shmemvv

0.1

Generated by Doxygen 1.11.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 test_options Struct Reference

Struct to hold selected tests options.

`#include <shmemvv.hpp>`

Public Member Functions

- test_options ()
  Constructor to initialize all flags to false.

Public Attributes

- bool test_all
- bool test_setup
- bool test_threads
- bool test_mem
- bool test_teams
- bool test_ctx
- bool test_remote
- bool test_atomics
- bool test_signaling
- bool test_collectives
- bool test_pt2pt_synch
- bool test_mem_ordering
- bool test_locking
- bool help

3.1.1 Detailed Description

Struct to hold selected tests options.

Definition at line 44 of file shmemvv.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 test_options()

```
test_options::test_options ()  [inline]
```

Constructor to initialize all flags to false.

Definition at line 63 of file shmemvv.hpp.

```
00063                     :
00064     test_all(false), test_setup(false), test_threads(false),
00065     test_mem(false), test_teams(false), test_ctx(false),
00066     test_remote(false), test_atomics(false), test_signaling(false),
00067     test_collectives(false), test_pt2pt_synch(false),
00068     test_mem_ordering(false), test_locking(false), help(false) {}
```

3.1.3 Member Data Documentation

3.1.3.1 help

```
bool test_options::help
```

Flag to display help message

Definition at line 58 of file shmemvv.hpp.

3.1.3.2 test_all

```
bool test_options::test_all
```

Flag to run all tests

Definition at line 45 of file shmemvv.hpp.

3.1.3.3 test_atomics

```
bool test_options::test_atomics
```

Flag to run atomic memory operations tests

Definition at line 52 of file shmemvv.hpp.

3.1.3.4 test_collectives

```
bool test_options::test_collectives
```

Flag to run collective operations tests

Definition at line 54 of file shmemvv.hpp.

3.1.3.5 test_ctx

```
bool test_options::test_ctx
```

Flag to run communication management tests

Definition at line 50 of file shmemvv.hpp.

3.1.3.6 test_locking

```
bool test_options::test_locking
```

Flag to run distributed locking tests

Definition at line 57 of file shmemvv.hpp.

3.1.3.7 test_mem

```
bool test_options::test_mem
```

Flag to run memory management tests

Definition at line 48 of file shmemvv.hpp.

3.1.3.8 test_mem_ordering

```
bool test_options::test_mem_ordering
```

Flag to run memory ordering tests

Definition at line 56 of file shmemvv.hpp.

3.1.3.9 test_pt2pt_synch

```
bool test_options::test_pt2pt_synch
```

Flag to run point-to-point synchronization tests

Definition at line 55 of file shmemvv.hpp.

3.1.3.10 test_remote

```
bool test_options::test_remote
```

Flag to run remote memory access tests

Definition at line 51 of file shmemvv.hpp.

3.1.3.11 test_setup

```
bool test_options::test_setup
```

Flag to run setup tests

Definition at line 46 of file shmemvv.hpp.

3.1.3.12 test_signaling

```
bool test_options::test_signaling
```

Flag to run signaling operations tests

Definition at line 53 of file shmemvv.hpp.

3.1.3.13 test_teams

```
bool test_options::test_teams
```

Flag to run team management tests

Definition at line 49 of file shmemvv.hpp.

3.1.3.14 test_threads

```
bool test_options::test_threads
```

Flag to run thread support tests

Definition at line 47 of file shmemvv.hpp.

The documentation for this struct was generated from the following file:

- src/include/shmemvv.hpp

Chapter 4

File Documentation

4.1 src/include/routines.hpp File Reference

Contains routine pointers to avoid compiler errors if a routine is not implemented.

```
#include <shmem.h>
```

Typedefs

- typedef void(* shmem_fake_routine_func) (void)
- typedef void(* shmem_init_func) (void)
- typedef void(* shmem_finalize_func) (void)
- typedef int(* shmem_my_pe_func) (void)
- typedef int(* shmem_n_pes_func) (void)
- typedef int(* shmem_pe_accessible_func) (int pe)
- typedef void(* shmem_barrier_all_func) (void)
- typedef void(* shmem_barrier_func) (int PE_start, int logPE_stride, int PE_size, long *pSync)
- typedef void(* shmem_info_get_version_func) (int *major, int *minor)
- typedef void(* shmem_info_get_name_func) (char *name)
- typedef void(* shmem_global_exit_func) (int status)
- typedef int(* shmem_init_thread_func) (int requested, int *provided)
- typedef int(* shmem_query_thread_func) (int *provided)
- typedef void *(* shmem_ptr_func) (const void *dest, int pe)
- typedef void *(* shmem_malloc_func) (size_t size)
- typedef void(* shmem_free_func) (void *ptr)
- typedef void *(* shmem_realloc_func) (void *ptr, size_t size)
- typedef void *(* shmem_align_func) (size_t alignment, size_t size)
- typedef void *(* shmem_malloc_with_hints_func) (size_t size, long hints)
- typedef void *(* shmem_calloc_func) (size_t count, size_t size)
- typedef int(* shmem_addr_accessible_func) (const void *addr, int pe)
- typedef int(* shmem_team_my_pe_func) (shmem_team_t team)
- typedef int(* shmem_team_n_pes_func) (shmem_team_t team)
- typedef void(* shmem_team_get_config_func) (shmem_team_t team, long config_mask, shmem_team_↩ config_t *config)
- typedef int(* shmem_team_translate_pe_func) (shmem_team_t src_team, int src_pe, shmem_team_t dest↩ _team)

- typedef shmem_team_t(* shmem_team_split_strided_func) (shmem_team_t parent_team, int start, int stride, int size, const shmem_team_config_t *config, long config_mask, shmem_team_t *new_team)
- typedef shmem_team_t(* shmem_team_split_2d_func) (shmem_team_t parent_team, int xrange, const shmem_team_config_t *xaxis_config, long xaxis_mask, shmem_team_t *xaxis_team, const shmem_team_config_t *yaxis_config, long yaxis_mask, shmem_team_t *yaxis_team)
- typedef void(* shmem_team_destroy_func) (shmem_team_t team)
- typedef int(* shmem_ctx_create_func) (long options, shmem_ctx_t *ctx)
- typedef int(* shmem_team_create_ctx_func) (shmem_team_t team, long options, shmem_ctx_t *ctx)
- typedef void(* shmem_ctx_destroy_func) (shmem_ctx_t ctx)
- typedef int(* shmem_ctx_get_team_func) (shmem_ctx_t ctx, shmem_team_t *team)
- typedef void(* shmem_long_put_func) (long *dest, const long *src, size_t nelems, int pe)
- typedef void(* shmem_long_p_func) (long *dest, long value, int pe)
- typedef void(* shmem_long_iput_func) (long *dest, const long *src, ptrdiff_t tst, ptrdiff_t sst, size_t nelems, int pe)
- typedef void(* shmem_long_get_func) (long *dest, const long *src, size_t nelems, int pe)
- typedef long(* shmem_long_g_func) (const long *src, int pe)
- typedef void(* shmem_long_iget_func) (long *dest, const long *src, ptrdiff_t tst, ptrdiff_t sst, size_t nelems, int pe)
- typedef void(* shmem_long_put_nbi_func) (long *dest, const long *src, size_t nelems, int pe)
- typedef void(* shmem_long_get_nbi_func) (long *dest, const long *src, size_t nelems, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_func) (const unsigned long *target, int pe)
- typedef void(* shmem_ulong_atomic_set_func) (unsigned long *target, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_compare_swap_func) (unsigned long *target, unsigned long cond, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_swap_func) (unsigned long *target, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_inc_func) (const unsigned long *target, int pe)
- typedef void(* shmem_ulong_atomic_inc_func) (unsigned long *target, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_add_func) (const unsigned long *target, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_add_func) (const unsigned long *target, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_and_func) (unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_and_func) (unsigned long *dest, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_or_func) (unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_or_func) (unsigned long *dest, unsigned long value, int pe)
- typedef unsigned long(* shmem_ulong_atomic_fetch_xor_func) (unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_xor_func) (unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_fetch_nbi_func) (unsigned long *dest, const unsigned long *target, int pe)
- typedef void(* shmem_ulong_atomic_compare_swap_nbi_func) (unsigned long *dest, unsigned long *target, unsigned long cond, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_swap_nbi_func) (unsigned long *dest, unsigned long *target, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_fetch_inc_nbi_func) (unsigned long *dest, const unsigned long *target, int pe)
- typedef void(* shmem_ulong_atomic_fetch_add_nbi_func) (unsigned long *dest, const unsigned long *target, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_fetch_and_nbi_func) (unsigned long *fetch, unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_fetch_or_nbi_func) (unsigned long *fetch, unsigned long *dest, unsigned long value, int pe)
- typedef void(* shmem_ulong_atomic_fetch_xor_nbi_func) (unsigned long *fetch, unsigned long *dest, unsigned long value, int pe)

- typedef void(* shmem_long_put_signal_func) (long *dest, const long *source, size_t nelems, uint64_t *sig_addr, uint64_t signal, int sig_op, int pe)
- typedef void(* shmem_long_put_signal_nbi_func) (long *dest, const long *source, size_t nelems, uint64_t *sig_addr, uint64_t signal, int sig_op, int pe)
- typedef long(* shmem_signal_fetch_func) (const uint64_t *sig_addr)
- typedef int(* shmem_sync_func) (int PE_start, int logPE_stride, int PE_size, long *pSync)
- typedef void(* shmem_sync_all_func) (void)
- typedef int(* shmem_long_alltoall_func) (shmem_team_t team, long *dest, const long *source, size_t nelems)
- typedef int(* shmem_long_alltoalls_func) (shmem_team_t team, long *dest, const long *source, ptrdiff_t dst, ptrdiff_t sst, size_t nelems)
- typedef int(* shmem_long_broadcast_func) (shmem_team_t team, long *dest, const long *source, size_t nelems, int PE_root)
- typedef int(* shmem_long_collect_func) (shmem_team_t team, long *dest, const long *source, size_t nelems)
- typedef int(* shmem_long_fcollect_func) (shmem_team_t team, long *dest, const long *source, size_t nelems)
- typedef int(* shmem_long_and_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_or_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_xor_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_max_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_min_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_sum_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef int(* shmem_long_prod_reduce_func) (shmem_team_t team, long *dest, const long *source, size_t nreduce)
- typedef void(* shmem_long_wait_until_func) (long *ivar, int cmp, long cmp_value)
- typedef void(* shmem_long_wait_until_all_func) (long *ivars, size_t nelems, const int *status, int cmp, long cmp_value)
- typedef size_t(* shmem_long_wait_until_any_func) (long *ivars, size_t nelems, const int *status, int cmp, long cmp_value)
- typedef size_t(* shmem_long_wait_until_some_func) (long *ivars, size_t nelems, size_t *indices, const int *status, int cmp, long cmp_value)
- typedef void(* shmem_long_wait_until_all_vector_func) (long *ivars, size_t nelems, const int *status, int cmp, long *cmp_values)
- typedef size_t(* shmem_long_wait_until_any_vector_func) (long *ivars, size_t nelems, const int *status, int cmp, long *cmp_values)
- typedef size_t(* shmem_long_wait_until_some_vector_func) (long *ivars, size_t nelems, size_t *indices, const int *status, int cmp, long *cmp_values)
- typedef int(* shmem_long_test_func) (long *ivar, int cmp, long cmp_value)
- typedef int(* shmem_long_test_all_func) (long *ivars, size_t nelems, const int *status, int cmp, long cmp_value)
- typedef size_t(* shmem_long_test_any_func) (long *ivars, size_t nelems, const int *status, int cmp, long cmp_value)
- typedef size_t(* shmem_long_test_some_func) (long *ivars, size_t nelems, size_t *indices, const int *status, int cmp, long cmp_value)
- typedef int(* shmem_long_test_all_vector_func) (long *ivars, size_t nelems, const int *status, int cmp, long *cmp_values)
- typedef size_t(* shmem_long_test_any_vector_func) (long *ivars, size_t nelems, const int *status, int cmp, long *cmp_values)

- typedef size_t(∗ shmem_long_test_some_vector_func) (long ∗ivars, size_t nelems, size_t ∗indices, const int ∗status, int cmp, long ∗cmp_values)
- typedef uint64_t(∗ shmem_signal_wait_until_func) (uint64_t ∗sig_addr, int cmp, uint64_t cmp_value)
- typedef void(∗ shmem_fence_func) (void)
- typedef void(∗ shmem_quiet_func) (void)
- typedef void(∗ shmem_set_lock_func) (long ∗lock)
- typedef void(∗ shmem_clear_lock_func) (long ∗lock)

Functions

- bool load_routines ()

  Loads the OpenSHMEM routines dynamically.

Variables

- shmem_fake_routine_func p_shmem_fake_routine
- shmem_init_func p_shmem_init
- shmem_init_thread_func p_shmem_init_thread
- shmem_finalize_func p_shmem_finalize
- shmem_my_pe_func p_shmem_my_pe
- shmem_n_pes_func p_shmem_n_pes
- shmem_pe_accessible_func p_shmem_pe_accessible
- shmem_barrier_all_func p_shmem_barrier_all
- shmem_barrier_func p_shmem_barrier
- shmem_info_get_version_func p_shmem_info_get_version
- shmem_info_get_name_func p_shmem_info_get_name
- shmem_global_exit_func p_shmem_global_exit
- shmem_query_thread_func p_shmem_query_thread
- shmem_ptr_func p_shmem_ptr
- shmem_malloc_func p_shmem_malloc
- shmem_free_func p_shmem_free
- shmem_realloc_func p_shmem_realloc
- shmem_align_func p_shmem_align
- shmem_malloc_with_hints_func p_shmem_malloc_with_hints
- shmem_calloc_func p_shmem_calloc
- shmem_addr_accessible_func p_shmem_addr_accessible
- shmem_team_my_pe_func p_shmem_team_my_pe
- shmem_team_n_pes_func p_shmem_team_n_pes
- shmem_team_get_config_func p_shmem_team_get_config
- shmem_team_translate_pe_func p_shmem_team_translate_pe
- shmem_team_split_strided_func p_shmem_team_split_strided
- shmem_team_split_2d_func p_shmem_team_split_2d
- shmem_team_destroy_func p_shmem_team_destroy
- shmem_ctx_create_func p_shmem_ctx_create
- shmem_team_create_ctx_func p_shmem_team_create_ctx
- shmem_ctx_destroy_func p_shmem_ctx_destroy
- shmem_ctx_get_team_func p_shmem_ctx_get_team
- shmem_long_put_func p_shmem_long_put
- shmem_long_p_func p_shmem_long_p
- shmem_long_iput_func p_shmem_long_iput
- shmem_long_get_func p_shmem_long_get
- shmem_long_g_func p_shmem_long_g
- shmem_long_iget_func p_shmem_long_iget

- shmem_long_put_nbi_func p_shmem_long_put_nbi
- shmem_long_get_nbi_func p_shmem_long_get_nbi
- shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch
- shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set
- shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap
- shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap
- shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc
- shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc
- shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add
- shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add
- shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and
- shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and
- shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or
- shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or
- shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor
- shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor
- shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi
- shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi
- shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi
- shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi
- shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi
- shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi
- shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi
- shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi
- shmem_long_put_signal_func p_shmem_long_put_signal
- shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi
- shmem_signal_fetch_func p_shmem_signal_fetch
- shmem_sync_func p_shmem_sync
- shmem_sync_all_func p_shmem_sync_all
- shmem_long_alltoall_func p_shmem_long_alltoall
- shmem_long_alltoalls_func p_shmem_long_alltoalls
- shmem_long_broadcast_func p_shmem_long_broadcast
- shmem_long_collect_func p_shmem_long_collect
- shmem_long_fcollect_func p_shmem_long_fcollect
- shmem_long_and_reduce_func p_shmem_long_and_reduce
- shmem_long_or_reduce_func p_shmem_long_or_reduce
- shmem_long_xor_reduce_func p_shmem_long_xor_reduce
- shmem_long_max_reduce_func p_shmem_long_max_reduce
- shmem_long_min_reduce_func p_shmem_long_min_reduce
- shmem_long_sum_reduce_func p_shmem_long_sum_reduce
- shmem_long_prod_reduce_func p_shmem_long_prod_reduce
- shmem_long_wait_until_func p_shmem_long_wait_until
- shmem_long_wait_until_all_func p_shmem_long_wait_until_all
- shmem_long_wait_until_any_func p_shmem_long_wait_until_any
- shmem_long_wait_until_some_func p_shmem_long_wait_until_some
- shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector
- shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector
- shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector
- shmem_long_test_func p_shmem_long_test
- shmem_long_test_all_func p_shmem_long_test_all
- shmem_long_test_any_func p_shmem_long_test_any
- shmem_long_test_some_func p_shmem_long_test_some
- shmem_long_test_all_vector_func p_shmem_long_test_all_vector
- shmem_long_test_any_vector_func p_shmem_long_test_any_vector
- shmem_long_test_some_vector_func p_shmem_long_test_some_vector

- shmem_signal_wait_until_func p_shmem_signal_wait_until
- shmem_quiet_func p_shmem_quiet
- shmem_fence_func p_shmem_fence
- shmem_set_lock_func p_shmem_set_lock
- shmem_clear_lock_func p_shmem_clear_lock

4.1.1 Detailed Description

Contains routine pointers to avoid compiler errors if a routine is not implemented.

Definition in file routines.hpp.

4.1.2 Typedef Documentation

4.1.2.1 shmem_addr_accessible_func

```
typedef int(* shmem_addr_accessible_func) (const void *addr, int pe)
```

Definition at line 42 of file routines.hpp.

4.1.2.2 shmem_align_func

```
typedef void *(* shmem_align_func) (size_t alignment, size_t size)
```

Definition at line 39 of file routines.hpp.

4.1.2.3 shmem_barrier_all_func

```
typedef void(* shmem_barrier_all_func) (void)
```

Definition at line 24 of file routines.hpp.

4.1.2.4 shmem_barrier_func

```
typedef void(* shmem_barrier_func) (int PE_start, int logPE_stride, int PE_size, long *pSync)
```

Definition at line 25 of file routines.hpp.

4.1.2.5 shmem_calloc_func

```
typedef void *(* shmem_calloc_func) (size_t count, size_t size)
```

Definition at line 41 of file routines.hpp.

4.1.2.6 shmem_clear_lock_func

```
typedef void(* shmem_clear_lock_func) (long *lock)
```

Definition at line 141 of file routines.hpp.

4.1.2.7 shmem_ctx_create_func

```
typedef int(* shmem_ctx_create_func) (long options, shmem_ctx_t *ctx)
```

Definition at line 54 of file routines.hpp.

4.1.2.8 shmem_ctx_destroy_func

```
typedef void(* shmem_ctx_destroy_func) (shmem_ctx_t ctx)
```

Definition at line 56 of file routines.hpp.

4.1.2.9 shmem_ctx_get_team_func

```
typedef int(* shmem_ctx_get_team_func) (shmem_ctx_t ctx, shmem_team_t *team)
```

Definition at line 57 of file routines.hpp.

4.1.2.10 shmem_fake_routine_func

```
typedef void(* shmem_fake_routine_func) (void)
```

Definition at line 16 of file routines.hpp.

4.1.2.11 shmem_fence_func

```
typedef void(* shmem_fence_func) (void)
```

Definition at line 136 of file routines.hpp.

4.1.2.12 shmem_finalize_func

```
typedef void(* shmem_finalize_func) (void)
```

Definition at line 20 of file routines.hpp.

4.1.2.13 shmem_free_func

```
typedef void(* shmem_free_func) (void *ptr)
```

Definition at line 37 of file routines.hpp.

4.1.2.14 shmem_global_exit_func

```
typedef void(* shmem_global_exit_func) (int status)
```

Definition at line 28 of file routines.hpp.

4.1.2.15 shmem_info_get_name_func

```
typedef void(* shmem_info_get_name_func) (char *name)
```

Definition at line 27 of file routines.hpp.

4.1.2.16 shmem_info_get_version_func

```
typedef void(* shmem_info_get_version_func) (int *major, int *minor)
```

Definition at line 26 of file routines.hpp.

4.1.2.17 shmem_init_func

```
typedef void(* shmem_init_func) (void)
```

Definition at line 19 of file routines.hpp.

4.1.2.18 shmem_init_thread_func

```
typedef int(* shmem_init_thread_func) (int requested, int *provided)
```

Definition at line 31 of file routines.hpp.

4.1.2.19 shmem_long_alltoall_func

```
typedef int(* shmem_long_alltoall_func) (shmem_team_t team, long *dest, const long *source,
size_t nelems)
```

Definition at line 104 of file routines.hpp.

4.1.2.20 shmem_long_alltoalls_func

```
typedef int(* shmem_long_alltoalls_func) (shmem_team_t team, long *dest, const long *source,
ptrdiff_t dst, ptrdiff_t sst, size_t nelems)
```

Definition at line 105 of file routines.hpp.

4.1.2.21 shmem_long_and_reduce_func

```
typedef int(* shmem_long_and_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 109 of file routines.hpp.

4.1.2.22 shmem_long_broadcast_func

```
typedef int(* shmem_long_broadcast_func) (shmem_team_t team, long *dest, const long *source,
size_t nelems, int PE_root)
```

Definition at line 106 of file routines.hpp.

4.1.2.23 shmem_long_collect_func

```
typedef int(* shmem_long_collect_func) (shmem_team_t team, long *dest, const long *source,
size_t nelems)
```

Definition at line 107 of file routines.hpp.

4.1.2.24 shmem_long_fcollect_func

```
typedef int(* shmem_long_fcollect_func) (shmem_team_t team, long *dest, const long *source,
size_t nelems)
```

Definition at line 108 of file routines.hpp.

4.1.2.25 shmem_long_g_func

```
typedef long(* shmem_long_g_func) (const long *src, int pe)
```

Definition at line 64 of file routines.hpp.

4.1.2.26 shmem_long_get_func

```
typedef void(* shmem_long_get_func) (long *dest, const long *src, size_t nelems, int pe)
```

Definition at line 63 of file routines.hpp.

4.1.2.27 shmem_long_get_nbi_func

```
typedef void(* shmem_long_get_nbi_func) (long *dest, const long *src, size_t nelems, int pe)
```

Definition at line 67 of file routines.hpp.

4.1.2.28 shmem_long_iget_func

```
typedef void(* shmem_long_iget_func) (long *dest, const long *src, ptrdiff_t tst, ptrdiff_↩
t sst, size_t nelems, int pe)
```

Definition at line 65 of file routines.hpp.

4.1.2.29 shmem_long_iput_func

```
typedef void(* shmem_long_iput_func) (long *dest, const long *src, ptrdiff_t tst, ptrdiff_↩
t sst, size_t nelems, int pe)
```

Definition at line 62 of file routines.hpp.

4.1.2.30 shmem_long_max_reduce_func

```
typedef int(* shmem_long_max_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 112 of file routines.hpp.

4.1.2.31 shmem_long_min_reduce_func

```
typedef int(* shmem_long_min_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 113 of file routines.hpp.

4.1.2.32 shmem_long_or_reduce_func

```
typedef int(* shmem_long_or_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 110 of file routines.hpp.

4.1.2.33 shmem_long_p_func

```
typedef void(* shmem_long_p_func) (long *dest, long value, int pe)
```

Definition at line 61 of file routines.hpp.

4.1.2.34 shmem_long_prod_reduce_func

```
typedef int(* shmem_long_prod_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 115 of file routines.hpp.

4.1.2.35 shmem_long_put_func

```
typedef void(* shmem_long_put_func) (long *dest, const long *src, size_t nelems, int pe)
```

Definition at line 60 of file routines.hpp.

4.1.2.36 shmem_long_put_nbi_func

```
typedef void(* shmem_long_put_nbi_func) (long *dest, const long *src, size_t nelems, int pe)
```

Definition at line 66 of file routines.hpp.

4.1.2.37 shmem_long_put_signal_func

```
typedef void(* shmem_long_put_signal_func) (long *dest, const long *source, size_t nelems,
uint64_t *sig_addr, uint64_t signal, int sig_op, int pe)
```

Definition at line 97 of file routines.hpp.

4.1.2.38 shmem_long_put_signal_nbi_func

```
typedef void(* shmem_long_put_signal_nbi_func) (long *dest, const long *source, size_t nelems,
uint64_t *sig_addr, uint64_t signal, int sig_op, int pe)
```

Definition at line 98 of file routines.hpp.

4.1.2.39 shmem_long_sum_reduce_func

```
typedef int(* shmem_long_sum_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 114 of file routines.hpp.

4.1.2.40 shmem_long_test_all_func

```
typedef int(* shmem_long_test_all_func) (long *ivars, size_t nelems, const int *status, int
cmp, long cmp_value)
```

Definition at line 126 of file routines.hpp.

4.1.2.41 shmem_long_test_all_vector_func

```
typedef int(* shmem_long_test_all_vector_func) (long *ivars, size_t nelems, const int *status,
int cmp, long *cmp_values)
```

Definition at line 129 of file routines.hpp.

4.1.2.42 shmem_long_test_any_func

```
typedef size_t(* shmem_long_test_any_func) (long *ivars, size_t nelems, const int *status, int
cmp, long cmp_value)
```

Definition at line 127 of file routines.hpp.

4.1.2.43 shmem_long_test_any_vector_func

```
typedef size_t(* shmem_long_test_any_vector_func) (long *ivars, size_t nelems, const int *status,
int cmp, long *cmp_values)
```

Definition at line 130 of file routines.hpp.

4.1.2.44 shmem_long_test_func

```
typedef int(* shmem_long_test_func) (long *ivar, int cmp, long cmp_value)
```

Definition at line 125 of file routines.hpp.

4.1.2.45 shmem_long_test_some_func

```
typedef size_t(* shmem_long_test_some_func) (long *ivars, size_t nelems, size_t *indices, const
int *status, int cmp, long cmp_value)
```

Definition at line 128 of file routines.hpp.

4.1.2.46 shmem_long_test_some_vector_func

```
typedef size_t(* shmem_long_test_some_vector_func) (long *ivars, size_t nelems, size_t *indices,
const int *status, int cmp, long *cmp_values)
```

Definition at line 131 of file routines.hpp.

4.1.2.47 shmem_long_wait_until_all_func

```
typedef void(* shmem_long_wait_until_all_func) (long *ivars, size_t nelems, const int *status,
int cmp, long cmp_value)
```

Definition at line 119 of file routines.hpp.

4.1.2.48 shmem_long_wait_until_all_vector_func

```
typedef void(* shmem_long_wait_until_all_vector_func) (long *ivars, size_t nelems, const int
*status, int cmp, long *cmp_values)
```

Definition at line 122 of file routines.hpp.

4.1.2.49 shmem_long_wait_until_any_func

```
typedef size_t(* shmem_long_wait_until_any_func) (long *ivars, size_t nelems, const int *status,
int cmp, long cmp_value)
```

Definition at line 120 of file routines.hpp.

4.1.2.50 shmem_long_wait_until_any_vector_func

```
typedef size_t(* shmem_long_wait_until_any_vector_func) (long *ivars, size_t nelems, const int
*status, int cmp, long *cmp_values)
```

Definition at line 123 of file routines.hpp.

4.1.2.51 shmem_long_wait_until_func

```
typedef void(* shmem_long_wait_until_func) (long *ivar, int cmp, long cmp_value)
```

Definition at line 118 of file routines.hpp.

4.1.2.52 shmem_long_wait_until_some_func

```
typedef size_t(* shmem_long_wait_until_some_func) (long *ivars, size_t nelems, size_t *indices,
const int *status, int cmp, long cmp_value)
```

Definition at line 121 of file routines.hpp.

4.1.2.53 shmem_long_wait_until_some_vector_func

```
typedef size_t(* shmem_long_wait_until_some_vector_func) (long *ivars, size_t nelems, size_↩
t *indices, const int *status, int cmp, long *cmp_values)
```

Definition at line 124 of file routines.hpp.

4.1.2.54 shmem_long_xor_reduce_func

```
typedef int(* shmem_long_xor_reduce_func) (shmem_team_t team, long *dest, const long *source,
size_t nreduce)
```

Definition at line 111 of file routines.hpp.

4.1.2.55 shmem_malloc_func

```
typedef void *(* shmem_malloc_func) (size_t size)
```

Definition at line 36 of file routines.hpp.

4.1.2.56 shmem_malloc_with_hints_func

```
typedef void *(* shmem_malloc_with_hints_func) (size_t size, long hints)
```

Definition at line 40 of file routines.hpp.

4.1.2.57 shmem_my_pe_func

```
typedef int(* shmem_my_pe_func) (void)
```

Definition at line 21 of file routines.hpp.

4.1.2.58 shmem_n_pes_func

```
typedef int(* shmem_n_pes_func) (void)
```

Definition at line 22 of file routines.hpp.

4.1.2.59 shmem_pe_accessible_func

```
typedef int(* shmem_pe_accessible_func) (int pe)
```

Definition at line 23 of file routines.hpp.

4.1.2.60 shmem_ptr_func

```
typedef void *(* shmem_ptr_func) (const void *dest, int pe)
```

Definition at line 35 of file routines.hpp.

4.1.2.61 shmem_query_thread_func

```
typedef int(* shmem_query_thread_func) (int *provided)
```

Definition at line 32 of file routines.hpp.

4.1.2.62 shmem_quiet_func

```
typedef void(* shmem_quiet_func) (void)
```

Definition at line 137 of file routines.hpp.

4.1.2.63 shmem_realloc_func

```
typedef void *(* shmem_realloc_func) (void *ptr, size_t size)
```

Definition at line 38 of file routines.hpp.

4.1.2.64 shmem_set_lock_func

```
typedef void(* shmem_set_lock_func) (long *lock)
```

Definition at line 140 of file routines.hpp.

4.1.2.65 shmem_signal_fetch_func

```
typedef long(* shmem_signal_fetch_func) (const uint64_t *sig_addr)
```

Definition at line 99 of file routines.hpp.

4.1.2.66 shmem_signal_wait_until_func

```
typedef uint64_t(* shmem_signal_wait_until_func) (uint64_t *sig_addr, int cmp, uint64_t cmp_↩
value)
```

Definition at line 133 of file routines.hpp.

4.1.2.67 shmem_sync_all_func

```
typedef void(* shmem_sync_all_func) (void)
```

Definition at line 103 of file routines.hpp.

4.1.2.68 shmem_sync_func

```
typedef int(* shmem_sync_func) (int PE_start, int logPE_stride, int PE_size, long *pSync)
```

Definition at line 102 of file routines.hpp.

4.1.2.69 shmem_team_create_ctx_func

```
typedef int(* shmem_team_create_ctx_func) (shmem_team_t team, long options, shmem_ctx_t *ctx)
```

Definition at line 55 of file routines.hpp.

4.1.2.70 shmem_team_destroy_func

```
typedef void(* shmem_team_destroy_func) (shmem_team_t team)
```

Definition at line 51 of file routines.hpp.

4.1.2.71 shmem_team_get_config_func

```
typedef void(* shmem_team_get_config_func) (shmem_team_t team, long config_mask, shmem_team_↩
config_t *config)
```

Definition at line 47 of file routines.hpp.

4.1.2.72 shmem_team_my_pe_func

```
typedef int(* shmem_team_my_pe_func) (shmem_team_t team)
```

Definition at line 45 of file routines.hpp.

4.1.2.73 shmem_team_n_pes_func

```
typedef int(* shmem_team_n_pes_func) (shmem_team_t team)
```

Definition at line 46 of file routines.hpp.

4.1.2.74 shmem_team_split_2d_func

```
typedef shmem_team_t(* shmem_team_split_2d_func) (shmem_team_t parent_team, int xrange, const
shmem_team_config_t *xaxis_config, long xaxis_mask, shmem_team_t *xaxis_team, const shmem_↩
team_config_t *yaxis_config, long yaxis_mask, shmem_team_t *yaxis_team)
```

Definition at line 50 of file routines.hpp.

4.1.2.75 shmem_team_split_strided_func

```
typedef shmem_team_t(* shmem_team_split_strided_func) (shmem_team_t parent_team, int start,
int stride, int size, const shmem_team_config_t *config, long config_mask, shmem_team_t *new_↩
team)
```

Definition at line 49 of file routines.hpp.

4.1.2.76 shmem_team_translate_pe_func

```
typedef int(* shmem_team_translate_pe_func) (shmem_team_t src_team, int src_pe, shmem_team_t
dest_team)
```

Definition at line 48 of file routines.hpp.

4.1.2.77 shmem_ulong_atomic_add_func

```
typedef void(* shmem_ulong_atomic_add_func) (const unsigned long *target, unsigned long value,
int pe)
```

Definition at line 77 of file routines.hpp.

4.1.2.78 shmem_ulong_atomic_and_func

```
typedef void(* shmem_ulong_atomic_and_func) (unsigned long *dest, unsigned long value, int pe)
```

Definition at line 80 of file routines.hpp.

4.1.2.79 shmem_ulong_atomic_compare_swap_func

```
typedef unsigned long(* shmem_ulong_atomic_compare_swap_func) (unsigned long *target, unsigned
long cond, unsigned long value, int pe)
```

Definition at line 72 of file routines.hpp.

4.1.2.80 shmem_ulong_atomic_compare_swap_nbi_func

```
typedef void(* shmem_ulong_atomic_compare_swap_nbi_func) (unsigned long *dest, unsigned long
*target, unsigned long cond, unsigned long value, int pe)
```

Definition at line 87 of file routines.hpp.

4.1.2.81 shmem_ulong_atomic_fetch_add_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_add_func) (const unsigned long *target, unsigned
long value, int pe)
```

Definition at line 76 of file routines.hpp.

4.1.2.82 shmem_ulong_atomic_fetch_add_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_add_nbi_func) (unsigned long *dest, const unsigned long
*target, unsigned long value, int pe)
```

Definition at line 90 of file routines.hpp.

4.1.2.83 shmem_ulong_atomic_fetch_and_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_and_func) (unsigned long *dest, unsigned long
value, int pe)
```

Definition at line 79 of file routines.hpp.

4.1.2.84 shmem_ulong_atomic_fetch_and_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_and_nbi_func) (unsigned long *fetch, unsigned long
*dest, unsigned long value, int pe)
```

Definition at line 92 of file routines.hpp.

4.1.2.85 shmem_ulong_atomic_fetch_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_func) (const unsigned long *target, int pe)
```

Definition at line 70 of file routines.hpp.

4.1.2.86 shmem_ulong_atomic_fetch_inc_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_inc_func) (const unsigned long *target, int
pe)
```

Definition at line 74 of file routines.hpp.

4.1.2.87 shmem_ulong_atomic_fetch_inc_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_inc_nbi_func) (unsigned long *dest, const unsigned long
*target, int pe)
```

Definition at line 89 of file routines.hpp.

4.1.2.88 shmem_ulong_atomic_fetch_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_nbi_func) (unsigned long *dest, const unsigned long
*target, int pe)
```

Definition at line 86 of file routines.hpp.

4.1.2.89 shmem_ulong_atomic_fetch_or_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_or_func) (unsigned long *dest, unsigned long
value, int pe)
```

Definition at line 81 of file routines.hpp.

4.1.2.90 shmem_ulong_atomic_fetch_or_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_or_nbi_func) (unsigned long *fetch, unsigned long
*dest, unsigned long value, int pe)
```

Definition at line 93 of file routines.hpp.

4.1.2.91 shmem_ulong_atomic_fetch_xor_func

```
typedef unsigned long(* shmem_ulong_atomic_fetch_xor_func) (unsigned long *dest, unsigned long
value, int pe)
```

Definition at line 83 of file routines.hpp.

4.1.2.92 shmem_ulong_atomic_fetch_xor_nbi_func

```
typedef void(* shmem_ulong_atomic_fetch_xor_nbi_func) (unsigned long *fetch, unsigned long
*dest, unsigned long value, int pe)
```

Definition at line 94 of file routines.hpp.

4.1.2.93 shmem_ulong_atomic_inc_func

```
typedef void(* shmem_ulong_atomic_inc_func) (unsigned long *target, int pe)
```

Definition at line 75 of file routines.hpp.

4.1.2.94 shmem_ulong_atomic_or_func

```
typedef void(* shmem_ulong_atomic_or_func) (unsigned long *dest, unsigned long value, int pe)
```

Definition at line 82 of file routines.hpp.

4.1.2.95 shmem_ulong_atomic_set_func

```
typedef void(* shmem_ulong_atomic_set_func) (unsigned long *target, unsigned long value, int
pe)
```

Definition at line 71 of file routines.hpp.

4.1.2.96 shmem_ulong_atomic_swap_func

```
typedef unsigned long(* shmem_ulong_atomic_swap_func) (unsigned long *target, unsigned long
value, int pe)
```

Definition at line 73 of file routines.hpp.

4.1.2.97 shmem_ulong_atomic_swap_nbi_func

```
typedef void(* shmem_ulong_atomic_swap_nbi_func) (unsigned long *dest, unsigned long *target,
unsigned long value, int pe)
```

Definition at line 88 of file routines.hpp.

4.1.2.98 shmem_ulong_atomic_xor_func

```
typedef void(* shmem_ulong_atomic_xor_func) (unsigned long *dest, unsigned long value, int pe)
```

Definition at line 84 of file routines.hpp.

4.1.3 Function Documentation

4.1.3.1 load_routines()

```
bool load_routines ()
```

Loads the OpenSHMEM routines dynamically.

Returns

True if successful, false if otherwise

This function loads the OpenSHMEM routines at runtime using dynamic linking.

Returns

True if successful, false otherwise.

Definition at line 143 of file routines.cpp.

```
00143                       {
00144   void *handle = dlopen(NULL, RTLD_LAZY);
00145   if (!handle) {
00146     std::cerr « "Failed to open handle: " « dlerror() « std::endl;
00147     return false;
00148   }
00149
00150   p_shmem_fake_routine = reinterpret_cast<shmem_fake_routine_func>(dlsym(handle,
      "shmem_fake_routine"));
00151
00152   /* Setup, Exit, and Query Routines */
00153   p_shmem_init = reinterpret_cast<shmem_init_func>(dlsym(handle, "shmem_init"));
00154   p_shmem_finalize = reinterpret_cast<shmem_finalize_func>(dlsym(handle, "shmem_finalize"));
00155   p_shmem_my_pe = reinterpret_cast<shmem_my_pe_func>(dlsym(handle, "shmem_my_pe"));
00156   p_shmem_n_pes = reinterpret_cast<shmem_n_pes_func>(dlsym(handle, "shmem_n_pes"));
00157   p_shmem_pe_accessible = reinterpret_cast<shmem_pe_accessible_func>(dlsym(handle,
      "shmem_pe_accessible"));
00158   p_shmem_barrier_all = reinterpret_cast<shmem_barrier_all_func>(dlsym(handle, "shmem_barrier_all"));
00159   p_shmem_barrier = reinterpret_cast<shmem_barrier_func>(dlsym(handle, "shmem_barrier"));
00160   p_shmem_info_get_version = reinterpret_cast<shmem_info_get_version_func>(dlsym(handle,
      "shmem_info_get_version"));
00161   p_shmem_info_get_name = reinterpret_cast<shmem_info_get_name_func>(dlsym(handle,
      "shmem_info_get_name"));
00162   p_shmem_global_exit = reinterpret_cast<shmem_global_exit_func>(dlsym(handle, "shmem_global_exit"));
00163
00164   /* Thread Support Routines */
00165   p_shmem_init_thread = reinterpret_cast<shmem_init_thread_func>(dlsym(handle, "shmem_init_thread"));
00166   p_shmem_query_thread = reinterpret_cast<shmem_query_thread_func>(dlsym(handle,
      "shmem_query_thread"));
00167
00168   /* Memory Management Routines */
00169   p_shmem_ptr = reinterpret_cast<shmem_ptr_func>(dlsym(handle, "shmem_ptr"));
00170   p_shmem_malloc = reinterpret_cast<shmem_malloc_func>(dlsym(handle, "shmem_malloc"));
00171   p_shmem_free = reinterpret_cast<shmem_free_func>(dlsym(handle, "shmem_free"));
00172   p_shmem_realloc = reinterpret_cast<shmem_realloc_func>(dlsym(handle, "shmem_realloc"));
00173   p_shmem_align = reinterpret_cast<shmem_align_func>(dlsym(handle, "shmem_align"));
00174   p_shmem_malloc_with_hints = reinterpret_cast<shmem_malloc_with_hints_func>(dlsym(handle,
      "shmem_malloc_with_hints"));
00175   p_shmem_calloc = reinterpret_cast<shmem_calloc_func>(dlsym(handle, "shmem_calloc"));
00176   p_shmem_addr_accessible = reinterpret_cast<shmem_addr_accessible_func>(dlsym(handle,
      "shmem_addr_accessible"));
00177
00178   /* Team Management Routines */
00179   p_shmem_team_my_pe = reinterpret_cast<shmem_team_my_pe_func>(dlsym(handle, "shmem_team_my_pe"));
00180   p_shmem_team_n_pes = reinterpret_cast<shmem_team_n_pes_func>(dlsym(handle, "shmem_team_n_pes"));
00181   p_shmem_team_get_config = reinterpret_cast<shmem_team_get_config_func>(dlsym(handle,
      "shmem_team_get_config"));
00182   p_shmem_team_translate_pe = reinterpret_cast<shmem_team_translate_pe_func>(dlsym(handle,
      "shmem_team_translate_pe"));
00183   p_shmem_team_split_strided = reinterpret_cast<shmem_team_split_strided_func>(dlsym(handle,
      "shmem_team_split_strided"));
00184   p_shmem_team_split_2d = reinterpret_cast<shmem_team_split_2d_func>(dlsym(handle,
      "shmem_team_split_2d"));
00185   p_shmem_team_destroy = reinterpret_cast<shmem_team_destroy_func>(dlsym(handle,
      "shmem_team_destroy"));
```

```
00186
00187   /* Communication/Context Management Routines */
00188   p_shmem_ctx_create = reinterpret_cast<shmem_ctx_create_func>(dlsym(handle, "shmem_ctx_create"));
00189   p_shmem_team_create_ctx = reinterpret_cast<shmem_team_create_ctx_func>(dlsym(handle,
      "shmem_team_create_ctx"));
00190   p_shmem_ctx_destroy = reinterpret_cast<shmem_ctx_destroy_func>(dlsym(handle, "shmem_ctx_destroy"));
00191   p_shmem_ctx_get_team = reinterpret_cast<shmem_ctx_get_team_func>(dlsym(handle,
      "shmem_ctx_get_team"));
00192
00193   /* Remote Access Routines */
00194   p_shmem_long_put = reinterpret_cast<shmem_long_put_func>(dlsym(handle, "shmem_long_put"));
00195   p_shmem_long_p = reinterpret_cast<shmem_long_p_func>(dlsym(handle, "shmem_long_p"));
00196   p_shmem_long_iput = reinterpret_cast<shmem_long_iput_func>(dlsym(handle, "shmem_long_iput"));
00197   p_shmem_long_get = reinterpret_cast<shmem_long_get_func>(dlsym(handle, "shmem_long_get"));
00198   p_shmem_long_g = reinterpret_cast<shmem_long_g_func>(dlsym(handle, "shmem_long_g"));
00199   p_shmem_long_iget = reinterpret_cast<shmem_long_iget_func>(dlsym(handle, "shmem_long_iget"));
00200   p_shmem_long_put_nbi = reinterpret_cast<shmem_long_put_nbi_func>(dlsym(handle,
      "shmem_long_put_nbi"));
00201   p_shmem_long_get_nbi = reinterpret_cast<shmem_long_get_nbi_func>(dlsym(handle,
      "shmem_long_get_nbi"));
00202
00203   /* Atomic Memory Operations */
00204   p_shmem_ulong_atomic_fetch = reinterpret_cast<shmem_ulong_atomic_fetch_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch"));
00205   p_shmem_ulong_atomic_set = reinterpret_cast<shmem_ulong_atomic_set_func>(dlsym(handle,
      "shmem_ulong_atomic_set"));
00206   p_shmem_ulong_atomic_compare_swap =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap"));
00207   p_shmem_ulong_atomic_swap = reinterpret_cast<shmem_ulong_atomic_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_swap"));
00208   p_shmem_ulong_atomic_fetch_inc = reinterpret_cast<shmem_ulong_atomic_fetch_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc"));
00209   p_shmem_ulong_atomic_inc = reinterpret_cast<shmem_ulong_atomic_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_inc"));
00210   p_shmem_ulong_atomic_fetch_add = reinterpret_cast<shmem_ulong_atomic_fetch_add_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add"));
00211   p_shmem_ulong_atomic_add = reinterpret_cast<shmem_ulong_atomic_add_func>(dlsym(handle,
      "shmem_ulong_atomic_add"));
00212   p_shmem_ulong_atomic_fetch_and = reinterpret_cast<shmem_ulong_atomic_fetch_and_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and"));
00213   p_shmem_ulong_atomic_and = reinterpret_cast<shmem_ulong_atomic_and_func>(dlsym(handle,
      "shmem_ulong_atomic_and"));
00214   p_shmem_ulong_atomic_fetch_or = reinterpret_cast<shmem_ulong_atomic_fetch_or_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or"));
00215   p_shmem_ulong_atomic_or = reinterpret_cast<shmem_ulong_atomic_or_func>(dlsym(handle,
      "shmem_ulong_atomic_or"));
00216   p_shmem_ulong_atomic_fetch_xor = reinterpret_cast<shmem_ulong_atomic_fetch_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor"));
00217   p_shmem_ulong_atomic_xor = reinterpret_cast<shmem_ulong_atomic_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_xor"));
00218
00219   p_shmem_ulong_atomic_fetch_nbi = reinterpret_cast<shmem_ulong_atomic_fetch_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_nbi"));
00220   p_shmem_ulong_atomic_compare_swap_nbi =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap_nbi"));
00221   p_shmem_ulong_atomic_swap_nbi = reinterpret_cast<shmem_ulong_atomic_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_swap_nbi"));
00222   p_shmem_ulong_atomic_fetch_inc_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_inc_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc_nbi"));
00223   p_shmem_ulong_atomic_fetch_add_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_add_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add_nbi"));
00224   p_shmem_ulong_atomic_fetch_and_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_and_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and_nbi"));
00225   p_shmem_ulong_atomic_fetch_or_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_or_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or_nbi"));
00226   p_shmem_ulong_atomic_fetch_xor_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_xor_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor_nbi"));
00227
00228   /* Signaling Operations */
00229   p_shmem_signal_fetch = reinterpret_cast<shmem_signal_fetch_func>(dlsym(handle,
      "shmem_signal_fetch"));
00230   p_shmem_long_put_signal = reinterpret_cast<shmem_long_put_signal_func>(dlsym(handle,
      "shmem_long_put_signal"));
00231   p_shmem_long_put_signal_nbi = reinterpret_cast<shmem_long_put_signal_nbi_func>(dlsym(handle,
      "shmem_long_put_signal_nbi"));
00232
00233   /* Collective Routines */
00234   p_shmem_sync = reinterpret_cast<shmem_sync_func>(dlsym(handle, "shmem_sync"));
00235   p_shmem_sync_all = reinterpret_cast<shmem_sync_all_func>(dlsym(handle, "shmem_sync_all"));
00236   p_shmem_long_alltoall = reinterpret_cast<shmem_long_alltoall_func>(dlsym(handle,
```

```
      "shmem_long_alltoall"));
00237   p_shmem_long_alltoalls = reinterpret_cast<shmem_long_alltoalls_func>(dlsym(handle,
      "shmem_long_alltoalls"));
00238   p_shmem_long_broadcast = reinterpret_cast<shmem_long_broadcast_func>(dlsym(handle,
      "shmem_long_broadcast"));
00239   p_shmem_long_collect = reinterpret_cast<shmem_long_collect_func>(dlsym(handle,
      "shmem_long_collect"));
00240   p_shmem_long_fcollect = reinterpret_cast<shmem_long_fcollect_func>(dlsym(handle,
      "shmem_long_fcollect"));
00241   p_shmem_long_and_reduce = reinterpret_cast<shmem_long_and_reduce_func>(dlsym(handle,
      "shmem_long_and_reduce"));
00242   p_shmem_long_or_reduce = reinterpret_cast<shmem_long_or_reduce_func>(dlsym(handle,
      "shmem_long_or_reduce"));
00243   p_shmem_long_xor_reduce = reinterpret_cast<shmem_long_xor_reduce_func>(dlsym(handle,
      "shmem_long_xor_reduce"));
00244   p_shmem_long_max_reduce = reinterpret_cast<shmem_long_max_reduce_func>(dlsym(handle,
      "shmem_long_max_reduce"));
00245   p_shmem_long_min_reduce = reinterpret_cast<shmem_long_min_reduce_func>(dlsym(handle,
      "shmem_long_min_reduce"));
00246   p_shmem_long_sum_reduce = reinterpret_cast<shmem_long_sum_reduce_func>(dlsym(handle,
      "shmem_long_sum_reduce"));
00247   p_shmem_long_prod_reduce = reinterpret_cast<shmem_long_prod_reduce_func>(dlsym(handle,
      "shmem_long_prod_reduce"));
00248
00249   /* Point-to-Point Synchronization Routines */
00250   p_shmem_long_wait_until = reinterpret_cast<shmem_long_wait_until_func>(dlsym(handle,
      "shmem_long_wait_until"));
00251   p_shmem_long_wait_until_all = reinterpret_cast<shmem_long_wait_until_all_func>(dlsym(handle,
      "shmem_long_wait_until_all"));
00252   p_shmem_long_wait_until_any = reinterpret_cast<shmem_long_wait_until_any_func>(dlsym(handle,
      "shmem_long_wait_until_any"));
00253   p_shmem_long_wait_until_some = reinterpret_cast<shmem_long_wait_until_some_func>(dlsym(handle,
      "shmem_long_wait_until_some"));
00254   p_shmem_long_wait_until_all_vector =
      reinterpret_cast<shmem_long_wait_until_all_vector_func>(dlsym(handle,
      "shmem_long_wait_until_all_vector"));
00255   p_shmem_long_wait_until_any_vector =
      reinterpret_cast<shmem_long_wait_until_any_vector_func>(dlsym(handle,
      "shmem_long_wait_until_any_vector"));
00256   p_shmem_long_wait_until_some_vector =
      reinterpret_cast<shmem_long_wait_until_some_vector_func>(dlsym(handle,
      "shmem_long_wait_until_some_vector"));
00257   p_shmem_long_test = reinterpret_cast<shmem_long_test_func>(dlsym(handle, "shmem_long_test"));
00258   p_shmem_long_test_all = reinterpret_cast<shmem_long_test_all_func>(dlsym(handle,
      "shmem_long_test_all"));
00259   p_shmem_long_test_any = reinterpret_cast<shmem_long_test_any_func>(dlsym(handle,
      "shmem_long_test_any"));
00260   p_shmem_long_test_some = reinterpret_cast<shmem_long_test_some_func>(dlsym(handle,
      "shmem_long_test_some"));
00261   p_shmem_long_test_all_vector = reinterpret_cast<shmem_long_test_all_vector_func>(dlsym(handle,
      "shmem_long_test_all_vector"));
00262   p_shmem_long_test_any_vector = reinterpret_cast<shmem_long_test_any_vector_func>(dlsym(handle,
      "shmem_long_test_any_vector"));
00263   p_shmem_long_test_some_vector = reinterpret_cast<shmem_long_test_some_vector_func>(dlsym(handle,
      "shmem_long_test_some_vector"));
00264   p_shmem_signal_wait_until = reinterpret_cast<shmem_signal_wait_until_func>(dlsym(handle,
      "shmem_signal_wait_until"));
00265
00266   /* Memory Ordering Routines */
00267   p_shmem_quiet = reinterpret_cast<shmem_quiet_func>(dlsym(handle, "shmem_quiet"));
00268   p_shmem_fence = reinterpret_cast<shmem_fence_func>(dlsym(handle, "shmem_fence"));
00269
00270   /* Distributed Locking Routines */
00271   p_shmem_set_lock = reinterpret_cast<shmem_set_lock_func>(dlsym(handle, "shmem_set_lock"));
00272   p_shmem_clear_lock = reinterpret_cast<shmem_clear_lock_func>(dlsym(handle, "shmem_clear_lock"));
00273
00274   const char *dlsym_error = dlerror();
00275   if (dlsym_error) {
00276     std::cerr « "Error loading functions: " « dlsym_error « std::endl;
00277     dlclose(handle);
00278     return false;
00279   }
00280
00281   return true;
00282 }
```

