Variadic templates, std::chrono

Alek Lewandowski, LizardFS.org

C++ User Group, 2013-12-10

LizardFS.org

- Sieciowy, rozproszony system plików
- https://github.com/lizardfs/lizardfs
- Fork MooseFS.org
  - C99
- C++11 !!!

Czas w sieciowym systemie plików

- Bezpieczeństwo, QoS, wysoka dostępność
- Timeouty
- Locki, granty
- Założenia związane ze spójnością

Problemy z mierzeniem czasu

- Przesunięcia
  - czas letni/zimowy
  - date -s
- Problemy przy użyciu wielu serwerów
  - Network Time Protocol
  - Różne częstotliwości taktowania

Off top

Google Spanner

- “...a novel time API that exposes clock uncertainty”
- http://research.google.com/archive/spanner.html
- https://www.usenix.org/conference/osdi12/elmo-building-globally-distributed-highly-available-database

MooseFS

```
alek@alek:~/src/moosefs-code$ find . -name '*.h' -o -name '*.c' | xargs grep --color=auto -i backward -B1
./mfscommon/main.c-	if (now<prevtime) {
./mfscommon/main.c:		// time went backward !!! - recalculate "nextevent" time
--
./mfsmount/writedata.c-		if ((uint32_t)(tv.tv_sec) < sec) {
./mfsmount/writedata.c:			// time went backward !!!
--
./mfsmaster/chartsdata.c:	if (uc.it_value.tv_sec<=999) {	// on fucken linux timers can go backward !!!
--
./mfschunkserver/chartsdata.c:	if (uc.it_value.tv_sec<=999) {	// on fucken linux timers can go backward !!!
```

MooseFS c.d.

```
alek@alek:~/src/moosefs-code$ find . -name '*.h' -o -name '*.c' | xargs grep --color=auto -iw '.*tick[s]\{0,1\}'
./mfsmount/readdata.c:#define USECTICK 333333
./mfsmount/readdata.c:#define REFRESHTICKS 15
./mfsmount/readdata.c:#define CLOSEDELAYTICKS 3
./mfsmount/readdata.c:             if (rrec->refcnt<REFRESHTICKS) {
./mfsmount/readdata.c:                          if (rrec->noaccesscnt==CLOSEDELAYTICKS) {
./mfsmount/readdata.c:        usleep(USECTICK);
./mfsmount/readdata.c:        rrec->noaccesscnt=CLOSEDELAYTICKS; // if no access then close socket as soon as possible
./mfsmount/readdata.c:        rrec->refcnt=REFRESHTICKS;         // force reconnect on forthcoming access
./mfsmount/readdata.c:   forcereconnect = (rrec->fd>=0 && rrec->refcnt==REFRESHTICKS)?1:0;
```

MooseFS c.d.

```
void* read_data_delayed_ops(void *arg) {
[...]
     for (;;) {
[...]
               if (rrec->refreshCounter < REFRESHTICKS) {
                    rrec->refreshCounter++;
               }
[...]
          usleep(USECTICK);
     }
}


pthread_create(&delayedOpsThread,&thattr,read_data_delayed_ops,NULL);
```

LizardFS

- Chcielibyśmy tak:

```
typedef std::chrono::steady_clock SteadyClock;
typedef SteadyClock::time_point SteadyTimePoint;
typedef SteadyClock::duration SteadyDuration;
```

- … no niestety jest troszkę trudniej, ale niewiele
  - (później `src/common/time_utils.cc`)
- `src/common/time_utils_unittest.cc`

