe wyniki:
  - $a = 0, b = 1$
  - $a = 1, b = 0$
  - $a = 1, b = 1$
- Byle nie: $a = 0, b = 0$!
- SC == możliwość zrozumienia kodu
- Niezależnie od optymalizacji :-)

Done!

Z punktu widzenia programisty...

- Standard == wspólna płaszczyzna
- Spójny, ustandaryzowany, przenośny...

Z punktu widzenia programisty...

- Standard == wspólna płaszczyzna
- Spójny, ustandaryzowany, przenośny...
- Definuje co oznacza „shared variable”

Z punktu widzenia programisty...

- Standard == wspólna płaszczyzna
- Spójny, ustandaryzowany, przenośny...
- Definuje co oznacza „shared variable”

```
1 struct Hmmm
2 {
3   char a;
4   char b;
5 };
6 Hmmm h;
7 // thread 1: h.a=42;
8 // thread 2: h.b=42;
```

- Wyścig czy nie?

Z punktu widzenia programisty...

- Standard == wspólna płaszczyzna
- Spójny, ustandaryzowany, przenośny...
- Definuje co oznacza „shared variable”

```
1 struct Hmmm
2 {
3   char a;
4   char b;
5 };
6 Hmmm h;
7 // thread 1: h.a=42;
8 // thread 2: h.b=42;
```

- Wyścig czy nie?
- C++11: nie
- pthreads: możliwe...

Prosty model (SC)

- Model „przeplatania wątków”...

```
1  void thread1(void)
2  {
3    x = 1;
4    a = y;
5  }
```

```
1  void thread2(void)
2  {
3    y = 1;
4    b = x;
5  }
```

Prosty model (SC)

- Model „przeplatania wątków”...

```
1  void thread1(void)
2  {
3    x = 1;
4    a = y;
5  }
```

```
1  void thread2(void)
2  {
3    y = 1;
4    b = x;
5  }
```

- Jeśli a, b, x, y to *int*
- Możliwe $a = b = 0$!

Prosty model (SC)

- Model „przeplatania wątków”...

```
1  void thread1(void)
2  {
3    x = 1;
4    a = y;
5  }
```

```
1  void thread2(void)
2  {
3    y = 1;
4    b = x;
5  }
```

- Jeśli a, b, x, y to *int*
- Możliwe $a = b = 0$!
- Pożądane optymalizacje
  - Kompilatory
  - Sprzęt

Prosty model (SC)

- Model „przeplatania wątków”...

```
1  void thread1(void)
2  {
3    x = 1;
4    a = y;
5  }
```

```
1  void thread2(void)
2  {
3    y = 1;
4    b = x;
5  }
```

- Jeśli a, b, x, y to *int*
- Możliwe $a = b = 0$!
- Pożądane optymalizacje
  - Kompilatory
  - Sprzęt
- Jak zapewnić SC?

Prosty model (SC)

- Model „przeplatania wątków”...

```
1  void thread1(void)
2  {
3    x = 1;
4    a = y;
5  }
```

```
1  void thread2(void)
2  {
3    y = 1;
4    b = x;
5  }
```

- Jeśli a, b, x, y to *int*
- Możliwe $a = b = 0$!
- Pożądane optymalizacje
  - Kompilatory
  - Sprzęt
- Jak zapewnić SC?
- Powiedzieć o tym kompilatorowi! :-)
  - x, y typu *std::atomic<int>*
  - Alternatywnie użyć *std::mutex*

Ograniczanie radosnych optymalizacji

- Kompilator nie może robić wszystkiego
  - Musi zapewnić SC (dla DRF)
  - Nie może wprowadzać DR

Ograniczanie radosnych optymalizacji

- Kompilator nie może robić wszystkiego
  - Musi zapewnić SC (dla DRF)
  - Nie może wprowadzać DR

```
1  // INPUT:
2  int foo(int a)
3  {
4    if(a<1)
5      b=2;
6    if(a==2)
7      b=2;
8    if(a>2)
9      b=2;
10   return b;
11
12
13 }
```

```
1  // OPT1:
2  int foo(int a)
3  {
4    if(a>2)
5      b=2;
6    else
7      if(a<1)
8        b=2;
9      else
10       if(a==2)
11         b=2;
12   return b;
13 }
```

```
1  // OPT2:
2  int foo(int a)
3  {
4    const int tmp=b;
5    b=2;
6    if(a==1)
7      b=tmp;
8    return b;
9
10
11
12
13 }
```

