⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit 64d1388

Browse files

stenzek committed 3 days ago Verified

```
Common: Add additional unit tests

And fix several logical errors.
```

1 parent f5191f5 commit 64d1388

Filter files...

- CMakeLists.txt
- CMakeModules
  - DuckStationBuildSum...
- src
  - common-tests
    - CMakeLists.txt
    - binary_reader_writer...
    - common-tests.vcxpr...
    - common-tests.vcxproj
    - heap_array_tests.cpp
    - small_string_tests.cpp
  - common
    - heap_array.h
    - small_string.cpp
    - small_string.h

11 files changed +3414 -66 lines changed

Search within code

+10

CMakeLists.txt

```
@@ -1,3 +1,13 @@
+ # SPDX-FileCopyrightText: 2019-2026 Connor McLaughlin <stenzek@gmail.com>
+ # SPDX-License-Identifier: CC-BY-NC-ND-4.0 + Packaging Restriction
+ #
+ # NOTE: In addition to the terms of CC-BY-NC-ND-4.0, you may not use this file to create
+ # packages or build recipes without explicit permission from the copyright holder.
+ #
+ # Unless otherwise specified, other files supporting the build system are covered under
+ # the same terms.
+ #
+
cmake_minimum_required(VERSION 3.19)
project(duckstation C CXX)
```

+14 -4

CMakeModules/DuckStationBuildSummary.cmake

```
@@ -26,13 +26,23 @@ if(BUILD_TESTS)
  message(STATUS "Building unit tests.")
endif()

- # Refuse to build in Arch package environments. My license does not allow for packages, and I'm sick of
- # dealing with people complaining about things broken by packagers. This is why we can't have nice things.
- if(DEFINED ENV{DEBUGINFOD_URLS})
-   if($ENV{DEBUGINFOD_URLS} MATCHES ".*archlinux.*")
+ # Refuse to build in hostile package environments. The code and build script licenses do not allow for
+ # packages, and I'm sick of dealing with people complaining about things broken by packagers, and then
+ # being attacked by package maintainers who violate their distribution's codes of conduct. Attempts to
+ # request removal of these packages have been unsuccessful, so we have to resort to this.
+ # NOTE: You do NOT have permission to distribute build scripts or patches that modify the build system
+ # without explicit permission from the copyright holder.
+ # DuckStation's code is public so it can be audited and learned from. Not to repackage.
+ # This is why we can't have nice things.
+ if(EXISTS /etc/os-release)
+   file(READ /etc/os-release OS_RELEASE_CONTENT)
+   if(OS_RELEASE_CONTENT MATCHES "ID=arch" OR OS_RELEASE_CONTENT MATCHES "ID_LIKE=arch" OR OS_RELEASE_CONTENT MATCHES "ID=nixos")
      message(FATAL_ERROR "Unsupported environment.")
    endif()
  endif()
+ if(DEFINED ENV{NIX_BUILD_TOP} OR DEFINED ENV{NIX_STORE} OR DEFINED ENV{IN_NIX_SHELL} OR EXISTS "/etc/NIXOS")
+   message(FATAL_ERROR "Unsupported environment.")
+ endif()

  if(DEFINED HOST_MIN_PAGE_SIZE AND DEFINED HOST_MAX_PAGE_SIZE)
    message(STATUS "Building with a dynamic page size of ${HOST_MIN_PAGE_SIZE} - ${HOST_MAX_PAGE_SIZE} bytes.")
```

+3

src/common-tests/CMakeLists.txt

```
@@ -1,11 +1,14 @@
  add_executable(common-tests
+   binary_reader_writer_tests.cpp
    bitutils_tests.cpp
    file_system_tests.cpp
    gsvector_tests.cpp
    gsvector_yuvtorgb_test.cpp
```

```
6   7       hash_tests.cpp
    8   +   heap_array_tests.cpp
7   9       path_tests.cpp
8   10      rectangle_tests.cpp
    11  +   small_string_tests.cpp
9   12      string_tests.cpp
10  13    )
11  14
```

+868

src/common-tests/binary_reader_writer_tests.cpp

Load Diff

Large diffs are not rendered by default.