References p_shmem_addr_accessible, p_shmem_align, p_shmem_barrier, p_shmem_barrier_all, p_shmem_calloc, p_shmem_clear_lock, p_shmem_ctx_create, p_shmem_ctx_destroy, p_shmem_ctx_get_team, p_shmem_fake_routine, p_shmem_fence, p_shmem_finalize, p_shmem_free, p_shmem_global_exit, p_shmem_info_get_name, p_shmem_info_get_version, p_shmem_init, p_shmem_init_thread, p_shmem_long_alltoall, p_shmem_long_alltoalls, p_shmem_long_and_reduce, p_shmem_long_broadcast, p_shmem_long_collect, p_shmem_long_fcollect, p_shmem_long_g, p_shmem_long_get, p_shmem_long_get_nbi, p_shmem_long_iget, p_shmem_long_iput, p_shmem_long_max_reduce, p_shmem_long_min_reduce, p_shmem_long_or_reduce, p_shmem_long_p,

p_shmem_long_prod_reduce, p_shmem_long_put, p_shmem_long_put_nbi, p_shmem_long_put_signal, p_shmem_long_put_signal_nbi, p_shmem_long_sum_reduce, p_shmem_long_test, p_shmem_long_test_all, p_shmem_long_test_all_vector, p_shmem_long_test_any, p_shmem_long_test_any_vector, p_shmem_long_test_some, p_shmem_long_test_some_vector, p_shmem_long_wait_until, p_shmem_long_wait_until_all, p_shmem_long_wait_until_all_vector, p_shmem_long_wait_until_any, p_shmem_long_wait_until_any_vector, p_shmem_long_wait_until_some, p_shmem_long_wait_until_some_vector, p_shmem_long_xor_reduce, p_shmem_malloc, p_shmem_malloc_with_hints, p_shmem_my_pe, p_shmem_n_pes, p_shmem_pe_accessible, p_shmem_ptr, p_shmem_query_thread, p_shmem_quiet, p_shmem_realloc, p_shmem_set_lock, p_shmem_signal_fetch, p_shmem_signal_wait_until, p_shmem_sync, p_shmem_sync_all, p_shmem_team_create_ctx, p_shmem_team_destroy, p_shmem_team_get_config, p_shmem_team_my_pe, p_shmem_team_n_pes, p_shmem_team_split_2d, p_shmem_team_split_strided, p_shmem_team_translate_pe, p_shmem_ulong_atomic_add, p_shmem_ulong_atomic_and, p_shmem_ulong_atomic_compare_swa p_shmem_ulong_atomic_compare_swap_nbi, p_shmem_ulong_atomic_fetch, p_shmem_ulong_atomic_fetch_add, p_shmem_ulong_atomic_fetch_add_nbi, p_shmem_ulong_atomic_fetch_and, p_shmem_ulong_atomic_fetch_and_nbi, p_shmem_ulong_atomic_fetch_inc, p_shmem_ulong_atomic_fetch_inc_nbi, p_shmem_ulong_atomic_fetch_nbi, p_shmem_ulong_atomic_fetch_or, p_shmem_ulong_atomic_fetch_or_nbi, p_shmem_ulong_atomic_fetch_xor, p_shmem_ulong_atomic_fetch_xor_nbi, p_shmem_ulong_atomic_inc, p_shmem_ulong_atomic_or, p_shmem_ulong_atomic_set, p_shmem_ulong_atomic_swap, p_shmem_ulong_atomic_swap_nbi, and p_shmem_ulong_atomic_xor.

4.1.4 Variable Documentation

4.1.4.1 p_shmem_addr_accessible

```
shmem_addr_accessible_func p_shmem_addr_accessible  [extern]
```

Definition at line 38 of file routines.cpp.

4.1.4.2 p_shmem_align

```
shmem_align_func p_shmem_align  [extern]
```

Definition at line 35 of file routines.cpp.

4.1.4.3 p_shmem_barrier

```
shmem_barrier_func p_shmem_barrier  [extern]
```

Definition at line 21 of file routines.cpp.

4.1.4.4 p_shmem_barrier_all

```
shmem_barrier_all_func p_shmem_barrier_all  [extern]
```

Definition at line 20 of file routines.cpp.

4.1.4.5 p_shmem_calloc

```
shmem_calloc_func p_shmem_calloc  [extern]
```

Definition at line 37 of file routines.cpp.

4.1.4.6 p_shmem_clear_lock

```
shmem_clear_lock_func p_shmem_clear_lock  [extern]
```

Definition at line 134 of file routines.cpp.

4.1.4.7 p_shmem_ctx_create

```
shmem_ctx_create_func p_shmem_ctx_create  [extern]
```

Definition at line 50 of file routines.cpp.

4.1.4.8 p_shmem_ctx_destroy

```
shmem_ctx_destroy_func p_shmem_ctx_destroy  [extern]
```

Definition at line 52 of file routines.cpp.

4.1.4.9 p_shmem_ctx_get_team

```
shmem_ctx_get_team_func p_shmem_ctx_get_team  [extern]
```

Definition at line 53 of file routines.cpp.

4.1.4.10 p_shmem_fake_routine

```
shmem_fake_routine_func p_shmem_fake_routine  [extern]
```

Definition at line 12 of file routines.cpp.

4.1.4.11 p_shmem_fence

```
shmem_fence_func p_shmem_fence  [extern]
```

Definition at line 130 of file routines.cpp.

4.1.4.12 p_shmem_finalize

```
shmem_finalize_func p_shmem_finalize  [extern]
```

Definition at line 16 of file routines.cpp.

4.1.4.13 p_shmem_free

```
shmem_free_func p_shmem_free  [extern]
```

Definition at line 33 of file routines.cpp.

4.1.4.14 p_shmem_global_exit

```
shmem_global_exit_func p_shmem_global_exit  [extern]
```

Definition at line 24 of file routines.cpp.

4.1.4.15 p_shmem_info_get_name

```
shmem_info_get_name_func p_shmem_info_get_name  [extern]
```

Definition at line 23 of file routines.cpp.

4.1.4.16 p_shmem_info_get_version

```
shmem_info_get_version_func p_shmem_info_get_version  [extern]
```

Definition at line 22 of file routines.cpp.

4.1.4.17 p_shmem_init

```
shmem_init_func p_shmem_init  [extern]
```

Definition at line 15 of file routines.cpp.

4.1.4.18 p_shmem_init_thread

```
shmem_init_thread_func p_shmem_init_thread  [extern]
```

Definition at line 27 of file routines.cpp.

4.1.4.19 p_shmem_long_alltoall

```
shmem_long_alltoall_func p_shmem_long_alltoall  [extern]
```

Definition at line 98 of file routines.cpp.

4.1.4.20 p_shmem_long_alltoalls

```
shmem_long_alltoalls_func p_shmem_long_alltoalls  [extern]
```

Definition at line 99 of file routines.cpp.

4.1.4.21 p_shmem_long_and_reduce

```
shmem_long_and_reduce_func p_shmem_long_and_reduce  [extern]
```

Definition at line 103 of file routines.cpp.

4.1.4.22 p_shmem_long_broadcast

shmem_long_broadcast_func p_shmem_long_broadcast [extern]

Definition at line 100 of file routines.cpp.

4.1.4.23 p_shmem_long_collect

shmem_long_collect_func p_shmem_long_collect [extern]

Definition at line 101 of file routines.cpp.

4.1.4.24 p_shmem_long_fcollect

shmem_long_fcollect_func p_shmem_long_fcollect [extern]

Definition at line 102 of file routines.cpp.

4.1.4.25 p_shmem_long_g

shmem_long_g_func p_shmem_long_g [extern]

Definition at line 60 of file routines.cpp.

4.1.4.26 p_shmem_long_get

shmem_long_get_func p_shmem_long_get [extern]

Definition at line 59 of file routines.cpp.

4.1.4.27 p_shmem_long_get_nbi

shmem_long_get_nbi_func p_shmem_long_get_nbi [extern]

Definition at line 63 of file routines.cpp.

4.1.4.28 p_shmem_long_iget

shmem_long_iget_func p_shmem_long_iget [extern]

Definition at line 61 of file routines.cpp.

4.1.4.29 p_shmem_long_iput

shmem_long_iput_func p_shmem_long_iput [extern]

Definition at line 58 of file routines.cpp.

4.1.4.30 p_shmem_long_max_reduce

```
shmem_long_max_reduce_func p_shmem_long_max_reduce  [extern]
```

Definition at line 106 of file routines.cpp.

4.1.4.31 p_shmem_long_min_reduce

```
shmem_long_min_reduce_func p_shmem_long_min_reduce  [extern]
```

Definition at line 107 of file routines.cpp.

4.1.4.32 p_shmem_long_or_reduce

```
shmem_long_or_reduce_func p_shmem_long_or_reduce  [extern]
```

Definition at line 104 of file routines.cpp.

4.1.4.33 p_shmem_long_p

```
shmem_long_p_func p_shmem_long_p  [extern]
```

Definition at line 57 of file routines.cpp.

4.1.4.34 p_shmem_long_prod_reduce

```
shmem_long_prod_reduce_func p_shmem_long_prod_reduce  [extern]
```

Definition at line 109 of file routines.cpp.

4.1.4.35 p_shmem_long_put

```
shmem_long_put_func p_shmem_long_put  [extern]
```

Definition at line 56 of file routines.cpp.

4.1.4.36 p_shmem_long_put_nbi

```
shmem_long_put_nbi_func p_shmem_long_put_nbi  [extern]
```

Definition at line 62 of file routines.cpp.

4.1.4.37 p_shmem_long_put_signal

```
shmem_long_put_signal_func p_shmem_long_put_signal  [extern]
```

Definition at line 92 of file routines.cpp.

4.1.4.38 p_shmem_long_put_signal_nbi

```
shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi  [extern]
```

Definition at line 93 of file routines.cpp.

4.1.4.39 p_shmem_long_sum_reduce

```
shmem_long_sum_reduce_func p_shmem_long_sum_reduce  [extern]
```

Definition at line 108 of file routines.cpp.

4.1.4.40 p_shmem_long_test

```
shmem_long_test_func p_shmem_long_test  [extern]
```

Definition at line 119 of file routines.cpp.

4.1.4.41 p_shmem_long_test_all

```
shmem_long_test_all_func p_shmem_long_test_all  [extern]
```

Definition at line 120 of file routines.cpp.

4.1.4.42 p_shmem_long_test_all_vector

```
shmem_long_test_all_vector_func p_shmem_long_test_all_vector  [extern]
```

Definition at line 123 of file routines.cpp.

4.1.4.43 p_shmem_long_test_any

```
shmem_long_test_any_func p_shmem_long_test_any  [extern]
```

Definition at line 121 of file routines.cpp.

4.1.4.44 p_shmem_long_test_any_vector

```
shmem_long_test_any_vector_func p_shmem_long_test_any_vector  [extern]
```

Definition at line 124 of file routines.cpp.

4.1.4.45 p_shmem_long_test_some

```
shmem_long_test_some_func p_shmem_long_test_some  [extern]
```

Definition at line 122 of file routines.cpp.

4.1.4.46 p_shmem_long_test_some_vector

```
shmem_long_test_some_vector_func p_shmem_long_test_some_vector  [extern]
```

Definition at line 125 of file routines.cpp.

4.1.4.47 p_shmem_long_wait_until

```
shmem_long_wait_until_func p_shmem_long_wait_until  [extern]
```

Definition at line 112 of file routines.cpp.

4.1.4.48 p_shmem_long_wait_until_all

```
shmem_long_wait_until_all_func p_shmem_long_wait_until_all  [extern]
```

Definition at line 113 of file routines.cpp.

4.1.4.49 p_shmem_long_wait_until_all_vector

```
shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector  [extern]
```

Definition at line 116 of file routines.cpp.

4.1.4.50 p_shmem_long_wait_until_any

```
shmem_long_wait_until_any_func p_shmem_long_wait_until_any  [extern]
```

Definition at line 114 of file routines.cpp.

4.1.4.51 p_shmem_long_wait_until_any_vector

```
shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector  [extern]
```

Definition at line 117 of file routines.cpp.

4.1.4.52 p_shmem_long_wait_until_some

```
shmem_long_wait_until_some_func p_shmem_long_wait_until_some  [extern]
```

Definition at line 115 of file routines.cpp.

4.1.4.53 p_shmem_long_wait_until_some_vector

```
shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector  [extern]
```

Definition at line 118 of file routines.cpp.

4.1.4.54 p_shmem_long_xor_reduce

`shmem_long_xor_reduce_func p_shmem_long_xor_reduce [extern]`

Definition at line 105 of file routines.cpp.

4.1.4.55 p_shmem_malloc

`shmem_malloc_func p_shmem_malloc [extern]`

Definition at line 32 of file routines.cpp.

4.1.4.56 p_shmem_malloc_with_hints

`shmem_malloc_with_hints_func p_shmem_malloc_with_hints [extern]`

Definition at line 36 of file routines.cpp.

4.1.4.57 p_shmem_my_pe

`shmem_my_pe_func p_shmem_my_pe [extern]`

Definition at line 17 of file routines.cpp.

4.1.4.58 p_shmem_n_pes

`shmem_n_pes_func p_shmem_n_pes [extern]`

Definition at line 18 of file routines.cpp.

4.1.4.59 p_shmem_pe_accessible

`shmem_pe_accessible_func p_shmem_pe_accessible [extern]`

Definition at line 19 of file routines.cpp.

4.1.4.60 p_shmem_ptr

`shmem_ptr_func p_shmem_ptr [extern]`

Definition at line 31 of file routines.cpp.

4.1.4.61 p_shmem_query_thread

`shmem_query_thread_func p_shmem_query_thread [extern]`

Definition at line 28 of file routines.cpp.

4.1.4.62 p_shmem_quiet

```
shmem_quiet_func p_shmem_quiet  [extern]
```

Definition at line 129 of file routines.cpp.

4.1.4.63 p_shmem_realloc

```
shmem_realloc_func p_shmem_realloc  [extern]
```

Definition at line 34 of file routines.cpp.

4.1.4.64 p_shmem_set_lock

```
shmem_set_lock_func p_shmem_set_lock  [extern]
```

Definition at line 133 of file routines.cpp.

4.1.4.65 p_shmem_signal_fetch

```
shmem_signal_fetch_func p_shmem_signal_fetch  [extern]
```

Definition at line 91 of file routines.cpp.

4.1.4.66 p_shmem_signal_wait_until

```
shmem_signal_wait_until_func p_shmem_signal_wait_until  [extern]
```

Definition at line 126 of file routines.cpp.

4.1.4.67 p_shmem_sync

```
shmem_sync_func p_shmem_sync  [extern]
```

Definition at line 96 of file routines.cpp.

4.1.4.68 p_shmem_sync_all

```
shmem_sync_all_func p_shmem_sync_all  [extern]
```

Definition at line 97 of file routines.cpp.

4.1.4.69 p_shmem_team_create_ctx

```
shmem_team_create_ctx_func p_shmem_team_create_ctx  [extern]
```

Definition at line 51 of file routines.cpp.

4.1.4.70 p_shmem_team_destroy

shmem_team_destroy_func p_shmem_team_destroy [extern]

Definition at line 47 of file routines.cpp.

4.1.4.71 p_shmem_team_get_config

shmem_team_get_config_func p_shmem_team_get_config [extern]

Definition at line 43 of file routines.cpp.

4.1.4.72 p_shmem_team_my_pe

shmem_team_my_pe_func p_shmem_team_my_pe [extern]

Definition at line 41 of file routines.cpp.

4.1.4.73 p_shmem_team_n_pes

shmem_team_n_pes_func p_shmem_team_n_pes [extern]

Definition at line 42 of file routines.cpp.

4.1.4.74 p_shmem_team_split_2d

shmem_team_split_2d_func p_shmem_team_split_2d [extern]

Definition at line 46 of file routines.cpp.

4.1.4.75 p_shmem_team_split_strided

shmem_team_split_strided_func p_shmem_team_split_strided [extern]

Definition at line 45 of file routines.cpp.

4.1.4.76 p_shmem_team_translate_pe

shmem_team_translate_pe_func p_shmem_team_translate_pe [extern]

Definition at line 44 of file routines.cpp.

4.1.4.77 p_shmem_ulong_atomic_add

shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add [extern]

Definition at line 74 of file routines.cpp.

4.1.4.78 p_shmem_ulong_atomic_and

shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and [extern]

Definition at line 76 of file routines.cpp.

4.1.4.79 p_shmem_ulong_atomic_compare_swap

shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap [extern]

Definition at line 69 of file routines.cpp.

4.1.4.80 p_shmem_ulong_atomic_compare_swap_nbi

shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi [extern]

Definition at line 82 of file routines.cpp.

4.1.4.81 p_shmem_ulong_atomic_fetch

shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch [extern]

Definition at line 67 of file routines.cpp.

4.1.4.82 p_shmem_ulong_atomic_fetch_add

shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add [extern]

Definition at line 73 of file routines.cpp.

4.1.4.83 p_shmem_ulong_atomic_fetch_add_nbi

shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi [extern]

Definition at line 85 of file routines.cpp.

4.1.4.84 p_shmem_ulong_atomic_fetch_and

shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and [extern]

Definition at line 75 of file routines.cpp.

4.1.4.85 p_shmem_ulong_atomic_fetch_and_nbi

shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi [extern]

Definition at line 86 of file routines.cpp.

4.1.4.86 p_shmem_ulong_atomic_fetch_inc

shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc [extern]

Definition at line 71 of file routines.cpp.

4.1.4.87 p_shmem_ulong_atomic_fetch_inc_nbi

shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi [extern]

Definition at line 84 of file routines.cpp.

4.1.4.88 p_shmem_ulong_atomic_fetch_nbi

shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi [extern]

Definition at line 81 of file routines.cpp.

4.1.4.89 p_shmem_ulong_atomic_fetch_or

shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or [extern]

Definition at line 77 of file routines.cpp.

4.1.4.90 p_shmem_ulong_atomic_fetch_or_nbi

shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi [extern]

Definition at line 87 of file routines.cpp.

4.1.4.91 p_shmem_ulong_atomic_fetch_xor

shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor [extern]

Definition at line 79 of file routines.cpp.

4.1.4.92 p_shmem_ulong_atomic_fetch_xor_nbi

shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi [extern]

Definition at line 88 of file routines.cpp.

4.1.4.93 p_shmem_ulong_atomic_inc

shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc [extern]

Definition at line 72 of file routines.cpp.

4.1.4.94 p_shmem_ulong_atomic_or

shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or [extern]

Definition at line 78 of file routines.cpp.

4.1.4.95 p_shmem_ulong_atomic_set

shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set [extern]

Definition at line 68 of file routines.cpp.

4.1.4.96 p_shmem_ulong_atomic_swap

shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap [extern]

Definition at line 70 of file routines.cpp.

4.1.4.97 p_shmem_ulong_atomic_swap_nbi

shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi [extern]

Definition at line 83 of file routines.cpp.

4.1.4.98 p_shmem_ulong_atomic_xor

shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor [extern]

Definition at line 80 of file routines.cpp.

4.2 routines.hpp

Go to the documentation of this file.

```
00001
00007 #pragma once
00008
00009 #include <shmem.h>
00010
00011 /************* Define function pointers for OpenSHMEM routines *************/
00012
00013 extern "C" {
00014
00015 /* Fake routine for testing */
00016 typedef void (*shmem_fake_routine_func)(void);
00017
00018 /* Library Setup, Exit, and Query Routines */
00019 typedef void (*shmem_init_func)(void);
00020 typedef void (*shmem_finalize_func)(void);
00021 typedef int (*shmem_my_pe_func)(void);
00022 typedef int (*shmem_n_pes_func)(void);
00023 typedef int (*shmem_pe_accessible_func)(int pe);
00024 typedef void (*shmem_barrier_all_func)(void);
00025 typedef void (*shmem_barrier_func)(int PE_start, int logPE_stride, int PE_size, long *pSync);
00026 typedef void (*shmem_info_get_version_func)(int *major, int *minor);
00027 typedef void (*shmem_info_get_name_func)(char *name);
00028 typedef void (*shmem_global_exit_func)(int status);
```

```
00029
00030 /* Thread Support */
00031 typedef int (*shmem_init_thread_func)(int requested, int *provided);
00032 typedef int (*shmem_query_thread_func)(int *provided);
00033
00034 /* Memory Management Routines */
00035 typedef void* (*shmem_ptr_func)(const void *dest, int pe);
00036 typedef void* (*shmem_malloc_func)(size_t size);
00037 typedef void (*shmem_free_func)(void *ptr);
00038 typedef void* (*shmem_realloc_func)(void *ptr, size_t size);
00039 typedef void* (*shmem_align_func)(size_t alignment, size_t size);
00040 typedef void* (*shmem_malloc_with_hints_func)(size_t size, long hints);
00041 typedef void* (*shmem_calloc_func)(size_t count, size_t size);
00042 typedef int (*shmem_addr_accessible_func)(const void *addr, int pe);
00043
00044 /* Team Management Routines */
00045 typedef int (*shmem_team_my_pe_func)(shmem_team_t team);
00046 typedef int (*shmem_team_n_pes_func)(shmem_team_t team);
00047 typedef void (*shmem_team_get_config_func)(shmem_team_t team, long config_mask, shmem_team_config_t
      *config);
00048 typedef int (*shmem_team_translate_pe_func)(shmem_team_t src_team, int src_pe, shmem_team_t
      dest_team);
00049 typedef shmem_team_t (*shmem_team_split_strided_func)(shmem_team_t parent_team, int start, int stride,
      int size, const shmem_team_config_t *config, long config_mask, shmem_team_t *new_team);
00050 typedef shmem_team_t (*shmem_team_split_2d_func)(shmem_team_t parent_team, int xrange, const
      shmem_team_config_t *xaxis_config, long xaxis_mask, shmem_team_t *xaxis_team, const
      shmem_team_config_t *yaxis_config, long yaxis_mask, shmem_team_t *yaxis_team);
00051 typedef void (*shmem_team_destroy_func)(shmem_team_t team);
00052
00053 /* Communication/Context Management Routines */
00054 typedef int (*shmem_ctx_create_func)(long options, shmem_ctx_t *ctx);
00055 typedef int (*shmem_team_create_ctx_func)(shmem_team_t team, long options, shmem_ctx_t *ctx);
00056 typedef void (*shmem_ctx_destroy_func)(shmem_ctx_t ctx);
00057 typedef int (*shmem_ctx_get_team_func)(shmem_ctx_t ctx, shmem_team_t *team);
00058
00059 /* Remote Access Routines */
00060 typedef void (*shmem_long_put_func)(long *dest, const long *src, size_t nelems, int pe);
00061 typedef void (*shmem_long_p_func)(long *dest, long value, int pe);
00062 typedef void (*shmem_long_iput_func)(long *dest, const long *src, ptrdiff_t tst, ptrdiff_t sst, size_t
      nelems, int pe);
00063 typedef void (*shmem_long_get_func)(long *dest, const long *src, size_t nelems, int pe);
00064 typedef long (*shmem_long_g_func)(const long *src, int pe);
00065 typedef void (*shmem_long_iget_func)(long *dest, const long *src, ptrdiff_t tst, ptrdiff_t sst, size_t
      nelems, int pe);
00066 typedef void (*shmem_long_put_nbi_func)(long *dest, const long *src, size_t nelems, int pe);
00067 typedef void (*shmem_long_get_nbi_func)(long *dest, const long *src, size_t nelems, int pe);
00068
00069 /* Atomic Memory Operations */
00070 typedef unsigned long (*shmem_ulong_atomic_fetch_func)(const unsigned long *target, int pe);
00071 typedef void (*shmem_ulong_atomic_set_func)(unsigned long *target, unsigned long value, int pe);
00072 typedef unsigned long (*shmem_ulong_atomic_compare_swap_func)(unsigned long *target, unsigned long
      cond, unsigned long value, int pe);
00073 typedef unsigned long (*shmem_ulong_atomic_swap_func)(unsigned long *target, unsigned long value, int
      pe);
00074 typedef unsigned long (*shmem_ulong_atomic_fetch_inc_func)(const unsigned long *target, int pe);
00075 typedef void (*shmem_ulong_atomic_inc_func)(unsigned long *target, int pe);
00076 typedef unsigned long (*shmem_ulong_atomic_fetch_add_func)(const unsigned long *target, unsigned long
      value, int pe);
00077 typedef void (*shmem_ulong_atomic_add_func)(const unsigned long *target, unsigned long value, int pe);
00078
00079 typedef unsigned long (*shmem_ulong_atomic_fetch_and_func)(unsigned long *dest, unsigned long value,
      int pe);
00080 typedef void (*shmem_ulong_atomic_and_func)(unsigned long *dest, unsigned long value, int pe);
00081 typedef unsigned long (*shmem_ulong_atomic_fetch_or_func)(unsigned long *dest, unsigned long value,
      int pe);
00082 typedef void (*shmem_ulong_atomic_or_func)(unsigned long *dest, unsigned long value, int pe);
00083 typedef unsigned long (*shmem_ulong_atomic_fetch_xor_func)(unsigned long *dest, unsigned long value,
      int pe);
00084 typedef void (*shmem_ulong_atomic_xor_func)(unsigned long *dest, unsigned long value, int pe);
00085
00086 typedef void (*shmem_ulong_atomic_fetch_nbi_func)(unsigned long *dest, const unsigned long *target,
      int pe);
00087 typedef void (*shmem_ulong_atomic_compare_swap_nbi_func)(unsigned long *dest, unsigned long *target,
      unsigned long cond, unsigned long value, int pe);
00088 typedef void (*shmem_ulong_atomic_swap_nbi_func)(unsigned long *dest, unsigned long *target, unsigned
      long value, int pe);
00089 typedef void (*shmem_ulong_atomic_fetch_inc_nbi_func)(unsigned long *dest, const unsigned long
      *target, int pe);
00090 typedef void (*shmem_ulong_atomic_fetch_add_nbi_func)(unsigned long *dest, const unsigned long
      *target, unsigned long value, int pe);
00091
00092 typedef void (*shmem_ulong_atomic_fetch_and_nbi_func)(unsigned long *fetch, unsigned long *dest,
      unsigned long value, int pe);
00093 typedef void (*shmem_ulong_atomic_fetch_or_nbi_func)(unsigned long *fetch, unsigned long *dest,
      unsigned long value, int pe);
00094 typedef void (*shmem_ulong_atomic_fetch_xor_nbi_func)(unsigned long *fetch, unsigned long *dest,
      unsigned long value, int pe);
```

```
00095
00096 /* Signaling Operations */
00097 typedef void (*shmem_long_put_signal_func)(long *dest, const long *source, size_t nelems, uint64_t
      *sig_addr, uint64_t signal, int sig_op, int pe);
00098 typedef void (*shmem_long_put_signal_nbi_func)(long *dest, const long *source, size_t nelems, uint64_t
      *sig_addr, uint64_t signal, int sig_op, int pe);
00099 typedef long (*shmem_signal_fetch_func)(const uint64_t *sig_addr);
00100
00101 /* Collective Routines */
00102 typedef int (*shmem_sync_func)(int PE_start, int logPE_stride, int PE_size, long *pSync);
00103 typedef void (*shmem_sync_all_func)(void);
00104 typedef int (*shmem_long_alltoall_func)(shmem_team_t team, long *dest, const long *source, size_t
      nelems);
00105 typedef int (*shmem_long_alltoalls_func)(shmem_team_t team, long *dest, const long *source, ptrdiff_t
      dst, ptrdiff_t sst, size_t nelems);
00106 typedef int (*shmem_long_broadcast_func)(shmem_team_t team, long *dest, const long *source, size_t
      nelems, int PE_root);
00107 typedef int (*shmem_long_collect_func)(shmem_team_t team, long *dest, const long *source, size_t
      nelems);
00108 typedef int (*shmem_long_fcollect_func)(shmem_team_t team, long *dest, const long *source, size_t
      nelems);
00109 typedef int (*shmem_long_and_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00110 typedef int (*shmem_long_or_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00111 typedef int (*shmem_long_xor_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00112 typedef int (*shmem_long_max_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00113 typedef int (*shmem_long_min_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00114 typedef int (*shmem_long_sum_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00115 typedef int (*shmem_long_prod_reduce_func)(shmem_team_t team, long *dest, const long *source, size_t
      nreduce);
00116
00117 /* Point-Point Synchronization Routines */
00118 typedef void (*shmem_long_wait_until_func)(long *ivar, int cmp, long cmp_value);
00119 typedef void (*shmem_long_wait_until_all_func)(long *ivars, size_t nelems, const int *status, int cmp,
      long cmp_value);
00120 typedef size_t (*shmem_long_wait_until_any_func)(long *ivars, size_t nelems, const int *status, int
      cmp, long cmp_value);
00121 typedef size_t (*shmem_long_wait_until_some_func)(long *ivars, size_t nelems, size_t *indices, const
      int *status, int cmp, long cmp_value);
00122 typedef void (*shmem_long_wait_until_all_vector_func)(long *ivars, size_t nelems, const int *status,
      int cmp, long *cmp_values);
00123 typedef size_t (*shmem_long_wait_until_any_vector_func)(long *ivars, size_t nelems, const int *status,
      int cmp, long *cmp_values);
00124 typedef size_t (*shmem_long_wait_until_some_vector_func)(long *ivars, size_t nelems, size_t *indices,
      const int *status, int cmp, long *cmp_values);
00125 typedef int (*shmem_long_test_func)(long *ivar, int cmp, long cmp_value);
00126 typedef int (*shmem_long_test_all_func)(long *ivars, size_t nelems, const int *status, int cmp, long
      cmp_value);
00127 typedef size_t (*shmem_long_test_any_func)(long *ivars, size_t nelems, const int *status, int cmp,long
      cmp_value);
00128 typedef size_t (*shmem_long_test_some_func)(long *ivars, size_t nelems, size_t *indices, const int
      *status, int cmp, long cmp_value);
00129 typedef int (*shmem_long_test_all_vector_func)(long *ivars, size_t nelems, const int *status, int cmp,
      long *cmp_values);
00130 typedef size_t (*shmem_long_test_any_vector_func)(long *ivars, size_t nelems, const int *status, int
      cmp, long *cmp_values);
00131 typedef size_t (*shmem_long_test_some_vector_func)(long *ivars, size_t nelems, size_t *indices, const
      int *status, int cmp, long *cmp_values);
00132
00133 typedef uint64_t (*shmem_signal_wait_until_func)(uint64_t *sig_addr, int cmp, uint64_t cmp_value);
00134
00135 /* Memory Ordering Routines */
00136 typedef void (*shmem_fence_func)(void);
00137 typedef void (*shmem_quiet_func)(void);
00138
00139 /* Distributed Locking Routines */
00140 typedef void (*shmem_set_lock_func)(long *lock);
00141 typedef void (*shmem_clear_lock_func)(long *lock);
00142
00143 }
00144
00145 /************* Declare global function pointers *************/
00146 /* Fake routine for testing */
00147 extern shmem_fake_routine_func p_shmem_fake_routine;
00148
00149 /* Library Setup, Exit, and Query Routines */
00150 extern shmem_init_func p_shmem_init;
00151 extern shmem_init_thread_func p_shmem_init_thread;
00152 extern shmem_finalize_func p_shmem_finalize;
00153 extern shmem_my_pe_func p_shmem_my_pe;
00154 extern shmem_n_pes_func p_shmem_n_pes;
00155 extern shmem_pe_accessible_func p_shmem_pe_accessible;
```

```
00156 extern shmem_barrier_all_func p_shmem_barrier_all;
00157 extern shmem_barrier_func p_shmem_barrier;
00158 extern shmem_info_get_version_func p_shmem_info_get_version;
00159 extern shmem_info_get_name_func p_shmem_info_get_name;
00160 extern shmem_global_exit_func p_shmem_global_exit;
00161
00162 /* Thread Support */
00163 extern shmem_query_thread_func p_shmem_query_thread;
00164
00165 /* Memory Management Routines */
00166 extern shmem_ptr_func p_shmem_ptr;
00167 extern shmem_malloc_func p_shmem_malloc;
00168 extern shmem_free_func p_shmem_free;
00169 extern shmem_realloc_func p_shmem_realloc;
00170 extern shmem_align_func p_shmem_align;
00171 extern shmem_malloc_with_hints_func p_shmem_malloc_with_hints;
00172 extern shmem_calloc_func p_shmem_calloc;
00173 extern shmem_addr_accessible_func p_shmem_addr_accessible;
00174
00175 /* Team Management Routines */
00176 extern shmem_team_my_pe_func p_shmem_team_my_pe;
00177 extern shmem_team_n_pes_func p_shmem_team_n_pes;
00178 extern shmem_team_get_config_func p_shmem_team_get_config;
00179 extern shmem_team_translate_pe_func p_shmem_team_translate_pe;
00180 extern shmem_team_split_strided_func p_shmem_team_split_strided;
00181 extern shmem_team_split_2d_func p_shmem_team_split_2d;
00182 extern shmem_team_destroy_func p_shmem_team_destroy;
00183
00184 /* Communication/Context Management Routines */
00185 extern shmem_ctx_create_func p_shmem_ctx_create;
00186 extern shmem_team_create_ctx_func p_shmem_team_create_ctx;
00187 extern shmem_ctx_destroy_func p_shmem_ctx_destroy;
00188 extern shmem_ctx_get_team_func p_shmem_ctx_get_team;
00189
00190 /* Remote Access Routines */
00191 extern shmem_long_put_func p_shmem_long_put;
00192 extern shmem_long_p_func p_shmem_long_p;
00193 extern shmem_long_iput_func p_shmem_long_iput;
00194 extern shmem_long_get_func p_shmem_long_get;
00195 extern shmem_long_g_func p_shmem_long_g;
00196 extern shmem_long_iget_func p_shmem_long_iget;
00197 extern shmem_long_put_nbi_func p_shmem_long_put_nbi;
00198 extern shmem_long_get_nbi_func p_shmem_long_get_nbi;
00199
00200 /* Atomic Memory Operations */
00201 extern shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch;
00202 extern shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set;
00203 extern shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap;
00204 extern shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap;
00205 extern shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc;
00206 extern shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc;
00207 extern shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add;
00208 extern shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add;
00209 extern shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and;
00210 extern shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and;
00211 extern shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or;
00212 extern shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or;
00213 extern shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor;
00214 extern shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor;
00215 extern shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi;
00216 extern shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi;
00217 extern shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi;
00218 extern shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi;
00219 extern shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi;
00220 extern shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi;
00221 extern shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi;
00222 extern shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi;
00223
00224 /* Signaling Operations */
00225 extern shmem_long_put_signal_func p_shmem_long_put_signal;
00226 extern shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi;
00227 extern shmem_signal_fetch_func p_shmem_signal_fetch;
00228
00229 /* Collective Routines */
00230 extern shmem_sync_func p_shmem_sync;
00231 extern shmem_sync_all_func p_shmem_sync_all;
00232 extern shmem_long_alltoall_func p_shmem_long_alltoall;
00233 extern shmem_long_alltoalls_func p_shmem_long_alltoalls;
00234 extern shmem_long_broadcast_func p_shmem_long_broadcast;
00235 extern shmem_long_collect_func p_shmem_long_collect;
00236 extern shmem_long_fcollect_func p_shmem_long_fcollect;
00237 extern shmem_long_and_reduce_func p_shmem_long_and_reduce;
00238 extern shmem_long_or_reduce_func p_shmem_long_or_reduce;
00239 extern shmem_long_xor_reduce_func p_shmem_long_xor_reduce;
00240 extern shmem_long_max_reduce_func p_shmem_long_max_reduce;
00241 extern shmem_long_min_reduce_func p_shmem_long_min_reduce;
00242 extern shmem_long_sum_reduce_func p_shmem_long_sum_reduce;
```

```
00243 extern shmem_long_prod_reduce_func p_shmem_long_prod_reduce;
00244
00245 /* Point-Point Synchronization Routines */
00246 extern shmem_long_wait_until_func p_shmem_long_wait_until;
00247 extern shmem_long_wait_until_all_func p_shmem_long_wait_until_all;
00248 extern shmem_long_wait_until_any_func p_shmem_long_wait_until_any;
00249 extern shmem_long_wait_until_some_func p_shmem_long_wait_until_some;
00250 extern shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector;
00251 extern shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector;
00252 extern shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector;
00253 extern shmem_long_test_func p_shmem_long_test;
00254 extern shmem_long_test_all_func p_shmem_long_test_all;
00255 extern shmem_long_test_any_func p_shmem_long_test_any;
00256 extern shmem_long_test_some_func p_shmem_long_test_some;
00257 extern shmem_long_test_all_vector_func p_shmem_long_test_all_vector;
00258 extern shmem_long_test_any_vector_func p_shmem_long_test_any_vector;
00259 extern shmem_long_test_some_vector_func p_shmem_long_test_some_vector;
00260 extern shmem_signal_wait_until_func p_shmem_signal_wait_until;
00261
00262 /* Memory Ordering Routines */
00263 extern shmem_quiet_func p_shmem_quiet;
00264 extern shmem_fence_func p_shmem_fence;
00265
00266 /* Distributed Locking Routines */
00267 extern shmem_set_lock_func p_shmem_set_lock;
00268 extern shmem_clear_lock_func p_shmem_clear_lock;
00269
00274 bool load_routines();
00275
```

4.3 src/include/shmemvv.hpp File Reference

Contains helper function declarations for the OpenSHMEM verification/validation test suite.

```
#include <shmem.h>
#include <iostream>
#include <getopt.h>
#include <string>
#include <cstring>
#include <vector>
#include <sstream>
#include <dlfcn.h>
#include "../tests/setup/setup_tests.hpp"
#include "../tests/threads/threads_tests.hpp"
#include "../tests/mem/mem_tests.hpp"
#include "../tests/teams/teams_tests.hpp"
#include "../tests/comms/comms_tests.hpp"
#include "../tests/remote/remote_tests.hpp"
#include "../tests/atomics/atomics_tests.hpp"
#include "../tests/signaling/signaling_tests.hpp"
#include "../tests/collectives/collectives_tests.hpp"
#include "../tests/pt2pt/pt2pt_tests.hpp"
#include "../tests/mem_ordering/mem_ordering_tests.hpp"
#include "../tests/locking/locking_tests.hpp"
```

Classes

- struct test_options

  Struct to hold selected tests options.