Biblioteka obsługi czasu

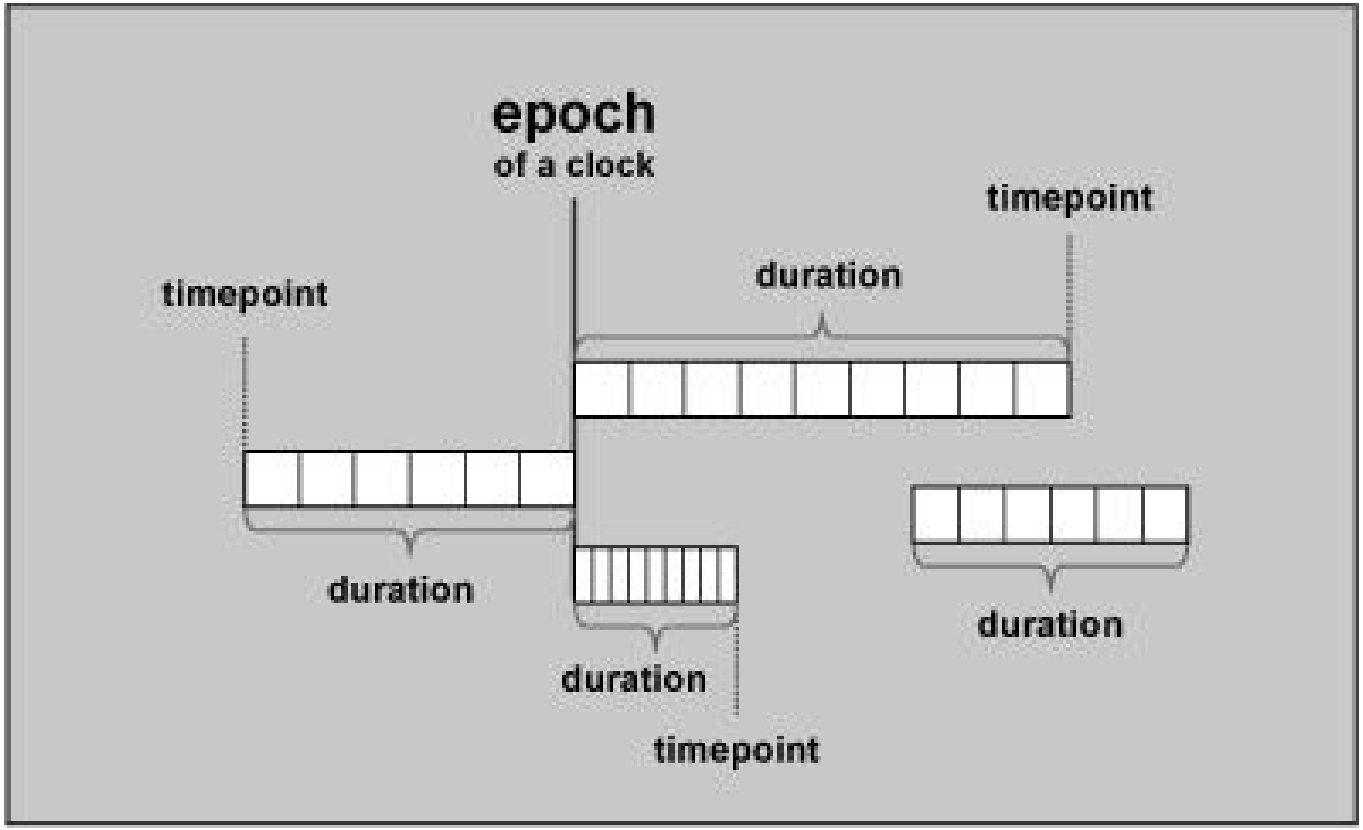
- Standardy C i POSIX biedne
- Precyzja (z upływem lat poprawiająca się)
  - W istniejących standardach przy zmienie precyzji (sekund -> milisekundy -> mikrosekundy -> nanosekundy) wprowadzane były nowe interfejsy

std::chrono

- Biblioteka czasu abstrahująca od precyzji
- Narzędzia do obsługi kalendarza
- Rozszerzenie do interfejsu wątków

Podstawowe byty

- duration
- clock - definiuje obiekt ‘epoch’ - początek czasu
- timepoint - kombinacja dwóch bytów
  - epoch
  - duration



Źródło: http://ptgmedia.pearsoncmg.com/images/chap5_9780321623218/elementLinks/05fig04_alt.jpg

std::chrono::duration

- Kombinacja
  - wartości reprezentującej liczbę tick’ów
  - ułamka reprezentującego jednostkę w sekundach

```
std::chrono::duration<int> twentySeconds(20);
std::chrono::duration<double,std::ratio<60>> halfAMinute(0.5);
std::chrono::duration<long,std::ratio<1,1000>> oneMillisecond(1);
```

std::chrono::duration c.d.

```
namespace std {
namespace chrono {
typedef duration<signed int-type >= 64 bits,nano>        nanoseconds;
typedef duration<signed int-type >= 55 bits,micro>       microseconds;
typedef duration<signed int-type >= 45 bits,milli>       milliseconds;
typedef duration<signed int-type >= 35 bits>             seconds;
typedef duration<signed int-type >= 29 bits,ratio<60>>   minutes;
typedef duration<signed int-type >= 23 bits,ratio<3600>> hours;
}
}
```

```
std::chrono::seconds      twentySeconds(20);
std::chrono::hours        aDay(24);
std::chrono::milliseconds oneMillisecond(1);
```

std::chrono::duration - operacje arytmetyczne

Wszystkie możliwe:

http://en.cppreference.com/w/cpp/chrono/duration

std::chrono::duration - c.d.