+3

src/common-tests/common-tests.vcxproj

```
    @@ -3,12 +3,15 @@
3   3     <Import Project="..\..\dep\msvc\vsprops\Configurations.props" />
4   4     <ItemGroup>
5   5       <ClCompile Include="..\..\dep\googletest\src\gtest_main.cc" />
    6   +   <ClCompile Include="binary_reader_writer_tests.cpp" />
6   7       <ClCompile Include="bitutils_tests.cpp" />
7   8       <ClCompile Include="file_system_tests.cpp" />
8   9       <ClCompile Include="gsvector_tests.cpp" />
    10  +   <ClCompile Include="heap_array_tests.cpp" />
9   11      <ClCompile Include="path_tests.cpp" />
10  12      <ClCompile Include="rectangle_tests.cpp" />
11  13      <ClCompile Include="hash_tests.cpp" />
    14  +   <ClCompile Include="small_string_tests.cpp" />
12  15      <ClCompile Include="string_tests.cpp" />
13  16      <ClCompile Include="gsvector_yuvtorgb_test.cpp" />
14  17    </ItemGroup>
```

+3

src/common-tests/common-tests.vcxproj.filters

```
    @@ -10,5 +10,8 @@
10  10      <ClCompile Include="gsvector_yuvtorgb_test.cpp" />
11  11      <ClCompile Include="hash_tests.cpp" />
12  12      <ClCompile Include="gsvector_tests.cpp" />
    13  +   <ClCompile Include="small_string_tests.cpp" />
    14  +   <ClCompile Include="binary_reader_writer_tests.cpp" />
    15  +   <ClCompile Include="heap_array_tests.cpp" />
13  16    </ItemGroup>
14  17    </Project>
```

+827

src/common-tests/heap_array_tests.cpp

Load Diff

Large diffs are not rendered by default.

+1,529

src/common-tests/small_string_tests.cpp

Load Diff

Large diffs are not rendered by default.