Macros

- #define RESET_COLOR "\033[0m"
- #define RED_COLOR "\033[31m"
- #define GREEN_COLOR "\033[32m"
- #define YELLOW_COLOR "\033[33m"
- #define HLINE "----------"

Functions

- bool parse_opts (int argc, char *argv[ ], test_options &opts)

  Parses command-line options.
- void display_help ()

  Displays usage information.
- void display_logo ()

  Displays the ASCII art logo.
- void display_test_header (std::string test_name)

  Displays a header for the test category.
- void display_test_info (std::string shmem_name, std::string shmem_version, int npes)

  Displays information about the test suite.
- bool check_if_exists (const std::string &routine_name)

  Checks whether the tested OpenSHMEM implementation has a given routine.
- void display_not_found_warning (std::string routine_name, bool required)

  Displays a warning message that the given routine is not avaible in the tested OpenSHMEM library.
- void display_not_enough_pes (std::string test_type)

  Print error message saying that there needs to be at least 2 PEs for the given test type.
- void display_test_result (std::string routine_name, bool passed, bool required)

  Displays whether the test passed.
- void finalize_shmemvv (int mype)

  Run finalization test.

4.3.1 Detailed Description

Contains helper function declarations for the OpenSHMEM verification/validation test suite.

Definition in file shmemvv.hpp.

4.3.2 Macro Definition Documentation

4.3.2.1 GREEN_COLOR

```
#define GREEN_COLOR "\033[32m"
```

Definition at line 35 of file shmemvv.hpp.

4.3.2.2 HLINE

```
#define HLINE "----------"
```

Definition at line 38 of file shmemvv.hpp.

4.3.2.3 RED_COLOR

```
#define RED_COLOR "\033[31m"
```

Definition at line 34 of file shmemvv.hpp.

4.3.2.4 RESET_COLOR

```
#define RESET_COLOR "\033[0m"
```

Definition at line 33 of file shmemvv.hpp.

4.3.2.5 YELLOW_COLOR

```
#define YELLOW_COLOR "\033[33m"
```

Definition at line 36 of file shmemvv.hpp.

4.3.3 Function Documentation

4.3.3.1 check_if_exists()

```
bool check_if_exists (
            const std::string & routine_name)
```

Checks whether the tested OpenSHMEM implementation has a given routine.

Parameters

routine_name	OpenSHMEM routine that we are making sure is present

Returns

true if it exists, false otherwise

4.3.3.2 display_help()

```
void display_help ()
```

Displays usage information.

4.3.3.3 display_logo()

```
void display_logo ()
```

Displays the ASCII art logo.

4.3.3.4 display_not_enough_pes()

```
void display_not_enough_pes (
            std::string test_type)
```

Print error message saying that there needs to be at least 2 PEs for the given test type.

Parameters

test_type	Category of tests

4.3.3.5 display_not_found_warning()

```
void display_not_found_warning (
            std::string routine_name,
            bool required)
```

Displays a warning message that the given routine is not avaible in the tested OpenSHMEM library.

Parameters

routine_name	OpenSHMEM routine
required	True if test is required, false otherwise

4.3.3.6 display_test_header()

```
void display_test_header (
            std::string test_name)
```

Displays a header for the test category.

Parameters

test_name	Name of the test category.

4.3.3.7 display_test_info()

```
void display_test_info (
            std::string shmem_name,
            std::string shmem_version,
            int npes)
```

Displays information about the test suite.

Parameters

shmem_name	Name of the OpenSHMEM library.
shmem_version	Version of the OpenSHMEM library.
npes	Number of PEs (Processing Elements).

4.3.3.8 display_test_result()

```
void display_test_result (
            std::string routine_name,
            bool passed,
            bool required)
```

Displays whether the test passed.

Parameters

routine_name	OpenSHMEM routine that was tested
passed	True if the test passed, false if the test failed
required	True if the test is required, false otherwise

4.3.3.9 finalize_shmemvv()

```
void finalize_shmemvv (
            int mype)
```

Run finalization test.

Parameters

mype	Current PE

4.3.3.10 parse_opts()

```
bool parse_opts (
            int argc,
            char * argv[],
            test_options & opts)
```

Parses command-line options.

Parameters

argc	Number of command-line arguments.
argv	Array of command-line argument strings.
opts	Reference to the test options structure.

Returns

True if parsing is successful, false otherwise.

4.4 shmemvv.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef SHMEMVV_HPP
00007 #define SHMEMVV_HPP
00008
00009 #include <shmem.h>
00010
00011 #include <iostream>
00012 #include <getopt.h>
00013 #include <string>
00014 #include <cstring>
00015 #include <vector>
00016 #include <sstream>
00017 #include <dlfcn.h>
```

```
00018
00019 #include "../tests/setup/setup_tests.hpp"
00020 #include "../tests/threads/threads_tests.hpp"
00021 #include "../tests/mem/mem_tests.hpp"
00022 #include "../tests/teams/teams_tests.hpp"
00023 #include "../tests/comms/comms_tests.hpp"
00024 #include "../tests/remote/remote_tests.hpp"
00025 #include "../tests/atomics/atomics_tests.hpp"
00026 #include "../tests/signaling/signaling_tests.hpp"
00027 #include "../tests/collectives/collectives_tests.hpp"
00028 #include "../tests/pt2pt/pt2pt_tests.hpp"
00029 #include "../tests/mem_ordering/mem_ordering_tests.hpp"
00030 #include "../tests/locking/locking_tests.hpp"
00031
00032 /* ANSI color codes for pretty output */
00033 #define RESET_COLOR "\033[0m"
00034 #define RED_COLOR "\033[31m"
00035 #define GREEN_COLOR "\033[32m"
00036 #define YELLOW_COLOR "\033[33m"
00037
00038 #define HLINE "----------------------------------------"
00039
00044 struct test_options {
00045   bool test_all;
00046   bool test_setup;
00047   bool test_threads;
00048   bool test_mem;
00049   bool test_teams;
00050   bool test_ctx;
00051   bool test_remote;
00052   bool test_atomics;
00053   bool test_signaling;
00054   bool test_collectives;
00055   bool test_pt2pt_synch;
00056   bool test_mem_ordering;
00057   bool test_locking;
00058   bool help;
00063   test_options() :
00064     test_all(false), test_setup(false), test_threads(false),
00065     test_mem(false), test_teams(false), test_ctx(false),
00066     test_remote(false), test_atomics(false), test_signaling(false),
00067     test_collectives(false), test_pt2pt_synch(false),
00068     test_mem_ordering(false), test_locking(false), help(false) {}
00069 };
00070
00078 bool parse_opts(int argc, char *argv[], test_options &opts);
00079
00083 void display_help();
00084
00088 void display_logo();
00089
00094 void display_test_header(std::string test_name);
00095
00102 void display_test_info(
00103   std::string shmem_name,
00104   std::string shmem_version,
00105   int npes
00106 );
00107
00113 bool check_if_exists(const std::string& routine_name);
00114
00121 void display_not_found_warning(std::string routine_name, bool required);
00122
00128 void display_not_enough_pes(std::string test_type);
00129
00136 void display_test_result(std::string routine_name, bool passed, bool required);
00137
00142  void finalize_shmemvv(int mype);
00143
00144
00145 #endif /* SHMEMVV_HPP */
```

4.5 src/main.cpp File Reference

Driver file for the test suite.

```
#include "shmemvv.hpp"
#include "routines.hpp"
```

Functions

- int main (int argc, char ∗argv[ ])

  Main function for running the test suite.

4.5.1 Detailed Description

Driver file for the test suite.

Definition in file main.cpp.

4.5.2 Function Documentation

4.5.2.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

Main function for running the test suite.

Parameters

argc	Number of command-line arguments.
argv	Array of command-line argument strings.

Returns

EXIT_SUCCESS on success, EXIT_FAILURE on failure.

Definition at line 15 of file main.cpp.

```
00015                                   {
00016   int mype = 0;
00017   int npes = 0;
00018   std::string version = "";
00019   std::string name = "";
00020   test_options opts;
00021
00022   /* Variables to hold test results */
00023   bool result_shmem_init = true;
00024   bool result_shmem_init_thread = true;
00025   bool result_shmem_barrier_all = true;
00026   bool result_shmem_barrier = true;
00027   bool result_shmem_my_pe = true;
00028   bool result_shmem_n_pes = true;
00029   bool result_shmem_pe_accessible = true;
00030   bool result_shmem_info_get_version = true;
00031   bool result_shmem_info_get_name = true;
00032
00033   /************************ SETUP *************************/
00034   void *handle = dlopen(NULL, RTLD_LAZY);
00035   if (!handle) {
00036     if (mype == 0) {
00037       std::cerr « "Failed to open handle: " « dlerror() « std::endl;
00038       return EXIT_FAILURE;
00039     }
00040   }
00041
00042   /* Load OpenSHMEM routines */
00043   if (!load_routines()) {
00044     std::cerr « "Failed to load OpenSHMEM routines" « std::endl;
00045     return EXIT_FAILURE;
```

```
00046   }
00047
00048   /* Initialize with shmem_init_thread() if THREADS tests were enabled */
00049   if (opts.test_threads) {
00050     if (!check_if_exists("shmem_init_thread")) {
00051       if (mype == 0) {
00052         display_not_found_warning("shmem_init_thread()", true);
00053       }
00054       return EXIT_FAILURE;
00055     }
00056     else {
00057       result_shmem_init_thread = test_shmem_init_thread();
00058       if (!result_shmem_init_thread) {
00059         display_test_result("shmem_init_thread()", result_shmem_init_thread, true);
00060         return EXIT_FAILURE;
00061       }
00062     }
00063   }
00064   else {
00065     /* Initialize with regular shmem_init() if THREADS tests are not enabled */
00066     if (!check_if_exists("shmem_init")) {
00067       if (mype == 0) {
00068         display_not_found_warning("shmem_init()", true);
00069       }
00070       return EXIT_FAILURE;
00071     }
00072     else {
00073       result_shmem_init = test_shmem_init();
00074       if (!result_shmem_init) {
00075         display_test_result("shmem_init()", result_shmem_init, true);
00076         return EXIT_FAILURE;
00077       }
00078     }
00079   }
00080
00081   /* Run shmem_barrier_all() test */
00082   if (!check_if_exists("shmem_barrier_all")) {
00083     if (mype == 0) {
00084       display_not_found_warning("shmem_barrier_all()", true);
00085     }
00086     shmem_finalize();
00087     return EXIT_FAILURE;
00088   }
00089   else {
00090     result_shmem_barrier_all = test_shmem_barrier_all();
00091     if (!result_shmem_barrier_all) {
00092       if (shmem_my_pe() == 0) {
00093         display_test_result("shmem_barrier_all()", result_shmem_barrier_all, true);
00094       }
00095       shmem_finalize();
00096       return EXIT_FAILURE;
00097     }
00098   }
00099
00100   /* Run shmem_my_pe() test */
00101   shmem_barrier_all();
00102   if (!check_if_exists("shmem_my_pe")) {
00103     if (mype == 0) {
00104       display_not_found_warning("shmem_my_pe()", true);
00105     }
00106     shmem_finalize();
00107     return EXIT_FAILURE;
00108   }
00109   else {
00110     mype = test_shmem_my_pe();
00111     result_shmem_my_pe = mype >= 0;
00112     if (!result_shmem_my_pe) {
00113       if (mype == 0) {
00114         display_test_result("shmem_my_pe()", result_shmem_my_pe, true);
00115       }
00116       shmem_finalize();
00117       return EXIT_FAILURE;
00118     }
00119   }
00120
00121   /* Run shmem_n_pes() test */
00122   shmem_barrier_all();
00123   if (!check_if_exists("shmem_n_pes")) {
00124     if (mype == 0) {
00125       display_not_found_warning("shmem_n_pes", true);
00126     }
00127     shmem_finalize();
00128     return EXIT_FAILURE;
00129   }
00130   else {
00131     /* Set npes */
00132     npes = test_shmem_n_pes();
```

```
00133     result_shmem_n_pes = npes > 0;
00134     if (!result_shmem_n_pes) {
00135       if (mype == 0) {
00136         display_test_result("shmem_n_pes()", result_shmem_n_pes, true);
00137       }
00138       shmem_finalize();
00139       return EXIT_FAILURE;
00140     }
00141   }
00142
00143   /* Run shmem_pe_accessible() test */
00144   shmem_barrier_all();
00145   if (!check_if_exists("shmem_pe_accessible")) {
00146     if (mype == 0) {
00147       display_not_found_warning("shmem_pe_accessible()", false);
00148     }
00149   }
00150   else {
00151     result_shmem_pe_accessible = test_shmem_pe_accessible();
00152     if (!result_shmem_pe_accessible) {
00153       if (mype == 0) {
00154         display_test_result("shmem_pe_accessible()", result_shmem_pe_accessible, true);
00155       }
00156       shmem_finalize();
00157       return EXIT_FAILURE;
00158     }
00159   }
00160
00161   /*
00162     Run test to make sure OpenSHMEM routines that aren't implemented
00163     don't throw compiler errors
00164   */
00165   #ifdef _DEBUG_
00166     shmem_barrier_all();
00167     if (!check_if_exists("shmem_fake_routine")) {
00168       if (mype == 0) {
00169         display_not_found_warning("shmem_fake_routine()", false);
00170       }
00171     }
00172     else {
00173       test_shmem_fake_routine();
00174     }
00175   #endif
00176
00177   /* Display help if requested */
00178   shmem_barrier_all();
00179   if (opts.help) {
00180     if (mype == 0) {
00181       display_help();
00182     }
00183     shmem_finalize();
00184     return EXIT_SUCCESS;
00185   }
00186
00187   /* Display ASCII art logo */
00188   shmem_barrier_all();
00189   if (mype == 0) {
00190     display_logo();
00191   }
00192
00193   /* Run shmem_barrier() test */
00194   shmem_barrier_all();
00195   if (!check_if_exists("shmem_barrier")) {
00196     if (mype == 0) {
00197       display_not_found_warning("shmem_barrier()", false);
00198     }
00199   }
00200   else {
00201     result_shmem_barrier = test_shmem_barrier();
00202     shmem_barrier_all();
00203   }
00204
00205   /* Run shmem_info_get_version() test */
00206   shmem_barrier_all();
00207   if (!check_if_exists("shmem_info_get_version")) {
00208     result_shmem_info_get_version = false;
00209     if (mype == 0) {
00210       display_not_found_warning("shmem_info_get_version()", false);
00211     }
00212   }
00213   else {
00214     version = test_shmem_info_get_version();
00215     if (version == "") {
00216       result_shmem_info_get_version = false;
00217     }
00218   }
00219
```

```
00220   /* Run shmem_info_get_name() test */
00221   shmem_barrier_all();
00222   if (!check_if_exists("shmem_info_get_name")) {
00223     result_shmem_info_get_name = false;
00224     if (mype == 0) {
00225       display_not_found_warning("shmem_info_get_name()", false);
00226     }
00227   }
00228   else {
00229     name = test_shmem_info_get_name();
00230     if (name == "") {
00231       result_shmem_info_get_name = false;
00232     }
00233   }
00234
00235   /* Parse command-line options */
00236   if (!parse_opts(argc, argv, opts)) {
00237     if (mype == 0) {
00238       display_help();
00239     }
00240     shmem_finalize();
00241     return EXIT_FAILURE;
00242   }
00243
00244   shmem_barrier_all();
00245
00246   /* Enable all tests if --all is specified or no specific test is selected */
00247   if (opts.test_all ||
00248       !(opts.test_setup || opts.test_threads || opts.test_mem || opts.test_teams ||
00249         opts.test_ctx || opts.test_remote || opts.test_atomics || opts.test_signaling ||
00250         opts.test_collectives || opts.test_pt2pt_synch || opts.test_mem_ordering ||
      opts.test_locking))
00251   {
00252     opts.test_setup = true; opts.test_threads = true; opts.test_mem = true; opts.test_teams = true;
00253     opts.test_ctx = true; opts.test_remote = true; opts.test_atomics = true; opts.test_signaling =
      true;
00254     opts.test_collectives = true; opts.test_pt2pt_synch = true; opts.test_mem_ordering = true;
      opts.test_locking = true;
00255   }
00256
00257   /* Display test information */
00258   shmem_barrier_all();
00259   if (mype == 0) {
00260     display_test_info(name, version, npes);
00261   }
00262
00263   /* Print setup tests header */
00264   shmem_barrier_all();
00265   if (mype == 0) {
00266     display_test_header("SETUP");
00267   }
00268
00269   /* shmem_init() and shmem_my_pe() tests passed */
00270   shmem_barrier_all();
00271   if (mype == 0) {
00272     if (!opts.test_threads) {
00273       display_test_result("shmem_init()", result_shmem_init, true);
00274     }
00275     display_test_result("shmem_barrier_all()", result_shmem_barrier_all, true);
00276     display_test_result("shmem_barrier()", result_shmem_barrier, false);
00277     display_test_result("shmem_my_pe()", result_shmem_my_pe, true);
00278     display_test_result("shmem_n_pes()", result_shmem_n_pes, true);
00279     display_test_result("shmem_pe_accessible()", result_shmem_pe_accessible, true);
00280     if (version != "1.5" && version != "1.50") {
00281       std::cerr « YELLOW_COLOR « "shmem_info_get_version() test did not return 1.5... Returned " «
      version « std::endl;
00282     }
00283     else {
00284       display_test_result("shmem_info_get_version()", result_shmem_info_get_version, false);
00285     }
00286     display_test_result("shmem_info_get_name()", result_shmem_info_get_name,false);
00287   }
00288
00289   /************************ THREADS TESTS *************************/
00290   if (opts.test_threads) {
00291     shmem_barrier_all();
00292     if (mype == 0) {
00293       display_test_header("THREADS");
00294     }
00295     shmem_barrier_all();
00296
00297     /* If we made it here shmem_init_thread() passed */
00298     if (mype == 0) {
00299       display_test_result("shmem_init_thread()", result_shmem_init_thread, true);
00300     }
00301
00302     /* Test shmem_query_thread() */
```

```
00303     shmem_barrier_all();
00304     if ( !check_if_exists("shmem_query_thread") ) {
00305       if (mype == 0) {
00306         display_not_found_warning("shmem_query_thread()", false);
00307       }
00308     }
00309     else {
00310       bool result_shmem_query_thread = test_shmem_query_thread();
00311       shmem_barrier_all();
00312       if (mype == 0) {
00313         display_test_result("shmem_query_thread()", result_shmem_query_thread, false);
00314       }
00315     }
00316   }
00317
00318   /************************ MEMORY TESTS *************************/
00319   if (opts.test_mem) {
00320     shmem_barrier_all();
00321     if (mype == 0) {
00322       display_test_header("MEMORY MANAGEMENT");
00323     }
00324
00325     /* Test shmem_malloc() and shmem_free() */
00326     shmem_barrier_all();
00327     if ( check_if_exists("shmem_malloc") && check_if_exists("shmem_free") ) {
00328       bool result_shmem_malloc_free = test_shmem_malloc_free();
00329       shmem_barrier_all();
00330       if (mype == 0) {
00331         display_test_result("shmem_malloc()", result_shmem_malloc_free, false);
00332         display_test_result("shmem_free()", result_shmem_malloc_free, false);
00333       }
00334     }
00335     else {
00336       if ( !check_if_exists("shmem_malloc") ) {
00337         if (mype == 0) {
00338           display_not_found_warning("shmem_malloc()", false);
00339         }
00340       }
00341       if ( !check_if_exists("shmem_free") ) {
00342         if (mype == 0) {
00343           display_not_found_warning("shmem_free()", false);
00344         }
00345       }
00346     }
00347
00348     /* Test shmem_ptr() */
00349     shmem_barrier_all();
00350     if ( !check_if_exists("shmem_ptr") ) {
00351       if (mype == 0) {
00352         display_not_found_warning("shmem_ptr()", false);
00353       }
00354     }
00355     else {
00356       bool result_shmem_ptr = test_shmem_ptr();
00357       shmem_barrier_all();
00358       if (mype == 0) {
00359         display_test_result("shmem_ptr()", result_shmem_ptr, false);
00360       }
00361     }
00362
00363     /* Test shmem_addr_accessible */
00364     shmem_barrier_all();
00365     if ( !check_if_exists("shmem_addr_accessible") ) {
00366       if (mype == 0) {
00367         display_not_found_warning("shmem_addr_accessible", false);
00368       }
00369     }
00370     else {
00371       bool result_shmem_addr_accessible = test_shmem_addr_accessible();
00372       shmem_barrier_all();
00373       if (mype == 0) {
00374         display_test_result("shmem_addr_accessible()", result_shmem_addr_accessible, false);
00375       }
00376     }
00377
00378     /* Test shmem_realloc() */
00379     shmem_barrier_all();
00380     if ( !check_if_exists("shmem_realloc") ) {
00381       if (mype == 0) {
00382         display_not_found_warning("shme_realloc()", false);
00383       }
00384     }
00385     else {
00386       bool result_shmem_realloc = test_shmem_realloc();
00387       shmem_barrier_all();
00388       if (mype == 0) {
00389         display_test_result("shmem_realloc()", result_shmem_realloc, false);
```

```
00390       }
00391     }
00392
00393     /* Test shmem_align() */
00394     shmem_barrier_all();
00395     if ( !check_if_exists("shmem_align") ) {
00396       if (mype == 0) {
00397         display_not_found_warning("shmem_align()", false);
00398       }
00399     }
00400     else {
00401       bool result_shmem_align = test_shmem_align();
00402       shmem_barrier_all();
00403       if (mype == 0) {
00404         display_test_result("shmem_align()", result_shmem_align, false);
00405       }
00406     }
00407
00408     /* Test shmem_malloc_with_hints() */
00409     shmem_barrier_all();
00410     if ( !check_if_exists("shmem_malloc_with_hints") ) {
00411       if (mype == 0) {
00412         display_not_found_warning("shmem_malloc_with_hints()", false);
00413       }
00414     }
00415     else {
00416       bool result_shmem_malloc_with_hints = test_shmem_malloc_with_hints();
00417       shmem_barrier_all();
00418       if (mype == 0) {
00419         display_test_result("shmem_malloc_with_hints()", result_shmem_malloc_with_hints, false);
00420       }
00421     }
00422
00423     /* Test shmem_calloc() */
00424     shmem_barrier_all();
00425     if ( !check_if_exists("shmem_calloc") ) {
00426       if (mype == 0) {
00427         display_not_found_warning("shmem_calloc()", false);
00428       }
00429     }
00430     else {
00431       bool result_shmem_calloc = test_shmem_calloc();
00432       shmem_barrier_all();
00433       if (mype == 0) {
00434         display_test_result("shmem_calloc()", result_shmem_calloc, false);
00435       }
00436     }
00437   }
00438
00439   /************************ TEAMS TESTS *************************/
00440   if (opts.test_teams) {
00441     shmem_barrier_all();
00442     if (mype == 0) {
00443       display_test_header("TEAMS MANAGEMENT");
00444     }
00445
00446     /* Run shmem_team_my_pe() test */
00447     shmem_barrier_all();
00448     if ( !check_if_exists("shmem_team_my_pe") ) {
00449       if (mype == 0) {
00450         display_not_found_warning("shmem_team_my_pe()", true);
00451       }
00452     }
00453     else {
00454       bool result_shmem_team_my_pe = test_shmem_team_my_pe();
00455       shmem_barrier_all();
00456       if (mype == 0) {
00457         display_test_result("shmem_team_my_pe()", result_shmem_team_my_pe, false);
00458       }
00459     }
00460
00461     /* Run shmem_team_n_pes() test */
00462     shmem_barrier_all();
00463     if ( !check_if_exists("shmem_team_n_pes") ) {
00464       if (mype == 0) {
00465         display_not_found_warning("shmem_team_n_pes()", true);
00466       }
00467     }
00468     else {
00469       bool result_shmem_team_n_pes = test_shmem_team_n_pes();
00470       shmem_barrier_all();
00471       if (mype == 0) {
00472         display_test_result("shmem_team_n_pes()", result_shmem_team_n_pes, false);
00473       }
00474     }
00475
00476     /* Run shmem_team_get_config() test */
```

```
00477     shmem_barrier_all();
00478     if ( !check_if_exists("shmem_team_get_config") ) {
00479       if (mype == 0) {
00480         display_not_found_warning("shmem_team_get_config()", false);
00481       }
00482     }
00483     else {
00484       bool result_shmem_team_get_config = test_shmem_team_get_config();
00485       shmem_barrier_all();
00486       if (mype == 0) {
00487         display_test_result("shmem_team_get_config()", result_shmem_team_get_config, false);
00488       }
00489     }
00490
00491     /* Run shmem_team_translate_pe() test */
00492     shmem_barrier_all();
00493     if ( !check_if_exists("shmem_team_translate_pe") ) {
00494       if (mype == 0) {
00495         display_not_found_warning("shmem_team_translate_pe()", false);
00496       }
00497     }
00498     else {
00499       bool result_shmem_team_translate_pe = test_shmem_team_translate_pe();
00500       shmem_barrier_all();
00501       if (mype == 0) {
00502         display_test_result("shmem_team_translate_pe()", result_shmem_team_translate_pe, false);
00503       }
00504     }
00505
00506     /* Run shmem_team_split_strided() test */
00507     shmem_barrier_all();
00508     if ( !check_if_exists("shmem_team_split_strided") ) {
00509       if (mype == 0) {
00510         display_not_found_warning("shmem_team_split_strided()", false);
00511       }
00512     }
00513     else {
00514       bool result_shmem_team_split_strided = test_shmem_team_split_strided();
00515       shmem_barrier_all();
00516       if (mype == 0) {
00517         display_test_result("shmem_team_split_strided()", result_shmem_team_split_strided, false);
00518       }
00519     }
00520
00521     /* Run shmem_team_split_2d() test */
00522     shmem_barrier_all();
00523     if ( !check_if_exists("shmem_team_split_2d") ) {
00524       if (mype == 0) {
00525         display_not_found_warning("shmem_team_split_2d()", false);
00526       }
00527     }
00528     else {
00529       bool result_shmem_team_split_2d = test_shmem_team_split_2d();
00530       shmem_barrier_all();
00531       if (mype == 0) {
00532         display_test_result("shmem_team_split_2d()", result_shmem_team_split_2d, false);
00533       }
00534     }
00535
00536     /* Run shmem_team_destroy() test */
00537     shmem_barrier_all();
00538     if ( !check_if_exists("shmem_team_destroy") ) {
00539       if (mype == 0) {
00540         display_not_found_warning("shmem_team_destroy()", false);
00541       }
00542     }
00543     else {
00544       bool result_shmem_team_destroy = test_shmem_team_destroy();
00545       shmem_barrier_all();
00546       if (mype == 0) {
00547         display_test_result("shmem_team_destroy()", result_shmem_team_destroy, false);
00548       }
00549     }
00550   }
00551
00552   /************************ CTX TESTS *************************/
00553   if (opts.test_ctx) {
00554     shmem_barrier_all();
00555     if (mype == 0) {
00556       display_test_header("COMMUNICATION / CONTEXT");
00557     }
00558
00559     /* Run shmem_ctx_create() test */
00560     shmem_barrier_all();
00561     if (!check_if_exists("shmem_ctx_create")) {
00562       if (mype == 0) {
00563         display_not_found_warning("shmem_ctx_create()", false);
```

```
00564       }
00565     }
00566     else {
00567       bool result_shmem_ctx_create = test_shmem_ctx_create();
00568       shmem_barrier_all();
00569       if (mype == 0) {
00570         display_test_result("shmem_ctx_create()", result_shmem_ctx_create, false);
00571       }
00572     }
00573
00574     /* Run shmem_team_create_ctx() test */
00575     shmem_barrier_all();
00576     if (!check_if_exists("shmem_team_create_ctx")) {
00577       if (mype == 0) {
00578         display_not_found_warning("shmem_team_create_ctx()", false);
00579       }
00580     }
00581     else {
00582       bool result_shmem_team_create_ctx = test_shmem_team_create_ctx();
00583       shmem_barrier_all();
00584       if (mype == 0) {
00585         display_test_result("shmem_team_create_ctx()", result_shmem_team_create_ctx, false);
00586       }
00587     }
00588
00589     /* Run shmem_ctx_destroy() test */
00590     shmem_barrier_all();
00591     if (!check_if_exists("shmem_ctx_destroy")) {
00592       if (mype == 0) {
00593         display_not_found_warning("shmem_ctx_destroy()", false);
00594       }
00595     }
00596     else {
00597       bool result_shmem_ctx_destroy = test_shmem_ctx_destroy();
00598       shmem_barrier_all();
00599       if (mype == 0) {
00600         display_test_result("shmem_ctx_destroy()", result_shmem_ctx_destroy, false);
00601       }
00602     }
00603
00604     /* Run shmem_ctx_get_team() test */
00605     shmem_barrier_all();
00606     if (!check_if_exists("shmem_ctx_get_team")) {
00607       if (mype == 0) {
00608         display_not_found_warning("shmem_ctx_get_team()", false);
00609       }
00610     }
00611     else {
00612       bool result_shmem_ctx_get_team = test_shmem_ctx_get_team();
00613       shmem_barrier_all();
00614       if (mype == 0) {
00615         display_test_result("shmem_ctx_get_team()", result_shmem_ctx_get_team, false);
00616       }
00617     }
00618   }
00619
00620   /************************ REMOTE TESTS *************************/
00621   if (opts.test_remote) {
00622     shmem_barrier_all();
00623     if (mype == 0) {
00624       display_test_header("REMOTE MEMORY ACCESS");
00625     }
00626
00627     /* Check to make sure there are at least 2 PEs */
00628     if ( !(npes > 1) ) {
00629       if (mype == 0) {
00630         display_not_enough_pes("REMOTE MEMORY ACCESS");
00631       }
00632     }
00633     else {
00634       /* Run shmem_put() test */
00635       shmem_barrier_all();
00636       if (!check_if_exists("shmem_long_put")) {
00637         if (mype == 0) {
00638           display_not_found_warning("shmem_long_put()", false);
00639         }
00640       }
00641       else {
00642         bool result_shmem_put = test_shmem_put();
00643         shmem_barrier_all();
00644         if (mype == 0) {
00645           display_test_result("shmem_put()", result_shmem_put, false);
00646         }
00647       }
00648
00649       /* Run shmem_p() test */
00650       shmem_barrier_all();
```

```
00651       if (!check_if_exists("shmem_long_p")) {
00652         if (mype == 0) {
00653           display_not_found_warning("shmem_long_p()", false);
00654         }
00655       }
00656       else {
00657         bool result_shmem_p = test_shmem_p();
00658         shmem_barrier_all();
00659         if (mype == 0) {
00660           display_test_result("shmem_p()", result_shmem_p, false);
00661         }
00662       }
00663
00664       /* Run shmem_iput() test */
00665       shmem_barrier_all();
00666       if (!check_if_exists("shmem_long_iput")) {
00667         if (mype == 0) {
00668           display_not_found_warning("shmem_long_iput()", false);
00669         }
00670       }
00671       else {
00672         bool result_shmem_iput = test_shmem_iput();
00673         shmem_barrier_all();
00674         if (mype == 0) {
00675           display_test_result("shmem_iput()", result_shmem_iput, false);
00676         }
00677       }
00678
00679       /* Run shmem_get() test */
00680       shmem_barrier_all();
00681       if (!check_if_exists("shmem_long_get")) {
00682         if (mype == 0) {
00683           display_not_found_warning("shmem_long_get()", false);
00684         }
00685       }
00686       else {
00687         bool result_shmem_get = test_shmem_get();
00688         shmem_barrier_all();
00689         if (mype == 0) {
00690           display_test_result("shmem_get()", result_shmem_get, false);
00691         }
00692       }
00693
00694       /* Run shmem_g() test */
00695       shmem_barrier_all();
00696       if (!check_if_exists("shmem_long_g")) {
00697         if (mype == 0) {
00698           display_not_found_warning("shmem_long_g()", false);
00699         }
00700       }
00701       else {
00702         bool result_shmem_g = test_shmem_g();
00703         shmem_barrier_all();
00704         if (mype == 0) {
00705           display_test_result("shmem_g()", result_shmem_g, false);
00706         }
00707       }
00708
00709       /* Run shmem_iget() test */
00710       shmem_barrier_all();
00711       if (!check_if_exists("shmem_long_iget")) {
00712         if (mype == 0) {
00713           display_not_found_warning("shmem_long_iget()", false);
00714         }
00715       }
00716       else {
00717         bool result_shmem_iget = test_shmem_iget();
00718         shmem_barrier_all();
00719         if (mype == 0) {
00720           display_test_result("shmem_iget()", result_shmem_iget, false);
00721         }
00722       }
00723
00724       /* Run shmem_put_nbi() test */
00725       shmem_barrier_all();
00726       if (!check_if_exists("shmem_long_put_nbi")) {
00727         if (mype == 0) {
00728           display_not_found_warning("shmem_long_put_nbi()", false);
00729         }
00730       }
00731       else {
00732         bool result_shmem_put_nbi = test_shmem_put_nbi();
00733         shmem_barrier_all();
00734         if (mype == 0) {
00735           display_test_result("shmem_put_nbi()", result_shmem_put_nbi, false);
00736         }
00737       }
```

```
00738
00739       /* Run shmem_get_nbi() test */
00740       shmem_barrier_all();
00741       if (!check_if_exists("shmem_long_get_nbi")) {
00742         if (mype == 0) {
00743           display_not_found_warning("shmem_long_get_nbi()", false);
00744         }
00745       }
00746       else {
00747         bool result_shmem_get_nbi = test_shmem_get_nbi();
00748         shmem_barrier_all();
00749         if (mype == 0) {
00750           display_test_result("shmem_get_nbi()", result_shmem_get_nbi, false);
00751         }
00752       }
00753     }
00754   }
00755   /************************ ATOMICS TESTS *************************/
00756   if (opts.test_atomics) {
00757     shmem_barrier_all();
00758     if (mype == 0) {
00759       display_test_header("ATOMIC MEMORY OPS");
00760     }
00761
00762     /* Make sure there are at least 2 PEs */
00763     if ( !(npes > 1) ) {
00764       if (mype == 0) {
00765         display_not_enough_pes("ATOMIC MEMORY OPS");
00766       }
00767     }
00768     else {
00769       /* Run shmem_atomic_fetch() test */
00770       shmem_barrier_all();
00771       if (!check_if_exists("shmem_ulong_atomic_fetch")) {
00772         if (mype == 0) {
00773           display_not_found_warning("shmem_atomic_fetch()", false);
00774         }
00775       }
00776       else {
00777         bool result_shmem_atomic_fetch = test_shmem_atomic_fetch();
00778         shmem_barrier_all();
00779         if (mype == 0) {
00780           display_test_result("shmem_atomic_fetch()", result_shmem_atomic_fetch, false);
00781         }
00782       }
00783
00784       /* Run shmem_atomic_set() test */
00785       shmem_barrier_all();
00786       if (!check_if_exists("shmem_ulong_atomic_set")) {
00787         if (mype == 0) {
00788           display_not_found_warning("shmem_atomic_set()", false);
00789         }
00790       }
00791       else {
00792         bool result_shmem_atomic_set = test_shmem_atomic_set();
00793         shmem_barrier_all();
00794         if (mype == 0) {
00795           display_test_result("shmem_atomic_set()", result_shmem_atomic_set, false);
00796         }
00797       }
00798
00799       /* Run shmem_atomic_compare_swap() test */
00800       shmem_barrier_all();
00801       if (!check_if_exists("shmem_ulong_atomic_compare_swap")) {
00802         if (mype == 0) {
00803           display_not_found_warning("shmem_atomic_compare_swap()", false);
00804         }
00805       }
00806       else {
00807         bool result_shmem_atomic_compare_swap = test_shmem_atomic_compare_swap();
00808         shmem_barrier_all();
00809         if (mype == 0) {
00810           display_test_result("shmem_atomic_compare_swap()", result_shmem_atomic_compare_swap, false);
00811         }
00812       }
00813
00814       /* Run shmem_atomic_swap() test */
00815       shmem_barrier_all();
00816       if (!check_if_exists("shmem_ulong_atomic_swap")) {
00817         if (mype == 0) {
00818           display_not_found_warning("shmem_atomic_swap()", false);
00819         }
00820       }
00821       else {
00822         bool result_shmem_atomic_swap = test_shmem_atomic_swap();
00823         shmem_barrier_all();
00824         if (mype == 0) {
```

```
00825           display_test_result("shmem_atomic_swap()", result_shmem_atomic_swap, false);
00826         }
00827       }
00828
00829       /* Run shmem_atomic_fetch_inc() test */
00830       shmem_barrier_all();
00831       if (!check_if_exists("shmem_ulong_atomic_fetch_inc")) {
00832         if (mype == 0) {
00833           display_not_found_warning("shmem_atomic_fetch_inc()", false);
00834         }
00835       }
00836       else {
00837         bool result_shmem_atomic_fetch_inc = test_shmem_atomic_fetch_inc();
00838         shmem_barrier_all();
00839         if (mype == 0) {
00840           display_test_result("shmem_atomic_fetch_inc()", result_shmem_atomic_fetch_inc, false);
00841         }
00842       }
00843
00844       /* Run shmem_atomic_inc() test */
00845       shmem_barrier_all();
00846       if (!check_if_exists("shmem_ulong_atomic_inc")) {
00847         if (mype == 0) {
00848           display_not_found_warning("shmem_atomic_inc()", false);
00849         }
00850       }
00851       else {
00852         bool result_shmem_atomic_inc = test_shmem_atomic_inc();
00853         shmem_barrier_all();
00854         if (mype == 0) {
00855           display_test_result("shmem_atomic_inc()", result_shmem_atomic_inc, false);
00856         }
00857       }
00858
00859       /* Run shmem_atomic_fetch_add() test */
00860       shmem_barrier_all();
00861       if (!check_if_exists("shmem_ulong_atomic_fetch_add")) {
00862         if (mype == 0) {
00863           display_not_found_warning("shmem_atomic_fetch_add()", false);
00864         }
00865       }
00866       else {
00867         bool result_shmem_atomic_fetch_add = test_shmem_atomic_fetch_add();
00868         shmem_barrier_all();
00869         if (mype == 0) {
00870           display_test_result("shmem_atomic_fetch_add()", result_shmem_atomic_fetch_add, false);
00871         }
00872       }
00873
00874       /* Run shmem_atomic_add() test */
00875       shmem_barrier_all();
00876       if (!check_if_exists("shmem_ulong_atomic_add")) {
00877         if (mype == 0) {
00878           display_not_found_warning("shmem_atomic_add()", false);
00879         }
00880       }
00881       else {
00882         bool result_shmem_atomic_add = test_shmem_atomic_add();
00883         shmem_barrier_all();
00884         if (mype == 0) {
00885           display_test_result("shmem_atomic_add()", result_shmem_atomic_add, false);
00886         }
00887       }
00888
00889       /* Run shmem_atomic_fetch_and() test */
00890       shmem_barrier_all();
00891       if (!check_if_exists("shmem_ulong_atomic_fetch_and")) {
00892         if (mype == 0) {
00893           display_not_found_warning("shmem_atomic_fetch_and()", false);
00894         }
00895       }
00896       else {
00897         bool result_shmem_atomic_fetch_and = test_shmem_atomic_fetch_and();
00898         shmem_barrier_all();
00899         if (mype == 0) {
00900           display_test_result("shmem_atomic_fetch_and()", result_shmem_atomic_fetch_and, false);
00901         }
00902       }
00903
00904       /* Run shmem_atomic_and() test */
00905       shmem_barrier_all();
00906       if (!check_if_exists("shmem_ulong_atomic_and")) {
00907         if (mype == 0) {
00908           display_not_found_warning("shmem_atomic_and()", false);
00909         }
00910       }
00911       else {
```

```
00912         bool result_shmem_atomic_and = test_shmem_atomic_and();
00913         shmem_barrier_all();
00914         if (mype == 0) {
00915           display_test_result("shmem_atomic_and()", result_shmem_atomic_and, false);
00916         }
00917       }
00918
00919       /* Run shmem_atomic_fetch_or() test */
00920       shmem_barrier_all();
00921       if (!check_if_exists("shmem_ulong_atomic_fetch_or")) {
00922         if (mype == 0) {
00923           display_not_found_warning("shmem_atomic_fetch_or()", false);
00924         }
00925       }
00926       else {
00927         bool result_shmem_atomic_fetch_or = test_shmem_atomic_fetch_or();
00928         shmem_barrier_all();
00929         if (mype == 0) {
00930           display_test_result("shmem_atomic_fetch_or()", result_shmem_atomic_fetch_or, false);
00931         }
00932       }
00933
00934       /* Run shmem_atomic_or() test */
00935       shmem_barrier_all();
00936       if (!check_if_exists("shmem_ulong_atomic_or")) {
00937         if (mype == 0) {
00938           display_not_found_warning("shmem_atomic_or()", false);
00939         }
00940       }
00941       else {
00942         bool result_shmem_atomic_or = test_shmem_atomic_or();
00943         shmem_barrier_all();
00944         if (mype == 0) {
00945           display_test_result("shmem_atomic_or()", result_shmem_atomic_or, false);
00946         }
00947       }
00948
00949       /* Run shmem_atomic_fetch_xor() test */
00950       shmem_barrier_all();
00951       if (!check_if_exists("shmem_ulong_atomic_fetch_xor")) {
00952         if (mype == 0) {
00953           display_not_found_warning("shmem_atomic_fetch_xor()", false);
00954         }
00955       }
00956       else {
00957         bool result_shmem_atomic_fetch_xor = test_shmem_atomic_fetch_xor();
00958         shmem_barrier_all();
00959         if (mype == 0) {
00960           display_test_result("shmem_atomic_fetch_xor()", result_shmem_atomic_fetch_xor, false);
00961         }
00962       }
00963
00964       /* Run shmem_atomic_xor() test */
00965       shmem_barrier_all();
00966       if (!check_if_exists("shmem_ulong_atomic_xor")) {
00967         if (mype == 0) {
00968           display_not_found_warning("shmem_atomic_xor()", false);
00969         }
00970       }
00971       else {
00972         bool result_shmem_atomic_xor = test_shmem_atomic_xor();
00973         shmem_barrier_all();
00974         if (mype == 0) {
00975           display_test_result("shmem_atomic_xor()", result_shmem_atomic_xor, false);
00976         }
00977       }
00978
00979       /* Run shmem_atomic_fetch_nbi() test */
00980       shmem_barrier_all();
00981       if (!check_if_exists("shmem_ulong_atomic_fetch_nbi")) {
00982         if (mype == 0) {
00983           display_not_found_warning("shmem_atomic_fetch_nbi()", false);
00984         }
00985       }
00986       else {
00987         bool result_shmem_atomic_fetch_nbi = test_shmem_atomic_fetch_nbi();
00988         shmem_barrier_all();
00989         if (mype == 0) {
00990           display_test_result("shmem_atomic_fetch_nbi()", result_shmem_atomic_fetch_nbi, false);
00991         }
00992       }
00993
00994       /* Run shmem_atomic_compare_swap_nbi() test */
00995       shmem_barrier_all();
00996       if (!check_if_exists("shmem_ulong_atomic_compare_swap_nbi")) {
00997         if (mype == 0) {
00998           display_not_found_warning("shmem_atomic_compare_swap_nbi()", false);
```

```
00999         }
01000       }
01001       else {
01002         bool result_shmem_atomic_compare_swap_nbi = test_shmem_atomic_compare_swap_nbi();
01003         shmem_barrier_all();
01004         if (mype == 0) {
01005           display_test_result("shmem_atomic_compare_swap_nbi()", result_shmem_atomic_compare_swap_nbi,
      false);
01006         }
01007       }
01008
01009       /* Run shmem_atomic_swap_nbi() test */
01010       shmem_barrier_all();
01011       if (!check_if_exists("shmem_ulong_atomic_swap_nbi")) {
01012         if (mype == 0) {
01013           display_not_found_warning("shmem_atomic_swap_nbi()", false);
01014         }
01015       }
01016       else {
01017         bool result_shmem_atomic_swap_nbi = test_shmem_atomic_swap_nbi();
01018         shmem_barrier_all();
01019         if (mype == 0) {
01020           display_test_result("shmem_atomic_swap_nbi()", result_shmem_atomic_swap_nbi, false);
01021         }
01022       }
01023
01024       /* Run shmem_atomic_fetch_inc_nbi() test */
01025       shmem_barrier_all();
01026       if (!check_if_exists("shmem_ulong_atomic_fetch_inc_nbi")) {
01027         if (mype == 0) {
01028           display_not_found_warning("shmem_atomic_fetch_inc_nbi()", false);
01029         }
01030       }
01031       else {
01032         bool result_shmem_atomic_fetch_inc_nbi = test_shmem_atomic_fetch_inc_nbi();
01033         shmem_barrier_all();
01034         if (mype == 0) {
01035           display_test_result("shmem_atomic_fetch_inc_nbi()", result_shmem_atomic_fetch_inc_nbi,
      false);
01036         }
01037       }
01038
01039       /* Run shmem_atomic_fetch_add_nbi() test */
01040       shmem_barrier_all();
01041       if (!check_if_exists("shmem_ulong_atomic_fetch_add_nbi")) {
01042         if (mype == 0) {
01043           display_not_found_warning("shmem_atomic_fetch_add_nbi()", false);
01044         }
01045       }
01046       else {
01047         bool result_shmem_atomic_fetch_add_nbi = test_shmem_atomic_fetch_add_nbi();
01048         shmem_barrier_all();
01049         if (mype == 0) {
01050           display_test_result("shmem_atomic_fetch_add_nbi()", result_shmem_atomic_fetch_add_nbi,
      false);
01051         }
01052       }
01053
01054       /* Run shmem_atomic_fetch_and_nbi() test */
01055       shmem_barrier_all();
01056       if (!check_if_exists("shmem_ulong_atomic_fetch_and_nbi")) {
01057         if (mype == 0) {
01058           display_not_found_warning("shmem_atomic_fetch_and_nbi()", false);
01059         }
01060       }
01061       else {
01062         bool result_shmem_atomic_fetch_and_nbi = test_shmem_atomic_fetch_and_nbi();
01063         shmem_barrier_all();
01064         if (mype == 0) {
01065           display_test_result("shmem_atomic_fetch_and_nbi()", result_shmem_atomic_fetch_and_nbi,
      false);
01066         }
01067       }
01068
01069       /* Run shmem_atomic_fetch_or_nbi() test */
01070       shmem_barrier_all();
01071       if (!check_if_exists("shmem_ulong_atomic_fetch_or_nbi")) {
01072         if (mype == 0) {
01073           display_not_found_warning("shmem_atomic_fetch_or_nbi()", false);
01074         }
01075       }
01076       else {
01077         bool result_shmem_atomic_fetch_or_nbi = test_shmem_atomic_fetch_or_nbi();
01078         shmem_barrier_all();
01079         if (mype == 0) {
01080           display_test_result("shmem_atomic_fetch_or_nbi()", result_shmem_atomic_fetch_or_nbi, false);
01081         }
```