- Jednostki dwóch różnych instancji mogą być różne!

```
chrono::duration<int,ratio<1,3>> d1(1); // 1 tick of 1/3 second
chrono::duration<int,ratio<1,5>> d2(1); // 1 tick of 1/5 second

d1 + d2;                                // 8 ticks of 1/15
```

- *Off top - przy okazji polecam std::ratio!!!*

std::chrono::duration

```
std::chrono::seconds twentySeconds(20);   // 20 seconds
std::chrono::hours   aDay(24);            // 24 hours

std::chrono::milliseconds ms;             // 0 milliseconds
ms += twentySeconds + aDay;               // 86,400,000 milliseconds
--ms;                                     // 86,399,999 milliseconds
ms *= 2;                                  // 172,839,998 milliseconds
std::cout << ms.count() << " ms" << std::endl;
std::cout << std::chrono::nanoseconds(ms).count() << " ns" << std::endl;
```

Output:

```
172839998 ms
172839998000000 ns
```

Clock, timepoint

- **Clock** definiuje epoch i tick period
  - Np. liczba sekund od 1970-01-01,00:00:00
  - `Now()` – zwraca obecny timepoint
- **Timepoint** reprezentuje konkretny punkt w czasie powiązując **clock** oraz **duration**
  - Interfejs pozwala wyłuskać: epoch, minimum, maksimum, dokładność arytmetyki

Rodzaje zagarów

- std::chrono::system_clock
  - interejs dostarcza m.in.
    - `to_time_t()`
    - `from_time_t()`
- std::chrono::steady_clock
  - nigdy się nie cofa!
- std::chrono::high_resulotion_clock
  - reprezentuje zegar z najmniejszym tick’iem na jaki pozwala używany system
- Standard nie mówi nic o tym jak dokładne są te zegary

```
template <typename C>
void printClockData ()
{
    cout << "- precision: ";
    // if time unit is less or equal one millisecond
    typedef typename C::period P; // type of time unit
    if (ratio_less_equal<P,milli>::value) {
       // convert to and print as milliseconds
       typedef typename ratio_multiply<P,kilo>::type TT;
       cout << fixed << double(TT::num)/TT::den
            << " milliseconds" << endl;
    }
    else {
        // print as seconds
        cout << fixed << double(P::num)/P::den << " seconds" << endl;
    }
    cout << "- is_steady: " << boolalpha << C::is_steady << endl;
}
```

```
int main() {
    std::cout << "system_clock: " << std::endl;
    printClockData<std::chrono::system_clock>();
    std::cout << "\nhigh_resolution_clock: " << std::endl;
    printClockData<std::chrono::high_resolution_clock>();
    std::cout << "\nsteady_clock: " << std::endl;
    printClockData<std::chrono::steady_clock>();
}
```

system_clock:
- precision: 0.000100 milliseconds
- is_steady: false

high_resolution_clock:
- precision: 0.000100 milliseconds
- is_steady: true

steady_clock:
- precision: 1.000000 milliseconds
- is_steady: true

Timery

- `sleep_for()`
- `sleep_until()`
- `try_lock_for() - std::mutex`
- `try_lock_until()  - std::mutex`
- `wait_for() - std::condition_variable`
- `wait_until() - std::condition_variable`

```
this_thread::sleep_until(
   chrono::system_clock::now() + chrono::seconds(10));
```

- dla system_clock mogłoby zadziałać niespodziewanie!

LizardFS

Obiecane krótkie omówienie

`src/common/time_utils.cc`

Część 2. [De-]serializacja pakietów sieciowych, variadic templates

- MooseFS - goły, binarny protokół
  - Żadne thrifty ani inne takie :(
- LizardFS jest kompatybilny wstecznie

MooseFS, LizardFS

LizardFS:src/common/packet.h

```
// Legacy MooseFS packet format:
//
// type
// length of data
// data (type-dependent)
//
// New packet header contains an additional version field.
// For backwards compatibility, length indicates total size
// of remaining part of the packet, including version.
//
// type
// length of version field and data
// version
// data
```

MooseFS

`mfscommon/MFSCommunication.h` …

MooseFS - przykładowy pakiet c.d.