+90 -38

src/common/heap_array.h

```
    @@ -13,6 +13,9 @@
13  13    template<typename T, std::size_t SIZE, std::size_t ALIGNMENT = 0>
14  14    class FixedHeapArray
15  15    {
    16  +   static_assert(std::is_trivially_copyable_v<T>, "T is trivially copyable");
    17  +   static_assert(std::is_standard_layout_v<T>, "T is standard layout");
    18  +
16  19    public:
17  20      using value_type = T;
18  21      using size_type = std::size_t;
    @@ -73,12 +76,12 @@ class FixedHeapArray
73  76
74  77      void swap(this_type& move) { std::swap(m_data, move.m_data); }
75  78
76      -   std::span<T, SIZE> span() { return std::span<T, SIZE>(m_data); }
77      -   std::span<const T, SIZE> cspan() const { return std::span<const T, SIZE>(m_data); }
```

```
79   +   std::span<T, SIZE> span() { return std::span<T,
         SIZE>(m_data, m_data + SIZE); }
80   +   std::span<const T, SIZE> cspan() const { return
         std::span<const T, SIZE>(m_data, m_data + SIZE); }
78   81
79   82      this_type& operator=(const this_type& rhs)
80   83      {
81       -     std::copy(begin(), end(), rhs.cbegin());
     84   +     std::copy(rhs.cbegin(), rhs.cend(), begin());
82   85        return *this;
83   86      }
84   87
          @@ -90,24 +93,12 @@ class FixedHeapArray
90   93        return *this;
91   94      }
92   95
93       - #define RELATIONAL_OPERATOR(op)
         \
94       -   bool operator op(const this_type& rhs) const
         \
95       -   {
         \
96       -     for (size_type i = 0; i < SIZE; i++)
         \
97       -     {
         \
98       -       if (!(m_data[i] op rhs.m_data[i]))
         \
99       -         return false;
         \
100      -     }
         \
101      -   }
102      -
103      -   RELATIONAL_OPERATOR(==);
104      -   RELATIONAL_OPERATOR(!=);
105      -   RELATIONAL_OPERATOR(<);
106      -   RELATIONAL_OPERATOR(<=);
107      -   RELATIONAL_OPERATOR(>);
108      -   RELATIONAL_OPERATOR(>=);
109      -
110      - #undef RELATIONAL_OPERATOR
     96   +   bool operator==(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) ==
         0); }
     97   +   bool operator!=(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) !=
         0); }
     98   +   bool operator<(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) <
         0); }
     99   +   bool operator<=(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) <=
         0); }
     100  +   bool operator>(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) <
         0); }
     101  +   bool operator>=(const this_type& rhs) const { return
         (std::memcmp(m_data, rhs.m_data, SIZE * sizeof(T)) >=
         0); }
111  102
112  103     private:
113  104       void allocate()
          @@ -372,26 +363,87 @@ class DynamicHeapArray
372  363       return *this;
373  364     }
374  365
375      - #define RELATIONAL_OPERATOR(op, size_op)
         \
376      -   bool operator op(const this_type& rhs) const
         \
377      -   {
         \
378      -     if (m_size != rhs.m_size)
         \
379      -       return m_size size_op rhs.m_size;
         \
380      -     for (size_type i = 0; i < m_size; i++)
         \
381      -     {
         \
382      -       if (!(m_data[i] op rhs.m_data[i]))
         \
383      -         return false;
         \
384      -     }
         \
     366  +   bool operator==(const this_type& rhs) const
     367  +   {
     368  +     if (m_size != rhs.m_size)
     369  +       return false;
     370  +
     371  +     if (m_size == 0)
     372  +       return true;
     373  +
     374  +     return (std::memcmp(m_data, rhs.m_data, m_size *
         sizeof(T)) == 0);
     375  +   }
     376  +
     377  +   bool operator!=(const this_type& rhs) const
     378  +   {
     379  +     if (m_size != rhs.m_size)
     380  +       return true;
```

```
381  +
382  +     if (m_size == 0)
383  +       return false;
384  +
385  +     return (std::memcmp(m_data, rhs.m_data, m_size *
       sizeof(T)) != 0);
386  +   }
387  +
388  +   bool operator<(const this_type& rhs) const
389  +   {
390  +     const size_type min_size = std::min(m_size,
       rhs.m_size);
391  +     for (size_type i = 0; i < min_size; i++)
392  +     {
393  +       if (!(m_data[i] < rhs.m_data[i]))
394  +         return false;
395  +     }
396  +
397  +     if (m_size != rhs.m_size)
398  +       return m_size < rhs.m_size;
399  +
400  +     return true;
401  +   }
402  +
403  +   bool operator<=(const this_type& rhs) const
404  +   {
405  +     const size_type min_size = std::min(m_size,
       rhs.m_size);
406  +     for (size_type i = 0; i < min_size; i++)
407  +     {
408  +       if (!(m_data[i] <= rhs.m_data[i]))
409  +         return false;
410  +     }
411  +
412  +     if (m_size != rhs.m_size)
413  +       return m_size <= rhs.m_size;
414  +
415  +     return true;
416  +   }
417  +
418  +   bool operator>(const this_type& rhs) const
419  +   {
420  +     const size_type min_size = std::min(m_size,
       rhs.m_size);
421  +     for (size_type i = 0; i < min_size; i++)
422  +     {
423  +       if (!(m_data[i] > rhs.m_data[i]))
424  +         return false;
425  +     }
426  +
427  +     if (m_size != rhs.m_size)
428  +       return m_size > rhs.m_size;
429  +
430  +     return true;
431       }
432
387  -   RELATIONAL_OPERATOR(==, !=);
388  -   RELATIONAL_OPERATOR(!=, ==);
389  -   RELATIONAL_OPERATOR(<, <);
390  -   RELATIONAL_OPERATOR(<=, <=);
391  -   RELATIONAL_OPERATOR(>, >);
392  -   RELATIONAL_OPERATOR(>=, >=);
433  +   bool operator>=(const this_type& rhs) const
434  +   {
435  +     const size_type min_size = std::min(m_size,
       rhs.m_size);
436  +     for (size_type i = 0; i < min_size; i++)
437  +     {
438  +       if (!(m_data[i] >= rhs.m_data[i]))
439  +         return false;
440  +     }
441
394  -  #undef RELATIONAL_OPERATOR
442  +     if (m_size != rhs.m_size)
443  +       return m_size >= rhs.m_size;
444  +
445  +     return true;
446  +   }
395  447
396  448    private:
397  449      void internal_resize(size_t size, T* prev_ptr,
         [[maybe_unused]] size_t prev_size)
```