```
01082       }
01083
01084       /* Run shmem_atomic_fetch_xor_nbi() test */
01085       shmem_barrier_all();
01086       if (!check_if_exists("shmem_ulong_atomic_fetch_xor_nbi")) {
01087         if (mype == 0) {
01088           display_not_found_warning("shmem_atomic_fetch_xor_nbi()", false);
01089         }
01090       }
01091       else {
01092         bool result_shmem_atomic_fetch_xor_nbi = test_shmem_atomic_fetch_xor_nbi();
01093         shmem_barrier_all();
01094         if (mype == 0) {
01095           display_test_result("shmem_atomic_fetch_xor_nbi()", result_shmem_atomic_fetch_xor_nbi,
      false);
01096         }
01097       }
01098     }
01099   }
01100
01101   /************************ SIGNALING TESTS *************************/
01102   if (opts.test_signaling) {
01103     shmem_barrier_all();
01104     if (mype == 0) {
01105       display_test_header("SIGNALING OPS");
01106     }
01107
01108     if (!(npes > 1)) {
01109       display_not_enough_pes("SIGNALING OPS");
01110     }
01111     else {
01112       /* Run shmem_put_signal() test */
01113       shmem_barrier_all();
01114       if (!check_if_exists("shmem_long_put_signal")) {
01115         if (mype == 0) {
01116           display_not_found_warning("shmem_long_put_signal()", false);
01117         }
01118       }
01119       else {
01120         bool result_shmem_put_signal = test_shmem_put_signal();
01121         shmem_barrier_all();
01122         if (mype == 0) {
01123           display_test_result("shmem_put_signal()", result_shmem_put_signal, false);
01124         }
01125       }
01126
01127       /* Run shmem_put_signal_nbi() test */
01128       shmem_barrier_all();
01129       if (!check_if_exists("shmem_long_put_signal_nbi")) {
01130         if (mype == 0) {
01131           display_not_found_warning("shmem_long_put_signal_nbi()", false);
01132         }
01133       }
01134       else {
01135         bool result_shmem_put_signal_nbi = test_shmem_put_signal_nbi();
01136         shmem_barrier_all();
01137         if (mype == 0) {
01138           display_test_result("shmem_put_signal_nbi()", result_shmem_put_signal_nbi, false);
01139         }
01140       }
01141
01142       /* Run shmem_signal_fetch() test */
01143       shmem_barrier_all();
01144       if (!check_if_exists("shmem_signal_fetch")) {
01145         if (mype == 0) {
01146           display_not_found_warning("shmem_signal_fetch()", false);
01147         }
01148       }
01149       else {
01150         bool result_shmem_signal_fetch = test_shmem_signal_fetch();
01151         shmem_barrier_all();
01152         if (mype == 0) {
01153           display_test_result("shmem_signal_fetch()", result_shmem_signal_fetch, false);
01154         }
01155       }
01156     }
01157   }
01158
01159   /************************ COLLECTIVES TESTS *************************/
01160   if (opts.test_collectives) {
01161     /* Print project header */
01162     shmem_barrier_all();
01163     if (mype == 0) {
01164       display_test_header("COLLECTIVE OPS");
01165     }
01166
01167     /* Check to make sure there are at least 2 PEs */
```

```
01168     if ( !(npes > 1) ) {
01169       if (mype == 0) {
01170         display_not_enough_pes("COLLECTIVE OPS");
01171       }
01172     }
01173     else {
01174       /* Run shmem_sync() test */
01175       shmem_barrier_all();
01176       if (!check_if_exists("shmem_sync")) {
01177         if (mype == 0) {
01178           display_not_found_warning("shmem_sync()", false);
01179         }
01180       }
01181       else {
01182         bool result_shmem_sync = test_shmem_sync();
01183         shmem_barrier_all();
01184         if (mype == 0) {
01185           display_test_result("shmem_sync()", result_shmem_sync, false);
01186         }
01187       }
01188
01189       /* Run shmem_sync_all() test */
01190       shmem_barrier_all();
01191       if (!check_if_exists("shmem_sync_all")) {
01192         if (mype == 0) {
01193           display_not_found_warning("shmem_sync_all()", false);
01194         }
01195       }
01196       else {
01197         bool result_shmem_sync_all = test_shmem_sync_all();
01198         shmem_barrier_all();
01199         if (mype == 0) {
01200           display_test_result("shmem_sync_all()", result_shmem_sync_all, false);
01201         }
01202       }
01203
01204       /* Run shmem_alltoall() test */
01205       shmem_barrier_all();
01206       if (!check_if_exists("shmem_long_alltoall")) {
01207         if (mype == 0) {
01208           display_not_found_warning("shmem_long_alltoall()", false);
01209         }
01210       }
01211       else {
01212         bool result_shmem_alltoall = test_shmem_alltoall();
01213         shmem_barrier_all();
01214         if (mype == 0) {
01215           display_test_result("shmem_alltoall()", result_shmem_alltoall, false);
01216         }
01217       }
01218
01219       /* Run shmem_alltoalls() test */
01220       shmem_barrier_all();
01221       if (!check_if_exists("shmem_long_alltoalls")) {
01222         if (mype == 0) {
01223           display_not_found_warning("shmem_long_alltoalls()", false);
01224         }
01225       }
01226       else {
01227         bool result_shmem_alltoalls = test_shmem_alltoalls();
01228         shmem_barrier_all();
01229         if (mype == 0) {
01230           display_test_result("shmem_alltoalls()", result_shmem_alltoalls, false);
01231         }
01232       }
01233
01234       /* Run shmem_broadcast() test */
01235       shmem_barrier_all();
01236       if (!check_if_exists("shmem_long_broadcast")) {
01237         if (mype == 0) {
01238           display_not_found_warning("shmem_long_broadcast()", false);
01239         }
01240       }
01241       else {
01242         bool result_shmem_broadcast = test_shmem_broadcast();
01243         shmem_barrier_all();
01244         if (mype == 0) {
01245           display_test_result("shmem_broadcast()", result_shmem_broadcast, false);
01246         }
01247       }
01248
01249       /* Run shmem_collect() test */
01250       shmem_barrier_all();
01251       if (!check_if_exists("shmem_long_collect")) {
01252         if (mype == 0) {
01253           display_not_found_warning("shmem_long_collect()", false);
01254         }
```

```
01255       }
01256       else {
01257         bool result_shmem_collect = test_shmem_collect();
01258         shmem_barrier_all();
01259         if (mype == 0) {
01260           display_test_result("shmem_collect()", result_shmem_collect, false);
01261         }
01262       }
01263
01264       /* Run shmem_fcollect() test */
01265       shmem_barrier_all();
01266       if (!check_if_exists("shmem_long_fcollect")) {
01267         if (mype == 0) {
01268           display_not_found_warning("shmem_long_fcollect()", false);
01269         }
01270       }
01271       else {
01272         bool result_shmem_fcollect = test_shmem_fcollect();
01273         shmem_barrier_all();
01274         if (mype == 0) {
01275           display_test_result("shmem_fcollect()", result_shmem_fcollect, false);
01276         }
01277       }
01278
01279       /* Run shmem_max_reduce() test */
01280       shmem_barrier_all();
01281       if (!check_if_exists("shmem_long_max_reduce")) {
01282         if (mype == 0) {
01283           display_not_found_warning("shmem_long_max_reduce()", false);
01284         }
01285       }
01286       else {
01287         bool result_shmem_max_reduce = test_shmem_max_reduce();
01288         shmem_barrier_all();
01289         if (mype == 0) {
01290           display_test_result("shmem_max_reduce()", result_shmem_max_reduce, false);
01291         }
01292       }
01293
01294       /* Run shmem_min_reduce() test */
01295       shmem_barrier_all();
01296       if (!check_if_exists("shmem_long_min_reduce")) {
01297         if (mype == 0) {
01298           display_not_found_warning("shmem_long_min_reduce()", false);
01299         }
01300       }
01301       else {
01302         bool result_shmem_min_reduce = test_shmem_min_reduce();
01303         shmem_barrier_all();
01304         if (mype == 0) {
01305           display_test_result("shmem_min_reduce()", result_shmem_min_reduce, false);
01306         }
01307       }
01308
01309       /* Run shmem_sum_reduce() test */
01310       shmem_barrier_all();
01311       if (!check_if_exists("shmem_long_sum_reduce")) {
01312         if (mype == 0) {
01313           display_not_found_warning("shmem_long_sum_reduce()", false);
01314         }
01315       }
01316       else {
01317         bool result_shmem_sum_reduce = test_shmem_sum_reduce();
01318         shmem_barrier_all();
01319         if (mype == 0) {
01320           display_test_result("shmem_sum_reduce()", result_shmem_sum_reduce, false);
01321         }
01322       }
01323
01324       /* Run shmem_prod_reduce() test */
01325       shmem_barrier_all();
01326       if (!check_if_exists("shmem_long_prod_reduce")) {
01327         if (mype == 0) {
01328           display_not_found_warning("shmem_long_prod_reduce()", false);
01329         }
01330       }
01331       else {
01332         bool result_shmem_prod_reduce = test_shmem_prod_reduce();
01333         shmem_barrier_all();
01334         if (mype == 0) {
01335           display_test_result("shmem_prod_reduce()", result_shmem_prod_reduce, false);
01336         }
01337       }
01338     }
01339   }
01340
01341   /************************ PT2PT TESTS *************************/
```

```
01342   if (opts.test_pt2pt_synch) {
01343     shmem_barrier_all();
01344     if (mype == 0) {
01345       display_test_header("POINT-TO-POINT SYNC OPS");
01346     }
01347
01348     if (!(npes > 1)) {
01349       display_not_enough_pes("POINT-TO-POINT SYNCH OPS");
01350     }
01351     else {
01352       /* Run shmem_wait_until() test */
01353       shmem_barrier_all();
01354       if (!check_if_exists("shmem_long_wait_until")) {
01355         if (mype == 0) {
01356           display_not_found_warning("shmem_long_wait_until()", false);
01357         }
01358       }
01359       else {
01360         bool result_shmem_wait_until = test_shmem_wait_until();
01361         shmem_barrier_all();
01362         if (mype == 0) {
01363           display_test_result("shmem_wait_until()", result_shmem_wait_until, false);
01364         }
01365       }
01366
01367       /* Run shmem_wait_until_all() test */
01368       shmem_barrier_all();
01369       if (!check_if_exists("shmem_long_wait_until_all")) {
01370         if (mype == 0) {
01371           display_not_found_warning("shmem_long_wait_until_all()", false);
01372         }
01373       }
01374       else {
01375         bool result_shmem_wait_until_all = test_shmem_wait_until_all();
01376         shmem_barrier_all();
01377         if (mype == 0) {
01378           display_test_result("shmem_wait_until_all()", result_shmem_wait_until_all, false);
01379         }
01380       }
01381
01382       /* Run shmem_wait_until_any() test */
01383       shmem_barrier_all();
01384       if (!check_if_exists("shmem_long_wait_until_any")) {
01385         if (mype == 0) {
01386           display_not_found_warning("shmem_long_wait_until_any()", false);
01387         }
01388       }
01389       else {
01390         bool result_shmem_wait_until_any = test_shmem_wait_until_any();
01391         shmem_barrier_all();
01392         if (mype == 0) {
01393           display_test_result("shmem_wait_until_any()", result_shmem_wait_until_any, false);
01394         }
01395       }
01396
01397       /* Run shmem_wait_until_some() test */
01398       shmem_barrier_all();
01399       if (!check_if_exists("shmem_long_wait_until_some")) {
01400         if (mype == 0) {
01401           display_not_found_warning("shmem_long_wait_until_some()", false);
01402         }
01403       }
01404       else {
01405         bool result_shmem_wait_until_some = test_shmem_wait_until_some();
01406         shmem_barrier_all();
01407         if (mype == 0) {
01408           display_test_result("shmem_wait_until_some()", result_shmem_wait_until_some, false);
01409         }
01410       }
01411
01412       /* Run shmem_wait_until_all_vector() test */
01413       shmem_barrier_all();
01414       if (!check_if_exists("shmem_long_wait_until_all_vector")) {
01415         if (mype == 0) {
01416           display_not_found_warning("shmem_long_wait_until_all_vector()", false);
01417         }
01418       }
01419       else {
01420         bool result_shmem_wait_until_all_vector = test_shmem_wait_until_all_vector();
01421         shmem_barrier_all();
01422         if (mype == 0) {
01423           display_test_result("shmem_wait_until_all_vector()", result_shmem_wait_until_all_vector,
      false);
01424         }
01425       }
01426
01427       /* Run shmem_wait_until_any_vector() test */
```

```
01428       shmem_barrier_all();
01429       if (!check_if_exists("shmem_long_wait_until_any_vector")) {
01430         if (mype == 0) {
01431           display_not_found_warning("shmem_long_wait_until_any_vector()", false);
01432         }
01433       }
01434       else {
01435         bool result_shmem_wait_until_any_vector = test_shmem_wait_until_any_vector();
01436         shmem_barrier_all();
01437         if (mype == 0) {
01438           display_test_result("shmem_wait_until_any_vector()", result_shmem_wait_until_any_vector,
      false);
01439         }
01440       }
01441
01442       /* Run shmem_wait_until_some_vector() test */
01443       shmem_barrier_all();
01444       if (!check_if_exists("shmem_long_wait_until_some_vector")) {
01445         if (mype == 0) {
01446           display_not_found_warning("shmem_long_wait_until_some_vector()", false);
01447         }
01448       }
01449       else {
01450         bool result_shmem_wait_until_some_vector = test_shmem_wait_until_some_vector();
01451         shmem_barrier_all();
01452         if (mype == 0) {
01453           display_test_result("shmem_wait_until_some_vector()", result_shmem_wait_until_some_vector,
      false);
01454         }
01455       }
01456
01457       /* Run shmem_test() test */
01458       shmem_barrier_all();
01459       if (!check_if_exists("shmem_long_test")) {
01460         if (mype == 0) {
01461           display_not_found_warning("shmem_long_test()", false);
01462         }
01463       }
01464       else {
01465         bool result_shmem_test = test_shmem_test();
01466         shmem_barrier_all();
01467         if (mype == 0) {
01468           display_test_result("shmem_test()", result_shmem_test, false);
01469         }
01470       }
01471
01472       /* Run shmem_test_all() test */
01473       shmem_barrier_all();
01474       if (!check_if_exists("shmem_long_test_all")) {
01475         if (mype == 0) {
01476           display_not_found_warning("shmem_long_test_all()", false);
01477         }
01478       }
01479       else {
01480         bool result_shmem_test_all = test_shmem_test_all();
01481         shmem_barrier_all();
01482         if (mype == 0) {
01483           display_test_result("shmem_test_all()", result_shmem_test_all, false);
01484         }
01485       }
01486
01487       /* Run shmem_test_any() test */
01488       shmem_barrier_all();
01489       if (!check_if_exists("shmem_long_test_any")) {
01490         if (mype == 0) {
01491           display_not_found_warning("shmem_long_test_any()", false);
01492         }
01493       }
01494       else {
01495         bool result_shmem_test_any = test_shmem_test_any();
01496         shmem_barrier_all();
01497         if (mype == 0) {
01498           display_test_result("shmem_test_any()", result_shmem_test_any, false);
01499         }
01500       }
01501
01502       /* Run shmem_test_some() test */
01503       shmem_barrier_all();
01504       if (!check_if_exists("shmem_long_test_some")) {
01505         if (mype == 0) {
01506           display_not_found_warning("shmem_long_test_some()", false);
01507         }
01508       }
01509       else {
01510         bool result_shmem_test_some = test_shmem_test_some();
01511         shmem_barrier_all();
01512         if (mype == 0) {
```

```
01513           display_test_result("shmem_test_some()", result_shmem_test_some, false);
01514         }
01515       }
01516
01517       /* Run shmem_test_all_vector() test */
01518       shmem_barrier_all();
01519       if (!check_if_exists("shmem_long_test_all_vector")) {
01520         if (mype == 0) {
01521           display_not_found_warning("shmem_long_test_all_vector()", false);
01522         }
01523       }
01524       else {
01525         bool result_shmem_test_all_vector = test_shmem_test_all_vector();
01526         shmem_barrier_all();
01527         if (mype == 0) {
01528           display_test_result("shmem_test_all_vector()", result_shmem_test_all_vector, false);
01529         }
01530       }
01531
01532       /* Run shmem_test_any_vector() test */
01533       shmem_barrier_all();
01534       if (!check_if_exists("shmem_long_test_any_vector")) {
01535         if (mype == 0) {
01536           display_not_found_warning("shmem_long_test_any_vector()", false);
01537         }
01538       }
01539       else {
01540         bool result_shmem_test_any_vector = test_shmem_test_any_vector();
01541         shmem_barrier_all();
01542         if (mype == 0) {
01543           display_test_result("shmem_test_any_vector()", result_shmem_test_any_vector, false);
01544         }
01545       }
01546
01547       /* Run shmem_test_some_vector() test */
01548       shmem_barrier_all();
01549       if (!check_if_exists("shmem_long_test_some_vector")) {
01550         if (mype == 0) {
01551           display_not_found_warning("shmem_long_test_some_vector()", false);
01552         }
01553       }
01554       else {
01555         bool result_shmem_test_some_vector = test_shmem_test_some_vector();
01556         shmem_barrier_all();
01557         if (mype == 0) {
01558           display_test_result("shmem_test_some_vector()", result_shmem_test_some_vector, false);
01559         }
01560       }
01561
01562       /* Run shmem_signal_wait_until() test */
01563       shmem_barrier_all();
01564       if (!check_if_exists("shmem_signal_wait_until")) {
01565         if (mype == 0) {
01566           display_not_found_warning("shmem_signal_wait_until()", false);
01567         }
01568       }
01569       else {
01570         bool result_shmem_signal_wait_until = test_shmem_signal_wait_until();
01571         shmem_barrier_all();
01572         if (mype == 0) {
01573           display_test_result("shmem_signal_wait_until()", result_shmem_signal_wait_until, false);
01574         }
01575       }
01576     }
01577   }
01578
01579   /************************ MEM ORDERING TESTS *************************/
01580   if (opts.test_mem_ordering) {
01581     shmem_barrier_all();
01582     if (mype == 0) {
01583       display_test_header("MEMORY ORDERING");
01584     }
01585
01586     /* Make sure there are at least 2 PEs */
01587     if ( !(npes > 1) ) {
01588       if (mype == 0) {
01589         display_not_enough_pes("MEMORY ORDERING");
01590       }
01591     }
01592     else {
01593       /* Run the shmem_fence() test */
01594       shmem_barrier_all();
01595       if ( !check_if_exists("shmem_fence") ) {
01596         if (mype == 0) {
01597           display_not_found_warning("shmem_fence()", false);
01598         }
01599       }
```

```
01600       else {
01601         bool result_shmem_fence = test_shmem_fence();
01602         shmem_barrier_all();
01603         if (mype == 0) {
01604           display_test_result("shmem_fence()", result_shmem_fence, false);
01605         }
01606       }
01607
01608       /* Run the shmem_quiet() test */
01609       shmem_barrier_all();
01610       if ( !check_if_exists("shmem_quiet") ) {
01611         if (mype == 0) {
01612           display_not_found_warning("shmem_quiet()", false);
01613         }
01614       }
01615       else {
01616         bool result_shmem_quiet = test_shmem_quiet();
01617         shmem_barrier_all();
01618         if (mype == 0) {
01619           display_test_result("shmem_quiet()", result_shmem_quiet, false);
01620         }
01621       }
01622     }
01623   }
01624
01625   /************************ DISTRIBUTED LOCKING TESTS *************************/
01626   if (opts.test_locking) {
01627     shmem_barrier_all();
01628     if (mype == 0) {
01629       display_test_header("DISTRIBUTED LOCKING");
01630     }
01631     shmem_barrier_all();
01632
01633     /* Make sure there are at least 2 PEs */
01634     if ( !(npes > 1) ) {
01635       if (mype == 0) {
01636         display_not_enough_pes("DISTRIBUTED LOCKING");
01637       }
01638     }
01639     else {
01640       /* Run the shmem_set_lock and shmem_clear_lock tests */
01641       shmem_barrier_all();
01642       if ( !check_if_exists("shmem_set_lock") ) {
01643         if (mype == 0) {
01644           display_not_found_warning("shmem_set_lock()", false);
01645         }
01646       }
01647       if ( !check_if_exists("shmem_clear_lock" )) {
01648         if (mype == 0) {
01649           display_not_found_warning("shmem_clear_lock()", false);
01650         }
01651       }
01652
01653       shmem_barrier_all();
01654       if ( check_if_exists("shmem_set_lock") && check_if_exists("shmem_clear_lock") ) {
01655         bool result_shmem_lock_unlock = test_shmem_lock_unlock();
01656         shmem_barrier_all();
01657         if (mype == 0) {
01658           display_test_result("shmem_set_lock()", result_shmem_lock_unlock, false);
01659           display_test_result("shmem_clear_lock()", result_shmem_lock_unlock, false);
01660         }
01661       }
01662     }
01663   }
01664
01665   /************************ FINALIZATION **************************/
01666   /* Run shmem_finalize() test */
01667   shmem_barrier_all();
01668
01669   if ( !check_if_exists("shmem_finalize") ) {
01670     display_not_found_warning("shmem_finalize()", true);
01671   }
01672   else {
01673     if (mype == 0) {
01674       display_test_header("FINALIZATION");
01675       display_test_result("shmem_finalize()", test_shmem_finalize(), false);
01676       std::cout « std::endl;
01677     }
01678   }
01679
01680   /* We made it! End the program. */
01681   return EXIT_SUCCESS;
01682 }
```

References check_if_exists(), display_help(), display_logo(), display_not_enough_pes(), display_not_found_warning(), display_test_header(), display_test_info(), display_test_result(), test_options::help, load_routines(), parse_opts(),

test_options::test_all, test_options::test_atomics, test_options::test_collectives, test_options::test_ctx, test_options::test_locking, test_options::test_mem, test_options::test_mem_ordering, test_options::test_pt2pt_synch, test_options::test_remote, test_options::test_setup, test_shmem_addr_accessible(), test_shmem_align(), test_shmem_alltoall(), test_shmem_alltoalls(), test_shmem_atomic_add(), test_shmem_atomic_and(), test_shmem_atomic_compare_swap(), test_shmem_atomic_compare_swap_ test_shmem_atomic_fetch(), test_shmem_atomic_fetch_add(), test_shmem_atomic_fetch_add_nbi(), test_shmem_atomic_fetch_and test_shmem_atomic_fetch_and_nbi(), test_shmem_atomic_fetch_inc(), test_shmem_atomic_fetch_inc_nbi(), test_shmem_atomic_fetch_nbi(), test_shmem_atomic_fetch_or(), test_shmem_atomic_fetch_or_nbi(), test_shmem_atomic_fetch_xo test_shmem_atomic_fetch_xor_nbi(), test_shmem_atomic_inc(), test_shmem_atomic_or(), test_shmem_atomic_set(), test_shmem_atomic_swap(), test_shmem_atomic_swap_nbi(), test_shmem_atomic_xor(), test_shmem_barrier(), test_shmem_barrier_all(), test_shmem_broadcast(), test_shmem_calloc(), test_shmem_collect(), test_shmem_ctx_create(), test_shmem_ctx_destroy(), test_shmem_ctx_get_team(), test_shmem_fake_routine(), test_shmem_fcollect(), test_shmem_fence(), test_shmem_finalize(), test_shmem_g(), test_shmem_get(), test_shmem_get_nbi(), test_shmem_iget(), test_shmem_info_get_name(), test_shmem_info_get_version(), test_shmem_init(), test_shmem_init_thread(), test_shmem_iput(), test_shmem_lock_unlock(), test_shmem_malloc_free(), test_shmem_malloc_with_hints(), test_shmem_max_reduce(), test_shmem_min_reduce(), test_shmem_my_pe(), test_shmem_n_pes(), test_shmem_p(), test_shmem_pe_accessible(), test_shmem_prod_reduce(), test_shmem_ptr(), test_shmem_put(), test_shmem_put_nbi(), test_shmem_put_signal(), test_shmem_put_signal_nbi(), test_shmem_query_thread(), test_shmem_quiet(), test_shmem_realloc(), test_shmem_signal_fetch(), test_shmem_signal_wait_until(), test_shmem_sum_reduce(), test_shmem_sync(), test_shmem_sync_all(), test_shmem_team_create_ctx(), test_shmem_team_destroy(), test_shmem_team_get_config(), test_shmem_team_my_pe(), test_shmem_team_n_pes(), test_shmem_team_split_2d(), test_shmem_team_split_strided(), test_shmem_team_translate_pe(), test_shmem_test(), test_shmem_test_all(), test_shmem_test_all_vector(), test_shmem_test_any(), test_shmem_test_any_vector(), test_shmem_test_some(), test_shmem_test_some_vector(), test_shmem_wait_until(), test_shmem_wait_until_all(), test_shmem_wait_until_all_vector(), test_shmem_wait_until_any(), test_shmem_wait_until_any_vector(), test_shmem_wait_until_some(), test_shmem_wait_until_some_v test_options::test_signaling, test_options::test_teams, test_options::test_threads, and YELLOW_COLOR.

4.6 main.cpp

Go to the documentation of this file.

```
00001
00006 #include "shmemvv.hpp"
00007 #include "routines.hpp"
00008
00015 int main(int argc, char *argv[]) {
00016   int mype = 0;
00017   int npes = 0;
00018   std::string version = "";
00019   std::string name = "";
00020   test_options opts;
00021
00022   /* Variables to hold test results */
00023   bool result_shmem_init = true;
00024   bool result_shmem_init_thread = true;
00025   bool result_shmem_barrier_all = true;
00026   bool result_shmem_barrier = true;
00027   bool result_shmem_my_pe = true;
00028   bool result_shmem_n_pes = true;
00029   bool result_shmem_pe_accessible = true;
00030   bool result_shmem_info_get_version = true;
00031   bool result_shmem_info_get_name = true;
00032
00033   /************************ SETUP *************************/
00034   void *handle = dlopen(NULL, RTLD_LAZY);
00035   if (!handle) {
00036     if (mype == 0) {
00037       std::cerr << "Failed to open handle: " << dlerror() << std::endl;
00038       return EXIT_FAILURE;
00039     }
00040   }
00041
00042   /* Load OpenSHMEM routines */
00043   if (!load_routines()) {
00044     std::cerr << "Failed to load OpenSHMEM routines" << std::endl;
00045     return EXIT_FAILURE;
00046   }
00047
00048   /* Initialize with shmem_init_thread() if THREADS tests were enabled */
00049   if (opts.test_threads) {
00050     if (!check_if_exists("shmem_init_thread")) {
```

```
00051       if (mype == 0) {
00052         display_not_found_warning("shmem_init_thread()", true);
00053       }
00054       return EXIT_FAILURE;
00055     }
00056     else {
00057       result_shmem_init_thread = test_shmem_init_thread();
00058       if (!result_shmem_init_thread) {
00059         display_test_result("shmem_init_thread()", result_shmem_init_thread, true);
00060         return EXIT_FAILURE;
00061       }
00062     }
00063   }
00064   else {
00065     /* Initialize with regular shmem_init() if THREADS tests are not enabled */
00066     if (!check_if_exists("shmem_init")) {
00067       if (mype == 0) {
00068         display_not_found_warning("shmem_init()", true);
00069       }
00070       return EXIT_FAILURE;
00071     }
00072     else {
00073       result_shmem_init = test_shmem_init();
00074       if (!result_shmem_init) {
00075         display_test_result("shmem_init()", result_shmem_init, true);
00076         return EXIT_FAILURE;
00077       }
00078     }
00079   }
00080
00081   /* Run shmem_barrier_all() test */
00082   if (!check_if_exists("shmem_barrier_all")) {
00083     if (mype == 0) {
00084       display_not_found_warning("shmem_barrier_all()", true);
00085     }
00086     shmem_finalize();
00087     return EXIT_FAILURE;
00088   }
00089   else {
00090     result_shmem_barrier_all = test_shmem_barrier_all();
00091     if (!result_shmem_barrier_all) {
00092       if (shmem_my_pe() == 0) {
00093         display_test_result("shmem_barrier_all()", result_shmem_barrier_all, true);
00094       }
00095       shmem_finalize();
00096       return EXIT_FAILURE;
00097     }
00098   }
00099
00100   /* Run shmem_my_pe() test */
00101   shmem_barrier_all();
00102   if (!check_if_exists("shmem_my_pe")) {
00103     if (mype == 0) {
00104       display_not_found_warning("shmem_my_pe()", true);
00105     }
00106     shmem_finalize();
00107     return EXIT_FAILURE;
00108   }
00109   else {
00110     mype = test_shmem_my_pe();
00111     result_shmem_my_pe = mype >= 0;
00112     if (!result_shmem_my_pe) {
00113       if (mype == 0) {
00114         display_test_result("shmem_my_pe()", result_shmem_my_pe, true);
00115       }
00116       shmem_finalize();
00117       return EXIT_FAILURE;
00118     }
00119   }
00120
00121   /* Run shmem_n_pes() test */
00122   shmem_barrier_all();
00123   if (!check_if_exists("shmem_n_pes")) {
00124     if (mype == 0) {
00125       display_not_found_warning("shmem_n_pes", true);
00126     }
00127     shmem_finalize();
00128     return EXIT_FAILURE;
00129   }
00130   else {
00131     /* Set npes */
00132     npes = test_shmem_n_pes();
00133     result_shmem_n_pes = npes > 0;
00134     if (!result_shmem_n_pes) {
00135       if (mype == 0) {
00136         display_test_result("shmem_n_pes()", result_shmem_n_pes, true);
00137       }
```

```
00138       shmem_finalize();
00139       return EXIT_FAILURE;
00140     }
00141   }
00142
00143   /* Run shmem_pe_accessible() test */
00144   shmem_barrier_all();
00145   if (!check_if_exists("shmem_pe_accessible")) {
00146     if (mype == 0) {
00147       display_not_found_warning("shmem_pe_accessible()", false);
00148     }
00149   }
00150   else {
00151     result_shmem_pe_accessible = test_shmem_pe_accessible();
00152     if (!result_shmem_pe_accessible) {
00153       if (mype == 0) {
00154         display_test_result("shmem_pe_accessible()", result_shmem_pe_accessible, true);
00155       }
00156       shmem_finalize();
00157       return EXIT_FAILURE;
00158     }
00159   }
00160
00161   /*
00162     Run test to make sure OpenSHMEM routines that aren't implemented
00163     don't throw compiler errors
00164   */
00165   #ifdef _DEBUG_
00166     shmem_barrier_all();
00167     if (!check_if_exists("shmem_fake_routine")) {
00168       if (mype == 0) {
00169         display_not_found_warning("shmem_fake_routine()", false);
00170       }
00171     }
00172     else {
00173       test_shmem_fake_routine();
00174     }
00175   #endif
00176
00177   /* Display help if requested */
00178   shmem_barrier_all();
00179   if (opts.help) {
00180     if (mype == 0) {
00181       display_help();
00182     }
00183     shmem_finalize();
00184     return EXIT_SUCCESS;
00185   }
00186
00187   /* Display ASCII art logo */
00188   shmem_barrier_all();
00189   if (mype == 0) {
00190     display_logo();
00191   }
00192
00193   /* Run shmem_barrier() test */
00194   shmem_barrier_all();
00195   if (!check_if_exists("shmem_barrier")) {
00196     if (mype == 0) {
00197       display_not_found_warning("shmem_barrier()", false);
00198     }
00199   }
00200   else {
00201     result_shmem_barrier = test_shmem_barrier();
00202     shmem_barrier_all();
00203   }
00204
00205   /* Run shmem_info_get_version() test */
00206   shmem_barrier_all();
00207   if (!check_if_exists("shmem_info_get_version")) {
00208     result_shmem_info_get_version = false;
00209     if (mype == 0) {
00210       display_not_found_warning("shmem_info_get_version()", false);
00211     }
00212   }
00213   else {
00214     version = test_shmem_info_get_version();
00215     if (version == "") {
00216       result_shmem_info_get_version = false;
00217     }
00218   }
00219
00220   /* Run shmem_info_get_name() test */
00221   shmem_barrier_all();
00222   if (!check_if_exists("shmem_info_get_name")) {
00223     result_shmem_info_get_name = false;
00224     if (mype == 0) {
```

```
00225       display_not_found_warning("shmem_info_get_name()", false);
00226     }
00227   }
00228   else {
00229     name = test_shmem_info_get_name();
00230     if (name == "") {
00231       result_shmem_info_get_name = false;
00232     }
00233   }
00234
00235   /* Parse command-line options */
00236   if (!parse_opts(argc, argv, opts)) {
00237     if (mype == 0) {
00238       display_help();
00239     }
00240     shmem_finalize();
00241     return EXIT_FAILURE;
00242   }
00243
00244   shmem_barrier_all();
00245
00246   /* Enable all tests if --all is specified or no specific test is selected */
00247   if (opts.test_all ||
00248       !(opts.test_setup || opts.test_threads || opts.test_mem || opts.test_teams ||
00249         opts.test_ctx || opts.test_remote || opts.test_atomics || opts.test_signaling ||
00250         opts.test_collectives || opts.test_pt2pt_synch || opts.test_mem_ordering ||
      opts.test_locking))
00251   {
00252     opts.test_setup = true; opts.test_threads = true; opts.test_mem = true; opts.test_teams = true;
00253     opts.test_ctx = true; opts.test_remote = true; opts.test_atomics = true; opts.test_signaling =
      true;
00254     opts.test_collectives = true; opts.test_pt2pt_synch = true; opts.test_mem_ordering = true;
      opts.test_locking = true;
00255   }
00256
00257   /* Display test information */
00258   shmem_barrier_all();
00259   if (mype == 0) {
00260     display_test_info(name, version, npes);
00261   }
00262
00263   /* Print setup tests header */
00264   shmem_barrier_all();
00265   if (mype == 0) {
00266     display_test_header("SETUP");
00267   }
00268
00269   /* shmem_init() and shmem_my_pe() tests passed */
00270   shmem_barrier_all();
00271   if (mype == 0) {
00272     if (!opts.test_threads) {
00273       display_test_result("shmem_init()", result_shmem_init, true);
00274     }
00275     display_test_result("shmem_barrier_all()", result_shmem_barrier_all, true);
00276     display_test_result("shmem_barrier()", result_shmem_barrier, false);
00277     display_test_result("shmem_my_pe()", result_shmem_my_pe, true);
00278     display_test_result("shmem_n_pes()", result_shmem_n_pes, true);
00279     display_test_result("shmem_pe_accessible()", result_shmem_pe_accessible, true);
00280     if (version != "1.5" && version != "1.50") {
00281       std::cerr << YELLOW_COLOR << "shmem_info_get_version() test did not return 1.5... Returned " <<
      version << std::endl;
00282     }
00283     else {
00284       display_test_result("shmem_info_get_version()", result_shmem_info_get_version, false);
00285     }
00286     display_test_result("shmem_info_get_name()", result_shmem_info_get_name,false);
00287   }
00288
00289   /************************ THREADS TESTS *************************/
00290   if (opts.test_threads) {
00291     shmem_barrier_all();
00292     if (mype == 0) {
00293       display_test_header("THREADS");
00294     }
00295     shmem_barrier_all();
00296
00297     /* If we made it here shmem_init_thread() passed */
00298     if (mype == 0) {
00299       display_test_result("shmem_init_thread()", result_shmem_init_thread, true);
00300     }
00301
00302     /* Test shmem_query_thread() */
00303     shmem_barrier_all();
00304     if ( !check_if_exists("shmem_query_thread") ) {
00305       if (mype == 0) {
00306         display_not_found_warning("shmem_query_thread()", false);
00307       }
```

```
00308     }
00309     else {
00310       bool result_shmem_query_thread = test_shmem_query_thread();
00311       shmem_barrier_all();
00312       if (mype == 0) {
00313         display_test_result("shmem_query_thread()", result_shmem_query_thread, false);
00314       }
00315     }
00316   }
00317
00318   /************************ MEMORY TESTS *************************/
00319   if (opts.test_mem) {
00320     shmem_barrier_all();
00321     if (mype == 0) {
00322       display_test_header("MEMORY MANAGEMENT");
00323     }
00324
00325     /* Test shmem_malloc() and shmem_free() */
00326     shmem_barrier_all();
00327     if ( check_if_exists("shmem_malloc") && check_if_exists("shmem_free") ) {
00328       bool result_shmem_malloc_free = test_shmem_malloc_free();
00329       shmem_barrier_all();
00330       if (mype == 0) {
00331         display_test_result("shmem_malloc()", result_shmem_malloc_free, false);
00332         display_test_result("shmem_free()", result_shmem_malloc_free, false);
00333       }
00334     }
00335     else {
00336       if ( !check_if_exists("shmem_malloc") ) {
00337         if (mype == 0) {
00338           display_not_found_warning("shmem_malloc()", false);
00339         }
00340       }
00341       if ( !check_if_exists("shmem_free") ) {
00342         if (mype == 0) {
00343           display_not_found_warning("shmem_free()", false);
00344         }
00345       }
00346     }
00347
00348     /* Test shmem_ptr() */
00349     shmem_barrier_all();
00350     if ( !check_if_exists("shmem_ptr") ) {
00351       if (mype == 0) {
00352         display_not_found_warning("shmem_ptr()", false);
00353       }
00354     }
00355     else {
00356       bool result_shmem_ptr = test_shmem_ptr();
00357       shmem_barrier_all();
00358       if (mype == 0) {
00359         display_test_result("shmem_ptr()", result_shmem_ptr, false);
00360       }
00361     }
00362
00363     /* Test shmem_addr_accessible */
00364     shmem_barrier_all();
00365     if ( !check_if_exists("shmem_addr_accessible") ) {
00366       if (mype == 0) {
00367         display_not_found_warning("shmem_addr_accessible", false);
00368       }
00369     }
00370     else {
00371       bool result_shmem_addr_accessible = test_shmem_addr_accessible();
00372       shmem_barrier_all();
00373       if (mype == 0) {
00374         display_test_result("shmem_addr_accessible()", result_shmem_addr_accessible, false);
00375       }
00376     }
00377
00378     /* Test shmem_realloc() */
00379     shmem_barrier_all();
00380     if ( !check_if_exists("shmem_realloc") ) {
00381       if (mype == 0) {
00382         display_not_found_warning("shme_realloc()", false);
00383       }
00384     }
00385     else {
00386       bool result_shmem_realloc = test_shmem_realloc();
00387       shmem_barrier_all();
00388       if (mype == 0) {
00389         display_test_result("shmem_realloc()", result_shmem_realloc, false);
00390       }
00391     }
00392
00393     /* Test shmem_align() */
00394     shmem_barrier_all();
```

```
00395     if ( !check_if_exists("shmem_align") ) {
00396       if (mype == 0) {
00397         display_not_found_warning("shmem_align()", false);
00398       }
00399     }
00400     else {
00401       bool result_shmem_align = test_shmem_align();
00402       shmem_barrier_all();
00403       if (mype == 0) {
00404         display_test_result("shmem_align()", result_shmem_align, false);
00405       }
00406     }
00407
00408     /* Test shmem_malloc_with_hints() */
00409     shmem_barrier_all();
00410     if ( !check_if_exists("shmem_malloc_with_hints") ) {
00411       if (mype == 0) {
00412         display_not_found_warning("shmem_malloc_with_hints()", false);
00413       }
00414     }
00415     else {
00416       bool result_shmem_malloc_with_hints = test_shmem_malloc_with_hints();
00417       shmem_barrier_all();
00418       if (mype == 0) {
00419         display_test_result("shmem_malloc_with_hints()", result_shmem_malloc_with_hints, false);
00420       }
00421     }
00422
00423     /* Test shmem_calloc() */
00424     shmem_barrier_all();
00425     if ( !check_if_exists("shmem_calloc") ) {
00426       if (mype == 0) {
00427         display_not_found_warning("shmem_calloc()", false);
00428       }
00429     }
00430     else {
00431       bool result_shmem_calloc = test_shmem_calloc();
00432       shmem_barrier_all();
00433       if (mype == 0) {
00434         display_test_result("shmem_calloc()", result_shmem_calloc, false);
00435       }
00436     }
00437   }
00438
00439   /************************ TEAMS TESTS *************************/
00440   if (opts.test_teams) {
00441     shmem_barrier_all();
00442     if (mype == 0) {
00443       display_test_header("TEAMS MANAGEMENT");
00444     }
00445
00446     /* Run shmem_team_my_pe() test */
00447     shmem_barrier_all();
00448     if ( !check_if_exists("shmem_team_my_pe") ) {
00449       if (mype == 0) {
00450         display_not_found_warning("shmem_team_my_pe()", true);
00451       }
00452     }
00453     else {
00454       bool result_shmem_team_my_pe = test_shmem_team_my_pe();
00455       shmem_barrier_all();
00456       if (mype == 0) {
00457         display_test_result("shmem_team_my_pe()", result_shmem_team_my_pe, false);
00458       }
00459     }
00460
00461     /* Run shmem_team_n_pes() test */
00462     shmem_barrier_all();
00463     if ( !check_if_exists("shmem_team_n_pes") ) {
00464       if (mype == 0) {
00465         display_not_found_warning("shmem_team_n_pes()", true);
00466       }
00467     }
00468     else {
00469       bool result_shmem_team_n_pes = test_shmem_team_n_pes();
00470       shmem_barrier_all();
00471       if (mype == 0) {
00472         display_test_result("shmem_team_n_pes()", result_shmem_team_n_pes, false);
00473       }
00474     }
00475
00476     /* Run shmem_team_get_config() test */
00477     shmem_barrier_all();
00478     if ( !check_if_exists("shmem_team_get_config") ) {
00479       if (mype == 0) {
00480         display_not_found_warning("shmem_team_get_config()", false);
00481       }
```

```
00482     }
00483     else {
00484       bool result_shmem_team_get_config = test_shmem_team_get_config();
00485       shmem_barrier_all();
00486       if (mype == 0) {
00487         display_test_result("shmem_team_get_config()", result_shmem_team_get_config, false);
00488       }
00489     }
00490
00491     /* Run shmem_team_translate_pe() test */
00492     shmem_barrier_all();
00493     if ( !check_if_exists("shmem_team_translate_pe") ) {
00494       if (mype == 0) {
00495         display_not_found_warning("shmem_team_translate_pe()", false);
00496       }
00497     }
00498     else {
00499       bool result_shmem_team_translate_pe = test_shmem_team_translate_pe();
00500       shmem_barrier_all();
00501       if (mype == 0) {
00502         display_test_result("shmem_team_translate_pe()", result_shmem_team_translate_pe, false);
00503       }
00504     }
00505
00506     /* Run shmem_team_split_strided() test */
00507     shmem_barrier_all();
00508     if ( !check_if_exists("shmem_team_split_strided") ) {
00509       if (mype == 0) {
00510         display_not_found_warning("shmem_team_split_strided()", false);
00511       }
00512     }
00513     else {
00514       bool result_shmem_team_split_strided = test_shmem_team_split_strided();
00515       shmem_barrier_all();
00516       if (mype == 0) {
00517         display_test_result("shmem_team_split_strided()", result_shmem_team_split_strided, false);
00518       }
00519     }
00520
00521     /* Run shmem_team_split_2d() test */
00522     shmem_barrier_all();
00523     if ( !check_if_exists("shmem_team_split_2d") ) {
00524       if (mype == 0) {
00525         display_not_found_warning("shmem_team_split_2d()", false);
00526       }
00527     }
00528     else {
00529       bool result_shmem_team_split_2d = test_shmem_team_split_2d();
00530       shmem_barrier_all();
00531       if (mype == 0) {
00532         display_test_result("shmem_team_split_2d()", result_shmem_team_split_2d, false);
00533       }
00534     }
00535
00536     /* Run shmem_team_destroy() test */
00537     shmem_barrier_all();
00538     if ( !check_if_exists("shmem_team_destroy") ) {
00539       if (mype == 0) {
00540         display_not_found_warning("shmem_team_destroy()", false);
00541       }
00542     }
00543     else {
00544       bool result_shmem_team_destroy = test_shmem_team_destroy();
00545       shmem_barrier_all();
00546       if (mype == 0) {
00547         display_test_result("shmem_team_destroy()", result_shmem_team_destroy, false);
00548       }
00549     }
00550   }
00551
00552   /************************ CTX TESTS *************************/
00553   if (opts.test_ctx) {
00554     shmem_barrier_all();
00555     if (mype == 0) {
00556       display_test_header("COMMUNICATION / CONTEXT");
00557     }
00558
00559     /* Run shmem_ctx_create() test */
00560     shmem_barrier_all();
00561     if (!check_if_exists("shmem_ctx_create")) {
00562       if (mype == 0) {
00563         display_not_found_warning("shmem_ctx_create()", false);
00564       }
00565     }
00566     else {
00567       bool result_shmem_ctx_create = test_shmem_ctx_create();
00568       shmem_barrier_all();
```