- Deklaracja:

```
// 0x01A0
#define CLTOMA_FUSE_MKNOD (PROTO_BASE+416)
// msgid:32 inode:32 name:NAME type:8 mode:16 uid:32 gid:32 rdev:32
```

MFS - przykładowy pakiet - serializacja

```
uint8_t *wptr;
const uint8_t *rptr;
wptr = fs_createpacket(rec,CLTOMA_FUSE_MKNOD,20+nleng); // malloc
put32bit(&wptr,parent);
put8bit(&wptr,nleng);
memcpy(wptr,name,nleng);
wptr+=nleng;
put8bit(&wptr,type);
put16bit(&wptr,mode);
put32bit(&wptr,uid);
put32bit(&wptr,gid);
put32bit(&wptr,rdev);
```

MFS - przykładowy pakiet - deserializacja

```
        if (length<24)
             syslog(LOG_NOTICE,"CLTOMA_FUSE_MKNOD - wrong size (%"PRIu32")",
length);
             [...]
        msgid = get32bit(&data);
        [...]
        nleng = get8bit(&data);
        if (length!=24U+nleng) {
             syslog(LOG_NOTICE,"CLTOMA_FUSE_MKNOD - wrong size [...]
             [...]
        name = data;
        data += nleng;
        type = get8bit(&data);
        mode = get16bit(&data);
```

MooseFS - smutno mi :(

```
alek@alek:~/src/moosefs-code$ egrep "([[:digit:]]+[[:blank:]]*\+){12,}" `CPPF`
./mfsmaster/matoclserv.c:
    uint8_t fsesrecord[43+SESSION_STATS*8];
    // 4+4+4+4+1+1+1+4+4+4+4+4+4+SESSION_STATS*4+SESSION_STATS*4
./mfsmaster/filesystem.c:
    uint8_t unodebuff[1+4+1+2+4+4+4+4+4+4+8+4+2+8*65536+4*65536+4];
./mfsmaster/filesystem.c:       if (fwrite(unodebuff,1,1+4+1+2+4+4+4+4+4+4+8+4+2,
fd)!=(size_t)(1+4+1+2+4+4+4+4+4+4+8+4+2)) {
./mfsmaster/filesystem.c:
    uint8_t unodebuff[4+1+2+4+4+4+4+4+4+8+4+2+8*65536+4*65536+4];
./mfsmetadump/mfsmetadump.c:
    uint8_t unodebuff[4+1+2+4+4+4+4+4+4+8+4+2+8*65536+4*65536+4];
```

MooseFS => LizardFS

```
void ChunkserverWriteChain::createInitialMessage(std::vector<uint8_t>& message,
    uint64_t chunkId, uint32_t version) {
-  size_t messageDataSize = 12 + 6 * (servers_.size() - 1);
-  message.resize(8 + messageDataSize); // 8 bytes of header + data
-
-  uint8_t* data = message.data();
-
-  // Message header
-  put32bit(&data, CLTOCS_WRITE);
-  put32bit(&data, messageDataSize);
-
-  // Message data: chunk ID, version and servers other then the first one
-  put64bit(&data, chunkId);
-  put32bit(&data, version);
-  for (size_t i = 1; i < servers_.size(); ++i) {
-    put32bit(&data, servers_[i].ip);
-    put16bit(&data, servers_[i].port);
-  }
+  serializeMooseFsPacket(message, CLTOCS_WRITE, chunkId, version,
+      makeSerializableRange(servers_.begin() + 1, servers_.end()));
}
```

LizardFS - c.d.

```
/ * Assembles a whole MooseFS packet (packet without version) */
template<class T, class... Data>
inline void serializeMooseFsPacket(std::vector<uint8_t>& buffer,
                const PacketHeader::Type& type,
                const T& t,
                const Data &...args) {
        uint32_t length = serializedSize(t, args...);
        serialize(buffer, type, length, t, args...);
}

inline void serializeMooseFsPacket(std::vector<uint8_t>& buffer,
                const PacketHeader::Type& type) {
        uint32_t length = 0;
        serialize(buffer, type, length);
}
```

LizardFS - c.d.

```
struct NetworkAddress {
     uint32_t ip;
     uint16_t port;
      [...]
};
inline uint32_t serializedSize(const NetworkAddress& server) {
     return serializedSize(server.ip, server.port);
}
inline void serialize(uint8_t** destination, const NetworkAddress& server) {
     return serialize(destination, server.ip, server.port);
}
inline void deserialize(const uint8_t** source, uint32_t& bytesLeftInBuffer,
          NetworkAddress& server) {
     deserialize(source, bytesLeftInBuffer, server.ip, server.port);
}
```

MooseFS - pytanie otwarte

Czy dało się to zrobić w cywilizowany sposób używając C?