+51 -20

src/common/small_string.cpp

```
     @@ -86,7 +86,7 @@ void
     SmallStringBase::reserve(u32 new_reserve)
86   86        m_on_heap = true;
87   87      }
88   88
89      -   m_buffer_size = new_reserve;
        89  +   m_buffer_size = real_reserve;
90   90    }
91   91
92   92    void SmallStringBase::shrink_to_fit()
     @@ -100,6 +100,7 @@ void
     SmallStringBase::shrink_to_fit()
100  100       std::free(m_buffer);
101  101       m_buffer = nullptr;
102  102       m_buffer_size = 0;
     103  +    m_on_heap = false;
103  104       return;
104  105     }
```

```
105  106
     @@ -130,7 +131,7 @@ std::string_view
     SmallStringBase::view() const
130  131
131  132  SmallStringBase& SmallStringBase::operator=
     (SmallStringBase&& move)
132  133  {
133       -    assign(move);
     134  +    assign(std::move(move));
134  135      return *this;
135  136  }
136  137
     @@ -160,15 +161,16 @@ SmallStringBase&
     SmallStringBase::operator=(const
     SmallStringBase& copy)
160  161
161  162  void SmallStringBase::make_room_for(u32 space)
162  163  {
163       -    const u32 required_size = m_length + space + 1;
164       -    if (m_buffer_size >= required_size)
     164  +    const u32 required_length = m_length + space;
     165  +    if (m_buffer_size > required_length)
165  166        return;
166  167
167       -    reserve(std::max(required_size, m_buffer_size *
     2));
     168  +    reserve(std::max(required_length, m_buffer_size *
     2));
168  169  }
169  170
170  171  void SmallStringBase::append(const char* str, u32
     length)
171  172  {
     173  +    DebugAssert(str != m_buffer); // appending self
     is not allowed
172  174      if (length == 0)
173  175        return;
174  176
     @@ -215,13 +217,16 @@ void
     SmallStringBase::append_hex(const void* data,
     size_t len, bool comma_separa
215  217          m_buffer[m_length++] = hex_char(bytes[i] &
     0xF);
216  218        }
217  219      }
     220  +
     221  +    m_buffer[m_length] = '\0';
218  222  }
219  223
220  224  void SmallStringBase::prepend(const char* str, u32
     length)
221  225  {
222  226      if (length == 0)
223  227        return;
224  228
     229  +    DebugAssert(str != m_buffer); // appending self
     is not allowed
225  230      make_room_for(length);
226  231
227  232      DebugAssert((length + m_length) < m_buffer_size);
     @@ -239,11 +244,13 @@ void
     SmallStringBase::append(char c)
239  244
240  245  void SmallStringBase::append(const SmallStringBase&
     str)
241  246  {
     247  +    DebugAssert(&str != this); // appending self is
     not allowed
242  248      append(str.m_buffer, str.m_length);
243  249  }
244  250
245  251  void SmallStringBase::append(const char* str)
246  252  {
     253  +    DebugAssert(str != m_buffer); // appending self
     is not allowed
247  254      append(str, static_cast<u32>(std::strlen(str)));
248  255  }
249  256
     @@ -254,6 +261,7 @@ void
     SmallStringBase::append(const std::string& str)
254  261
255  262  void SmallStringBase::append(const std::string_view
     str)
256  263  {
     264  +    DebugAssert(str.data() != m_buffer); // appending
     self is not allowed
257  265      append(str.data(), static_cast<u32>
     (str.length()));
258  266  }
259  267
     @@ -307,11 +315,13 @@ void
     SmallStringBase::prepend(char c)
307  315
308  316  void SmallStringBase::prepend(const
     SmallStringBase& str)
309  317  {
     318  +    DebugAssert(&str != this); // prepending self is
     not allowed
310  319      prepend(str.m_buffer, str.m_length);
311  320  }
312  321
313  322  void SmallStringBase::prepend(const char* str)
314  323  {
```

```diff
+  DebugAssert(str != m_buffer); // prepending self is not allowed
   prepend(str, static_cast<u32>(std::strlen(str)));
 }
 
@@ -322,6 +332,7 @@ void SmallStringBase::prepend(const std::string& str)
 
 void SmallStringBase::prepend(const std::string_view str)
 {
+  DebugAssert(str.data() != m_buffer); // prepending self is not allowed
   prepend(str.data(), static_cast<u32>(str.length()));
 }
 
@@ -345,7 +356,10 @@ void SmallStringBase::prepend_vsprintf(const char* format, va_list ArgPtr)
 
   for (;;)
   {
-    int ret = std::vsnprintf(buffer, buffer_size, format, ArgPtr);
+    std::va_list ap_copy;
+    va_copy(ap_copy, ArgPtr);
+    int ret = std::vsnprintf(buffer, buffer_size, format, ap_copy);
+    va_end(ap_copy);
     if (ret < 0 || (static_cast<u32>(ret) >= (buffer_size - 1)))
     {
       buffer_size *= 2;
@@ -367,11 +381,13 @@ void SmallStringBase::prepend_vsprintf(const char* format, va_list ArgPtr)
 
 void SmallStringBase::insert(s32 offset, const char* str)
 {
+  DebugAssert(str != m_buffer); // inserting self is not allowed
   insert(offset, str, static_cast<u32>(std::strlen(str)));
 }
 
 void SmallStringBase::insert(s32 offset, const SmallStringBase& str)
 {
+  DebugAssert(&str != this); // inserting self is not allowed
   insert(offset, str, str.m_length);
 }
 
@@ -393,7 +409,7 @@ void SmallStringBase::insert(s32 offset, const char* str, u32 length)
   DebugAssert(real_offset <= m_length);
   const u32 chars_after_offset = m_length - real_offset;
   if (chars_after_offset > 0)
-    std::memmove(m_buffer + offset + length, m_buffer + offset, chars_after_offset);
+    std::memmove(m_buffer + real_offset + length, m_buffer + real_offset, chars_after_offset);
 
   // insert the string
   std::memcpy(m_buffer + real_offset, str, length);
@@ -410,6 +426,7 @@ void SmallStringBase::insert(s32 offset, const std::string& str)
 