Ograniczanie radosnych optymalizacji

- Kompilator nie może robić wszystkiego
  - Musi zapewnić SC (dla DRF)
  - Nie może wprowadzać DR

```
1  // INPUT:
2  int foo(int a)
3  {
4    if(a<1)
5      b=2;
6    if(a==2)
7      b=2;
8    if(a>2)
9      b=2;
10   return b;
11
12
13 }
```

```
1  // OPT1:
2  int foo(int a)
3  {
4    if(a>2)
5      b=2;
6    else
7      if(a<1)
8        b=2;
9      else
10       if(a==2)
11         b=2;
12   return b;
13 }
```

```
1  // OPT2:
2  int foo(int a)
3  {
4    const int tmp=b;
5    b=2;
6    if(a==1)
7      b=tmp;
8    return b;
9
10
11
12
13 }
```

- Poprawne czy nie?

Ograniczanie radosnych optymalizacji

- Kompilator nie może robić wszystkiego
  - Musi zapewnić SC (dla DRF)
  - Nie może wprowadzać DR

```
1  // INPUT:
2  int foo(int a)
3  {
4    if(a<1)
5      b=2;
6    if(a==2)
7      b=2;
8    if(a>2)
9      b=2;
10   return b;
11
12
13 }
```

```
1  // OPT1:
2  int foo(int a)
3  {
4    if(a>2)
5      b=2;
6    else
7      if(a<1)
8        b=2;
9      else
10       if(a==2)
11         b=2;
12   return b;
13 }
```

```
1  // OPT2:
2  int foo(int a)
3  {
4    const int tmp=b;
5    b=2;
6    if(a==1)
7      b=tmp;
8    return b;
9
10
11
12
13 }
```

- Poprawne czy nie?
- Tylko *opt1*

Const czy nie-const

- C++98/03:
  - const-logiczny („obserwowalny”)
  - const-faktyczny („binarny”)

Const czy nie-const

- C++98/03:
  - const-logiczny („obserwowalny”)
  - const-faktyczny („binarny”)
- C++11:
  - const-binarny
  - const-synchronizowany

Const czy nie-const

- C++98/03:
  - const-logiczny („obserwowalny”)
  - const-faktyczny („binarny”)
- C++11:
  - const-binarny
  - const-synchronizowany
- Nie jest wymagane ale...
- Inaczej:
  - Ograniczone użycie...
  - Możliwy DR...

Const czy nie-const

- C++98/03:
  - const-logiczny („obserwowalny”)
  - const-faktyczny („binarny”)
- C++11:
  - const-binarny
  - const-synchronizowany
- Nie jest wymagane ale...
- Inaczej:
  - Ograniczone użycie...
  - Możliwy DR...
- STL zakłada const-binarny
- Również dla typów użytkownika

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

- Co może pójść nie tak?

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

- Co może pójść nie tak?
- DR na *done*

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

- Co może pójść nie tak?
- DR na *done*
- Dlaczego nie DR na *answer*?

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

- Co może pójść nie tak?
- DR na *done*
- Dlaczego nie DR na *answer*?
- Może *volatile bool done*?

Hej – skończyłem!

- Przekazywanie sygnału gotowości:

```
1 void thread1(void)
2 {
3   answer = askComputer();
4   done   = true;
5 }
```

```
1 void thread2(void)
2 {
3   while(not done)
4     yield();
5   cout << "the answer is "
6        << answer << endl;
7 }
```

- Co może pójść nie tak?
- DR na *done*
- Dlaczego nie DR na *answer*?
- Może *volatile bool done*?
- *volatile* ! = *std::atomic*!

Kończymy pracę?

- Współpraca z odłączonym wątkiem 2:

```
1  int main(void)
2  {
3    ThreadSafeQueue q;
4    thread th2( [&]()
5                {
6                  thread2(q);
7                } );
8    th2.detach();
9    for(int i=0; i<10; ++i)
10   {
11     q.push(i+42);
12     yield();
13   }
14 }
```

```
1  void thread2(ThreadSafeQueue& q)
2  {
3    while(true)
4    {
5      auto e=q.pop(); // blocks
6      process(e);
7    }
8  }
```

Kończymy pracę?

- Współpraca z odłączonym wątkiem 2:

```
1  int main(void)
2  {
3    ThreadSafeQueue q;
4    thread th2( [&]()
5                {
6                  thread2(q);
7                } );
8    th2.detach();
9    for(int i=0; i<10; ++i)
10   {
11     q.push(i+42);
12     yield();
13   }
14 }
```

```
1  void thread2(ThreadSafeQueue& q)
2  {
3    while(true)
4    {
5      auto e=q.pop(); // blocks
6      process(e);
7    }
8  }
```

- Co może pójść nie tak?