- Generacja kodu sieciowego w innym języku?
- Generacja kodu w makrami?
- …?

Boost tuple

http://www.boost.org/doc/libs/1_40_0/boost/tuple/detail/tuple_basic.hpp

```
[...]

  // - tuple forward declaration
  ------------------------------------------------
  template <
    class T0 = null_type, class T1 = null_type, class T2 = null_type,
    class T3 = null_type, class T4 = null_type, class T5 = null_type,
    class T6 = null_type, class T7 = null_type, class T8 = null_type,
    class T9 = null_type>
  class tuple;

[...]
```

Variadic templates

```
template <typename... Ts>
class C {
    ...
};
template <typename... Ts>
void fun(const Ts&... vs) {
    ...
}
```

Nowy byt w języku - lista parametrów

```
typedef Ts MyList; // błąd!
Ts var;            // błąd!
auto copy = vs;    // błąd!
```

- `Ts` jest aliasem na listę typów
- `vs` jest aliasem na listę wartości
- Obie listy mogą być potencjalnie puste
- Na obu możemy wykonywać odpowiednie (różne) operacje

Użycie

- Można przyłożyć sizeof...

```
size_t items = sizeof...(Ts); // or vs
```

- Można rozwijać

```
template <typename... Ts>
void fun(Ts&&... vs) {
     gun(3.14, std::forward<Ts>(vs)..., 6.28);
}
```

- … I to w zasadzie tyle!

Zasady rozwijania

Użycie	Rozwinięcie
`Ts...`	`T1, . . . , Tn`
`Ts&&...`	`T1&&, . . . , Tn&&`
`x<Ts, Y>::z...`	`x<T1, Y>::z, . . . , x<Tn, Y>::z`
`x<Ts&, Us>...`	`x<T1&, U1>, . . . , x<Tn&, Un>`
`func(5, vs)...`	`func(5, v1), . . . , func(5, vn)`

Przypadki użycia

- Listy inicjalizacyjne

```
any a[] = { vs... };
```

- Dziedziczenie

```
template <typename... Ts>
struct C : Ts... {};

template <typename... Ts>
struct C : Box<Ts>... {};
```

- Listy inicjalizacyjne konstruktora

```
template <typename... Us> D(Us... vs) : Box<Ts>(vs)... {}
```

Przypadki użycia c.d.

- Argumenty template'ów

```
std::map<Ts...> m;
```

- Capture lists

```
template <class... Ts> void fun(Ts... vs) {
     auto g = [&vs...] { return gun(vs...); }
     g();
}
```

- Specyfikowanie wyjątków

```
template<class...X> void func(int arg) throw(X...)
```

:)

```
template <class... Ts> void fun(Ts... vs) {
    gun(A<Ts...>::hun(vs)...);
    gun(A<Ts...>::hun(vs...));
    gun(A<Ts>::hun(vs)...);
}
```

Jak używać variadic template?

- Tak jak zawsze template'ów - przez dopasowywanie wzorców!

```
template <class T1, class T2>
     bool isOneOf(T1&& a, T2&& b) {
     return a == b;
}

template <class T1, class T2, class... Ts>
     bool isOneOf(T1&& a, T2&& b, Ts&&... vs) {
     return a == b || isOneOf(a, vs...);
}

assert(isOneOf(1, 2, 3.5, 4, 1, 2));
```

Printf

Standardowy printf:

- Wydajny
- W miarę wygodny
- Dobrze znany
- Thread-safe
- Niebezpieczny!