```
00569       if (mype == 0) {
00570         display_test_result("shmem_ctx_create()", result_shmem_ctx_create, false);
00571       }
00572     }
00573
00574     /* Run shmem_team_create_ctx() test */
00575     shmem_barrier_all();
00576     if (!check_if_exists("shmem_team_create_ctx")) {
00577       if (mype == 0) {
00578         display_not_found_warning("shmem_team_create_ctx()", false);
00579       }
00580     }
00581     else {
00582       bool result_shmem_team_create_ctx = test_shmem_team_create_ctx();
00583       shmem_barrier_all();
00584       if (mype == 0) {
00585         display_test_result("shmem_team_create_ctx()", result_shmem_team_create_ctx, false);
00586       }
00587     }
00588
00589     /* Run shmem_ctx_destroy() test */
00590     shmem_barrier_all();
00591     if (!check_if_exists("shmem_ctx_destroy")) {
00592       if (mype == 0) {
00593         display_not_found_warning("shmem_ctx_destroy()", false);
00594       }
00595     }
00596     else {
00597       bool result_shmem_ctx_destroy = test_shmem_ctx_destroy();
00598       shmem_barrier_all();
00599       if (mype == 0) {
00600         display_test_result("shmem_ctx_destroy()", result_shmem_ctx_destroy, false);
00601       }
00602     }
00603
00604     /* Run shmem_ctx_get_team() test */
00605     shmem_barrier_all();
00606     if (!check_if_exists("shmem_ctx_get_team")) {
00607       if (mype == 0) {
00608         display_not_found_warning("shmem_ctx_get_team()", false);
00609       }
00610     }
00611     else {
00612       bool result_shmem_ctx_get_team = test_shmem_ctx_get_team();
00613       shmem_barrier_all();
00614       if (mype == 0) {
00615         display_test_result("shmem_ctx_get_team()", result_shmem_ctx_get_team, false);
00616       }
00617     }
00618   }
00619
00620   /************************ REMOTE TESTS ************************/
00621   if (opts.test_remote) {
00622     shmem_barrier_all();
00623     if (mype == 0) {
00624       display_test_header("REMOTE MEMORY ACCESS");
00625     }
00626
00627     /* Check to make sure there are at least 2 PEs */
00628     if ( !(npes > 1) ) {
00629       if (mype == 0) {
00630         display_not_enough_pes("REMOTE MEMORY ACCESS");
00631       }
00632     }
00633     else {
00634       /* Run shmem_put() test */
00635       shmem_barrier_all();
00636       if (!check_if_exists("shmem_long_put")) {
00637         if (mype == 0) {
00638           display_not_found_warning("shmem_long_put()", false);
00639         }
00640       }
00641       else {
00642         bool result_shmem_put = test_shmem_put();
00643         shmem_barrier_all();
00644         if (mype == 0) {
00645           display_test_result("shmem_put()", result_shmem_put, false);
00646         }
00647       }
00648
00649       /* Run shmem_p() test */
00650       shmem_barrier_all();
00651       if (!check_if_exists("shmem_long_p")) {
00652         if (mype == 0) {
00653           display_not_found_warning("shmem_long_p()", false);
00654         }
00655       }
```

```
00656       else {
00657         bool result_shmem_p = test_shmem_p();
00658         shmem_barrier_all();
00659         if (mype == 0) {
00660           display_test_result("shmem_p()", result_shmem_p, false);
00661         }
00662       }
00663
00664       /* Run shmem_iput() test */
00665       shmem_barrier_all();
00666       if (!check_if_exists("shmem_long_iput")) {
00667         if (mype == 0) {
00668           display_not_found_warning("shmem_long_iput()", false);
00669         }
00670       }
00671       else {
00672         bool result_shmem_iput = test_shmem_iput();
00673         shmem_barrier_all();
00674         if (mype == 0) {
00675           display_test_result("shmem_iput()", result_shmem_iput, false);
00676         }
00677       }
00678
00679       /* Run shmem_get() test */
00680       shmem_barrier_all();
00681       if (!check_if_exists("shmem_long_get")) {
00682         if (mype == 0) {
00683           display_not_found_warning("shmem_long_get()", false);
00684         }
00685       }
00686       else {
00687         bool result_shmem_get = test_shmem_get();
00688         shmem_barrier_all();
00689         if (mype == 0) {
00690           display_test_result("shmem_get()", result_shmem_get, false);
00691         }
00692       }
00693
00694       /* Run shmem_g() test */
00695       shmem_barrier_all();
00696       if (!check_if_exists("shmem_long_g")) {
00697         if (mype == 0) {
00698           display_not_found_warning("shmem_long_g()", false);
00699         }
00700       }
00701       else {
00702         bool result_shmem_g = test_shmem_g();
00703         shmem_barrier_all();
00704         if (mype == 0) {
00705           display_test_result("shmem_g()", result_shmem_g, false);
00706         }
00707       }
00708
00709       /* Run shmem_iget() test */
00710       shmem_barrier_all();
00711       if (!check_if_exists("shmem_long_iget")) {
00712         if (mype == 0) {
00713           display_not_found_warning("shmem_long_iget()", false);
00714         }
00715       }
00716       else {
00717         bool result_shmem_iget = test_shmem_iget();
00718         shmem_barrier_all();
00719         if (mype == 0) {
00720           display_test_result("shmem_iget()", result_shmem_iget, false);
00721         }
00722       }
00723
00724       /* Run shmem_put_nbi() test */
00725       shmem_barrier_all();
00726       if (!check_if_exists("shmem_long_put_nbi")) {
00727         if (mype == 0) {
00728           display_not_found_warning("shmem_long_put_nbi()", false);
00729         }
00730       }
00731       else {
00732         bool result_shmem_put_nbi = test_shmem_put_nbi();
00733         shmem_barrier_all();
00734         if (mype == 0) {
00735           display_test_result("shmem_put_nbi()", result_shmem_put_nbi, false);
00736         }
00737       }
00738
00739       /* Run shmem_get_nbi() test */
00740       shmem_barrier_all();
00741       if (!check_if_exists("shmem_long_get_nbi")) {
00742         if (mype == 0) {
```

```
00743           display_not_found_warning("shmem_long_get_nbi()", false);
00744         }
00745       }
00746       else {
00747         bool result_shmem_get_nbi = test_shmem_get_nbi();
00748         shmem_barrier_all();
00749         if (mype == 0) {
00750           display_test_result("shmem_get_nbi()", result_shmem_get_nbi, false);
00751         }
00752       }
00753     }
00754   }
00755   /************************ ATOMICS TESTS *************************/
00756   if (opts.test_atomics) {
00757     shmem_barrier_all();
00758     if (mype == 0) {
00759       display_test_header("ATOMIC MEMORY OPS");
00760     }
00761
00762     /* Make sure there are at least 2 PEs */
00763     if ( !(npes > 1) ) {
00764       if (mype == 0) {
00765         display_not_enough_pes("ATOMIC MEMORY OPS");
00766       }
00767     }
00768     else {
00769       /* Run shmem_atomic_fetch() test */
00770       shmem_barrier_all();
00771       if (!check_if_exists("shmem_ulong_atomic_fetch")) {
00772         if (mype == 0) {
00773           display_not_found_warning("shmem_atomic_fetch()", false);
00774         }
00775       }
00776       else {
00777         bool result_shmem_atomic_fetch = test_shmem_atomic_fetch();
00778         shmem_barrier_all();
00779         if (mype == 0) {
00780           display_test_result("shmem_atomic_fetch()", result_shmem_atomic_fetch, false);
00781         }
00782       }
00783
00784       /* Run shmem_atomic_set() test */
00785       shmem_barrier_all();
00786       if (!check_if_exists("shmem_ulong_atomic_set")) {
00787         if (mype == 0) {
00788           display_not_found_warning("shmem_atomic_set()", false);
00789         }
00790       }
00791       else {
00792         bool result_shmem_atomic_set = test_shmem_atomic_set();
00793         shmem_barrier_all();
00794         if (mype == 0) {
00795           display_test_result("shmem_atomic_set()", result_shmem_atomic_set, false);
00796         }
00797       }
00798
00799       /* Run shmem_atomic_compare_swap() test */
00800       shmem_barrier_all();
00801       if (!check_if_exists("shmem_ulong_atomic_compare_swap")) {
00802         if (mype == 0) {
00803           display_not_found_warning("shmem_atomic_compare_swap()", false);
00804         }
00805       }
00806       else {
00807         bool result_shmem_atomic_compare_swap = test_shmem_atomic_compare_swap();
00808         shmem_barrier_all();
00809         if (mype == 0) {
00810           display_test_result("shmem_atomic_compare_swap()", result_shmem_atomic_compare_swap, false);
00811         }
00812       }
00813
00814       /* Run shmem_atomic_swap() test */
00815       shmem_barrier_all();
00816       if (!check_if_exists("shmem_ulong_atomic_swap")) {
00817         if (mype == 0) {
00818           display_not_found_warning("shmem_atomic_swap()", false);
00819         }
00820       }
00821       else {
00822         bool result_shmem_atomic_swap = test_shmem_atomic_swap();
00823         shmem_barrier_all();
00824         if (mype == 0) {
00825           display_test_result("shmem_atomic_swap()", result_shmem_atomic_swap, false);
00826         }
00827       }
00828
00829       /* Run shmem_atomic_fetch_inc() test */
```

```
00830       shmem_barrier_all();
00831       if (!check_if_exists("shmem_ulong_atomic_fetch_inc")) {
00832         if (mype == 0) {
00833           display_not_found_warning("shmem_atomic_fetch_inc()", false);
00834         }
00835       }
00836       else {
00837         bool result_shmem_atomic_fetch_inc = test_shmem_atomic_fetch_inc();
00838         shmem_barrier_all();
00839         if (mype == 0) {
00840           display_test_result("shmem_atomic_fetch_inc()", result_shmem_atomic_fetch_inc, false);
00841         }
00842       }
00843
00844       /* Run shmem_atomic_inc() test */
00845       shmem_barrier_all();
00846       if (!check_if_exists("shmem_ulong_atomic_inc")) {
00847         if (mype == 0) {
00848           display_not_found_warning("shmem_atomic_inc()", false);
00849         }
00850       }
00851       else {
00852         bool result_shmem_atomic_inc = test_shmem_atomic_inc();
00853         shmem_barrier_all();
00854         if (mype == 0) {
00855           display_test_result("shmem_atomic_inc()", result_shmem_atomic_inc, false);
00856         }
00857       }
00858
00859       /* Run shmem_atomic_fetch_add() test */
00860       shmem_barrier_all();
00861       if (!check_if_exists("shmem_ulong_atomic_fetch_add")) {
00862         if (mype == 0) {
00863           display_not_found_warning("shmem_atomic_fetch_add()", false);
00864         }
00865       }
00866       else {
00867         bool result_shmem_atomic_fetch_add = test_shmem_atomic_fetch_add();
00868         shmem_barrier_all();
00869         if (mype == 0) {
00870           display_test_result("shmem_atomic_fetch_add()", result_shmem_atomic_fetch_add, false);
00871         }
00872       }
00873
00874       /* Run shmem_atomic_add() test */
00875       shmem_barrier_all();
00876       if (!check_if_exists("shmem_ulong_atomic_add")) {
00877         if (mype == 0) {
00878           display_not_found_warning("shmem_atomic_add()", false);
00879         }
00880       }
00881       else {
00882         bool result_shmem_atomic_add = test_shmem_atomic_add();
00883         shmem_barrier_all();
00884         if (mype == 0) {
00885           display_test_result("shmem_atomic_add()", result_shmem_atomic_add, false);
00886         }
00887       }
00888
00889       /* Run shmem_atomic_fetch_and() test */
00890       shmem_barrier_all();
00891       if (!check_if_exists("shmem_ulong_atomic_fetch_and")) {
00892         if (mype == 0) {
00893           display_not_found_warning("shmem_atomic_fetch_and()", false);
00894         }
00895       }
00896       else {
00897         bool result_shmem_atomic_fetch_and = test_shmem_atomic_fetch_and();
00898         shmem_barrier_all();
00899         if (mype == 0) {
00900           display_test_result("shmem_atomic_fetch_and()", result_shmem_atomic_fetch_and, false);
00901         }
00902       }
00903
00904       /* Run shmem_atomic_and() test */
00905       shmem_barrier_all();
00906       if (!check_if_exists("shmem_ulong_atomic_and")) {
00907         if (mype == 0) {
00908           display_not_found_warning("shmem_atomic_and()", false);
00909         }
00910       }
00911       else {
00912         bool result_shmem_atomic_and = test_shmem_atomic_and();
00913         shmem_barrier_all();
00914         if (mype == 0) {
00915           display_test_result("shmem_atomic_and()", result_shmem_atomic_and, false);
00916         }
```

```
00917       }
00918
00919       /* Run shmem_atomic_fetch_or() test */
00920       shmem_barrier_all();
00921       if (!check_if_exists("shmem_ulong_atomic_fetch_or")) {
00922         if (mype == 0) {
00923           display_not_found_warning("shmem_atomic_fetch_or()", false);
00924         }
00925       }
00926       else {
00927         bool result_shmem_atomic_fetch_or = test_shmem_atomic_fetch_or();
00928         shmem_barrier_all();
00929         if (mype == 0) {
00930           display_test_result("shmem_atomic_fetch_or()", result_shmem_atomic_fetch_or, false);
00931         }
00932       }
00933
00934       /* Run shmem_atomic_or() test */
00935       shmem_barrier_all();
00936       if (!check_if_exists("shmem_ulong_atomic_or")) {
00937         if (mype == 0) {
00938           display_not_found_warning("shmem_atomic_or()", false);
00939         }
00940       }
00941       else {
00942         bool result_shmem_atomic_or = test_shmem_atomic_or();
00943         shmem_barrier_all();
00944         if (mype == 0) {
00945           display_test_result("shmem_atomic_or()", result_shmem_atomic_or, false);
00946         }
00947       }
00948
00949       /* Run shmem_atomic_fetch_xor() test */
00950       shmem_barrier_all();
00951       if (!check_if_exists("shmem_ulong_atomic_fetch_xor")) {
00952         if (mype == 0) {
00953           display_not_found_warning("shmem_atomic_fetch_xor()", false);
00954         }
00955       }
00956       else {
00957         bool result_shmem_atomic_fetch_xor = test_shmem_atomic_fetch_xor();
00958         shmem_barrier_all();
00959         if (mype == 0) {
00960           display_test_result("shmem_atomic_fetch_xor()", result_shmem_atomic_fetch_xor, false);
00961         }
00962       }
00963
00964       /* Run shmem_atomic_xor() test */
00965       shmem_barrier_all();
00966       if (!check_if_exists("shmem_ulong_atomic_xor")) {
00967         if (mype == 0) {
00968           display_not_found_warning("shmem_atomic_xor()", false);
00969         }
00970       }
00971       else {
00972         bool result_shmem_atomic_xor = test_shmem_atomic_xor();
00973         shmem_barrier_all();
00974         if (mype == 0) {
00975           display_test_result("shmem_atomic_xor()", result_shmem_atomic_xor, false);
00976         }
00977       }
00978
00979       /* Run shmem_atomic_fetch_nbi() test */
00980       shmem_barrier_all();
00981       if (!check_if_exists("shmem_ulong_atomic_fetch_nbi")) {
00982         if (mype == 0) {
00983           display_not_found_warning("shmem_atomic_fetch_nbi()", false);
00984         }
00985       }
00986       else {
00987         bool result_shmem_atomic_fetch_nbi = test_shmem_atomic_fetch_nbi();
00988         shmem_barrier_all();
00989         if (mype == 0) {
00990           display_test_result("shmem_atomic_fetch_nbi()", result_shmem_atomic_fetch_nbi, false);
00991         }
00992       }
00993
00994       /* Run shmem_atomic_compare_swap_nbi() test */
00995       shmem_barrier_all();
00996       if (!check_if_exists("shmem_ulong_atomic_compare_swap_nbi")) {
00997         if (mype == 0) {
00998           display_not_found_warning("shmem_atomic_compare_swap_nbi()", false);
00999         }
01000       }
01001       else {
01002         bool result_shmem_atomic_compare_swap_nbi = test_shmem_atomic_compare_swap_nbi();
01003         shmem_barrier_all();
```

```
01004         if (mype == 0) {
01005           display_test_result("shmem_atomic_compare_swap_nbi()", result_shmem_atomic_compare_swap_nbi,
      false);
01006         }
01007       }
01008
01009       /* Run shmem_atomic_swap_nbi() test */
01010       shmem_barrier_all();
01011       if (!check_if_exists("shmem_ulong_atomic_swap_nbi")) {
01012         if (mype == 0) {
01013           display_not_found_warning("shmem_atomic_swap_nbi()", false);
01014         }
01015       }
01016       else {
01017         bool result_shmem_atomic_swap_nbi = test_shmem_atomic_swap_nbi();
01018         shmem_barrier_all();
01019         if (mype == 0) {
01020           display_test_result("shmem_atomic_swap_nbi()", result_shmem_atomic_swap_nbi, false);
01021         }
01022       }
01023
01024       /* Run shmem_atomic_fetch_inc_nbi() test */
01025       shmem_barrier_all();
01026       if (!check_if_exists("shmem_ulong_atomic_fetch_inc_nbi")) {
01027         if (mype == 0) {
01028           display_not_found_warning("shmem_atomic_fetch_inc_nbi()", false);
01029         }
01030       }
01031       else {
01032         bool result_shmem_atomic_fetch_inc_nbi = test_shmem_atomic_fetch_inc_nbi();
01033         shmem_barrier_all();
01034         if (mype == 0) {
01035           display_test_result("shmem_atomic_fetch_inc_nbi()", result_shmem_atomic_fetch_inc_nbi,
      false);
01036         }
01037       }
01038
01039       /* Run shmem_atomic_fetch_add_nbi() test */
01040       shmem_barrier_all();
01041       if (!check_if_exists("shmem_ulong_atomic_fetch_add_nbi")) {
01042         if (mype == 0) {
01043           display_not_found_warning("shmem_atomic_fetch_add_nbi()", false);
01044         }
01045       }
01046       else {
01047         bool result_shmem_atomic_fetch_add_nbi = test_shmem_atomic_fetch_add_nbi();
01048         shmem_barrier_all();
01049         if (mype == 0) {
01050           display_test_result("shmem_atomic_fetch_add_nbi()", result_shmem_atomic_fetch_add_nbi,
      false);
01051         }
01052       }
01053
01054       /* Run shmem_atomic_fetch_and_nbi() test */
01055       shmem_barrier_all();
01056       if (!check_if_exists("shmem_ulong_atomic_fetch_and_nbi")) {
01057         if (mype == 0) {
01058           display_not_found_warning("shmem_atomic_fetch_and_nbi()", false);
01059         }
01060       }
01061       else {
01062         bool result_shmem_atomic_fetch_and_nbi = test_shmem_atomic_fetch_and_nbi();
01063         shmem_barrier_all();
01064         if (mype == 0) {
01065           display_test_result("shmem_atomic_fetch_and_nbi()", result_shmem_atomic_fetch_and_nbi,
      false);
01066         }
01067       }
01068
01069       /* Run shmem_atomic_fetch_or_nbi() test */
01070       shmem_barrier_all();
01071       if (!check_if_exists("shmem_ulong_atomic_fetch_or_nbi")) {
01072         if (mype == 0) {
01073           display_not_found_warning("shmem_atomic_fetch_or_nbi()", false);
01074         }
01075       }
01076       else {
01077         bool result_shmem_atomic_fetch_or_nbi = test_shmem_atomic_fetch_or_nbi();
01078         shmem_barrier_all();
01079         if (mype == 0) {
01080           display_test_result("shmem_atomic_fetch_or_nbi()", result_shmem_atomic_fetch_or_nbi, false);
01081         }
01082       }
01083
01084       /* Run shmem_atomic_fetch_xor_nbi() test */
01085       shmem_barrier_all();
01086       if (!check_if_exists("shmem_ulong_atomic_fetch_xor_nbi")) {
```

```
01087         if (mype == 0) {
01088           display_not_found_warning("shmem_atomic_fetch_xor_nbi()", false);
01089         }
01090       }
01091       else {
01092         bool result_shmem_atomic_fetch_xor_nbi = test_shmem_atomic_fetch_xor_nbi();
01093         shmem_barrier_all();
01094         if (mype == 0) {
01095           display_test_result("shmem_atomic_fetch_xor_nbi()", result_shmem_atomic_fetch_xor_nbi,
      false);
01096         }
01097       }
01098     }
01099   }
01100
01101   /************************ SIGNALING TESTS *************************/
01102   if (opts.test_signaling) {
01103     shmem_barrier_all();
01104     if (mype == 0) {
01105       display_test_header("SIGNALING OPS");
01106     }
01107
01108     if (!(npes > 1)) {
01109       display_not_enough_pes("SIGNALING OPS");
01110     }
01111     else {
01112       /* Run shmem_put_signal() test */
01113       shmem_barrier_all();
01114       if (!check_if_exists("shmem_long_put_signal")) {
01115         if (mype == 0) {
01116           display_not_found_warning("shmem_long_put_signal()", false);
01117         }
01118       }
01119       else {
01120         bool result_shmem_put_signal = test_shmem_put_signal();
01121         shmem_barrier_all();
01122         if (mype == 0) {
01123           display_test_result("shmem_put_signal()", result_shmem_put_signal, false);
01124         }
01125       }
01126
01127       /* Run shmem_put_signal_nbi() test */
01128       shmem_barrier_all();
01129       if (!check_if_exists("shmem_long_put_signal_nbi")) {
01130         if (mype == 0) {
01131           display_not_found_warning("shmem_long_put_signal_nbi()", false);
01132         }
01133       }
01134       else {
01135         bool result_shmem_put_signal_nbi = test_shmem_put_signal_nbi();
01136         shmem_barrier_all();
01137         if (mype == 0) {
01138           display_test_result("shmem_put_signal_nbi()", result_shmem_put_signal_nbi, false);
01139         }
01140       }
01141
01142       /* Run shmem_signal_fetch() test */
01143       shmem_barrier_all();
01144       if (!check_if_exists("shmem_signal_fetch")) {
01145         if (mype == 0) {
01146           display_not_found_warning("shmem_signal_fetch()", false);
01147         }
01148       }
01149       else {
01150         bool result_shmem_signal_fetch = test_shmem_signal_fetch();
01151         shmem_barrier_all();
01152         if (mype == 0) {
01153           display_test_result("shmem_signal_fetch()", result_shmem_signal_fetch, false);
01154         }
01155       }
01156     }
01157   }
01158
01159   /************************ COLLECTIVES TESTS *************************/
01160   if (opts.test_collectives) {
01161     /* Print project header */
01162     shmem_barrier_all();
01163     if (mype == 0) {
01164       display_test_header("COLLECTIVE OPS");
01165     }
01166
01167     /* Check to make sure there are at least 2 PEs */
01168     if ( !(npes > 1) ) {
01169       if (mype == 0) {
01170         display_not_enough_pes("COLLECTIVE OPS");
01171       }
01172     }
```

```
01173     else {
01174       /* Run shmem_sync() test */
01175       shmem_barrier_all();
01176       if (!check_if_exists("shmem_sync")) {
01177         if (mype == 0) {
01178           display_not_found_warning("shmem_sync()", false);
01179         }
01180       }
01181       else {
01182         bool result_shmem_sync = test_shmem_sync();
01183         shmem_barrier_all();
01184         if (mype == 0) {
01185           display_test_result("shmem_sync()", result_shmem_sync, false);
01186         }
01187       }
01188
01189       /* Run shmem_sync_all() test */
01190       shmem_barrier_all();
01191       if (!check_if_exists("shmem_sync_all")) {
01192         if (mype == 0) {
01193           display_not_found_warning("shmem_sync_all()", false);
01194         }
01195       }
01196       else {
01197         bool result_shmem_sync_all = test_shmem_sync_all();
01198         shmem_barrier_all();
01199         if (mype == 0) {
01200           display_test_result("shmem_sync_all()", result_shmem_sync_all, false);
01201         }
01202       }
01203
01204       /* Run shmem_alltoall() test */
01205       shmem_barrier_all();
01206       if (!check_if_exists("shmem_long_alltoall")) {
01207         if (mype == 0) {
01208           display_not_found_warning("shmem_long_alltoall()", false);
01209         }
01210       }
01211       else {
01212         bool result_shmem_alltoall = test_shmem_alltoall();
01213         shmem_barrier_all();
01214         if (mype == 0) {
01215           display_test_result("shmem_alltoall()", result_shmem_alltoall, false);
01216         }
01217       }
01218
01219       /* Run shmem_alltoalls() test */
01220       shmem_barrier_all();
01221       if (!check_if_exists("shmem_long_alltoalls")) {
01222         if (mype == 0) {
01223           display_not_found_warning("shmem_long_alltoalls()", false);
01224         }
01225       }
01226       else {
01227         bool result_shmem_alltoalls = test_shmem_alltoalls();
01228         shmem_barrier_all();
01229         if (mype == 0) {
01230           display_test_result("shmem_alltoalls()", result_shmem_alltoalls, false);
01231         }
01232       }
01233
01234       /* Run shmem_broadcast() test */
01235       shmem_barrier_all();
01236       if (!check_if_exists("shmem_long_broadcast")) {
01237         if (mype == 0) {
01238           display_not_found_warning("shmem_long_broadcast()", false);
01239         }
01240       }
01241       else {
01242         bool result_shmem_broadcast = test_shmem_broadcast();
01243         shmem_barrier_all();
01244         if (mype == 0) {
01245           display_test_result("shmem_broadcast()", result_shmem_broadcast, false);
01246         }
01247       }
01248
01249       /* Run shmem_collect() test */
01250       shmem_barrier_all();
01251       if (!check_if_exists("shmem_long_collect")) {
01252         if (mype == 0) {
01253           display_not_found_warning("shmem_long_collect()", false);
01254         }
01255       }
01256       else {
01257         bool result_shmem_collect = test_shmem_collect();
01258         shmem_barrier_all();
01259         if (mype == 0) {
```

```
01260           display_test_result("shmem_collect()", result_shmem_collect, false);
01261         }
01262       }
01263
01264       /* Run shmem_fcollect() test */
01265       shmem_barrier_all();
01266       if (!check_if_exists("shmem_long_fcollect")) {
01267         if (mype == 0) {
01268           display_not_found_warning("shmem_long_fcollect()", false);
01269         }
01270       }
01271       else {
01272         bool result_shmem_fcollect = test_shmem_fcollect();
01273         shmem_barrier_all();
01274         if (mype == 0) {
01275           display_test_result("shmem_fcollect()", result_shmem_fcollect, false);
01276         }
01277       }
01278
01279       /* Run shmem_max_reduce() test */
01280       shmem_barrier_all();
01281       if (!check_if_exists("shmem_long_max_reduce")) {
01282         if (mype == 0) {
01283           display_not_found_warning("shmem_long_max_reduce()", false);
01284         }
01285       }
01286       else {
01287         bool result_shmem_max_reduce = test_shmem_max_reduce();
01288         shmem_barrier_all();
01289         if (mype == 0) {
01290           display_test_result("shmem_max_reduce()", result_shmem_max_reduce, false);
01291         }
01292       }
01293
01294       /* Run shmem_min_reduce() test */
01295       shmem_barrier_all();
01296       if (!check_if_exists("shmem_long_min_reduce")) {
01297         if (mype == 0) {
01298           display_not_found_warning("shmem_long_min_reduce()", false);
01299         }
01300       }
01301       else {
01302         bool result_shmem_min_reduce = test_shmem_min_reduce();
01303         shmem_barrier_all();
01304         if (mype == 0) {
01305           display_test_result("shmem_min_reduce()", result_shmem_min_reduce, false);
01306         }
01307       }
01308
01309       /* Run shmem_sum_reduce() test */
01310       shmem_barrier_all();
01311       if (!check_if_exists("shmem_long_sum_reduce")) {
01312         if (mype == 0) {
01313           display_not_found_warning("shmem_long_sum_reduce()", false);
01314         }
01315       }
01316       else {
01317         bool result_shmem_sum_reduce = test_shmem_sum_reduce();
01318         shmem_barrier_all();
01319         if (mype == 0) {
01320           display_test_result("shmem_sum_reduce()", result_shmem_sum_reduce, false);
01321         }
01322       }
01323
01324       /* Run shmem_prod_reduce() test */
01325       shmem_barrier_all();
01326       if (!check_if_exists("shmem_long_prod_reduce")) {
01327         if (mype == 0) {
01328           display_not_found_warning("shmem_long_prod_reduce()", false);
01329         }
01330       }
01331       else {
01332         bool result_shmem_prod_reduce = test_shmem_prod_reduce();
01333         shmem_barrier_all();
01334         if (mype == 0) {
01335           display_test_result("shmem_prod_reduce()", result_shmem_prod_reduce, false);
01336         }
01337       }
01338     }
01339   }
01340
01341   /************************ PT2PT TESTS *************************/
01342   if (opts.test_pt2pt_synch) {
01343     shmem_barrier_all();
01344     if (mype == 0) {
01345       display_test_header("POINT-TO-POINT SYNC OPS");
01346     }
```

```
01347
01348     if (!(npes > 1)) {
01349       display_not_enough_pes("POINT-TO-POINT SYNCH OPS");
01350     }
01351     else {
01352       /* Run shmem_wait_until() test */
01353       shmem_barrier_all();
01354       if (!check_if_exists("shmem_long_wait_until")) {
01355         if (mype == 0) {
01356           display_not_found_warning("shmem_long_wait_until()", false);
01357         }
01358       }
01359       else {
01360         bool result_shmem_wait_until = test_shmem_wait_until();
01361         shmem_barrier_all();
01362         if (mype == 0) {
01363           display_test_result("shmem_wait_until()", result_shmem_wait_until, false);
01364         }
01365       }
01366
01367       /* Run shmem_wait_until_all() test */
01368       shmem_barrier_all();
01369       if (!check_if_exists("shmem_long_wait_until_all")) {
01370         if (mype == 0) {
01371           display_not_found_warning("shmem_long_wait_until_all()", false);
01372         }
01373       }
01374       else {
01375         bool result_shmem_wait_until_all = test_shmem_wait_until_all();
01376         shmem_barrier_all();
01377         if (mype == 0) {
01378           display_test_result("shmem_wait_until_all()", result_shmem_wait_until_all, false);
01379         }
01380       }
01381
01382       /* Run shmem_wait_until_any() test */
01383       shmem_barrier_all();
01384       if (!check_if_exists("shmem_long_wait_until_any")) {
01385         if (mype == 0) {
01386           display_not_found_warning("shmem_long_wait_until_any()", false);
01387         }
01388       }
01389       else {
01390         bool result_shmem_wait_until_any = test_shmem_wait_until_any();
01391         shmem_barrier_all();
01392         if (mype == 0) {
01393           display_test_result("shmem_wait_until_any()", result_shmem_wait_until_any, false);
01394         }
01395       }
01396
01397       /* Run shmem_wait_until_some() test */
01398       shmem_barrier_all();
01399       if (!check_if_exists("shmem_long_wait_until_some")) {
01400         if (mype == 0) {
01401           display_not_found_warning("shmem_long_wait_until_some()", false);
01402         }
01403       }
01404       else {
01405         bool result_shmem_wait_until_some = test_shmem_wait_until_some();
01406         shmem_barrier_all();
01407         if (mype == 0) {
01408           display_test_result("shmem_wait_until_some()", result_shmem_wait_until_some, false);
01409         }
01410       }
01411
01412       /* Run shmem_wait_until_all_vector() test */
01413       shmem_barrier_all();
01414       if (!check_if_exists("shmem_long_wait_until_all_vector")) {
01415         if (mype == 0) {
01416           display_not_found_warning("shmem_long_wait_until_all_vector()", false);
01417         }
01418       }
01419       else {
01420         bool result_shmem_wait_until_all_vector = test_shmem_wait_until_all_vector();
01421         shmem_barrier_all();
01422         if (mype == 0) {
01423           display_test_result("shmem_wait_until_all_vector()", result_shmem_wait_until_all_vector,
      false);
01424         }
01425       }
01426
01427       /* Run shmem_wait_until_any_vector() test */
01428       shmem_barrier_all();
01429       if (!check_if_exists("shmem_long_wait_until_any_vector")) {
01430         if (mype == 0) {
01431           display_not_found_warning("shmem_long_wait_until_any_vector()", false);
01432         }
```

```
01433       }
01434       else {
01435         bool result_shmem_wait_until_any_vector = test_shmem_wait_until_any_vector();
01436         shmem_barrier_all();
01437         if (mype == 0) {
01438           display_test_result("shmem_wait_until_any_vector()", result_shmem_wait_until_any_vector,
      false);
01439         }
01440       }
01441
01442       /* Run shmem_wait_until_some_vector() test */
01443       shmem_barrier_all();
01444       if (!check_if_exists("shmem_long_wait_until_some_vector")) {
01445         if (mype == 0) {
01446           display_not_found_warning("shmem_long_wait_until_some_vector()", false);
01447         }
01448       }
01449       else {
01450         bool result_shmem_wait_until_some_vector = test_shmem_wait_until_some_vector();
01451         shmem_barrier_all();
01452         if (mype == 0) {
01453           display_test_result("shmem_wait_until_some_vector()", result_shmem_wait_until_some_vector,
      false);
01454         }
01455       }
01456
01457       /* Run shmem_test() test */
01458       shmem_barrier_all();
01459       if (!check_if_exists("shmem_long_test")) {
01460         if (mype == 0) {
01461           display_not_found_warning("shmem_long_test()", false);
01462         }
01463       }
01464       else {
01465         bool result_shmem_test = test_shmem_test();
01466         shmem_barrier_all();
01467         if (mype == 0) {
01468           display_test_result("shmem_test()", result_shmem_test, false);
01469         }
01470       }
01471
01472       /* Run shmem_test_all() test */
01473       shmem_barrier_all();
01474       if (!check_if_exists("shmem_long_test_all")) {
01475         if (mype == 0) {
01476           display_not_found_warning("shmem_long_test_all()", false);
01477         }
01478       }
01479       else {
01480         bool result_shmem_test_all = test_shmem_test_all();
01481         shmem_barrier_all();
01482         if (mype == 0) {
01483           display_test_result("shmem_test_all()", result_shmem_test_all, false);
01484         }
01485       }
01486
01487       /* Run shmem_test_any() test */
01488       shmem_barrier_all();
01489       if (!check_if_exists("shmem_long_test_any")) {
01490         if (mype == 0) {
01491           display_not_found_warning("shmem_long_test_any()", false);
01492         }
01493       }
01494       else {
01495         bool result_shmem_test_any = test_shmem_test_any();
01496         shmem_barrier_all();
01497         if (mype == 0) {
01498           display_test_result("shmem_test_any()", result_shmem_test_any, false);
01499         }
01500       }
01501
01502       /* Run shmem_test_some() test */
01503       shmem_barrier_all();
01504       if (!check_if_exists("shmem_long_test_some")) {
01505         if (mype == 0) {
01506           display_not_found_warning("shmem_long_test_some()", false);
01507         }
01508       }
01509       else {
01510         bool result_shmem_test_some = test_shmem_test_some();
01511         shmem_barrier_all();
01512         if (mype == 0) {
01513           display_test_result("shmem_test_some()", result_shmem_test_some, false);
01514         }
01515       }
01516
01517       /* Run shmem_test_all_vector() test */
```

```
01518       shmem_barrier_all();
01519       if (!check_if_exists("shmem_long_test_all_vector")) {
01520         if (mype == 0) {
01521           display_not_found_warning("shmem_long_test_all_vector()", false);
01522         }
01523       }
01524       else {
01525         bool result_shmem_test_all_vector = test_shmem_test_all_vector();
01526         shmem_barrier_all();
01527         if (mype == 0) {
01528           display_test_result("shmem_test_all_vector()", result_shmem_test_all_vector, false);
01529         }
01530       }
01531
01532       /* Run shmem_test_any_vector() test */
01533       shmem_barrier_all();
01534       if (!check_if_exists("shmem_long_test_any_vector")) {
01535         if (mype == 0) {
01536           display_not_found_warning("shmem_long_test_any_vector()", false);
01537         }
01538       }
01539       else {
01540         bool result_shmem_test_any_vector = test_shmem_test_any_vector();
01541         shmem_barrier_all();
01542         if (mype == 0) {
01543           display_test_result("shmem_test_any_vector()", result_shmem_test_any_vector, false);
01544         }
01545       }
01546
01547       /* Run shmem_test_some_vector() test */
01548       shmem_barrier_all();
01549       if (!check_if_exists("shmem_long_test_some_vector")) {
01550         if (mype == 0) {
01551           display_not_found_warning("shmem_long_test_some_vector()", false);
01552         }
01553       }
01554       else {
01555         bool result_shmem_test_some_vector = test_shmem_test_some_vector();
01556         shmem_barrier_all();
01557         if (mype == 0) {
01558           display_test_result("shmem_test_some_vector()", result_shmem_test_some_vector, false);
01559         }
01560       }
01561
01562       /* Run shmem_signal_wait_until() test */
01563       shmem_barrier_all();
01564       if (!check_if_exists("shmem_signal_wait_until")) {
01565         if (mype == 0) {
01566           display_not_found_warning("shmem_signal_wait_until()", false);
01567         }
01568       }
01569       else {
01570         bool result_shmem_signal_wait_until = test_shmem_signal_wait_until();
01571         shmem_barrier_all();
01572         if (mype == 0) {
01573           display_test_result("shmem_signal_wait_until()", result_shmem_signal_wait_until, false);
01574         }
01575       }
01576     }
01577   }
01578
01579   /************************ MEM ORDERING TESTS *************************/
01580   if (opts.test_mem_ordering) {
01581     shmem_barrier_all();
01582     if (mype == 0) {
01583       display_test_header("MEMORY ORDERING");
01584     }
01585
01586     /* Make sure there are at least 2 PEs */
01587     if ( !(npes > 1) ) {
01588       if (mype == 0) {
01589         display_not_enough_pes("MEMORY ORDERING");
01590       }
01591     }
01592     else {
01593       /* Run the shmem_fence() test */
01594       shmem_barrier_all();
01595       if ( !check_if_exists("shmem_fence") ) {
01596         if (mype == 0) {
01597           display_not_found_warning("shmem_fence()", false);
01598         }
01599       }
01600       else {
01601         bool result_shmem_fence = test_shmem_fence();
01602         shmem_barrier_all();
01603         if (mype == 0) {
01604           display_test_result("shmem_fence()", result_shmem_fence, false);
```

```
01605         }
01606       }
01607
01608       /* Run the shmem_quiet() test */
01609       shmem_barrier_all();
01610       if ( !check_if_exists("shmem_quiet") ) {
01611         if (mype == 0) {
01612           display_not_found_warning("shmem_quiet()", false);
01613         }
01614       }
01615       else {
01616         bool result_shmem_quiet = test_shmem_quiet();
01617         shmem_barrier_all();
01618         if (mype == 0) {
01619           display_test_result("shmem_quiet()", result_shmem_quiet, false);
01620         }
01621       }
01622     }
01623   }
01624
01625   /************************ DISTRIBUTED LOCKING TESTS *************************/
01626   if (opts.test_locking) {
01627     shmem_barrier_all();
01628     if (mype == 0) {
01629       display_test_header("DISTRIBUTED LOCKING");
01630     }
01631     shmem_barrier_all();
01632
01633     /* Make sure there are at least 2 PEs */
01634     if ( !(npes > 1) ) {
01635       if (mype == 0) {
01636         display_not_enough_pes("DISTRIBUTED LOCKING");
01637       }
01638     }
01639     else {
01640       /* Run the shmem_set_lock and shmem_clear_lock tests */
01641       shmem_barrier_all();
01642       if ( !check_if_exists("shmem_set_lock") ) {
01643         if (mype == 0) {
01644           display_not_found_warning("shmem_set_lock()", false);
01645         }
01646       }
01647       if ( !check_if_exists("shmem_clear_lock" )) {
01648         if (mype == 0) {
01649           display_not_found_warning("shmem_clear_lock()", false);
01650         }
01651       }
01652
01653       shmem_barrier_all();
01654       if ( check_if_exists("shmem_set_lock") && check_if_exists("shmem_clear_lock") ) {
01655         bool result_shmem_lock_unlock = test_shmem_lock_unlock();
01656         shmem_barrier_all();
01657         if (mype == 0) {
01658           display_test_result("shmem_set_lock()", result_shmem_lock_unlock, false);
01659           display_test_result("shmem_clear_lock()", result_shmem_lock_unlock, false);
01660         }
01661       }
01662     }
01663   }
01664
01665   /************************ FINALIZATION **************************/
01666   /* Run shmem_finalize() test */
01667   shmem_barrier_all();
01668
01669   if ( !check_if_exists("shmem_finalize") ) {
01670     display_not_found_warning("shmem_finalize()", true);
01671   }
01672   else {
01673     if (mype == 0) {
01674       display_test_header("FINALIZATION");
01675       display_test_result("shmem_finalize()", test_shmem_finalize(), false);
01676       std::cout « std::endl;
01677     }
01678   }
01679
01680   /* We made it! End the program. */
01681   return EXIT_SUCCESS;
01682 }
```

4.7 src/routines.cpp File Reference

Contains function pointer declarations and routine loading function for the OpenSHMEM library.

```
#include "routines.hpp"
#include <iostream>
#include <dlfcn.h>
```

Functions

- bool load_routines ()

  Loads the OpenSHMEM routines dynamically.

Variables

- shmem_fake_routine_func p_shmem_fake_routine = nullptr
- shmem_init_func p_shmem_init = nullptr
- shmem_finalize_func p_shmem_finalize = nullptr
- shmem_my_pe_func p_shmem_my_pe = nullptr
- shmem_n_pes_func p_shmem_n_pes = nullptr
- shmem_pe_accessible_func p_shmem_pe_accessible = nullptr
- shmem_barrier_all_func p_shmem_barrier_all = nullptr
- shmem_barrier_func p_shmem_barrier = nullptr
- shmem_info_get_version_func p_shmem_info_get_version = nullptr
- shmem_info_get_name_func p_shmem_info_get_name = nullptr
- shmem_global_exit_func p_shmem_global_exit = nullptr
- shmem_init_thread_func p_shmem_init_thread = nullptr
- shmem_query_thread_func p_shmem_query_thread = nullptr
- shmem_ptr_func p_shmem_ptr = nullptr
- shmem_malloc_func p_shmem_malloc = nullptr
- shmem_free_func p_shmem_free = nullptr
- shmem_realloc_func p_shmem_realloc = nullptr
- shmem_align_func p_shmem_align = nullptr
- shmem_malloc_with_hints_func p_shmem_malloc_with_hints = nullptr
- shmem_calloc_func p_shmem_calloc = nullptr
- shmem_addr_accessible_func p_shmem_addr_accessible = nullptr
- shmem_team_my_pe_func p_shmem_team_my_pe = nullptr
- shmem_team_n_pes_func p_shmem_team_n_pes = nullptr
- shmem_team_get_config_func p_shmem_team_get_config = nullptr
- shmem_team_translate_pe_func p_shmem_team_translate_pe = nullptr
- shmem_team_split_strided_func p_shmem_team_split_strided = nullptr
- shmem_team_split_2d_func p_shmem_team_split_2d = nullptr
- shmem_team_destroy_func p_shmem_team_destroy = nullptr
- shmem_ctx_create_func p_shmem_ctx_create = nullptr
- shmem_team_create_ctx_func p_shmem_team_create_ctx = nullptr
- shmem_ctx_destroy_func p_shmem_ctx_destroy = nullptr
- shmem_ctx_get_team_func p_shmem_ctx_get_team = nullptr
- shmem_long_put_func p_shmem_long_put = nullptr
- shmem_long_p_func p_shmem_long_p = nullptr
- shmem_long_iput_func p_shmem_long_iput = nullptr
- shmem_long_get_func p_shmem_long_get = nullptr
- shmem_long_g_func p_shmem_long_g = nullptr
- shmem_long_iget_func p_shmem_long_iget = nullptr
- shmem_long_put_nbi_func p_shmem_long_put_nbi = nullptr
- shmem_long_get_nbi_func p_shmem_long_get_nbi = nullptr
- shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch = nullptr

- shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set = nullptr
- shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap = nullptr
- shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap = nullptr
- shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc = nullptr
- shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc = nullptr
- shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add = nullptr
- shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add = nullptr
- shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and = nullptr
- shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and = nullptr
- shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or = nullptr
- shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or = nullptr
- shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor = nullptr
- shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor = nullptr
- shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi = nullptr
- shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi = nullptr
- shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi = nullptr
- shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi = nullptr
- shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi = nullptr
- shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi = nullptr
- shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi = nullptr
- shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi = nullptr
- shmem_signal_fetch_func p_shmem_signal_fetch = nullptr
- shmem_long_put_signal_func p_shmem_long_put_signal = nullptr
- shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi = nullptr
- shmem_sync_func p_shmem_sync = nullptr
- shmem_sync_all_func p_shmem_sync_all = nullptr
- shmem_long_alltoall_func p_shmem_long_alltoall = nullptr
- shmem_long_alltoalls_func p_shmem_long_alltoalls = nullptr
- shmem_long_broadcast_func p_shmem_long_broadcast = nullptr
- shmem_long_collect_func p_shmem_long_collect = nullptr
- shmem_long_fcollect_func p_shmem_long_fcollect = nullptr
- shmem_long_and_reduce_func p_shmem_long_and_reduce = nullptr
- shmem_long_or_reduce_func p_shmem_long_or_reduce = nullptr
- shmem_long_xor_reduce_func p_shmem_long_xor_reduce = nullptr
- shmem_long_max_reduce_func p_shmem_long_max_reduce = nullptr
- shmem_long_min_reduce_func p_shmem_long_min_reduce = nullptr
- shmem_long_sum_reduce_func p_shmem_long_sum_reduce = nullptr
- shmem_long_prod_reduce_func p_shmem_long_prod_reduce = nullptr
- shmem_long_wait_until_func p_shmem_long_wait_until = nullptr
- shmem_long_wait_until_all_func p_shmem_long_wait_until_all = nullptr
- shmem_long_wait_until_any_func p_shmem_long_wait_until_any = nullptr
- shmem_long_wait_until_some_func p_shmem_long_wait_until_some = nullptr
- shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector = nullptr
- shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector = nullptr
- shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector = nullptr
- shmem_long_test_func p_shmem_long_test = nullptr
- shmem_long_test_all_func p_shmem_long_test_all = nullptr
- shmem_long_test_any_func p_shmem_long_test_any = nullptr
- shmem_long_test_some_func p_shmem_long_test_some = nullptr
- shmem_long_test_all_vector_func p_shmem_long_test_all_vector = nullptr
- shmem_long_test_any_vector_func p_shmem_long_test_any_vector = nullptr
- shmem_long_test_some_vector_func p_shmem_long_test_some_vector = nullptr
- shmem_signal_wait_until_func p_shmem_signal_wait_until = nullptr
- shmem_quiet_func p_shmem_quiet = nullptr
- shmem_fence_func p_shmem_fence = nullptr
- shmem_set_lock_func p_shmem_set_lock = nullptr
- shmem_clear_lock_func p_shmem_clear_lock = nullptr

4.7.1 Detailed Description

Contains function pointer declarations and routine loading function for the OpenSHMEM library.

Definition in file routines.cpp.

4.7.2 Function Documentation

4.7.2.1 load_routines()

```
bool load_routines ()
```

Loads the OpenSHMEM routines dynamically.

This function loads the OpenSHMEM routines at runtime using dynamic linking.

Returns

True if successful, false otherwise.

Definition at line 143 of file routines.cpp.