 void SmallStringBase::insert(s32 offset, const std::string_view str)
 {
+  DebugAssert(str.data() != m_buffer); // inserting self is not allowed
   insert(offset, str.data(), static_cast<u32>(str.size()));
 }
 
@@ -497,6 +514,7 @@ void SmallStringBase::assign(const std::wstring_view wstr)
   }
 
   m_length = static_cast<u32>(mblen);
+  m_buffer[m_length] = '\0';
 }
 
 std::wstring SmallStringBase::wstring() const
@@ -572,7 +590,8 @@ bool SmallStringBase::iequals(const char* otherText) const
 
 bool SmallStringBase::iequals(const SmallStringBase& str) const
 {
-  return (m_length == str.m_length && (m_length == 0 || std::strcmp(m_buffer, str.m_buffer) == 0));
+  return (m_length == str.m_length &&
+          (m_length == 0 || StringUtil::Strncasecmp(m_buffer, str.m_buffer,
```

```
                m_length) == 0));
576  595      }
577  596
578  597      bool SmallStringBase::iequals(const
              std::string_view str) const
          @@ -771,6 +790,9 @@ bool
          SmallStringBase::ends_with(const std::string&
          str, bool case_sensitive) con
771  790
772  791      void SmallStringBase::clear()
773  792      {
     793  +     if (m_buffer_size == 0)
     794  +       return;
     795  +
774  796        // in debug, zero whole string, in release, zero
              only the first character
775  797      #if _DEBUG
776  798        std::memset(m_buffer, 0, m_buffer_size);
          @@ -823,6 +845,9 @@ u32
          SmallStringBase::count(char ch) const
823  845      u32 SmallStringBase::replace(const char* search,
              const char* replacement)
824  846      {
825  847        const u32 search_length = static_cast<u32>
              (std::strlen(search));
     848  +     if (search_length == 0)
     849  +       return 0;
     850  +
826  851        const u32 replacement_length = static_cast<u32>
              (std::strlen(replacement));
827  852
828  853        s32 offset = 0;
          @@ -833,17 +858,19 @@ u32
          SmallStringBase::replace(const char* search,
          const char* replacement)
833  858        if (offset < 0)
834  859          break;
835  860
     861  +     const u32 chars_after_offset = (m_length -
              static_cast<u32>(offset));
     862  +     DebugAssert(chars_after_offset >=
              search_length);
     863  +
836  864        const u32 new_length = m_length - search_length
              + replacement_length;
837  865        reserve(new_length);
838  866        m_length = new_length;
839  867
840       -     const u32 chars_after_offset = (m_length -
              static_cast<u32>(offset));
841       -     DebugAssert(chars_after_offset >=
              search_length);
842  868        if (chars_after_offset > search_length)
843  869        {
844  870          std::memmove(&m_buffer[static_cast<u32>
              (offset) + replacement_length],
845  871                       &m_buffer[static_cast<u32>
              (offset) + search_length], chars_after_offset -
              search_length);
846  872          std::memcpy(&m_buffer[static_cast<u32>
              (offset)], replacement, replacement_length);
     873  +       m_buffer[m_length] = '\0';
847  874        }
848  875        else
849  876        {
          @@ -861,22 +888,26 @@ u32
          SmallStringBase::replace(const char* search,
          const char* replacement)
861  888
862  889    void SmallStringBase::resize(u32 new_size, char
              fill, bool shrink_if_smaller)
863  890    {
864       -   // if going larger, or we don't own the buffer,
              realloc
865       -   if (new_size >= m_buffer_size)
     891  +   if (new_size > m_length)
866  892      {
     893  +     // expanding - ensure we have space
867  894        reserve(new_size);
868  895
869       -     if (m_length < new_size)
870       -     {
871       -       std::memset(m_buffer + m_length, fill,
              m_buffer_size - m_length - 1);
872       -     }
873       -
     896  +     // fill the expanded area with the fill
              character
     897  +     std::memset(m_buffer + m_length, fill, new_size
              - m_length);
874  898        m_length = new_size;
     899  +
     900  + #ifdef _DEBUG
     901  +     // zero remaining unused buffer in debug
     902  +     std::memset(m_buffer + m_length, 0,
              m_buffer_size - new_size);
     903  + #else
     904  +     m_buffer[m_length] = 0;
     905  + #endif
875  906      }
876  907      else
877  908      {
878       -     // update length and terminator
879       - #if _DEBUG
```

```
       909   +     // shrinking or same size - update length and
                   terminator
       910   + #ifdef _DEBUG
880    911         std::memset(m_buffer + new_size, 0,
                   m_buffer_size - new_size);
881    912       #else
882    913         m_buffer[new_size] = 0;
           @@ -975,7 +1006,7 @@ void
           SmallStringBase::erase(s32 offset, s32 count)
975    1006        const u32 after_erase_block = m_length -
                   real_offset - real_count;
976    1007        DebugAssert(after_erase_block > 0);
977    1008
978          -     std::memmove(m_buffer + offset, m_buffer +
                   real_offset + real_count, after_erase_block);
       1009  +     std::memmove(m_buffer + real_offset, m_buffer +
                   real_offset + real_count, after_erase_block);
979    1010        m_length = m_length - real_count;
980    1011
981    1012      #ifdef _DEBUG
```