Bezpieczny printf

- Chcemy zbudować funkcję:

```
template <typename... Ts>
int safe_printf(const char * f, const Ts&... ts)
```

- Zbudujmy metodę walidującą

```
template <typename... Ts>
int check_printf(const char * f, const Ts&... ts)
```

Bezpieczny printf c.d.

```
void check_printf(const char * f) {
        for (; *f; ++f) {
                if (*f != '%' || *++f == '%') continue;
                throw Exc("Bad format");
        }
}
```

Bezpieczny printf c.d.

```
template <class T, typename... Ts>
void check_printf(const char * f, const T& t,
      const Ts&... ts) {
   for (; *f; ++f) {
      if (*f != '%' || *++f == '%') continue;
      switch (*f) {
         default: throw Exc("Invalid format char: %", *f);
         case 'f': case 'g':
            ENFORCE(is_floating_point<T>::value);
            break;
         case 's': . . .
      }
      return check_printf(++f, ts...); // AHA!!!
   }
   throw Exc("Too few format specifiers.");
}
```

Bezpieczny printf - c.d.

```
template <typename... Ts>
int safe_printf(const char * f, const Ts&... ts) {
     check_printf(f, ts...);
     return printf(f, ts...);
}
```

- W internecie sporo implementacji: flagi, precyzja, dziwne zachowania (long jako adres itd.)
- Inne pokrewne funkcje, safe_scanf
- `#ifndef NDEBUG`

std::tuple

- “Produkt” pakujący dowolną liczbę heterogenicznych obiektów
- Generaliazacja std::pair
- Ułożenie w pamięci niewyspecyfikowane
  - Obecne implementacje nie robią tego mądrze

std::tuple - interfejs

Functions

Object creation

make_tuple - Construct tuple (function template)

forward_as_tuple - Forward as tuple (function template)

tie - Tie arguments to tuple elements (function template)

tuple_cat - Concatenate tuples (function template)

Element access

get - Get element (function template)

std::tuple - przykład użycia get<Int>

```
tuple<int, string, double> t;
static_assert(tuple_size<decltype(t)>::value == 3, "Rupture in the
Universe.");
get<0>(t) = 42;
assert(get<0>(t) == 42);
get<1>(t) = "forty-two";
get<2>(t) = 0.42;
```

std::tuple - implementacja

```
template <class... Ts> class tuple {};

template <class T, class... Ts>
class tuple<T, Ts...> : private tuple<Ts...> {
private:
     T head_;
     ...
};
```

std::tuple - typ get<N>

```
template <size_t, class> struct tuple_element;
template <class T, class... Ts>
struct tuple_element<0, tuple<T, Ts...>> {
    typedef T type;
};

template <size_t k, class T, class... Ts>
struct tuple_element<k, tuple<T, Ts...>> {
    typedef
        typename tuple_element<k-1, tuple<Ts...>>::type
        type;
};
```

std::tuple - implementacja get<N>

```
template <size_t k, class... Ts>
typename enable_if<k == 0,
    typename tuple_element<0, tuple<Ts...>>::type&>::type
get(tuple<Ts...>& t) {
    return t.head();
}

template <size_t k, class T, class... Ts>
typename enable_if<k != 0,
    typename tuple_element<k, tuple<T, Ts...>>::type&>::type
get(tuple<T, Ts...>& t) {
    tuple<Ts...> & super = t; // get must be friend
    return get<k - 1>(super);
}
```

std::<vector|...>::emplace

template< class... Args > iterator emplace(const_iterator pos, Args&&... args);

(since C++11)

Inserts a new element into the container directly before pos. The element is constructed in-place, i.e. no copy or move operations are performed. The constructor of the element is called with the arguments std::forward<Args>(args).... The element type must be EmplaceConstructible, MoveInsertable and MoveAssignable.

If the new size() is greater than capacity(), all iterators and references are invalidated. Otherwise, only the iterators and references before the insertion point remain valid. The past-the-end iterator is also invalidated.

std::vector::emplace

```
#ifdef __GXX_EXPERIMENTAL_CXX0X__
  template<typename _Tp, typename _Alloc>
    template<typename... _Args>
      void
      vector<_Tp, _Alloc>::
      emplace_back(_Args&&... __args)
      {
          if (this->_M_impl._M_finish != this->_M_impl._M_end_of_storage)
           {
             this->_M_impl.construct(this->_M_impl._M_finish,
                                     std::forward<_Args>(__args)...);
             ++this->_M_impl._M_finish;
           }
          else
           _M_insert_aux(end(), std::forward<_Args>(__args)...);
      }
#endif
```

Bibliografia

- Korzystałem bardzo intensywnie z wykładu A. Alexandrescu nt. variadic templates http://www.youtube.com/watch?v=_zgq6_zFNGY
  - Bardzo polecam prezentacje z konferencji ‘Go native’ na youtube!
- Draft standardu
- http://www.informit.com/articles/article.aspx?p=1881386&seqNum=2
- MooseFS.org
- LizardFS.org