```
00143                       {
00144   void *handle = dlopen(NULL, RTLD_LAZY);
00145   if (!handle) {
00146     std::cerr « "Failed to open handle: " « dlerror() « std::endl;
00147     return false;
00148   }
00149
00150   p_shmem_fake_routine = reinterpret_cast<shmem_fake_routine_func>(dlsym(handle,
      "shmem_fake_routine"));
00151
00152   /* Setup, Exit, and Query Routines */
00153   p_shmem_init = reinterpret_cast<shmem_init_func>(dlsym(handle, "shmem_init"));
00154   p_shmem_finalize = reinterpret_cast<shmem_finalize_func>(dlsym(handle, "shmem_finalize"));
00155   p_shmem_my_pe = reinterpret_cast<shmem_my_pe_func>(dlsym(handle, "shmem_my_pe"));
00156   p_shmem_n_pes = reinterpret_cast<shmem_n_pes_func>(dlsym(handle, "shmem_n_pes"));
00157   p_shmem_pe_accessible = reinterpret_cast<shmem_pe_accessible_func>(dlsym(handle,
      "shmem_pe_accessible"));
00158   p_shmem_barrier_all = reinterpret_cast<shmem_barrier_all_func>(dlsym(handle, "shmem_barrier_all"));
00159   p_shmem_barrier = reinterpret_cast<shmem_barrier_func>(dlsym(handle, "shmem_barrier"));
00160   p_shmem_info_get_version = reinterpret_cast<shmem_info_get_version_func>(dlsym(handle,
      "shmem_info_get_version"));
00161   p_shmem_info_get_name = reinterpret_cast<shmem_info_get_name_func>(dlsym(handle,
      "shmem_info_get_name"));
00162   p_shmem_global_exit = reinterpret_cast<shmem_global_exit_func>(dlsym(handle, "shmem_global_exit"));
00163
00164   /* Thread Support Routines */
00165   p_shmem_init_thread = reinterpret_cast<shmem_init_thread_func>(dlsym(handle, "shmem_init_thread"));
00166   p_shmem_query_thread = reinterpret_cast<shmem_query_thread_func>(dlsym(handle,
      "shmem_query_thread"));
00167
00168   /* Memory Management Routines */
00169   p_shmem_ptr = reinterpret_cast<shmem_ptr_func>(dlsym(handle, "shmem_ptr"));
00170   p_shmem_malloc = reinterpret_cast<shmem_malloc_func>(dlsym(handle, "shmem_malloc"));
00171   p_shmem_free = reinterpret_cast<shmem_free_func>(dlsym(handle, "shmem_free"));
00172   p_shmem_realloc = reinterpret_cast<shmem_realloc_func>(dlsym(handle, "shmem_realloc"));
00173   p_shmem_align = reinterpret_cast<shmem_align_func>(dlsym(handle, "shmem_align"));
00174   p_shmem_malloc_with_hints = reinterpret_cast<shmem_malloc_with_hints_func>(dlsym(handle,
      "shmem_malloc_with_hints"));
00175   p_shmem_calloc = reinterpret_cast<shmem_calloc_func>(dlsym(handle, "shmem_calloc"));
00176   p_shmem_addr_accessible = reinterpret_cast<shmem_addr_accessible_func>(dlsym(handle,
      "shmem_addr_accessible"));
00177
00178   /* Team Management Routines */
00179   p_shmem_team_my_pe = reinterpret_cast<shmem_team_my_pe_func>(dlsym(handle, "shmem_team_my_pe"));
00180   p_shmem_team_n_pes = reinterpret_cast<shmem_team_n_pes_func>(dlsym(handle, "shmem_team_n_pes"));
00181   p_shmem_team_get_config = reinterpret_cast<shmem_team_get_config_func>(dlsym(handle,
      "shmem_team_get_config"));
00182   p_shmem_team_translate_pe = reinterpret_cast<shmem_team_translate_pe_func>(dlsym(handle,
      "shmem_team_translate_pe"));
00183   p_shmem_team_split_strided = reinterpret_cast<shmem_team_split_strided_func>(dlsym(handle,
      "shmem_team_split_strided"));
```

```
00184   p_shmem_team_split_2d = reinterpret_cast<shmem_team_split_2d_func>(dlsym(handle,
      "shmem_team_split_2d"));
00185   p_shmem_team_destroy = reinterpret_cast<shmem_team_destroy_func>(dlsym(handle,
      "shmem_team_destroy"));
00186
00187   /* Communication/Context Management Routines */
00188   p_shmem_ctx_create = reinterpret_cast<shmem_ctx_create_func>(dlsym(handle, "shmem_ctx_create"));
00189   p_shmem_team_create_ctx = reinterpret_cast<shmem_team_create_ctx_func>(dlsym(handle,
      "shmem_team_create_ctx"));
00190   p_shmem_ctx_destroy = reinterpret_cast<shmem_ctx_destroy_func>(dlsym(handle, "shmem_ctx_destroy"));
00191   p_shmem_ctx_get_team = reinterpret_cast<shmem_ctx_get_team_func>(dlsym(handle,
      "shmem_ctx_get_team"));
00192
00193   /* Remote Access Routines */
00194   p_shmem_long_put = reinterpret_cast<shmem_long_put_func>(dlsym(handle, "shmem_long_put"));
00195   p_shmem_long_p = reinterpret_cast<shmem_long_p_func>(dlsym(handle, "shmem_long_p"));
00196   p_shmem_long_iput = reinterpret_cast<shmem_long_iput_func>(dlsym(handle, "shmem_long_iput"));
00197   p_shmem_long_get = reinterpret_cast<shmem_long_get_func>(dlsym(handle, "shmem_long_get"));
00198   p_shmem_long_g = reinterpret_cast<shmem_long_g_func>(dlsym(handle, "shmem_long_g"));
00199   p_shmem_long_iget = reinterpret_cast<shmem_long_iget_func>(dlsym(handle, "shmem_long_iget"));
00200   p_shmem_long_put_nbi = reinterpret_cast<shmem_long_put_nbi_func>(dlsym(handle,
      "shmem_long_put_nbi"));
00201   p_shmem_long_get_nbi = reinterpret_cast<shmem_long_get_nbi_func>(dlsym(handle,
      "shmem_long_get_nbi"));
00202
00203   /* Atomic Memory Operations */
00204   p_shmem_ulong_atomic_fetch = reinterpret_cast<shmem_ulong_atomic_fetch_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch"));
00205   p_shmem_ulong_atomic_set = reinterpret_cast<shmem_ulong_atomic_set_func>(dlsym(handle,
      "shmem_ulong_atomic_set"));
00206   p_shmem_ulong_atomic_compare_swap =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap"));
00207   p_shmem_ulong_atomic_swap = reinterpret_cast<shmem_ulong_atomic_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_swap"));
00208   p_shmem_ulong_atomic_fetch_inc = reinterpret_cast<shmem_ulong_atomic_fetch_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc"));
00209   p_shmem_ulong_atomic_inc = reinterpret_cast<shmem_ulong_atomic_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_inc"));
00210   p_shmem_ulong_atomic_fetch_add = reinterpret_cast<shmem_ulong_atomic_fetch_add_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add"));
00211   p_shmem_ulong_atomic_add = reinterpret_cast<shmem_ulong_atomic_add_func>(dlsym(handle,
      "shmem_ulong_atomic_add"));
00212   p_shmem_ulong_atomic_fetch_and = reinterpret_cast<shmem_ulong_atomic_fetch_and_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and"));
00213   p_shmem_ulong_atomic_and = reinterpret_cast<shmem_ulong_atomic_and_func>(dlsym(handle,
      "shmem_ulong_atomic_and"));
00214   p_shmem_ulong_atomic_fetch_or = reinterpret_cast<shmem_ulong_atomic_fetch_or_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or"));
00215   p_shmem_ulong_atomic_or = reinterpret_cast<shmem_ulong_atomic_or_func>(dlsym(handle,
      "shmem_ulong_atomic_or"));
00216   p_shmem_ulong_atomic_fetch_xor = reinterpret_cast<shmem_ulong_atomic_fetch_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor"));
00217   p_shmem_ulong_atomic_xor = reinterpret_cast<shmem_ulong_atomic_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_xor"));
00218
00219   p_shmem_ulong_atomic_fetch_nbi = reinterpret_cast<shmem_ulong_atomic_fetch_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_nbi"));
00220   p_shmem_ulong_atomic_compare_swap_nbi =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap_nbi"));
00221   p_shmem_ulong_atomic_swap_nbi = reinterpret_cast<shmem_ulong_atomic_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_swap_nbi"));
00222   p_shmem_ulong_atomic_fetch_inc_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_inc_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc_nbi"));
00223   p_shmem_ulong_atomic_fetch_add_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_add_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add_nbi"));
00224   p_shmem_ulong_atomic_fetch_and_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_and_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and_nbi"));
00225   p_shmem_ulong_atomic_fetch_or_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_or_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or_nbi"));
00226   p_shmem_ulong_atomic_fetch_xor_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_xor_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor_nbi"));
00227
00228   /* Signaling Operations */
00229   p_shmem_signal_fetch = reinterpret_cast<shmem_signal_fetch_func>(dlsym(handle,
      "shmem_signal_fetch"));
00230   p_shmem_long_put_signal = reinterpret_cast<shmem_long_put_signal_func>(dlsym(handle,
      "shmem_long_put_signal"));
00231   p_shmem_long_put_signal_nbi = reinterpret_cast<shmem_long_put_signal_nbi_func>(dlsym(handle,
      "shmem_long_put_signal_nbi"));
00232
```

```
00233   /* Collective Routines */
00234   p_shmem_sync = reinterpret_cast<shmem_sync_func>(dlsym(handle, "shmem_sync"));
00235   p_shmem_sync_all = reinterpret_cast<shmem_sync_all_func>(dlsym(handle, "shmem_sync_all"));
00236   p_shmem_long_alltoall = reinterpret_cast<shmem_long_alltoall_func>(dlsym(handle,
      "shmem_long_alltoall"));
00237   p_shmem_long_alltoalls = reinterpret_cast<shmem_long_alltoalls_func>(dlsym(handle,
      "shmem_long_alltoalls"));
00238   p_shmem_long_broadcast = reinterpret_cast<shmem_long_broadcast_func>(dlsym(handle,
      "shmem_long_broadcast"));
00239   p_shmem_long_collect = reinterpret_cast<shmem_long_collect_func>(dlsym(handle,
      "shmem_long_collect"));
00240   p_shmem_long_fcollect = reinterpret_cast<shmem_long_fcollect_func>(dlsym(handle,
      "shmem_long_fcollect"));
00241   p_shmem_long_and_reduce = reinterpret_cast<shmem_long_and_reduce_func>(dlsym(handle,
      "shmem_long_and_reduce"));
00242   p_shmem_long_or_reduce = reinterpret_cast<shmem_long_or_reduce_func>(dlsym(handle,
      "shmem_long_or_reduce"));
00243   p_shmem_long_xor_reduce = reinterpret_cast<shmem_long_xor_reduce_func>(dlsym(handle,
      "shmem_long_xor_reduce"));
00244   p_shmem_long_max_reduce = reinterpret_cast<shmem_long_max_reduce_func>(dlsym(handle,
      "shmem_long_max_reduce"));
00245   p_shmem_long_min_reduce = reinterpret_cast<shmem_long_min_reduce_func>(dlsym(handle,
      "shmem_long_min_reduce"));
00246   p_shmem_long_sum_reduce = reinterpret_cast<shmem_long_sum_reduce_func>(dlsym(handle,
      "shmem_long_sum_reduce"));
00247   p_shmem_long_prod_reduce = reinterpret_cast<shmem_long_prod_reduce_func>(dlsym(handle,
      "shmem_long_prod_reduce"));
00248
00249   /* Point-to-Point Synchronization Routines */
00250   p_shmem_long_wait_until = reinterpret_cast<shmem_long_wait_until_func>(dlsym(handle,
      "shmem_long_wait_until"));
00251   p_shmem_long_wait_until_all = reinterpret_cast<shmem_long_wait_until_all_func>(dlsym(handle,
      "shmem_long_wait_until_all"));
00252   p_shmem_long_wait_until_any = reinterpret_cast<shmem_long_wait_until_any_func>(dlsym(handle,
      "shmem_long_wait_until_any"));
00253   p_shmem_long_wait_until_some = reinterpret_cast<shmem_long_wait_until_some_func>(dlsym(handle,
      "shmem_long_wait_until_some"));
00254   p_shmem_long_wait_until_all_vector =
      reinterpret_cast<shmem_long_wait_until_all_vector_func>(dlsym(handle,
      "shmem_long_wait_until_all_vector"));
00255   p_shmem_long_wait_until_any_vector =
      reinterpret_cast<shmem_long_wait_until_any_vector_func>(dlsym(handle,
      "shmem_long_wait_until_any_vector"));
00256   p_shmem_long_wait_until_some_vector =
      reinterpret_cast<shmem_long_wait_until_some_vector_func>(dlsym(handle,
      "shmem_long_wait_until_some_vector"));
00257   p_shmem_long_test = reinterpret_cast<shmem_long_test_func>(dlsym(handle, "shmem_long_test"));
00258   p_shmem_long_test_all = reinterpret_cast<shmem_long_test_all_func>(dlsym(handle,
      "shmem_long_test_all"));
00259   p_shmem_long_test_any = reinterpret_cast<shmem_long_test_any_func>(dlsym(handle,
      "shmem_long_test_any"));
00260   p_shmem_long_test_some = reinterpret_cast<shmem_long_test_some_func>(dlsym(handle,
      "shmem_long_test_some"));
00261   p_shmem_long_test_all_vector = reinterpret_cast<shmem_long_test_all_vector_func>(dlsym(handle,
      "shmem_long_test_all_vector"));
00262   p_shmem_long_test_any_vector = reinterpret_cast<shmem_long_test_any_vector_func>(dlsym(handle,
      "shmem_long_test_any_vector"));
00263   p_shmem_long_test_some_vector = reinterpret_cast<shmem_long_test_some_vector_func>(dlsym(handle,
      "shmem_long_test_some_vector"));
00264   p_shmem_signal_wait_until = reinterpret_cast<shmem_signal_wait_until_func>(dlsym(handle,
      "shmem_signal_wait_until"));
00265
00266   /* Memory Ordering Routines */
00267   p_shmem_quiet = reinterpret_cast<shmem_quiet_func>(dlsym(handle, "shmem_quiet"));
00268   p_shmem_fence = reinterpret_cast<shmem_fence_func>(dlsym(handle, "shmem_fence"));
00269
00270   /* Distributed Locking Routines */
00271   p_shmem_set_lock = reinterpret_cast<shmem_set_lock_func>(dlsym(handle, "shmem_set_lock"));
00272   p_shmem_clear_lock = reinterpret_cast<shmem_clear_lock_func>(dlsym(handle, "shmem_clear_lock"));
00273
00274   const char *dlsym_error = dlerror();
00275   if (dlsym_error) {
00276     std::cerr « "Error loading functions: " « dlsym_error « std::endl;
00277     dlclose(handle);
00278     return false;
00279   }
00280
00281   return true;
00282 }
```

References p_shmem_addr_accessible, p_shmem_align, p_shmem_barrier, p_shmem_barrier_all, p_shmem_calloc, p_shmem_clear_lock, p_shmem_ctx_create, p_shmem_ctx_destroy, p_shmem_ctx_get_team, p_shmem_fake_routine, p_shmem_fence, p_shmem_finalize, p_shmem_free, p_shmem_global_exit, p_shmem_info_get_name, p_shmem_info_get_version, p_shmem_init, p_shmem_init_thread, p_shmem_long_alltoall, p_shmem_long_alltoalls, p_shmem_long_and_reduce, p_shmem_long_broadcast, p_shmem_long_collect, p_shmem_long_fcollect,

p_shmem_long_g, p_shmem_long_get, p_shmem_long_get_nbi, p_shmem_long_iget, p_shmem_long_iput, p_shmem_long_max_reduce, p_shmem_long_min_reduce, p_shmem_long_or_reduce, p_shmem_long_p, p_shmem_long_prod_reduce, p_shmem_long_put, p_shmem_long_put_nbi, p_shmem_long_put_signal, p_shmem_long_put_signal_nbi, p_shmem_long_sum_reduce, p_shmem_long_test, p_shmem_long_test_all, p_shmem_long_test_all_vector, p_shmem_long_test_any, p_shmem_long_test_any_vector, p_shmem_long_test_some, p_shmem_long_test_some_vector, p_shmem_long_wait_until, p_shmem_long_wait_until_all, p_shmem_long_wait_until_all_vector, p_shmem_long_wait_until_any, p_shmem_long_wait_until_any_vector, p_shmem_long_wait_until_some, p_shmem_long_wait_until_some_vector, p_shmem_long_xor_reduce, p_shmem_malloc, p_shmem_malloc_with_hints, p_shmem_my_pe, p_shmem_n_pes, p_shmem_pe_accessible, p_shmem_ptr, p_shmem_query_thread, p_shmem_quiet, p_shmem_realloc, p_shmem_set_lock, p_shmem_signal_fetch, p_shmem_signal_wait_until, p_shmem_sync, p_shmem_sync_all, p_shmem_team_create_ctx, p_shmem_team_destroy, p_shmem_team_get_config, p_shmem_team_my_pe, p_shmem_team_n_pes, p_shmem_team_split_2d, p_shmem_team_split_strided, p_shmem_team_translate_pe, p_shmem_ulong_atomic_add, p_shmem_ulong_atomic_and, p_shmem_ulong_atomic_compare_swa
p_shmem_ulong_atomic_compare_swap_nbi, p_shmem_ulong_atomic_fetch, p_shmem_ulong_atomic_fetch_add, p_shmem_ulong_atomic_fetch_add_nbi, p_shmem_ulong_atomic_fetch_and, p_shmem_ulong_atomic_fetch_and_nbi, p_shmem_ulong_atomic_fetch_inc, p_shmem_ulong_atomic_fetch_inc_nbi, p_shmem_ulong_atomic_fetch_nbi, p_shmem_ulong_atomic_fetch_or, p_shmem_ulong_atomic_fetch_or_nbi, p_shmem_ulong_atomic_fetch_xor, p_shmem_ulong_atomic_fetch_xor_nbi, p_shmem_ulong_atomic_inc, p_shmem_ulong_atomic_or, p_shmem_ulong_atomic_set, p_shmem_ulong_atomic_swap, p_shmem_ulong_atomic_swap_nbi, and p_shmem_ulong_atomic_xor.

4.7.3 Variable Documentation

4.7.3.1 p_shmem_addr_accessible

```
shmem_addr_accessible_func p_shmem_addr_accessible = nullptr
```

Definition at line 38 of file routines.cpp.

4.7.3.2 p_shmem_align

```
shmem_align_func p_shmem_align = nullptr
```

Definition at line 35 of file routines.cpp.

4.7.3.3 p_shmem_barrier

```
shmem_barrier_func p_shmem_barrier = nullptr
```

Definition at line 21 of file routines.cpp.

4.7.3.4 p_shmem_barrier_all

```
shmem_barrier_all_func p_shmem_barrier_all = nullptr
```

Definition at line 20 of file routines.cpp.

4.7.3.5 p_shmem_calloc

```
shmem_calloc_func p_shmem_calloc = nullptr
```

Definition at line 37 of file routines.cpp.

4.7.3.6 p_shmem_clear_lock

```
shmem_clear_lock_func p_shmem_clear_lock = nullptr
```

Definition at line 134 of file routines.cpp.

4.7.3.7 p_shmem_ctx_create

```
shmem_ctx_create_func p_shmem_ctx_create = nullptr
```

Definition at line 50 of file routines.cpp.

4.7.3.8 p_shmem_ctx_destroy

```
shmem_ctx_destroy_func p_shmem_ctx_destroy = nullptr
```

Definition at line 52 of file routines.cpp.

4.7.3.9 p_shmem_ctx_get_team

```
shmem_ctx_get_team_func p_shmem_ctx_get_team = nullptr
```

Definition at line 53 of file routines.cpp.

4.7.3.10 p_shmem_fake_routine

```
shmem_fake_routine_func p_shmem_fake_routine = nullptr
```

Definition at line 12 of file routines.cpp.

4.7.3.11 p_shmem_fence

```
shmem_fence_func p_shmem_fence = nullptr
```

Definition at line 130 of file routines.cpp.

4.7.3.12 p_shmem_finalize

```
shmem_finalize_func p_shmem_finalize = nullptr
```

Definition at line 16 of file routines.cpp.

4.7.3.13 p_shmem_free

```
shmem_free_func p_shmem_free = nullptr
```

Definition at line 33 of file routines.cpp.

4.7.3.14 p_shmem_global_exit

```
shmem_global_exit_func p_shmem_global_exit = nullptr
```

Definition at line 24 of file routines.cpp.

4.7.3.15 p_shmem_info_get_name

```
shmem_info_get_name_func p_shmem_info_get_name = nullptr
```

Definition at line 23 of file routines.cpp.

4.7.3.16 p_shmem_info_get_version

```
shmem_info_get_version_func p_shmem_info_get_version = nullptr
```

Definition at line 22 of file routines.cpp.

4.7.3.17 p_shmem_init

```
shmem_init_func p_shmem_init = nullptr
```

Definition at line 15 of file routines.cpp.

4.7.3.18 p_shmem_init_thread

```
shmem_init_thread_func p_shmem_init_thread = nullptr
```

Definition at line 27 of file routines.cpp.

4.7.3.19 p_shmem_long_alltoall

```
shmem_long_alltoall_func p_shmem_long_alltoall = nullptr
```

Definition at line 98 of file routines.cpp.

4.7.3.20 p_shmem_long_alltoalls

```
shmem_long_alltoalls_func p_shmem_long_alltoalls = nullptr
```

Definition at line 99 of file routines.cpp.

4.7.3.21 p_shmem_long_and_reduce

```
shmem_long_and_reduce_func p_shmem_long_and_reduce = nullptr
```

Definition at line 103 of file routines.cpp.

4.7.3.22 p_shmem_long_broadcast

```
shmem_long_broadcast_func p_shmem_long_broadcast = nullptr
```

Definition at line 100 of file routines.cpp.

4.7.3.23 p_shmem_long_collect

```
shmem_long_collect_func p_shmem_long_collect = nullptr
```

Definition at line 101 of file routines.cpp.

4.7.3.24 p_shmem_long_fcollect

```
shmem_long_fcollect_func p_shmem_long_fcollect = nullptr
```

Definition at line 102 of file routines.cpp.

4.7.3.25 p_shmem_long_g

```
shmem_long_g_func p_shmem_long_g = nullptr
```

Definition at line 60 of file routines.cpp.

4.7.3.26 p_shmem_long_get

```
shmem_long_get_func p_shmem_long_get = nullptr
```

Definition at line 59 of file routines.cpp.

4.7.3.27 p_shmem_long_get_nbi

```
shmem_long_get_nbi_func p_shmem_long_get_nbi = nullptr
```

Definition at line 63 of file routines.cpp.

4.7.3.28 p_shmem_long_iget

```
shmem_long_iget_func p_shmem_long_iget = nullptr
```

Definition at line 61 of file routines.cpp.

4.7.3.29 p_shmem_long_iput

```
shmem_long_iput_func p_shmem_long_iput = nullptr
```

Definition at line 58 of file routines.cpp.

4.7.3.30 p_shmem_long_max_reduce

```
shmem_long_max_reduce_func p_shmem_long_max_reduce = nullptr
```

Definition at line 106 of file routines.cpp.

4.7.3.31 p_shmem_long_min_reduce

```
shmem_long_min_reduce_func p_shmem_long_min_reduce = nullptr
```

Definition at line 107 of file routines.cpp.

4.7.3.32 p_shmem_long_or_reduce

```
shmem_long_or_reduce_func p_shmem_long_or_reduce = nullptr
```

Definition at line 104 of file routines.cpp.

4.7.3.33 p_shmem_long_p

```
shmem_long_p_func p_shmem_long_p = nullptr
```

Definition at line 57 of file routines.cpp.

4.7.3.34 p_shmem_long_prod_reduce

```
shmem_long_prod_reduce_func p_shmem_long_prod_reduce = nullptr
```

Definition at line 109 of file routines.cpp.

4.7.3.35 p_shmem_long_put

```
shmem_long_put_func p_shmem_long_put = nullptr
```

Definition at line 56 of file routines.cpp.

4.7.3.36 p_shmem_long_put_nbi

```
shmem_long_put_nbi_func p_shmem_long_put_nbi = nullptr
```

Definition at line 62 of file routines.cpp.

4.7.3.37 p_shmem_long_put_signal

```
shmem_long_put_signal_func p_shmem_long_put_signal = nullptr
```

Definition at line 92 of file routines.cpp.

4.7.3.38 p_shmem_long_put_signal_nbi

```
shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi = nullptr
```

Definition at line 93 of file routines.cpp.

4.7.3.39 p_shmem_long_sum_reduce

```
shmem_long_sum_reduce_func p_shmem_long_sum_reduce = nullptr
```

Definition at line 108 of file routines.cpp.

4.7.3.40 p_shmem_long_test

```
shmem_long_test_func p_shmem_long_test = nullptr
```

Definition at line 119 of file routines.cpp.

4.7.3.41 p_shmem_long_test_all

```
shmem_long_test_all_func p_shmem_long_test_all = nullptr
```

Definition at line 120 of file routines.cpp.

4.7.3.42 p_shmem_long_test_all_vector

```
shmem_long_test_all_vector_func p_shmem_long_test_all_vector = nullptr
```

Definition at line 123 of file routines.cpp.

4.7.3.43 p_shmem_long_test_any

```
shmem_long_test_any_func p_shmem_long_test_any = nullptr
```

Definition at line 121 of file routines.cpp.

4.7.3.44 p_shmem_long_test_any_vector

```
shmem_long_test_any_vector_func p_shmem_long_test_any_vector = nullptr
```

Definition at line 124 of file routines.cpp.

4.7.3.45 p_shmem_long_test_some

```
shmem_long_test_some_func p_shmem_long_test_some = nullptr
```

Definition at line 122 of file routines.cpp.

4.7.3.46 p_shmem_long_test_some_vector

```
shmem_long_test_some_vector_func p_shmem_long_test_some_vector = nullptr
```

Definition at line 125 of file routines.cpp.

4.7.3.47 p_shmem_long_wait_until

```
shmem_long_wait_until_func p_shmem_long_wait_until = nullptr
```

Definition at line 112 of file routines.cpp.

4.7.3.48 p_shmem_long_wait_until_all

```
shmem_long_wait_until_all_func p_shmem_long_wait_until_all = nullptr
```

Definition at line 113 of file routines.cpp.

4.7.3.49 p_shmem_long_wait_until_all_vector

```
shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector = nullptr
```

Definition at line 116 of file routines.cpp.

4.7.3.50 p_shmem_long_wait_until_any

```
shmem_long_wait_until_any_func p_shmem_long_wait_until_any = nullptr
```

Definition at line 114 of file routines.cpp.

4.7.3.51 p_shmem_long_wait_until_any_vector

```
shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector = nullptr
```

Definition at line 117 of file routines.cpp.

4.7.3.52 p_shmem_long_wait_until_some

```
shmem_long_wait_until_some_func p_shmem_long_wait_until_some = nullptr
```

Definition at line 115 of file routines.cpp.

4.7.3.53 p_shmem_long_wait_until_some_vector

```
shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector = nullptr
```

Definition at line 118 of file routines.cpp.

4.7.3.54 p_shmem_long_xor_reduce

```
shmem_long_xor_reduce_func p_shmem_long_xor_reduce = nullptr
```

Definition at line 105 of file routines.cpp.

4.7.3.55 p_shmem_malloc

```
shmem_malloc_func p_shmem_malloc = nullptr
```

Definition at line 32 of file routines.cpp.

4.7.3.56 p_shmem_malloc_with_hints

```
shmem_malloc_with_hints_func p_shmem_malloc_with_hints = nullptr
```

Definition at line 36 of file routines.cpp.

4.7.3.57 p_shmem_my_pe

```
shmem_my_pe_func p_shmem_my_pe = nullptr
```

Definition at line 17 of file routines.cpp.

4.7.3.58 p_shmem_n_pes

```
shmem_n_pes_func p_shmem_n_pes = nullptr
```

Definition at line 18 of file routines.cpp.

4.7.3.59 p_shmem_pe_accessible

```
shmem_pe_accessible_func p_shmem_pe_accessible = nullptr
```

Definition at line 19 of file routines.cpp.

4.7.3.60 p_shmem_ptr

```
shmem_ptr_func p_shmem_ptr = nullptr
```

Definition at line 31 of file routines.cpp.

4.7.3.61 p_shmem_query_thread

```
shmem_query_thread_func p_shmem_query_thread = nullptr
```

Definition at line 28 of file routines.cpp.

4.7.3.62 p_shmem_quiet

```
shmem_quiet_func p_shmem_quiet = nullptr
```

Definition at line 129 of file routines.cpp.

4.7.3.63 p_shmem_realloc

```
shmem_realloc_func p_shmem_realloc = nullptr
```

Definition at line 34 of file routines.cpp.

4.7.3.64 p_shmem_set_lock

```
shmem_set_lock_func p_shmem_set_lock = nullptr
```

Definition at line 133 of file routines.cpp.

4.7.3.65 p_shmem_signal_fetch

```
shmem_signal_fetch_func p_shmem_signal_fetch = nullptr
```

Definition at line 91 of file routines.cpp.

4.7.3.66 p_shmem_signal_wait_until

```
shmem_signal_wait_until_func p_shmem_signal_wait_until = nullptr
```

Definition at line 126 of file routines.cpp.

4.7.3.67 p_shmem_sync

```
shmem_sync_func p_shmem_sync = nullptr
```

Definition at line 96 of file routines.cpp.

4.7.3.68 p_shmem_sync_all

```
shmem_sync_all_func p_shmem_sync_all = nullptr
```

Definition at line 97 of file routines.cpp.

4.7.3.69 p_shmem_team_create_ctx

```
shmem_team_create_ctx_func p_shmem_team_create_ctx = nullptr
```

Definition at line 51 of file routines.cpp.

4.7.3.70 p_shmem_team_destroy

```
shmem_team_destroy_func p_shmem_team_destroy = nullptr
```

Definition at line 47 of file routines.cpp.

4.7.3.71 p_shmem_team_get_config

```
shmem_team_get_config_func p_shmem_team_get_config = nullptr
```

Definition at line 43 of file routines.cpp.

4.7.3.72 p_shmem_team_my_pe

```
shmem_team_my_pe_func p_shmem_team_my_pe = nullptr
```

Definition at line 41 of file routines.cpp.

4.7.3.73 p_shmem_team_n_pes

```
shmem_team_n_pes_func p_shmem_team_n_pes = nullptr
```

Definition at line 42 of file routines.cpp.

4.7.3.74 p_shmem_team_split_2d

```
shmem_team_split_2d_func p_shmem_team_split_2d = nullptr
```

Definition at line 46 of file routines.cpp.

4.7.3.75 p_shmem_team_split_strided

```
shmem_team_split_strided_func p_shmem_team_split_strided = nullptr
```

Definition at line 45 of file routines.cpp.

4.7.3.76 p_shmem_team_translate_pe

```
shmem_team_translate_pe_func p_shmem_team_translate_pe = nullptr
```

Definition at line 44 of file routines.cpp.

4.7.3.77 p_shmem_ulong_atomic_add

```
shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add = nullptr
```

Definition at line 74 of file routines.cpp.

4.7.3.78 p_shmem_ulong_atomic_and

```
shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and = nullptr
```

Definition at line 76 of file routines.cpp.

4.7.3.79 p_shmem_ulong_atomic_compare_swap

```
shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap = nullptr
```

Definition at line 69 of file routines.cpp.

4.7.3.80 p_shmem_ulong_atomic_compare_swap_nbi

```
shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi = nullptr
```

Definition at line 82 of file routines.cpp.

4.7.3.81 p_shmem_ulong_atomic_fetch

```
shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch = nullptr
```

Definition at line 67 of file routines.cpp.

4.7.3.82 p_shmem_ulong_atomic_fetch_add

```
shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add = nullptr
```

Definition at line 73 of file routines.cpp.

4.7.3.83 p_shmem_ulong_atomic_fetch_add_nbi

```
shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi = nullptr
```

Definition at line 85 of file routines.cpp.

4.7.3.84 p_shmem_ulong_atomic_fetch_and

```
shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and = nullptr
```

Definition at line 75 of file routines.cpp.

4.7.3.85 p_shmem_ulong_atomic_fetch_and_nbi

```
shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi = nullptr
```

Definition at line 86 of file routines.cpp.

4.7.3.86 p_shmem_ulong_atomic_fetch_inc

```
shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc = nullptr
```

Definition at line 71 of file routines.cpp.

4.7.3.87 p_shmem_ulong_atomic_fetch_inc_nbi

```
shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi = nullptr
```

Definition at line 84 of file routines.cpp.

4.7.3.88 p_shmem_ulong_atomic_fetch_nbi

```
shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi = nullptr
```

Definition at line 81 of file routines.cpp.

4.7.3.89 p_shmem_ulong_atomic_fetch_or

```
shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or = nullptr
```

Definition at line 77 of file routines.cpp.

4.7.3.90 p_shmem_ulong_atomic_fetch_or_nbi

```
shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi = nullptr
```

Definition at line 87 of file routines.cpp.

4.7.3.91 p_shmem_ulong_atomic_fetch_xor

```
shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor = nullptr
```

Definition at line 79 of file routines.cpp.

4.7.3.92 p_shmem_ulong_atomic_fetch_xor_nbi

```
shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi = nullptr
```

Definition at line 88 of file routines.cpp.

4.7.3.93 p_shmem_ulong_atomic_inc

```
shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc = nullptr
```

Definition at line 72 of file routines.cpp.

4.7.3.94 p_shmem_ulong_atomic_or

```
shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or = nullptr
```

Definition at line 78 of file routines.cpp.

4.7.3.95 p_shmem_ulong_atomic_set

```
shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set = nullptr
```

Definition at line 68 of file routines.cpp.

4.7.3.96 p_shmem_ulong_atomic_swap

```
shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap = nullptr
```

Definition at line 70 of file routines.cpp.

4.7.3.97 p_shmem_ulong_atomic_swap_nbi

```
shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi = nullptr
```

Definition at line 83 of file routines.cpp.

4.7.3.98 p_shmem_ulong_atomic_xor

```
shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor = nullptr
```

Definition at line 80 of file routines.cpp.

4.8 routines.cpp

Go to the documentation of this file.

```
00001
00006 #include "routines.hpp"
00007
00008 #include <iostream>
00009 #include <dlfcn.h>
00010
00011 /* Define global function pointers */
00012 shmem_fake_routine_func p_shmem_fake_routine = nullptr;
00013
00014 /* Setup, Exit, and Query Routines */
00015 shmem_init_func p_shmem_init = nullptr;
00016 shmem_finalize_func p_shmem_finalize = nullptr;
00017 shmem_my_pe_func p_shmem_my_pe = nullptr;
00018 shmem_n_pes_func p_shmem_n_pes = nullptr;
00019 shmem_pe_accessible_func p_shmem_pe_accessible = nullptr;
00020 shmem_barrier_all_func p_shmem_barrier_all = nullptr;
00021 shmem_barrier_func p_shmem_barrier = nullptr;
00022 shmem_info_get_version_func p_shmem_info_get_version = nullptr;
00023 shmem_info_get_name_func p_shmem_info_get_name = nullptr;
00024 shmem_global_exit_func p_shmem_global_exit = nullptr;
00025
00026 /* Thread Support Routines */
00027 shmem_init_thread_func p_shmem_init_thread = nullptr;
```

```
00028 shmem_query_thread_func p_shmem_query_thread = nullptr;
00029
00030 /* Memory Management Routines */
00031 shmem_ptr_func p_shmem_ptr = nullptr;
00032 shmem_malloc_func p_shmem_malloc = nullptr;
00033 shmem_free_func p_shmem_free = nullptr;
00034 shmem_realloc_func p_shmem_realloc = nullptr;
00035 shmem_align_func p_shmem_align = nullptr;
00036 shmem_malloc_with_hints_func p_shmem_malloc_with_hints = nullptr;
00037 shmem_calloc_func p_shmem_calloc = nullptr;
00038 shmem_addr_accessible_func p_shmem_addr_accessible = nullptr;
00039
00040 /* Team Management Routines */
00041 shmem_team_my_pe_func p_shmem_team_my_pe = nullptr;
00042 shmem_team_n_pes_func p_shmem_team_n_pes = nullptr;
00043 shmem_team_get_config_func p_shmem_team_get_config = nullptr;
00044 shmem_team_translate_pe_func p_shmem_team_translate_pe = nullptr;
00045 shmem_team_split_strided_func p_shmem_team_split_strided = nullptr;
00046 shmem_team_split_2d_func p_shmem_team_split_2d = nullptr;
00047 shmem_team_destroy_func p_shmem_team_destroy = nullptr;
00048
00049 /* Communication/Context Management Routines */
00050 shmem_ctx_create_func p_shmem_ctx_create = nullptr;
00051 shmem_team_create_ctx_func p_shmem_team_create_ctx = nullptr;
00052 shmem_ctx_destroy_func p_shmem_ctx_destroy = nullptr;
00053 shmem_ctx_get_team_func p_shmem_ctx_get_team = nullptr;
00054
00055 /* Remote Access Routines */
00056 shmem_long_put_func p_shmem_long_put = nullptr;
00057 shmem_long_p_func p_shmem_long_p = nullptr;
00058 shmem_long_iput_func p_shmem_long_iput = nullptr;
00059 shmem_long_get_func p_shmem_long_get = nullptr;
00060 shmem_long_g_func p_shmem_long_g = nullptr;
00061 shmem_long_iget_func p_shmem_long_iget = nullptr;
00062 shmem_long_put_nbi_func p_shmem_long_put_nbi = nullptr;
00063 shmem_long_get_nbi_func p_shmem_long_get_nbi = nullptr;
00064
00065
00066 /* Atomic Memory Operations */
00067 shmem_ulong_atomic_fetch_func p_shmem_ulong_atomic_fetch = nullptr;
00068 shmem_ulong_atomic_set_func p_shmem_ulong_atomic_set = nullptr;
00069 shmem_ulong_atomic_compare_swap_func p_shmem_ulong_atomic_compare_swap = nullptr;
00070 shmem_ulong_atomic_swap_func p_shmem_ulong_atomic_swap = nullptr;
00071 shmem_ulong_atomic_fetch_inc_func p_shmem_ulong_atomic_fetch_inc = nullptr;
00072 shmem_ulong_atomic_inc_func p_shmem_ulong_atomic_inc = nullptr;
00073 shmem_ulong_atomic_fetch_add_func p_shmem_ulong_atomic_fetch_add = nullptr;
00074 shmem_ulong_atomic_add_func p_shmem_ulong_atomic_add = nullptr;
00075 shmem_ulong_atomic_fetch_and_func p_shmem_ulong_atomic_fetch_and = nullptr;
00076 shmem_ulong_atomic_and_func p_shmem_ulong_atomic_and = nullptr;
00077 shmem_ulong_atomic_fetch_or_func p_shmem_ulong_atomic_fetch_or = nullptr;
00078 shmem_ulong_atomic_or_func p_shmem_ulong_atomic_or = nullptr;
00079 shmem_ulong_atomic_fetch_xor_func p_shmem_ulong_atomic_fetch_xor = nullptr;
00080 shmem_ulong_atomic_xor_func p_shmem_ulong_atomic_xor = nullptr;
00081 shmem_ulong_atomic_fetch_nbi_func p_shmem_ulong_atomic_fetch_nbi = nullptr;
00082 shmem_ulong_atomic_compare_swap_nbi_func p_shmem_ulong_atomic_compare_swap_nbi = nullptr;
00083 shmem_ulong_atomic_swap_nbi_func p_shmem_ulong_atomic_swap_nbi = nullptr;
00084 shmem_ulong_atomic_fetch_inc_nbi_func p_shmem_ulong_atomic_fetch_inc_nbi = nullptr;
00085 shmem_ulong_atomic_fetch_add_nbi_func p_shmem_ulong_atomic_fetch_add_nbi = nullptr;
00086 shmem_ulong_atomic_fetch_and_nbi_func p_shmem_ulong_atomic_fetch_and_nbi = nullptr;
00087 shmem_ulong_atomic_fetch_or_nbi_func p_shmem_ulong_atomic_fetch_or_nbi = nullptr;
00088 shmem_ulong_atomic_fetch_xor_nbi_func p_shmem_ulong_atomic_fetch_xor_nbi = nullptr;
00089
00090 /* Signaling Operations */
00091 shmem_signal_fetch_func p_shmem_signal_fetch = nullptr;
00092 shmem_long_put_signal_func p_shmem_long_put_signal = nullptr;
00093 shmem_long_put_signal_nbi_func p_shmem_long_put_signal_nbi = nullptr;
00094
00095 /* Collective Routines */
00096 shmem_sync_func p_shmem_sync = nullptr;
00097 shmem_sync_all_func p_shmem_sync_all = nullptr;
00098 shmem_long_alltoall_func p_shmem_long_alltoall = nullptr;
00099 shmem_long_alltoalls_func p_shmem_long_alltoalls = nullptr;
00100 shmem_long_broadcast_func p_shmem_long_broadcast = nullptr;
00101 shmem_long_collect_func p_shmem_long_collect = nullptr;
00102 shmem_long_fcollect_func p_shmem_long_fcollect = nullptr;
00103 shmem_long_and_reduce_func p_shmem_long_and_reduce = nullptr;
00104 shmem_long_or_reduce_func p_shmem_long_or_reduce = nullptr;
00105 shmem_long_xor_reduce_func p_shmem_long_xor_reduce = nullptr;
00106 shmem_long_max_reduce_func p_shmem_long_max_reduce = nullptr;
00107 shmem_long_min_reduce_func p_shmem_long_min_reduce = nullptr;
00108 shmem_long_sum_reduce_func p_shmem_long_sum_reduce = nullptr;
00109 shmem_long_prod_reduce_func p_shmem_long_prod_reduce = nullptr;
00110
00111 /* Point-to-Point Synchronization Routines */
00112 shmem_long_wait_until_func p_shmem_long_wait_until = nullptr;
00113 shmem_long_wait_until_all_func p_shmem_long_wait_until_all = nullptr;
00114 shmem_long_wait_until_any_func p_shmem_long_wait_until_any = nullptr;
```