Kończymy pracę?

- Współpraca z odłączonym wątkiem 2:

```
1  int main(void)
2  {
3    ThreadSafeQueue q;
4    thread th2( [&]()
5                {
6                  thread2(q);
7                } );
8    th2.detach();
9    for(int i=0; i<10; ++i)
10   {
11     q.push(i+42);
12     yield();
13   }
14 }
```

```
1  void thread2(ThreadSafeQueue& q)
2  {
3    while(true)
4    {
5      auto e=q.pop(); // blocks
6      process(e);
7    }
8  }
```

- Co może pójść nie tak?
- DR podczas niszczenia q
- Jak to naprawić?

Sprawdźmy coś

- Początkowo $x = y = 0$
- Zmienne typu *int*

Sprawdźmy coś

- Początkowo $x = y = 0$
- Zmienne typu *int*

```
1  void thread1(void)
2  {
3    if(x)
4      y=1;
5  }
```

```
1  void thread2(void)
2  {
3    if(y)
4      x=1;
5  }
```

Sprawdźmy coś

- Początkowo $x = y = 0$
- Zmienne typu *int*

```
1  void thread1(void)
2  {
3    if(x)
4      y=1;
5  }
```

```
1  void thread2(void)
2  {
3    if(y)
4      x=1;
5  }
```

- Jaki problem?

Sprawdźmy coś

- Początkowo $x = y = 0$
- Zmienne typu *int*

```
1  void thread1(void)
2  {
3    if(x)
4      y=1;
5  }
```

```
1  void thread2(void)
2  {
3    if(y)
4      x=1;
5  }
```

- Jaki problem?
- Żaden!
  - Występuje „conflicting access”
  - Brak jednoczesności

Sprawdźmy coś

- Początkowo $x = y = 0$
- Zmienne typu *int*

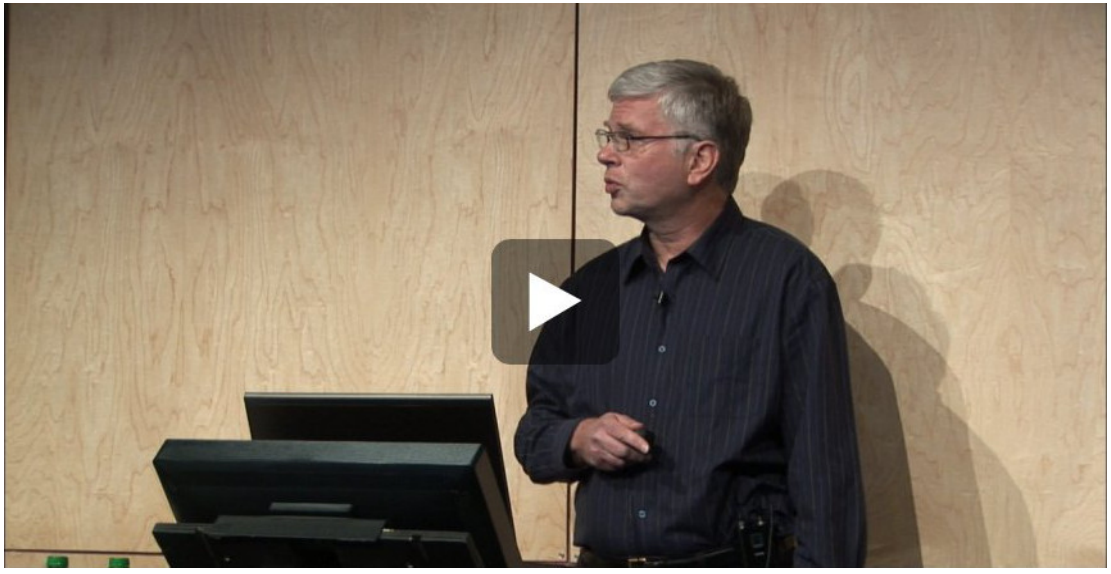
```
1  void thread1(void)
2  {
3    if(x)
4      y=1;
5  }
```

```
1  void thread2(void)
2  {
3    if(y)
4      x=1;
5  }
```

- Jaki problem?
- Żaden!
  - Występuje „conflicting access”
  - Brak jednoczesności
- Ale $x = y = 2$ implikuje DR...

Warto zobaczyć

- *C++ And Beyond 2012*
- *Threads and Shared Variables in C++11*
- Hans Boehm



- `http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/Threads-and-Shared-Variables-in-C-11`

Pytania?