src/common/small_string.h +16 -4

```
           @@ -184,6 +184,9 @@ class SmallStringBase
184    184       // returns the end of the string (pointer is past
                 the last character)
185    185       ALWAYS_INLINE const char* end_ptr() const { return
                 m_buffer + m_length; }
186    186
       187   +   // returns true if the string is heap-allocated
       188   +   ALWAYS_INLINE bool is_heap_allocated() const {
                 return m_on_heap; }
       189   +
187    190       // STL adapters
188    191       ALWAYS_INLINE char& front() { return m_buffer[0]; }
189    192       ALWAYS_INLINE const char& front() const { return
                 m_buffer[0]; }
           @@ -287,7 +290,7 @@ class SmallStackString :
           public SmallStringBase
287    290       ALWAYS_INLINE SmallStackString(SmallStringBase&&
                 move)
288    291       {
289    292         init();
290          -     assign(move);
       293   +     move_assign(std::move(move));
291    294       }
292    295
293    296       ALWAYS_INLINE explicit SmallStackString(const
                 SmallStackString& copy)
           @@ -299,7 +302,7 @@ class SmallStackString :
           public SmallStringBase
299    302       ALWAYS_INLINE explicit
                 SmallStackString(SmallStackString&& move)
300    303       {
301    304         init();
302          -     assign(move);
       305   +     move_assign(std::move(move));
303    306       }
304    307
305    308       ALWAYS_INLINE explicit SmallStackString(const
                 std::string& str)
           @@ -322,7 +325,7 @@ class SmallStackString :
           public SmallStringBase
322    325
323    326       ALWAYS_INLINE SmallStackString& operator=
                 (SmallStringBase&& move)
324    327       {
325          -     assign(move);
       328   +     move_assign(std::move(move));
326    329         return *this;
327    330       }
328    331
           @@ -334,7 +337,7 @@ class SmallStackString :
           public SmallStringBase
334    337
335    338       ALWAYS_INLINE SmallStackString& operator=
                 (SmallStackString&& move)
336    339       {
337          -     assign(move);
       340   +     move_assign(std::move(move));
338    341         return *this;
339    342       }
340    343
           @@ -378,6 +381,15 @@ class SmallStackString :
           public SmallStringBase
378    381         m_stack_buffer[0] = '\0';
379    382     #endif
380    383       }
       384   +
       385   +   ALWAYS_INLINE void move_assign(SmallStringBase&&
                 move)
       386   +   {
       387   +     // only move if on the heap, otherwise copy
       388   +     if (move.is_heap_allocated())
       389   +       SmallStringBase::assign(std::move(move));
       390   +     else
       391   +       assign(move.data(), move.length());
       392   +   }
381    393     };
382    394
383    395     #ifdef _MSC_VER
```

Comments 0

Please sign in to comment.