```
00115 shmem_long_wait_until_some_func p_shmem_long_wait_until_some = nullptr;
00116 shmem_long_wait_until_all_vector_func p_shmem_long_wait_until_all_vector = nullptr;
00117 shmem_long_wait_until_any_vector_func p_shmem_long_wait_until_any_vector = nullptr;
00118 shmem_long_wait_until_some_vector_func p_shmem_long_wait_until_some_vector = nullptr;
00119 shmem_long_test_func p_shmem_long_test = nullptr;
00120 shmem_long_test_all_func p_shmem_long_test_all = nullptr;
00121 shmem_long_test_any_func p_shmem_long_test_any = nullptr;
00122 shmem_long_test_some_func p_shmem_long_test_some = nullptr;
00123 shmem_long_test_all_vector_func p_shmem_long_test_all_vector = nullptr;
00124 shmem_long_test_any_vector_func p_shmem_long_test_any_vector = nullptr;
00125 shmem_long_test_some_vector_func p_shmem_long_test_some_vector = nullptr;
00126 shmem_signal_wait_until_func p_shmem_signal_wait_until = nullptr;
00127
00128 /* Memory Ordering Routines */
00129 shmem_quiet_func p_shmem_quiet = nullptr;
00130 shmem_fence_func p_shmem_fence = nullptr;
00131
00132 /* Distributed Locking Routines */
00133 shmem_set_lock_func p_shmem_set_lock = nullptr;
00134 shmem_clear_lock_func p_shmem_clear_lock = nullptr;
00135
00143 bool load_routines() {
00144   void *handle = dlopen(NULL, RTLD_LAZY);
00145   if (!handle) {
00146     std::cerr « "Failed to open handle: " « dlerror() « std::endl;
00147     return false;
00148   }
00149
00150   p_shmem_fake_routine = reinterpret_cast<shmem_fake_routine_func>(dlsym(handle,
      "shmem_fake_routine"));
00151
00152   /* Setup, Exit, and Query Routines */
00153   p_shmem_init = reinterpret_cast<shmem_init_func>(dlsym(handle, "shmem_init"));
00154   p_shmem_finalize = reinterpret_cast<shmem_finalize_func>(dlsym(handle, "shmem_finalize"));
00155   p_shmem_my_pe = reinterpret_cast<shmem_my_pe_func>(dlsym(handle, "shmem_my_pe"));
00156   p_shmem_n_pes = reinterpret_cast<shmem_n_pes_func>(dlsym(handle, "shmem_n_pes"));
00157   p_shmem_pe_accessible = reinterpret_cast<shmem_pe_accessible_func>(dlsym(handle,
      "shmem_pe_accessible"));
00158   p_shmem_barrier_all = reinterpret_cast<shmem_barrier_all_func>(dlsym(handle, "shmem_barrier_all"));
00159   p_shmem_barrier = reinterpret_cast<shmem_barrier_func>(dlsym(handle, "shmem_barrier"));
00160   p_shmem_info_get_version = reinterpret_cast<shmem_info_get_version_func>(dlsym(handle,
      "shmem_info_get_version"));
00161   p_shmem_info_get_name = reinterpret_cast<shmem_info_get_name_func>(dlsym(handle,
      "shmem_info_get_name"));
00162   p_shmem_global_exit = reinterpret_cast<shmem_global_exit_func>(dlsym(handle, "shmem_global_exit"));
00163
00164   /* Thread Support Routines */
00165   p_shmem_init_thread = reinterpret_cast<shmem_init_thread_func>(dlsym(handle, "shmem_init_thread"));
00166   p_shmem_query_thread = reinterpret_cast<shmem_query_thread_func>(dlsym(handle,
      "shmem_query_thread"));
00167
00168   /* Memory Management Routines */
00169   p_shmem_ptr = reinterpret_cast<shmem_ptr_func>(dlsym(handle, "shmem_ptr"));
00170   p_shmem_malloc = reinterpret_cast<shmem_malloc_func>(dlsym(handle, "shmem_malloc"));
00171   p_shmem_free = reinterpret_cast<shmem_free_func>(dlsym(handle, "shmem_free"));
00172   p_shmem_realloc = reinterpret_cast<shmem_realloc_func>(dlsym(handle, "shmem_realloc"));
00173   p_shmem_align = reinterpret_cast<shmem_align_func>(dlsym(handle, "shmem_align"));
00174   p_shmem_malloc_with_hints = reinterpret_cast<shmem_malloc_with_hints_func>(dlsym(handle,
      "shmem_malloc_with_hints"));
00175   p_shmem_calloc = reinterpret_cast<shmem_calloc_func>(dlsym(handle, "shmem_calloc"));
00176   p_shmem_addr_accessible = reinterpret_cast<shmem_addr_accessible_func>(dlsym(handle,
      "shmem_addr_accessible"));
00177
00178   /* Team Management Routines */
00179   p_shmem_team_my_pe = reinterpret_cast<shmem_team_my_pe_func>(dlsym(handle, "shmem_team_my_pe"));
00180   p_shmem_team_n_pes = reinterpret_cast<shmem_team_n_pes_func>(dlsym(handle, "shmem_team_n_pes"));
00181   p_shmem_team_get_config = reinterpret_cast<shmem_team_get_config_func>(dlsym(handle,
      "shmem_team_get_config"));
00182   p_shmem_team_translate_pe = reinterpret_cast<shmem_team_translate_pe_func>(dlsym(handle,
      "shmem_team_translate_pe"));
00183   p_shmem_team_split_strided = reinterpret_cast<shmem_team_split_strided_func>(dlsym(handle,
      "shmem_team_split_strided"));
00184   p_shmem_team_split_2d = reinterpret_cast<shmem_team_split_2d_func>(dlsym(handle,
      "shmem_team_split_2d"));
00185   p_shmem_team_destroy = reinterpret_cast<shmem_team_destroy_func>(dlsym(handle,
      "shmem_team_destroy"));
00186
00187   /* Communication/Context Management Routines */
00188   p_shmem_ctx_create = reinterpret_cast<shmem_ctx_create_func>(dlsym(handle, "shmem_ctx_create"));
00189   p_shmem_team_create_ctx = reinterpret_cast<shmem_team_create_ctx_func>(dlsym(handle,
      "shmem_team_create_ctx"));
00190   p_shmem_ctx_destroy = reinterpret_cast<shmem_ctx_destroy_func>(dlsym(handle, "shmem_ctx_destroy"));
00191   p_shmem_ctx_get_team = reinterpret_cast<shmem_ctx_get_team_func>(dlsym(handle,
      "shmem_ctx_get_team"));
00192
00193   /* Remote Access Routines */
00194   p_shmem_long_put = reinterpret_cast<shmem_long_put_func>(dlsym(handle, "shmem_long_put"));
```

```
00195   p_shmem_long_p = reinterpret_cast<shmem_long_p_func>(dlsym(handle, "shmem_long_p"));
00196   p_shmem_long_iput = reinterpret_cast<shmem_long_iput_func>(dlsym(handle, "shmem_long_iput"));
00197   p_shmem_long_get = reinterpret_cast<shmem_long_get_func>(dlsym(handle, "shmem_long_get"));
00198   p_shmem_long_g = reinterpret_cast<shmem_long_g_func>(dlsym(handle, "shmem_long_g"));
00199   p_shmem_long_iget = reinterpret_cast<shmem_long_iget_func>(dlsym(handle, "shmem_long_iget"));
00200   p_shmem_long_put_nbi = reinterpret_cast<shmem_long_put_nbi_func>(dlsym(handle,
      "shmem_long_put_nbi"));
00201   p_shmem_long_get_nbi = reinterpret_cast<shmem_long_get_nbi_func>(dlsym(handle,
      "shmem_long_get_nbi"));
00202
00203   /* Atomic Memory Operations */
00204   p_shmem_ulong_atomic_fetch = reinterpret_cast<shmem_ulong_atomic_fetch_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch"));
00205   p_shmem_ulong_atomic_set = reinterpret_cast<shmem_ulong_atomic_set_func>(dlsym(handle,
      "shmem_ulong_atomic_set"));
00206   p_shmem_ulong_atomic_compare_swap =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap"));
00207   p_shmem_ulong_atomic_swap = reinterpret_cast<shmem_ulong_atomic_swap_func>(dlsym(handle,
      "shmem_ulong_atomic_swap"));
00208   p_shmem_ulong_atomic_fetch_inc = reinterpret_cast<shmem_ulong_atomic_fetch_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc"));
00209   p_shmem_ulong_atomic_inc = reinterpret_cast<shmem_ulong_atomic_inc_func>(dlsym(handle,
      "shmem_ulong_atomic_inc"));
00210   p_shmem_ulong_atomic_fetch_add = reinterpret_cast<shmem_ulong_atomic_fetch_add_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add"));
00211   p_shmem_ulong_atomic_add = reinterpret_cast<shmem_ulong_atomic_add_func>(dlsym(handle,
      "shmem_ulong_atomic_add"));
00212   p_shmem_ulong_atomic_fetch_and = reinterpret_cast<shmem_ulong_atomic_fetch_and_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and"));
00213   p_shmem_ulong_atomic_and = reinterpret_cast<shmem_ulong_atomic_and_func>(dlsym(handle,
      "shmem_ulong_atomic_and"));
00214   p_shmem_ulong_atomic_fetch_or = reinterpret_cast<shmem_ulong_atomic_fetch_or_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or"));
00215   p_shmem_ulong_atomic_or = reinterpret_cast<shmem_ulong_atomic_or_func>(dlsym(handle,
      "shmem_ulong_atomic_or"));
00216   p_shmem_ulong_atomic_fetch_xor = reinterpret_cast<shmem_ulong_atomic_fetch_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor"));
00217   p_shmem_ulong_atomic_xor = reinterpret_cast<shmem_ulong_atomic_xor_func>(dlsym(handle,
      "shmem_ulong_atomic_xor"));
00218
00219   p_shmem_ulong_atomic_fetch_nbi = reinterpret_cast<shmem_ulong_atomic_fetch_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_nbi"));
00220   p_shmem_ulong_atomic_compare_swap_nbi =
      reinterpret_cast<shmem_ulong_atomic_compare_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_compare_swap_nbi"));
00221   p_shmem_ulong_atomic_swap_nbi = reinterpret_cast<shmem_ulong_atomic_swap_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_swap_nbi"));
00222   p_shmem_ulong_atomic_fetch_inc_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_inc_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_inc_nbi"));
00223   p_shmem_ulong_atomic_fetch_add_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_add_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_add_nbi"));
00224   p_shmem_ulong_atomic_fetch_and_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_and_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_and_nbi"));
00225   p_shmem_ulong_atomic_fetch_or_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_or_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_or_nbi"));
00226   p_shmem_ulong_atomic_fetch_xor_nbi =
      reinterpret_cast<shmem_ulong_atomic_fetch_xor_nbi_func>(dlsym(handle,
      "shmem_ulong_atomic_fetch_xor_nbi"));
00227
00228   /* Signaling Operations */
00229   p_shmem_signal_fetch = reinterpret_cast<shmem_signal_fetch_func>(dlsym(handle,
      "shmem_signal_fetch"));
00230   p_shmem_long_put_signal = reinterpret_cast<shmem_long_put_signal_func>(dlsym(handle,
      "shmem_long_put_signal"));
00231   p_shmem_long_put_signal_nbi = reinterpret_cast<shmem_long_put_signal_nbi_func>(dlsym(handle,
      "shmem_long_put_signal_nbi"));
00232
00233   /* Collective Routines */
00234   p_shmem_sync = reinterpret_cast<shmem_sync_func>(dlsym(handle, "shmem_sync"));
00235   p_shmem_sync_all = reinterpret_cast<shmem_sync_all_func>(dlsym(handle, "shmem_sync_all"));
00236   p_shmem_long_alltoall = reinterpret_cast<shmem_long_alltoall_func>(dlsym(handle,
      "shmem_long_alltoall"));
00237   p_shmem_long_alltoalls = reinterpret_cast<shmem_long_alltoalls_func>(dlsym(handle,
      "shmem_long_alltoalls"));
00238   p_shmem_long_broadcast = reinterpret_cast<shmem_long_broadcast_func>(dlsym(handle,
      "shmem_long_broadcast"));
00239   p_shmem_long_collect = reinterpret_cast<shmem_long_collect_func>(dlsym(handle,
      "shmem_long_collect"));
00240   p_shmem_long_fcollect = reinterpret_cast<shmem_long_fcollect_func>(dlsym(handle,
      "shmem_long_fcollect"));
00241   p_shmem_long_and_reduce = reinterpret_cast<shmem_long_and_reduce_func>(dlsym(handle,
      "shmem_long_and_reduce"));
```

```
00242   p_shmem_long_or_reduce = reinterpret_cast<shmem_long_or_reduce_func>(dlsym(handle,
      "shmem_long_or_reduce"));
00243   p_shmem_long_xor_reduce = reinterpret_cast<shmem_long_xor_reduce_func>(dlsym(handle,
      "shmem_long_xor_reduce"));
00244   p_shmem_long_max_reduce = reinterpret_cast<shmem_long_max_reduce_func>(dlsym(handle,
      "shmem_long_max_reduce"));
00245   p_shmem_long_min_reduce = reinterpret_cast<shmem_long_min_reduce_func>(dlsym(handle,
      "shmem_long_min_reduce"));
00246   p_shmem_long_sum_reduce = reinterpret_cast<shmem_long_sum_reduce_func>(dlsym(handle,
      "shmem_long_sum_reduce"));
00247   p_shmem_long_prod_reduce = reinterpret_cast<shmem_long_prod_reduce_func>(dlsym(handle,
      "shmem_long_prod_reduce"));
00248
00249   /* Point-to-Point Synchronization Routines */
00250   p_shmem_long_wait_until = reinterpret_cast<shmem_long_wait_until_func>(dlsym(handle,
      "shmem_long_wait_until"));
00251   p_shmem_long_wait_until_all = reinterpret_cast<shmem_long_wait_until_all_func>(dlsym(handle,
      "shmem_long_wait_until_all"));
00252   p_shmem_long_wait_until_any = reinterpret_cast<shmem_long_wait_until_any_func>(dlsym(handle,
      "shmem_long_wait_until_any"));
00253   p_shmem_long_wait_until_some = reinterpret_cast<shmem_long_wait_until_some_func>(dlsym(handle,
      "shmem_long_wait_until_some"));
00254   p_shmem_long_wait_until_all_vector =
      reinterpret_cast<shmem_long_wait_until_all_vector_func>(dlsym(handle,
      "shmem_long_wait_until_all_vector"));
00255   p_shmem_long_wait_until_any_vector =
      reinterpret_cast<shmem_long_wait_until_any_vector_func>(dlsym(handle,
      "shmem_long_wait_until_any_vector"));
00256   p_shmem_long_wait_until_some_vector =
      reinterpret_cast<shmem_long_wait_until_some_vector_func>(dlsym(handle,
      "shmem_long_wait_until_some_vector"));
00257   p_shmem_long_test = reinterpret_cast<shmem_long_test_func>(dlsym(handle, "shmem_long_test"));
00258   p_shmem_long_test_all = reinterpret_cast<shmem_long_test_all_func>(dlsym(handle,
      "shmem_long_test_all"));
00259   p_shmem_long_test_any = reinterpret_cast<shmem_long_test_any_func>(dlsym(handle,
      "shmem_long_test_any"));
00260   p_shmem_long_test_some = reinterpret_cast<shmem_long_test_some_func>(dlsym(handle,
      "shmem_long_test_some"));
00261   p_shmem_long_test_all_vector = reinterpret_cast<shmem_long_test_all_vector_func>(dlsym(handle,
      "shmem_long_test_all_vector"));
00262   p_shmem_long_test_any_vector = reinterpret_cast<shmem_long_test_any_vector_func>(dlsym(handle,
      "shmem_long_test_any_vector"));
00263   p_shmem_long_test_some_vector = reinterpret_cast<shmem_long_test_some_vector_func>(dlsym(handle,
      "shmem_long_test_some_vector"));
00264   p_shmem_signal_wait_until = reinterpret_cast<shmem_signal_wait_until_func>(dlsym(handle,
      "shmem_signal_wait_until"));
00265
00266   /* Memory Ordering Routines */
00267   p_shmem_quiet = reinterpret_cast<shmem_quiet_func>(dlsym(handle, "shmem_quiet"));
00268   p_shmem_fence = reinterpret_cast<shmem_fence_func>(dlsym(handle, "shmem_fence"));
00269
00270   /* Distributed Locking Routines */
00271   p_shmem_set_lock = reinterpret_cast<shmem_set_lock_func>(dlsym(handle, "shmem_set_lock"));
00272   p_shmem_clear_lock = reinterpret_cast<shmem_clear_lock_func>(dlsym(handle, "shmem_clear_lock"));
00273
00274   const char *dlsym_error = dlerror();
00275   if (dlsym_error) {
00276     std::cerr « "Error loading functions: " « dlsym_error « std::endl;
00277     dlclose(handle);
00278     return false;
00279   }
00280
00281   return true;
00282 }
```

4.9 src/tests/atomics/atomics_tests.cpp File Reference

Contains tests for OpenSHMEM atomic routines.

```
#include "atomics_tests.hpp"
```

Functions

- bool test_shmem_atomic_fetch ()

Tests the shmem_atomic_fetch() routine.

- bool test_shmem_atomic_set ()

  Tests the shmem_atomic_set() routine.
- bool test_shmem_atomic_compare_swap ()

  Tests the shmem_atomic_compare_swap() routine.
- bool test_shmem_atomic_swap ()

  Tests the shmem_atomic_swap() routine.
- bool test_shmem_atomic_fetch_inc ()

  Tests the shmem_atomic_fetch_inc() routine.
- bool test_shmem_atomic_inc ()

  Tests the shmem_atomic_inc() routine.
- bool test_shmem_atomic_fetch_add ()

  Tests the shmem_atomic_fetch_add() routine.
- bool test_shmem_atomic_add ()

  Tests the shmem_atomic_add() routine.
- bool test_shmem_atomic_fetch_and ()

  Tests the shmem_atomic_fetch_and() routine.
- bool test_shmem_atomic_and ()

  Tests the shmem_atomic_and() routine.
- bool test_shmem_atomic_fetch_or ()

  Tests the shmem_atomic_fetch_or() routine.
- bool test_shmem_atomic_or ()

  Tests the shmem_atomic_or() routine.
- bool test_shmem_atomic_fetch_xor ()

  Tests the shmem_atomic_fetch_xor() routine.
- bool test_shmem_atomic_xor ()

  Tests the shmem_atomic_xor() routine.
- bool test_shmem_atomic_fetch_nbi ()

  Tests the shmem_atomic_fetch_nbi() routine.
- bool test_shmem_atomic_compare_swap_nbi ()

  Tests the shmem_atomic_compare_swap_nbi() routine.
- bool test_shmem_atomic_swap_nbi ()

  Tests the shmem_atomic_swap_nbi() routine.
- bool test_shmem_atomic_fetch_inc_nbi ()

  Tests the shmem_atomic_fetch_inc_nbi() routine.
- bool test_shmem_atomic_fetch_add_nbi ()

  Tests the shmem_atomic_fetch_add_nbi() routine.
- bool test_shmem_atomic_fetch_and_nbi ()

  Tests the shmem_atomic_fetch_and_nbi() routine.
- bool test_shmem_atomic_fetch_or_nbi ()

  Tests the shmem_atomic_fetch_or_nbi() routine.
- bool test_shmem_atomic_fetch_xor_nbi ()

  Tests the shmem_atomic_fetch_xor_nbi() routine.

4.9.1 Detailed Description

Contains tests for OpenSHMEM atomic routines.

Definition in file atomics_tests.cpp.

4.9.2 Function Documentation

4.9.2.1 test_shmem_atomic_add()

```
bool test_shmem_atomic_add (
            void )
```

Tests the shmem_atomic_add() routine.

This test verifies that the shmem_atomic_add() routine correctly adds a value to the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 175 of file atomics_tests.cpp.

```
00175                                {
00176   static ulong *dest;
00177   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00178   ulong value = 42, add_val = 10;
00179   *dest = value;
00180   p_shmem_barrier_all();
00181   int mype = p_shmem_my_pe();
00182   p_shmem_ulong_atomic_add(dest, add_val, mype);
00183   p_shmem_barrier_all();
00184   bool success = (*dest == value + add_val);
00185   p_shmem_free(dest);
00186   return success;
00187 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_add.

4.9.2.2 test_shmem_atomic_and()

```
bool test_shmem_atomic_and (
            void )
```

Tests the shmem_atomic_and() routine.

This test verifies that the shmem_atomic_and() routine correctly performs a bitwise AND operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 220 of file atomics_tests.cpp.

```
00220                                {
00221   static ulong *dest;
00222   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00223   ulong value = 42, and_val = 15;
00224   *dest = value;
00225   p_shmem_barrier_all();
00226   int mype = p_shmem_my_pe();
00227   p_shmem_ulong_atomic_and(dest, and_val, mype);
00228   p_shmem_barrier_all();
00229   bool success = (*dest == (value & and_val));
00230   p_shmem_free(dest);
00231   return success;
00232 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_and.

4.9.2.3 test_shmem_atomic_compare_swap()

```
bool test_shmem_atomic_compare_swap (
            void )
```

Tests the shmem_atomic_compare_swap() routine.

This test verifies that the shmem_atomic_compare_swap() routine correctly swaps the value at a remote memory location if it matches the expected value.

Returns

True if the test is successful, false otherwise.

Definition at line 60 of file atomics_tests.cpp.

```
00060                                       {
00061   static ulong *dest;
00062   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00063   ulong old = 42, new_val = 43;
00064   *dest = old;
00065   p_shmem_barrier_all();
00066   int mype = p_shmem_my_pe();
00067   int npes = p_shmem_n_pes();
00068   p_shmem_barrier_all();
00069   ulong swapped = p_shmem_ulong_atomic_compare_swap(dest, old, new_val, (mype + 1) % npes);
00070   p_shmem_barrier_all();
00071   bool success = (swapped == old && *dest == new_val);
00072   p_shmem_barrier_all();
00073   p_shmem_free(dest);
00074   return success;
00075 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_ulong_atomic_compare_swap.

4.9.2.4 test_shmem_atomic_compare_swap_nbi()

```
bool test_shmem_atomic_compare_swap_nbi (
            void )
```

Tests the shmem_atomic_compare_swap_nbi() routine.

This test verifies that the shmem_atomic_compare_swap_nbi() routine correctly swaps the value at a remote memory location in a non-blocking manner if it matches the expected value.

Returns

True if the test is successful, false otherwise.

Definition at line 356 of file atomics_tests.cpp.

```
00356                                           {
00357   static ulong *dest;
00358   static ulong fetch;
00359   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00360   fetch = 0;
00361   ulong old = 42, new_val = 43;
00362   *dest = old;
00363   p_shmem_barrier_all();
00364   int mype = p_shmem_my_pe();
00365   p_shmem_ulong_atomic_compare_swap_nbi(&fetch, dest, old, new_val, mype);
00366   p_shmem_quiet();
00367   p_shmem_barrier_all();
00368   bool success = (fetch == old && *dest == new_val);
00369   p_shmem_free(dest);
00370   return success;
00371 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_compare_swap_nbi.

4.9.2.5 test_shmem_atomic_fetch()

```
bool test_shmem_atomic_fetch (
            void )
```

Tests the shmem_atomic_fetch() routine.

This test verifies that the shmem_atomic_fetch() routine correctly retrieves the value from a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file atomics_tests.cpp.

```
00016                                   {
00017   static ulong *dest;
00018   static ulong fetch;
00019   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00020   ulong value = 42;
00021   *dest = value;
00022   p_shmem_barrier_all();
00023   int mype = p_shmem_my_pe();
00024   fetch = p_shmem_ulong_atomic_fetch(dest, mype);
00025   p_shmem_barrier_all();
00026   bool success = (fetch == value);
00027   p_shmem_free(dest);
00028   return success;
00029 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch.

4.9.2.6 test_shmem_atomic_fetch_add()

```
bool test_shmem_atomic_fetch_add (
            void )
```

Tests the shmem_atomic_fetch_add() routine.

This test verifies that the shmem_atomic_fetch_add() routine correctly adds a value to the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 152 of file atomics_tests.cpp.

```
00152                                       {
00153   static ulong *dest;
00154   static ulong fetch;
00155   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00156   ulong value = 42, add_val = 10;
00157   *dest = value;
00158   p_shmem_barrier_all();
00159   int mype = p_shmem_my_pe();
00160   fetch = p_shmem_ulong_atomic_fetch_add(dest, add_val, mype);
00161   p_shmem_barrier_all();
00162   bool success = (fetch == value && *dest == value + add_val);
00163   p_shmem_free(dest);
00164   return success;
00165 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_add.

4.9.2.7 test_shmem_atomic_fetch_add_nbi()

```
bool test_shmem_atomic_fetch_add_nbi (
            void )
```

Tests the shmem_atomic_fetch_add_nbi() routine.

This test verifies that the shmem_atomic_fetch_add_nbi() routine correctly adds a value to the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 431 of file atomics_tests.cpp.

```
00431                                          {
00432   static ulong *dest;
00433   static ulong fetch;
00434   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00435   fetch = 0;
00436   ulong value = 42, add_val = 10;
00437   *dest = value;
00438   p_shmem_barrier_all();
00439   int mype = p_shmem_my_pe();
00440   p_shmem_ulong_atomic_fetch_add_nbi(&fetch, dest, add_val, mype);
00441   p_shmem_quiet();
00442   p_shmem_barrier_all();
00443   bool success = (fetch == value && *dest == value + add_val);
00444   p_shmem_free(dest);
00445   return success;
00446 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_add_nbi.

4.9.2.8 test_shmem_atomic_fetch_and()

```
bool test_shmem_atomic_fetch_and (
            void )
```

Tests the shmem_atomic_fetch_and() routine.

This test verifies that the shmem_atomic_fetch_and() routine correctly performs a bitwise AND operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 197 of file atomics_tests.cpp.

```
00197                                       {
00198   static ulong *dest;
00199   static ulong fetch;
00200   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00201   ulong value = 42, and_val = 15;
00202   *dest = value;
00203   p_shmem_barrier_all();
00204   int mype = p_shmem_my_pe();
00205   fetch = p_shmem_ulong_atomic_fetch_and(dest, and_val, mype);
00206   p_shmem_barrier_all();
00207   bool success = (fetch == value && *dest == (value & and_val));
00208   p_shmem_free(dest);
00209   return success;
00210 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_and.

4.9.2.9 test_shmem_atomic_fetch_and_nbi()

```
bool test_shmem_atomic_fetch_and_nbi (
            void )
```

Tests the shmem_atomic_fetch_and_nbi() routine.

This test verifies that the shmem_atomic_fetch_and_nbi() routine correctly performs a bitwise AND operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 456 of file atomics_tests.cpp.

```
00456                                          {
00457   static ulong *dest;
00458   static ulong fetch;
00459   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00460   fetch = 0;
00461   ulong value = 42, and_val = 15;
00462   *dest = value;
00463   p_shmem_barrier_all();
00464   int mype = p_shmem_my_pe();
00465   p_shmem_ulong_atomic_fetch_and_nbi(&fetch, dest, and_val, mype);
00466   p_shmem_quiet();
00467   p_shmem_barrier_all();
00468   bool success = (fetch == value && *dest == (value & and_val));
00469   p_shmem_free(dest);
00470   return success;
00471 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_and_nbi.

4.9.2.10 test_shmem_atomic_fetch_inc()

```
bool test_shmem_atomic_fetch_inc (
            void )
```

Tests the shmem_atomic_fetch_inc() routine.

This test verifies that the shmem_atomic_fetch_inc() routine correctly increments the value at a remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 107 of file atomics_tests.cpp.

```
00107                                       {
00108   static ulong *dest;
00109   static ulong fetch;
00110   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00111   ulong value = 42;
00112   *dest = value;
00113   p_shmem_barrier_all();
00114   int mype = p_shmem_my_pe();
00115   fetch = p_shmem_ulong_atomic_fetch_inc(dest, mype);
00116   p_shmem_barrier_all();
00117   bool success = (fetch == value && *dest == value + 1);
00118   p_shmem_free(dest);
00119   return success;
00120 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_inc.

4.9.2.11 test_shmem_atomic_fetch_inc_nbi()

```
bool test_shmem_atomic_fetch_inc_nbi (
            void )
```

Tests the shmem_atomic_fetch_inc_nbi() routine.

This test verifies that the shmem_atomic_fetch_inc_nbi() routine correctly increments the value at a remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 406 of file atomics_tests.cpp.

```
00406                                           {
00407   static ulong *dest;
00408   static ulong fetch;
00409   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00410   fetch = 0;
00411   ulong value = 42;
00412   *dest = value;
00413   p_shmem_barrier_all();
00414   int mype = p_shmem_my_pe();
00415   p_shmem_ulong_atomic_fetch_inc_nbi(&fetch, dest, mype);
00416   p_shmem_quiet();
00417   p_shmem_barrier_all();
00418   bool success = (fetch == value && *dest == value + 1);
00419   p_shmem_free(dest);
00420   return success;
00421 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_inc_nbi.

4.9.2.12 test_shmem_atomic_fetch_nbi()

```
bool test_shmem_atomic_fetch_nbi (
            void )
```

Tests the shmem_atomic_fetch_nbi() routine.

This test verifies that the shmem_atomic_fetch_nbi() routine correctly retrieves the value from a remote memory location in a non-blocking manner.

Returns

True if the test is successful, false otherwise.

Definition at line 332 of file atomics_tests.cpp.

```
00332                                           {
00333   static ulong *dest;
00334   static ulong fetch;
00335   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00336   ulong value = 42;
00337   *dest = value;
00338   p_shmem_barrier_all();
00339   int mype = p_shmem_my_pe();
00340   p_shmem_ulong_atomic_fetch_nbi(&fetch, dest, mype);
00341   p_shmem_quiet();
00342   p_shmem_barrier_all();
00343   bool success = (fetch == value);
00344   p_shmem_free(dest);
00345   return success;
00346 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_nbi.

4.9.2.13 test_shmem_atomic_fetch_or()

```
bool test_shmem_atomic_fetch_or (
            void )
```

Tests the shmem_atomic_fetch_or() routine.

This test verifies that the shmem_atomic_fetch_or() routine correctly performs a bitwise OR operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 242 of file atomics_tests.cpp.

```
00242                                    {
00243   static ulong *dest;
00244   static ulong fetch;
00245   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00246   ulong value = 42, or_val = 15;
00247   *dest = value;
00248   p_shmem_barrier_all();
00249   int mype = p_shmem_my_pe();
00250   fetch = p_shmem_ulong_atomic_fetch_or(dest, or_val, mype);
00251   p_shmem_barrier_all();
00252   bool success = (fetch == value && *dest == (value | or_val));
00253   p_shmem_free(dest);
00254   return success;
00255 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_or.

4.9.2.14 test_shmem_atomic_fetch_or_nbi()

```
bool test_shmem_atomic_fetch_or_nbi (
            void )
```

Tests the shmem_atomic_fetch_or_nbi() routine.

This test verifies that the shmem_atomic_fetch_or_nbi() routine correctly performs a bitwise OR operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 481 of file atomics_tests.cpp.

```
00481                                        {
00482   static ulong *dest;
00483   static ulong fetch;
00484   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00485   fetch = 0;
00486   ulong value = 42, or_val = 15;
00487   *dest = value;
00488   p_shmem_barrier_all();
00489   int mype = p_shmem_my_pe();
00490   p_shmem_ulong_atomic_fetch_or_nbi(&fetch, dest, or_val, mype);
00491   p_shmem_quiet();
00492   p_shmem_barrier_all();
00493   bool success = (fetch == value && *dest == (value | or_val));
00494   p_shmem_free(dest);
00495   return success;
00496 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_or_nbi.

4.9.2.15 test_shmem_atomic_fetch_xor()

```
bool test_shmem_atomic_fetch_xor (
            void )
```

Tests the shmem_atomic_fetch_xor() routine.

This test verifies that the shmem_atomic_fetch_xor() routine correctly performs a bitwise XOR operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 287 of file atomics_tests.cpp.

```
00287                                       {
00288   static ulong *dest;
00289   static ulong fetch;
00290   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00291   ulong value = 42, xor_val = 15;
00292   *dest = value;
00293   p_shmem_barrier_all();
00294   int mype = p_shmem_my_pe();
00295   fetch = p_shmem_ulong_atomic_fetch_xor(dest, xor_val, mype);
00296   p_shmem_barrier_all();
00297   bool success = (fetch == value && *dest == (value ^ xor_val));
00298   p_shmem_free(dest);
00299   return success;
00300 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_xor.

4.9.2.16 test_shmem_atomic_fetch_xor_nbi()

```
bool test_shmem_atomic_fetch_xor_nbi (
            void )
```

Tests the shmem_atomic_fetch_xor_nbi() routine.

This test verifies that the shmem_atomic_fetch_xor_nbi() routine correctly performs a bitwise XOR operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 506 of file atomics_tests.cpp.

```
00506                                           {
00507   static ulong *dest;
00508   static ulong fetch;
00509   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00510   fetch = 0;
00511   ulong value = 42, xor_val = 15;
00512   *dest = value;
00513   p_shmem_barrier_all();
00514   int mype = p_shmem_my_pe();
00515   p_shmem_ulong_atomic_fetch_xor_nbi(&fetch, dest, xor_val, mype);
00516   p_shmem_quiet();
00517   p_shmem_barrier_all();
00518   bool success = (fetch == value && *dest == (value ^ xor_val));
00519   p_shmem_free(dest);
00520   return success;
00521 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_xor_nbi.

4.9.2.17 test_shmem_atomic_inc()

```
bool test_shmem_atomic_inc (
            void )
```

Tests the shmem_atomic_inc() routine.

This test verifies that the shmem_atomic_inc() routine correctly increments the value at a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 130 of file atomics_tests.cpp.

```
00130                                {
00131   static ulong *dest;
00132   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00133   ulong value = 42;
00134   *dest = value;
00135   p_shmem_barrier_all();
00136   int mype = p_shmem_my_pe();
00137   p_shmem_ulong_atomic_inc(dest, mype);
00138   p_shmem_barrier_all();
00139   bool success = (*dest == value + 1);
00140   p_shmem_free(dest);
00141   return success;
00142 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_inc.

4.9.2.18 test_shmem_atomic_or()

```
bool test_shmem_atomic_or (
            void )
```

Tests the shmem_atomic_or() routine.

This test verifies that the shmem_atomic_or() routine correctly performs a bitwise OR operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 265 of file atomics_tests.cpp.

```
00265                                {
00266   static ulong *dest;
00267   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00268   ulong value = 42, or_val = 15;
00269   *dest = value;
00270   p_shmem_barrier_all();
00271   int mype = p_shmem_my_pe();
00272   p_shmem_ulong_atomic_or(dest, or_val, mype);
00273   p_shmem_barrier_all();
00274   bool success = (*dest == (value | or_val));
00275   p_shmem_free(dest);
00276   return success;
00277 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_or.

4.9.2.19 test_shmem_atomic_set()

```
bool test_shmem_atomic_set (
            void )
```

Tests the shmem_atomic_set() routine.

This test verifies that the shmem_atomic_set() routine correctly sets the value at a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 39 of file atomics_tests.cpp.

```
00039                                {
00040   static ulong *dest;
00041   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00042   ulong value = 42;
00043   p_shmem_barrier_all();
00044   int mype = p_shmem_my_pe();
00045   p_shmem_ulong_atomic_set(dest, value, mype);
00046   p_shmem_barrier_all();
00047   bool success = (*dest == value);
00048   p_shmem_free(dest);
00049   return success;
00050 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_set.

4.9.2.20 test_shmem_atomic_swap()

```
bool test_shmem_atomic_swap (
            void )
```

Tests the shmem_atomic_swap() routine.

This test verifies that the shmem_atomic_swap() routine correctly swaps the value at a remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 85 of file atomics_tests.cpp.

```
00085                                {
00086   static ulong *dest;
00087   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00088   ulong value = 42, new_val = 43;
00089   *dest = value;
00090   p_shmem_barrier_all();
00091   int mype = p_shmem_my_pe();
00092   ulong swapped = p_shmem_ulong_atomic_swap(dest, new_val, mype);
00093   p_shmem_barrier_all();
00094   bool success = (swapped == value && *dest == new_val);
00095   p_shmem_free(dest);
00096   return success;
00097 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_swap.

4.9.2.21 test_shmem_atomic_swap_nbi()

```
bool test_shmem_atomic_swap_nbi (
            void )
```

Tests the shmem_atomic_swap_nbi() routine.

This test verifies that the shmem_atomic_swap_nbi() routine correctly swaps the value at a remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 381 of file atomics_tests.cpp.

```
00381                                    {
00382   static ulong *dest;
00383   static ulong fetch;
00384   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00385   fetch = 0;
00386   ulong value = 42, new_val = 43;
00387   *dest = value;
00388   p_shmem_barrier_all();
00389   int mype = p_shmem_my_pe();
00390   p_shmem_ulong_atomic_swap_nbi(&fetch, dest, new_val, mype);
00391   p_shmem_quiet();
00392   p_shmem_barrier_all();
00393   bool success = (fetch == value && *dest == new_val);
00394   p_shmem_free(dest);
00395   return success;
00396 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_swap_nbi.

4.9.2.22 test_shmem_atomic_xor()

```
bool test_shmem_atomic_xor (
            void )
```

Tests the shmem_atomic_xor() routine.

This test verifies that the shmem_atomic_xor() routine correctly performs a bitwise XOR operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 310 of file atomics_tests.cpp.

```
00310                                {
00311   static ulong *dest;
00312   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00313   ulong value = 42, xor_val = 15;
00314   *dest = value;
00315   p_shmem_barrier_all();
00316   int mype = p_shmem_my_pe();
00317   p_shmem_ulong_atomic_xor(dest, xor_val, mype);
00318   p_shmem_barrier_all();
00319   bool success = (*dest == (value ^ xor_val));
00320   p_shmem_free(dest);
00321   return success;
00322 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_xor.

4.10 atomics_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "atomics_tests.hpp"
00007
00016 bool test_shmem_atomic_fetch() {
00017   static ulong *dest;
00018   static ulong fetch;
00019   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00020   ulong value = 42;
00021   *dest = value;
00022   p_shmem_barrier_all();
00023   int mype = p_shmem_my_pe();
00024   fetch = p_shmem_ulong_atomic_fetch(dest, mype);
00025   p_shmem_barrier_all();
00026   bool success = (fetch == value);
00027   p_shmem_free(dest);
00028   return success;
00029 }
00030
00039 bool test_shmem_atomic_set() {
00040   static ulong *dest;
00041   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00042   ulong value = 42;
00043   p_shmem_barrier_all();
00044   int mype = p_shmem_my_pe();
00045   p_shmem_ulong_atomic_set(dest, value, mype);
00046   p_shmem_barrier_all();
00047   bool success = (*dest == value);
00048   p_shmem_free(dest);
00049   return success;
00050 }
00051
00060 bool test_shmem_atomic_compare_swap() {
00061   static ulong *dest;
00062   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00063   ulong old = 42, new_val = 43;
00064   *dest = old;
00065   p_shmem_barrier_all();
00066   int mype = p_shmem_my_pe();
00067   int npes = p_shmem_n_pes();
00068   p_shmem_barrier_all();
00069   ulong swapped = p_shmem_ulong_atomic_compare_swap(dest, old, new_val, (mype + 1) % npes);
00070   p_shmem_barrier_all();
00071   bool success = (swapped == old && *dest == new_val);
00072   p_shmem_barrier_all();
00073   p_shmem_free(dest);
00074   return success;
00075 }
00076
00085 bool test_shmem_atomic_swap() {
00086   static ulong *dest;
00087   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00088   ulong value = 42, new_val = 43;
00089   *dest = value;
00090   p_shmem_barrier_all();
00091   int mype = p_shmem_my_pe();
00092   ulong swapped = p_shmem_ulong_atomic_swap(dest, new_val, mype);
00093   p_shmem_barrier_all();
00094   bool success = (swapped == value && *dest == new_val);
00095   p_shmem_free(dest);
00096   return success;
00097 }
00098
00107 bool test_shmem_atomic_fetch_inc() {
00108   static ulong *dest;
00109   static ulong fetch;
00110   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00111   ulong value = 42;
00112   *dest = value;
00113   p_shmem_barrier_all();
00114   int mype = p_shmem_my_pe();
00115   fetch = p_shmem_ulong_atomic_fetch_inc(dest, mype);
00116   p_shmem_barrier_all();
00117   bool success = (fetch == value && *dest == value + 1);
00118   p_shmem_free(dest);
00119   return success;
00120 }
00121
00130 bool test_shmem_atomic_inc() {
00131   static ulong *dest;
00132   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00133   ulong value = 42;
00134   *dest = value;
```

```
00135   p_shmem_barrier_all();
00136   int mype = p_shmem_my_pe();
00137   p_shmem_ulong_atomic_inc(dest, mype);
00138   p_shmem_barrier_all();
00139   bool success = (*dest == value + 1);
00140   p_shmem_free(dest);
00141   return success;
00142 }
00143
00152 bool test_shmem_atomic_fetch_add() {
00153   static ulong *dest;
00154   static ulong fetch;
00155   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00156   ulong value = 42, add_val = 10;
00157   *dest = value;
00158   p_shmem_barrier_all();
00159   int mype = p_shmem_my_pe();
00160   fetch = p_shmem_ulong_atomic_fetch_add(dest, add_val, mype);
00161   p_shmem_barrier_all();
00162   bool success = (fetch == value && *dest == value + add_val);
00163   p_shmem_free(dest);
00164   return success;
00165 }
00166
00175 bool test_shmem_atomic_add() {
00176   static ulong *dest;
00177   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00178   ulong value = 42, add_val = 10;
00179   *dest = value;
00180   p_shmem_barrier_all();
00181   int mype = p_shmem_my_pe();
00182   p_shmem_ulong_atomic_add(dest, add_val, mype);
00183   p_shmem_barrier_all();
00184   bool success = (*dest == value + add_val);
00185   p_shmem_free(dest);
00186   return success;
00187 }
00188
00197 bool test_shmem_atomic_fetch_and() {
00198   static ulong *dest;
00199   static ulong fetch;
00200   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00201   ulong value = 42, and_val = 15;
00202   *dest = value;
00203   p_shmem_barrier_all();
00204   int mype = p_shmem_my_pe();
00205   fetch = p_shmem_ulong_atomic_fetch_and(dest, and_val, mype);
00206   p_shmem_barrier_all();
00207   bool success = (fetch == value && *dest == (value & and_val));
00208   p_shmem_free(dest);
00209   return success;
00210 }
00211
00220 bool test_shmem_atomic_and() {
00221   static ulong *dest;
00222   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00223   ulong value = 42, and_val = 15;
00224   *dest = value;
00225   p_shmem_barrier_all();
00226   int mype = p_shmem_my_pe();
00227   p_shmem_ulong_atomic_and(dest, and_val, mype);
00228   p_shmem_barrier_all();
00229   bool success = (*dest == (value & and_val));
00230   p_shmem_free(dest);
00231   return success;
00232 }
00233
00242 bool test_shmem_atomic_fetch_or() {
00243   static ulong *dest;
00244   static ulong fetch;
00245   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00246   ulong value = 42, or_val = 15;
00247   *dest = value;
00248   p_shmem_barrier_all();
00249   int mype = p_shmem_my_pe();
00250   fetch = p_shmem_ulong_atomic_fetch_or(dest, or_val, mype);
00251   p_shmem_barrier_all();
00252   bool success = (fetch == value && *dest == (value | or_val));
00253   p_shmem_free(dest);
00254   return success;
00255 }
00256
00265 bool test_shmem_atomic_or() {
00266   static ulong *dest;
00267   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00268   ulong value = 42, or_val = 15;
00269   *dest = value;
```

```
00270   p_shmem_barrier_all();
00271   int mype = p_shmem_my_pe();
00272   p_shmem_ulong_atomic_or(dest, or_val, mype);
00273   p_shmem_barrier_all();
00274   bool success = (*dest == (value | or_val));
00275   p_shmem_free(dest);
00276   return success;
00277 }
00278
00287 bool test_shmem_atomic_fetch_xor() {
00288   static ulong *dest;
00289   static ulong fetch;
00290   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00291   ulong value = 42, xor_val = 15;
00292   *dest = value;
00293   p_shmem_barrier_all();
00294   int mype = p_shmem_my_pe();
00295   fetch = p_shmem_ulong_atomic_fetch_xor(dest, xor_val, mype);
00296   p_shmem_barrier_all();
00297   bool success = (fetch == value && *dest == (value ^ xor_val));
00298   p_shmem_free(dest);
00299   return success;
00300 }
00301
00310 bool test_shmem_atomic_xor() {
00311   static ulong *dest;
00312   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00313   ulong value = 42, xor_val = 15;
00314   *dest = value;
00315   p_shmem_barrier_all();
00316   int mype = p_shmem_my_pe();
00317   p_shmem_ulong_atomic_xor(dest, xor_val, mype);
00318   p_shmem_barrier_all();
00319   bool success = (*dest == (value ^ xor_val));
00320   p_shmem_free(dest);
00321   return success;
00322 }
00323
00332 bool test_shmem_atomic_fetch_nbi() {
00333   static ulong *dest;
00334   static ulong fetch;
00335   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00336   ulong value = 42;
00337   *dest = value;
00338   p_shmem_barrier_all();
00339   int mype = p_shmem_my_pe();
00340   p_shmem_ulong_atomic_fetch_nbi(&fetch, dest, mype);
00341   p_shmem_quiet();
00342   p_shmem_barrier_all();
00343   bool success = (fetch == value);
00344   p_shmem_free(dest);
00345   return success;
00346 }
00347
00356 bool test_shmem_atomic_compare_swap_nbi() {
00357   static ulong *dest;
00358   static ulong fetch;
00359   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00360   fetch = 0;
00361   ulong old = 42, new_val = 43;
00362   *dest = old;
00363   p_shmem_barrier_all();
00364   int mype = p_shmem_my_pe();
00365   p_shmem_ulong_atomic_compare_swap_nbi(&fetch, dest, old, new_val, mype);
00366   p_shmem_quiet();
00367   p_shmem_barrier_all();
00368   bool success = (fetch == old && *dest == new_val);
00369   p_shmem_free(dest);
00370   return success;
00371 }
00372
00381 bool test_shmem_atomic_swap_nbi() {
00382   static ulong *dest;
00383   static ulong fetch;
00384   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00385   fetch = 0;
00386   ulong value = 42, new_val = 43;
00387   *dest = value;
00388   p_shmem_barrier_all();
00389   int mype = p_shmem_my_pe();
00390   p_shmem_ulong_atomic_swap_nbi(&fetch, dest, new_val, mype);
00391   p_shmem_quiet();
00392   p_shmem_barrier_all();
00393   bool success = (fetch == value && *dest == new_val);
00394   p_shmem_free(dest);
00395   return success;
00396 }
```

```
00397
00406 bool test_shmem_atomic_fetch_inc_nbi() {
00407   static ulong *dest;
00408   static ulong fetch;
00409   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00410   fetch = 0;
00411   ulong value = 42;
00412   *dest = value;
00413   p_shmem_barrier_all();
00414   int mype = p_shmem_my_pe();
00415   p_shmem_ulong_atomic_fetch_inc_nbi(&fetch, dest, mype);
00416   p_shmem_quiet();
00417   p_shmem_barrier_all();
00418   bool success = (fetch == value && *dest == value + 1);
00419   p_shmem_free(dest);
00420   return success;
00421 }
00422
00431 bool test_shmem_atomic_fetch_add_nbi() {
00432   static ulong *dest;
00433   static ulong fetch;
00434   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00435   fetch = 0;
00436   ulong value = 42, add_val = 10;
00437   *dest = value;
00438   p_shmem_barrier_all();
00439   int mype = p_shmem_my_pe();
00440   p_shmem_ulong_atomic_fetch_add_nbi(&fetch, dest, add_val, mype);
00441   p_shmem_quiet();
00442   p_shmem_barrier_all();
00443   bool success = (fetch == value && *dest == value + add_val);
00444   p_shmem_free(dest);
00445   return success;
00446 }
00447
00456 bool test_shmem_atomic_fetch_and_nbi() {
00457   static ulong *dest;
00458   static ulong fetch;
00459   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00460   fetch = 0;
00461   ulong value = 42, and_val = 15;
00462   *dest = value;
00463   p_shmem_barrier_all();
00464   int mype = p_shmem_my_pe();
00465   p_shmem_ulong_atomic_fetch_and_nbi(&fetch, dest, and_val, mype);
00466   p_shmem_quiet();
00467   p_shmem_barrier_all();
00468   bool success = (fetch == value && *dest == (value & and_val));
00469   p_shmem_free(dest);
00470   return success;
00471 }
00472
00481 bool test_shmem_atomic_fetch_or_nbi() {
00482   static ulong *dest;
00483   static ulong fetch;
00484   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00485   fetch = 0;
00486   ulong value = 42, or_val = 15;
00487   *dest = value;
00488   p_shmem_barrier_all();
00489   int mype = p_shmem_my_pe();
00490   p_shmem_ulong_atomic_fetch_or_nbi(&fetch, dest, or_val, mype);
00491   p_shmem_quiet();
00492   p_shmem_barrier_all();
00493   bool success = (fetch == value && *dest == (value | or_val));
00494   p_shmem_free(dest);
00495   return success;
00496 }
00497
00506 bool test_shmem_atomic_fetch_xor_nbi() {
00507   static ulong *dest;
00508   static ulong fetch;
00509   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00510   fetch = 0;
00511   ulong value = 42, xor_val = 15;
00512   *dest = value;
00513   p_shmem_barrier_all();
00514   int mype = p_shmem_my_pe();
00515   p_shmem_ulong_atomic_fetch_xor_nbi(&fetch, dest, xor_val, mype);
00516   p_shmem_quiet();
00517   p_shmem_barrier_all();
00518   bool success = (fetch == value && *dest == (value ^ xor_val));
00519   p_shmem_free(dest);
00520   return success;
00521 }
```

4.11 src/tests/atomics/atomics_tests.hpp File Reference

Contains function declarations for the OpenSHMEM atomic memory operations tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
```

Functions

- bool test_shmem_atomic_fetch (void)

  Tests the shmem_atomic_fetch() routine.
- bool test_shmem_atomic_set (void)

  Tests the shmem_atomic_set() routine.
- bool test_shmem_atomic_compare_swap (void)

  Tests the shmem_atomic_compare_swap() routine.
- bool test_shmem_atomic_swap (void)

  Tests the shmem_atomic_swap() routine.
- bool test_shmem_atomic_fetch_inc (void)

  Tests the shmem_atomic_fetch_inc() routine.
- bool test_shmem_atomic_inc (void)

  Tests the shmem_atomic_inc() routine.
- bool test_shmem_atomic_fetch_add (void)

  Tests the shmem_atomic_fetch_add() routine.
- bool test_shmem_atomic_add (void)

  Tests the shmem_atomic_add() routine.
- bool test_shmem_atomic_fetch_and (void)

  Tests the shmem_atomic_fetch_and() routine.
- bool test_shmem_atomic_and (void)

  Tests the shmem_atomic_and() routine.
- bool test_shmem_atomic_fetch_or (void)

  Tests the shmem_atomic_fetch_or() routine.
- bool test_shmem_atomic_or (void)

  Tests the shmem_atomic_or() routine.
- bool test_shmem_atomic_fetch_xor (void)

  Tests the shmem_atomic_fetch_xor() routine.
- bool test_shmem_atomic_xor (void)

  Tests the shmem_atomic_xor() routine.
- bool test_shmem_atomic_fetch_nbi (void)

  Tests the shmem_atomic_fetch_nbi() routine.
- bool test_shmem_atomic_compare_swap_nbi (void)

  Tests the shmem_atomic_compare_swap_nbi() routine.
- bool test_shmem_atomic_swap_nbi (void)

  Tests the shmem_atomic_swap_nbi() routine.
- bool test_shmem_atomic_fetch_inc_nbi (void)

  Tests the shmem_atomic_fetch_inc_nbi() routine.
- bool test_shmem_atomic_fetch_add_nbi (void)

  Tests the shmem_atomic_fetch_add_nbi() routine.
- bool test_shmem_atomic_fetch_and_nbi (void)

  Tests the shmem_atomic_fetch_and_nbi() routine.
- bool test_shmem_atomic_fetch_or_nbi (void)

  Tests the shmem_atomic_fetch_or_nbi() routine.
- bool test_shmem_atomic_fetch_xor_nbi (void)

  Tests the shmem_atomic_fetch_xor_nbi() routine.

4.11.1 Detailed Description

Contains function declarations for the OpenSHMEM atomic memory operations tests.

Definition in file atomics_tests.hpp.

4.11.2 Function Documentation

4.11.2.1 test_shmem_atomic_add()

```
bool test_shmem_atomic_add (
            void )
```

Tests the shmem_atomic_add() routine.

This test verifies that the shmem_atomic_add() routine correctly adds a value to the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 175 of file atomics_tests.cpp.

```
00175                                {
00176   static ulong *dest;
00177   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00178   ulong value = 42, add_val = 10;
00179   *dest = value;
00180   p_shmem_barrier_all();
00181   int mype = p_shmem_my_pe();
00182   p_shmem_ulong_atomic_add(dest, add_val, mype);
00183   p_shmem_barrier_all();
00184   bool success = (*dest == value + add_val);
00185   p_shmem_free(dest);
00186   return success;
00187 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_add.

4.11.2.2 test_shmem_atomic_and()

```
bool test_shmem_atomic_and (
            void )
```

Tests the shmem_atomic_and() routine.

This test verifies that the shmem_atomic_and() routine correctly performs a bitwise AND operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 220 of file atomics_tests.cpp.

```
00220                                {
00221   static ulong *dest;
00222   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00223   ulong value = 42, and_val = 15;
00224   *dest = value;
00225   p_shmem_barrier_all();
00226   int mype = p_shmem_my_pe();
00227   p_shmem_ulong_atomic_and(dest, and_val, mype);
00228   p_shmem_barrier_all();
00229   bool success = (*dest == (value & and_val));
00230   p_shmem_free(dest);
00231   return success;
00232 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_and.

4.11.2.3 test_shmem_atomic_compare_swap()

```
bool test_shmem_atomic_compare_swap (
            void )
```

Tests the shmem_atomic_compare_swap() routine.

This test verifies that the shmem_atomic_compare_swap() routine correctly swaps the value at a remote memory location if it matches the expected value.

Returns

True if the test is successful, false otherwise.

Definition at line 60 of file atomics_tests.cpp.

```
00060                                       {
00061   static ulong *dest;
00062   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00063   ulong old = 42, new_val = 43;
00064   *dest = old;
00065   p_shmem_barrier_all();
00066   int mype = p_shmem_my_pe();
00067   int npes = p_shmem_n_pes();
00068   p_shmem_barrier_all();
00069   ulong swapped = p_shmem_ulong_atomic_compare_swap(dest, old, new_val, (mype + 1) % npes);
00070   p_shmem_barrier_all();
00071   bool success = (swapped == old && *dest == new_val);
00072   p_shmem_barrier_all();
00073   p_shmem_free(dest);
00074   return success;
00075 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_ulong_atomic_compare_swap.

4.11.2.4 test_shmem_atomic_compare_swap_nbi()

```
bool test_shmem_atomic_compare_swap_nbi (
            void )
```

Tests the shmem_atomic_compare_swap_nbi() routine.

This test verifies that the shmem_atomic_compare_swap_nbi() routine correctly swaps the value at a remote memory location in a non-blocking manner if it matches the expected value.

Returns

True if the test is successful, false otherwise.

Definition at line 356 of file atomics_tests.cpp.

```
00356                                           {
00357   static ulong *dest;
00358   static ulong fetch;
00359   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00360   fetch = 0;
00361   ulong old = 42, new_val = 43;
00362   *dest = old;
00363   p_shmem_barrier_all();
00364   int mype = p_shmem_my_pe();
00365   p_shmem_ulong_atomic_compare_swap_nbi(&fetch, dest, old, new_val, mype);
00366   p_shmem_quiet();
00367   p_shmem_barrier_all();
00368   bool success = (fetch == old && *dest == new_val);
00369   p_shmem_free(dest);
00370   return success;
00371 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_compare_swap_nbi.

4.11.2.5 test_shmem_atomic_fetch()

```
bool test_shmem_atomic_fetch (
             void )
```

Tests the shmem_atomic_fetch() routine.

This test verifies that the shmem_atomic_fetch() routine correctly retrieves the value from a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file atomics_tests.cpp.

```
00016                                    {
00017   static ulong *dest;
00018   static ulong fetch;
00019   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00020   ulong value = 42;
00021   *dest = value;
00022   p_shmem_barrier_all();
00023   int mype = p_shmem_my_pe();
00024   fetch = p_shmem_ulong_atomic_fetch(dest, mype);
00025   p_shmem_barrier_all();
00026   bool success = (fetch == value);
00027   p_shmem_free(dest);
00028   return success;
00029 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch.

4.11.2.6 test_shmem_atomic_fetch_add()

```
bool test_shmem_atomic_fetch_add (
             void )
```

Tests the shmem_atomic_fetch_add() routine.

This test verifies that the shmem_atomic_fetch_add() routine correctly adds a value to the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 152 of file atomics_tests.cpp.

```
00152                                        {
00153   static ulong *dest;
00154   static ulong fetch;
00155   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00156   ulong value = 42, add_val = 10;
00157   *dest = value;
00158   p_shmem_barrier_all();
00159   int mype = p_shmem_my_pe();
00160   fetch = p_shmem_ulong_atomic_fetch_add(dest, add_val, mype);
00161   p_shmem_barrier_all();
00162   bool success = (fetch == value && *dest == value + add_val);
00163   p_shmem_free(dest);
00164   return success;
00165 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_add.

4.11.2.7 test_shmem_atomic_fetch_add_nbi()

```
bool test_shmem_atomic_fetch_add_nbi (
            void )
```

Tests the shmem_atomic_fetch_add_nbi() routine.

This test verifies that the shmem_atomic_fetch_add_nbi() routine correctly adds a value to the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 431 of file atomics_tests.cpp.

```
00431                                          {
00432   static ulong *dest;
00433   static ulong fetch;
00434   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00435   fetch = 0;
00436   ulong value = 42, add_val = 10;
00437   *dest = value;
00438   p_shmem_barrier_all();
00439   int mype = p_shmem_my_pe();
00440   p_shmem_ulong_atomic_fetch_add_nbi(&fetch, dest, add_val, mype);
00441   p_shmem_quiet();
00442   p_shmem_barrier_all();
00443   bool success = (fetch == value && *dest == value + add_val);
00444   p_shmem_free(dest);
00445   return success;
00446 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_add_nbi.

4.11.2.8 test_shmem_atomic_fetch_and()

```
bool test_shmem_atomic_fetch_and (
            void )
```

Tests the shmem_atomic_fetch_and() routine.

This test verifies that the shmem_atomic_fetch_and() routine correctly performs a bitwise AND operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 197 of file atomics_tests.cpp.

```
00197                                       {
00198   static ulong *dest;
00199   static ulong fetch;
00200   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00201   ulong value = 42, and_val = 15;
00202   *dest = value;
00203   p_shmem_barrier_all();
00204   int mype = p_shmem_my_pe();
00205   fetch = p_shmem_ulong_atomic_fetch_and(dest, and_val, mype);
00206   p_shmem_barrier_all();
00207   bool success = (fetch == value && *dest == (value & and_val));
00208   p_shmem_free(dest);
00209   return success;
00210 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_and.

4.11.2.9 test_shmem_atomic_fetch_and_nbi()

```
bool test_shmem_atomic_fetch_and_nbi (
            void )
```

Tests the shmem_atomic_fetch_and_nbi() routine.

This test verifies that the shmem_atomic_fetch_and_nbi() routine correctly performs a bitwise AND operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 456 of file atomics_tests.cpp.

```
00456                                           {
00457   static ulong *dest;
00458   static ulong fetch;
00459   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00460   fetch = 0;
00461   ulong value = 42, and_val = 15;
00462   *dest = value;
00463   p_shmem_barrier_all();
00464   int mype = p_shmem_my_pe();
00465   p_shmem_ulong_atomic_fetch_and_nbi(&fetch, dest, and_val, mype);
00466   p_shmem_quiet();
00467   p_shmem_barrier_all();
00468   bool success = (fetch == value && *dest == (value & and_val));
00469   p_shmem_free(dest);
00470   return success;
00471 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_and_nbi.

4.11.2.10 test_shmem_atomic_fetch_inc()

```
bool test_shmem_atomic_fetch_inc (
            void )
```

Tests the shmem_atomic_fetch_inc() routine.

This test verifies that the shmem_atomic_fetch_inc() routine correctly increments the value at a remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 107 of file atomics_tests.cpp.

```
00107                                           {
00108   static ulong *dest;
00109   static ulong fetch;
00110   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00111   ulong value = 42;
00112   *dest = value;
00113   p_shmem_barrier_all();
00114   int mype = p_shmem_my_pe();
00115   fetch = p_shmem_ulong_atomic_fetch_inc(dest, mype);
00116   p_shmem_barrier_all();
00117   bool success = (fetch == value && *dest == value + 1);
00118   p_shmem_free(dest);
00119   return success;
00120 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_inc.

4.11.2.11 test_shmem_atomic_fetch_inc_nbi()

```
bool test_shmem_atomic_fetch_inc_nbi (
            void )
```

Tests the shmem_atomic_fetch_inc_nbi() routine.

This test verifies that the shmem_atomic_fetch_inc_nbi() routine correctly increments the value at a remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 406 of file atomics_tests.cpp.

```
00406                                           {
00407   static ulong *dest;
00408   static ulong fetch;
00409   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00410   fetch = 0;
00411   ulong value = 42;
00412   *dest = value;
00413   p_shmem_barrier_all();
00414   int mype = p_shmem_my_pe();
00415   p_shmem_ulong_atomic_fetch_inc_nbi(&fetch, dest, mype);
00416   p_shmem_quiet();
00417   p_shmem_barrier_all();
00418   bool success = (fetch == value && *dest == value + 1);
00419   p_shmem_free(dest);
00420   return success;
00421 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_inc_nbi.

4.11.2.12 test_shmem_atomic_fetch_nbi()

```
bool test_shmem_atomic_fetch_nbi (
            void )
```

Tests the shmem_atomic_fetch_nbi() routine.

This test verifies that the shmem_atomic_fetch_nbi() routine correctly retrieves the value from a remote memory location in a non-blocking manner.

Returns

True if the test is successful, false otherwise.

Definition at line 332 of file atomics_tests.cpp.

```
00332                                           {
00333   static ulong *dest;
00334   static ulong fetch;
00335   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00336   ulong value = 42;
00337   *dest = value;
00338   p_shmem_barrier_all();
00339   int mype = p_shmem_my_pe();
00340   p_shmem_ulong_atomic_fetch_nbi(&fetch, dest, mype);
00341   p_shmem_quiet();
00342   p_shmem_barrier_all();
00343   bool success = (fetch == value);
00344   p_shmem_free(dest);
00345   return success;
00346 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_nbi.

4.11.2.13 test_shmem_atomic_fetch_or()

```
bool test_shmem_atomic_fetch_or (
            void )
```

Tests the shmem_atomic_fetch_or() routine.

This test verifies that the shmem_atomic_fetch_or() routine correctly performs a bitwise OR operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 242 of file atomics_tests.cpp.

```
00242                                    {
00243   static ulong *dest;
00244   static ulong fetch;
00245   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00246   ulong value = 42, or_val = 15;
00247   *dest = value;
00248   p_shmem_barrier_all();
00249   int mype = p_shmem_my_pe();
00250   fetch = p_shmem_ulong_atomic_fetch_or(dest, or_val, mype);
00251   p_shmem_barrier_all();
00252   bool success = (fetch == value && *dest == (value | or_val));
00253   p_shmem_free(dest);
00254   return success;
00255 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_or.

4.11.2.14 test_shmem_atomic_fetch_or_nbi()

```
bool test_shmem_atomic_fetch_or_nbi (
            void )
```

Tests the shmem_atomic_fetch_or_nbi() routine.

This test verifies that the shmem_atomic_fetch_or_nbi() routine correctly performs a bitwise OR operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 481 of file atomics_tests.cpp.

```
00481                                        {
00482   static ulong *dest;
00483   static ulong fetch;
00484   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00485   fetch = 0;
00486   ulong value = 42, or_val = 15;
00487   *dest = value;
00488   p_shmem_barrier_all();
00489   int mype = p_shmem_my_pe();
00490   p_shmem_ulong_atomic_fetch_or_nbi(&fetch, dest, or_val, mype);
00491   p_shmem_quiet();
00492   p_shmem_barrier_all();
00493   bool success = (fetch == value && *dest == (value | or_val));
00494   p_shmem_free(dest);
00495   return success;
00496 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_or_nbi.

4.11.2.15 test_shmem_atomic_fetch_xor()

```
bool test_shmem_atomic_fetch_xor (
            void )
```

Tests the shmem_atomic_fetch_xor() routine.

This test verifies that the shmem_atomic_fetch_xor() routine correctly performs a bitwise XOR operation with the remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 287 of file atomics_tests.cpp.

```
00287                                       {
00288   static ulong *dest;
00289   static ulong fetch;
00290   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00291   ulong value = 42, xor_val = 15;
00292   *dest = value;
00293   p_shmem_barrier_all();
00294   int mype = p_shmem_my_pe();
00295   fetch = p_shmem_ulong_atomic_fetch_xor(dest, xor_val, mype);
00296   p_shmem_barrier_all();
00297   bool success = (fetch == value && *dest == (value ^ xor_val));
00298   p_shmem_free(dest);
00299   return success;
00300 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_fetch_xor.

4.11.2.16 test_shmem_atomic_fetch_xor_nbi()

```
bool test_shmem_atomic_fetch_xor_nbi (
            void )
```

Tests the shmem_atomic_fetch_xor_nbi() routine.

This test verifies that the shmem_atomic_fetch_xor_nbi() routine correctly performs a bitwise XOR operation with the remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 506 of file atomics_tests.cpp.

```
00506                                           {
00507   static ulong *dest;
00508   static ulong fetch;
00509   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00510   fetch = 0;
00511   ulong value = 42, xor_val = 15;
00512   *dest = value;
00513   p_shmem_barrier_all();
00514   int mype = p_shmem_my_pe();
00515   p_shmem_ulong_atomic_fetch_xor_nbi(&fetch, dest, xor_val, mype);
00516   p_shmem_quiet();
00517   p_shmem_barrier_all();
00518   bool success = (fetch == value && *dest == (value ^ xor_val));
00519   p_shmem_free(dest);
00520   return success;
00521 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_fetch_xor_nbi.

4.11.2.17 test_shmem_atomic_inc()

```
bool test_shmem_atomic_inc (
            void )
```

Tests the shmem_atomic_inc() routine.

This test verifies that the shmem_atomic_inc() routine correctly increments the value at a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 130 of file atomics_tests.cpp.

```
00130                                {
00131   static ulong *dest;
00132   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00133   ulong value = 42;
00134   *dest = value;
00135   p_shmem_barrier_all();
00136   int mype = p_shmem_my_pe();
00137   p_shmem_ulong_atomic_inc(dest, mype);
00138   p_shmem_barrier_all();
00139   bool success = (*dest == value + 1);
00140   p_shmem_free(dest);
00141   return success;
00142 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_inc.

4.11.2.18 test_shmem_atomic_or()

```
bool test_shmem_atomic_or (
            void )
```

Tests the shmem_atomic_or() routine.

This test verifies that the shmem_atomic_or() routine correctly performs a bitwise OR operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 265 of file atomics_tests.cpp.

```
00265                                {
00266   static ulong *dest;
00267   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00268   ulong value = 42, or_val = 15;
00269   *dest = value;
00270   p_shmem_barrier_all();
00271   int mype = p_shmem_my_pe();
00272   p_shmem_ulong_atomic_or(dest, or_val, mype);
00273   p_shmem_barrier_all();
00274   bool success = (*dest == (value | or_val));
00275   p_shmem_free(dest);
00276   return success;
00277 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_or.

4.11.2.19 test_shmem_atomic_set()

```
bool test_shmem_atomic_set (
            void )
```

Tests the shmem_atomic_set() routine.

This test verifies that the shmem_atomic_set() routine correctly sets the value at a remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 39 of file atomics_tests.cpp.

```
00039                                {
00040   static ulong *dest;
00041   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00042   ulong value = 42;
00043   p_shmem_barrier_all();
00044   int mype = p_shmem_my_pe();
00045   p_shmem_ulong_atomic_set(dest, value, mype);
00046   p_shmem_barrier_all();
00047   bool success = (*dest == value);
00048   p_shmem_free(dest);
00049   return success;
00050 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_set.

4.11.2.20 test_shmem_atomic_swap()

```
bool test_shmem_atomic_swap (
            void )
```

Tests the shmem_atomic_swap() routine.

This test verifies that the shmem_atomic_swap() routine correctly swaps the value at a remote memory location and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 85 of file atomics_tests.cpp.

```
00085                                {
00086   static ulong *dest;
00087   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00088   ulong value = 42, new_val = 43;
00089   *dest = value;
00090   p_shmem_barrier_all();
00091   int mype = p_shmem_my_pe();
00092   ulong swapped = p_shmem_ulong_atomic_swap(dest, new_val, mype);
00093   p_shmem_barrier_all();
00094   bool success = (swapped == value && *dest == new_val);
00095   p_shmem_free(dest);
00096   return success;
00097 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_swap.

4.11.2.21 test_shmem_atomic_swap_nbi()

```
bool test_shmem_atomic_swap_nbi (
            void )
```

Tests the shmem_atomic_swap_nbi() routine.

This test verifies that the shmem_atomic_swap_nbi() routine correctly swaps the value at a remote memory location in a non-blocking manner and returns the old value.

Returns

True if the test is successful, false otherwise.

Definition at line 381 of file atomics_tests.cpp.

```
00381                                    {
00382   static ulong *dest;
00383   static ulong fetch;
00384   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00385   fetch = 0;
00386   ulong value = 42, new_val = 43;
00387   *dest = value;
00388   p_shmem_barrier_all();
00389   int mype = p_shmem_my_pe();
00390   p_shmem_ulong_atomic_swap_nbi(&fetch, dest, new_val, mype);
00391   p_shmem_quiet();
00392   p_shmem_barrier_all();
00393   bool success = (fetch == value && *dest == new_val);
00394   p_shmem_free(dest);
00395   return success;
00396 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and p_shmem_ulong_atomic_swap_nbi.

4.11.2.22 test_shmem_atomic_xor()

```
bool test_shmem_atomic_xor (
            void )
```

Tests the shmem_atomic_xor() routine.

This test verifies that the shmem_atomic_xor() routine correctly performs a bitwise XOR operation with the remote memory location.

Returns

True if the test is successful, false otherwise.

Definition at line 310 of file atomics_tests.cpp.

```
00310                                {
00311   static ulong *dest;
00312   dest = (ulong *)p_shmem_malloc(sizeof(ulong));
00313   ulong value = 42, xor_val = 15;
00314   *dest = value;
00315   p_shmem_barrier_all();
00316   int mype = p_shmem_my_pe();
00317   p_shmem_ulong_atomic_xor(dest, xor_val, mype);
00318   p_shmem_barrier_all();
00319   bool success = (*dest == (value ^ xor_val));
00320   p_shmem_free(dest);
00321   return success;
00322 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_ulong_atomic_xor.

4.12 atomics_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef ATOMICS_TESTS_HPP
00007 #define ATOMICS_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012
00021 bool test_shmem_atomic_fetch(void);
00022
00031 bool test_shmem_atomic_set(void);
00032
00041 bool test_shmem_atomic_compare_swap(void);
00042
00051 bool test_shmem_atomic_swap(void);
00052
00061 bool test_shmem_atomic_fetch_inc(void);
00062
00071 bool test_shmem_atomic_inc(void);
00072
00081 bool test_shmem_atomic_fetch_add(void);
00082
00091 bool test_shmem_atomic_add(void);
00092
00101 bool test_shmem_atomic_fetch_and(void);
00102
00111 bool test_shmem_atomic_and(void);
00112
00121 bool test_shmem_atomic_fetch_or(void);
00122
00131 bool test_shmem_atomic_or(void);
00132
00141 bool test_shmem_atomic_fetch_xor(void);
00142
00151 bool test_shmem_atomic_xor(void);
00152
00161 bool test_shmem_atomic_fetch_nbi(void);
00162
00171 bool test_shmem_atomic_compare_swap_nbi(void);
00172
00181 bool test_shmem_atomic_swap_nbi(void);
00182
00191 bool test_shmem_atomic_fetch_inc_nbi(void);
00192
00201 bool test_shmem_atomic_fetch_add_nbi(void);
00202
00211 bool test_shmem_atomic_fetch_and_nbi(void);
00212
00221 bool test_shmem_atomic_fetch_or_nbi(void);
00222
00231 bool test_shmem_atomic_fetch_xor_nbi(void);
00232
00233 #endif /* ATOMICS_TESTS_HPP */
```

4.13 src/tests/collectives/collectives_tests.cpp File Reference

Contains tests for various OpenSHMEM collective routines.

```
#include "collectives_tests.hpp"
```

Functions

- bool test_shmem_sync (void)

  Tests the shmem_sync() routine.
- bool test_shmem_sync_all (void)

  Tests the shmem_sync_all() routine.
- bool test_shmem_alltoall (void)

Tests the shmem_alltoall() routine.

- bool test_shmem_alltoalls (void)

  Tests the shmem_alltoalls() routine.
- bool test_shmem_broadcast (void)

  Tests the shmem_broadcast() routine.
- bool test_shmem_collect (void)

  Tests the shmem_collect() routine.
- bool test_shmem_fcollect (void)

  Tests the shmem_fcollect() routine.
- bool test_shmem_sum_reduce (void)

  Tests the shmem_sum_reduce() routine.
- bool test_shmem_prod_reduce (void)

  Tests the shmem_prod_reduce() routine.
- bool test_shmem_min_reduce (void)

  Tests the shmem_min_reduce() routine.
- bool test_shmem_max_reduce (void)

  Tests the shmem_max_reduce() routine.

4.13.1 Detailed Description

Contains tests for various OpenSHMEM collective routines.

Definition in file collectives_tests.cpp.

4.13.2 Function Documentation

4.13.2.1 test_shmem_alltoall()

```
bool test_shmem_alltoall (
            void )
```

Tests the shmem_alltoall() routine.

This test verifies that the shmem_alltoall() routine correctly performs an all-to-all data exchange among all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 45 of file collectives_tests.cpp.

```
00045                                {
00046   int npes = p_shmem_n_pes();
00047   int mype = p_shmem_my_pe();
00048
00049   long *src = (long *)p_shmem_malloc(npes * sizeof(long));
00050   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00051
00052   for (int i = 0; i < npes; ++i) {
00053     src[i] = mype + i;
00054   }
00055
00056   p_shmem_long_alltoall(SHMEM_TEAM_WORLD, dest, src, 1);
00057
00058   bool success = true;
00059   for (int i = 0; i < npes; ++i) {
00060     if (dest[i] != mype + i) {
00061       success = false;
00062       break;
00063     }
00064   }
00065
00066   p_shmem_free(src);
00067   p_shmem_free(dest);
00068
00069   return success;
00070 }
```

References p_shmem_free, p_shmem_long_alltoall, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.2 test_shmem_alltoalls()

```
bool test_shmem_alltoalls (
            void )
```

Tests the shmem_alltoalls() routine.

This test verifies that the shmem_alltoalls() routine correctly performs a strided all-to-all data exchange among all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 80 of file collectives_tests.cpp.

```
00080                                   {
00081   int npes = p_shmem_n_pes();
00082   int mype = p_shmem_my_pe();
00083
00084   long *src = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00085   long *dest = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00086
00087   for (int i = 0; i < npes; ++i) {
00088     src[i] = mype + i * npes;
00089   }
00090
00091   p_shmem_long_alltoalls(SHMEM_TEAM_WORLD, dest, src, 1, 1, npes);
00092
00093   bool success = true;
00094   for (int i = 0; i < npes; ++i) {
00095     if (dest[i] != i * npes + mype) {
00096       success = false;
00097       break;
00098     }
00099   }
00100
00101   p_shmem_free(src);
00102   p_shmem_free(dest);
00103
00104   return success;
00105 }
```

References p_shmem_free, p_shmem_long_alltoalls, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.3 test_shmem_broadcast()

```
bool test_shmem_broadcast (
            void )
```

Tests the shmem_broadcast() routine.

This test verifies that the shmem_broadcast() routine correctly broadcasts data from the root PE to all other PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 115 of file collectives_tests.cpp.

```
00115                                {
00116   int npes = p_shmem_n_pes();
00117   int mype = p_shmem_my_pe();
00118
00119   long *src = (long *)p_shmem_malloc(4 * sizeof(long));
00120   long *dest = (long *)p_shmem_malloc(4 * sizeof(long));
00121
00122   if (mype == 0) {
00123     for (int i = 0; i < 4; ++i) {
00124       src[i] = i + 1;
00125     }
00126   }
00127
00128   for (int i = 0; i < 4; ++i) {
00129     dest[i] = -1;
00130   }
00131
00132   p_shmem_barrier_all();
00133
00134   p_shmem_long_broadcast(SHMEM_TEAM_WORLD, dest, src, 4, 0);
00135
00136   p_shmem_barrier_all();
00137
00138   bool success = true;
00139   for (int i = 0; i < 4; ++i) {
00140     if (dest[i] != i + 1) {
00141       success = false;
00142       break;
00143     }
00144   }
00145
00146   p_shmem_free(src);
00147   p_shmem_free(dest);
00148
00149   return success;
00150 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_broadcast, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.4 test_shmem_collect()

```
bool test_shmem_collect (
             void )
```

Tests the shmem_collect() routine.

This test verifies that the shmem_collect() routine correctly collects data from all PEs to a single PE.

Returns

True if the test is successful, false otherwise.

Definition at line 160 of file collectives_tests.cpp.

```
00160                                {
00161   int npes = p_shmem_n_pes();
00162   int mype = p_shmem_my_pe();
00163
00164   long *src = (long *)p_shmem_malloc(sizeof(long));
00165   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00166
00167   src[0] = mype;
00168
00169   p_shmem_long_collect(SHMEM_TEAM_WORLD, dest, src, 1);
00170
00171   bool success = true;
00172   for (int i = 0; i < npes; ++i) {
00173     if (dest[i] != i) {
```

```
00174       success = false;
00175       break;
00176     }
00177   }
00178
00179   p_shmem_free(src);
00180   p_shmem_free(dest);
00181
00182   return success;
00183 }
```

References p_shmem_free, p_shmem_long_collect, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.5 test_shmem_fcollect()

```
bool test_shmem_fcollect (
            void )
```

Tests the shmem_fcollect() routine.

This test verifies that the shmem_fcollect() routine correctly collects data from all PEs to a single PE in a more efficient manner than shmem_collect().

Returns

True if the test is successful, false otherwise.

Definition at line 193 of file collectives_tests.cpp.

```
00193                                 {
00194   int npes = p_shmem_n_pes();
00195   int mype = p_shmem_my_pe();
00196
00197   long *src = (long *)p_shmem_malloc(sizeof(long));
00198   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00199
00200   src[0] = mype;
00201
00202   p_shmem_long_fcollect(SHMEM_TEAM_WORLD, dest, src, 1);
00203
00204   bool success = true;
00205   for (int i = 0; i < npes; ++i) {
00206     if (dest[i] != i) {
00207       success = false;
00208       break;
00209     }
00210   }
00211
00212   p_shmem_free(src);
00213   p_shmem_free(dest);
00214
00215   return success;
00216 }
```

References p_shmem_free, p_shmem_long_fcollect, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.6 test_shmem_max_reduce()

```
bool test_shmem_max_reduce (
            void )
```

Tests the shmem_max_reduce() routine.

This test verifies that the shmem_max_reduce() routine correctly computes the maximum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 313 of file collectives_tests.cpp.

```
00313                                   {
00314   int npes = p_shmem_n_pes();
00315   int mype = p_shmem_my_pe();
00316
00317   long *src = (long *)p_shmem_malloc(sizeof(long));
00318   long *dest = (long *)p_shmem_malloc(sizeof(long));
00319
00320   *src = mype;
00321
00322   p_shmem_long_max_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00323
00324   bool success = (*dest == npes - 1);
00325
00326   p_shmem_free(src);
00327   p_shmem_free(dest);
00328
00329   return success;
00330 }
```

References p_shmem_free, p_shmem_long_max_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.7 test_shmem_min_reduce()

```
bool test_shmem_min_reduce (
            void )
```

Tests the shmem_min_reduce() routine.

This test verifies that the shmem_min_reduce() routine correctly computes the minimum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 286 of file collectives_tests.cpp.

```
00286                                   {
00287   int npes = p_shmem_n_pes();
00288   int mype = p_shmem_my_pe();
00289
00290   long *src = (long *)p_shmem_malloc(sizeof(long));
00291   long *dest = (long *)p_shmem_malloc(sizeof(long));
00292
00293   *src = mype;
00294
00295   p_shmem_long_min_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00296
00297   bool success = (*dest == 0);
00298
00299   p_shmem_free(src);
00300   p_shmem_free(dest);
00301
00302   return success;
00303 }
```

References p_shmem_free, p_shmem_long_min_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.8 test_shmem_prod_reduce()

```
bool test_shmem_prod_reduce (
            void )
```

Tests the shmem_prod_reduce() routine.

This test verifies that the shmem_prod_reduce() routine correctly computes the product of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 254 of file collectives_tests.cpp.

```
00254                                   {
00255   int npes = p_shmem_n_pes();
00256   int mype = p_shmem_my_pe();
00257
00258   long *src = (long *)p_shmem_malloc(sizeof(long));
00259   long *dest = (long *)p_shmem_malloc(sizeof(long));
00260
00261   *src = mype + 1;
00262
00263   p_shmem_long_prod_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00264
00265   long expected_prod = 1;
00266   for (int i = 1; i <= npes; i++) {
00267     expected_prod *= i;
00268   }
00269
00270   bool success = (*dest == expected_prod);
00271
00272   p_shmem_free(src);
00273   p_shmem_free(dest);
00274
00275   return success;
00276 }
```

References p_shmem_free, p_shmem_long_prod_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.9 test_shmem_sum_reduce()

```
bool test_shmem_sum_reduce (
            void )
```

Tests the shmem_sum_reduce() routine.

This test verifies that the shmem_sum_reduce() routine correctly computes the sum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 226 of file collectives_tests.cpp.

```
00226                                   {
00227   int npes = p_shmem_n_pes();
00228   int mype = p_shmem_my_pe();
00229
00230   long *src = (long *)p_shmem_malloc(sizeof(long));
00231   long *dest = (long *)p_shmem_malloc(sizeof(long));
00232
00233   *src = mype;
00234
00235   p_shmem_long_sum_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00236
00237   long expected_sum = npes * (npes - 1) / 2;
00238   bool success = (*dest == expected_sum);
00239
00240   p_shmem_free(src);
00241   p_shmem_free(dest);
00242
00243   return success;
00244 }
```

References p_shmem_free, p_shmem_long_sum_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.13.2.10 test_shmem_sync()

```
bool test_shmem_sync (
             void )
```

Tests the shmem_sync() routine.

This test verifies that the shmem_sync() routine correctly synchronizes all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file collectives_tests.cpp.

```
00015                                {
00016   static long pSync[SHMEM_SYNC_SIZE];
00017   for (int i = 0; i < SHMEM_SYNC_SIZE; i++) {
00018     pSync[i] = SHMEM_SYNC_VALUE;
00019   }
00020   p_shmem_barrier_all();
00021   p_shmem_sync(0, 0, p_shmem_n_pes(), pSync);
00022   return true;
00023 }
```

References p_shmem_barrier_all, p_shmem_n_pes, and p_shmem_sync.

4.13.2.11 test_shmem_sync_all()

```
bool test_shmem_sync_all (
             void )
```

Tests the shmem_sync_all() routine.

This test verifies that the shmem_sync_all() routine correctly synchronizes all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 32 of file collectives_tests.cpp.

```
00032                                    {
00033   p_shmem_sync_all();
00034   return true;
00035 }
```

References p_shmem_sync_all.

4.14 collectives_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "collectives_tests.hpp"
00007
00015 bool test_shmem_sync(void) {
00016   static long pSync[SHMEM_SYNC_SIZE];
00017   for (int i = 0; i < SHMEM_SYNC_SIZE; i++) {
00018     pSync[i] = SHMEM_SYNC_VALUE;
00019   }
00020   p_shmem_barrier_all();
00021   p_shmem_sync(0, 0, p_shmem_n_pes(), pSync);
00022   return true;
00023 }
00024
00032 bool test_shmem_sync_all(void) {
00033   p_shmem_sync_all();
00034   return true;
00035 }
00036
00045 bool test_shmem_alltoall(void) {
00046   int npes = p_shmem_n_pes();
00047   int mype = p_shmem_my_pe();
00048
00049   long *src = (long *)p_shmem_malloc(npes * sizeof(long));
00050   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00051
00052   for (int i = 0; i < npes; ++i) {
00053     src[i] = mype + i;
00054   }
00055
00056   p_shmem_long_alltoall(SHMEM_TEAM_WORLD, dest, src, 1);
00057
00058   bool success = true;
00059   for (int i = 0; i < npes; ++i) {
00060     if (dest[i] != mype + i) {
00061       success = false;
00062       break;
00063     }
00064   }
00065
00066   p_shmem_free(src);
00067   p_shmem_free(dest);
00068
00069   return success;
00070 }
00071
00080 bool test_shmem_alltoalls(void) {
00081   int npes = p_shmem_n_pes();
00082   int mype = p_shmem_my_pe();
00083
00084   long *src = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00085   long *dest = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00086
00087   for (int i = 0; i < npes; ++i) {
00088     src[i] = mype + i * npes;
00089   }
00090
00091   p_shmem_long_alltoalls(SHMEM_TEAM_WORLD, dest, src, 1, 1, npes);
00092
00093   bool success = true;
00094   for (int i = 0; i < npes; ++i) {
00095     if (dest[i] != i * npes + mype) {
00096       success = false;
00097       break;
00098     }
00099   }
00100
00101   p_shmem_free(src);
00102   p_shmem_free(dest);
00103
00104   return success;
00105 }
00106
00115 bool test_shmem_broadcast(void) {
00116   int npes = p_shmem_n_pes();
00117   int mype = p_shmem_my_pe();
00118
00119   long *src = (long *)p_shmem_malloc(4 * sizeof(long));
00120   long *dest = (long *)p_shmem_malloc(4 * sizeof(long));
00121
00122   if (mype == 0) {
00123     for (int i = 0; i < 4; ++i) {
00124       src[i] = i + 1;
```

```
00125     }
00126   }
00127
00128   for (int i = 0; i < 4; ++i) {
00129     dest[i] = -1;
00130   }
00131
00132   p_shmem_barrier_all();
00133
00134   p_shmem_long_broadcast(SHMEM_TEAM_WORLD, dest, src, 4, 0);
00135
00136   p_shmem_barrier_all();
00137
00138   bool success = true;
00139   for (int i = 0; i < 4; ++i) {
00140     if (dest[i] != i + 1) {
00141       success = false;
00142       break;
00143     }
00144   }
00145
00146   p_shmem_free(src);
00147   p_shmem_free(dest);
00148
00149   return success;
00150 }
00151
00160 bool test_shmem_collect(void) {
00161   int npes = p_shmem_n_pes();
00162   int mype = p_shmem_my_pe();
00163
00164   long *src = (long *)p_shmem_malloc(sizeof(long));
00165   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00166
00167   src[0] = mype;
00168
00169   p_shmem_long_collect(SHMEM_TEAM_WORLD, dest, src, 1);
00170
00171   bool success = true;
00172   for (int i = 0; i < npes; ++i) {
00173     if (dest[i] != i) {
00174       success = false;
00175       break;
00176     }
00177   }
00178
00179   p_shmem_free(src);
00180   p_shmem_free(dest);
00181
00182   return success;
00183 }
00184
00193 bool test_shmem_fcollect(void) {
00194   int npes = p_shmem_n_pes();
00195   int mype = p_shmem_my_pe();
00196
00197   long *src = (long *)p_shmem_malloc(sizeof(long));
00198   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00199
00200   src[0] = mype;
00201
00202   p_shmem_long_fcollect(SHMEM_TEAM_WORLD, dest, src, 1);
00203
00204   bool success = true;
00205   for (int i = 0; i < npes; ++i) {
00206     if (dest[i] != i) {
00207       success = false;
00208       break;
00209     }
00210   }
00211
00212   p_shmem_free(src);
00213   p_shmem_free(dest);
00214
00215   return success;
00216 }
00217
00226 bool test_shmem_sum_reduce(void) {
00227   int npes = p_shmem_n_pes();
00228   int mype = p_shmem_my_pe();
00229
00230   long *src = (long *)p_shmem_malloc(sizeof(long));
00231   long *dest = (long *)p_shmem_malloc(sizeof(long));
00232
00233   *src = mype;
00234
00235   p_shmem_long_sum_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
```

```
00236
00237   long expected_sum = npes * (npes - 1) / 2;
00238   bool success = (*dest == expected_sum);
00239
00240   p_shmem_free(src);
00241   p_shmem_free(dest);
00242
00243   return success;
00244 }
00245
00254 bool test_shmem_prod_reduce(void) {
00255   int npes = p_shmem_n_pes();
00256   int mype = p_shmem_my_pe();
00257
00258   long *src = (long *)p_shmem_malloc(sizeof(long));
00259   long *dest = (long *)p_shmem_malloc(sizeof(long));
00260
00261   *src = mype + 1;
00262
00263   p_shmem_long_prod_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00264
00265   long expected_prod = 1;
00266   for (int i = 1; i <= npes; i++) {
00267     expected_prod *= i;
00268   }
00269
00270   bool success = (*dest == expected_prod);
00271
00272   p_shmem_free(src);
00273   p_shmem_free(dest);
00274
00275   return success;
00276 }
00277
00286 bool test_shmem_min_reduce(void) {
00287   int npes = p_shmem_n_pes();
00288   int mype = p_shmem_my_pe();
00289
00290   long *src = (long *)p_shmem_malloc(sizeof(long));
00291   long *dest = (long *)p_shmem_malloc(sizeof(long));
00292
00293   *src = mype;
00294
00295   p_shmem_long_min_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00296
00297   bool success = (*dest == 0);
00298
00299   p_shmem_free(src);
00300   p_shmem_free(dest);
00301
00302   return success;
00303 }
00304
00313 bool test_shmem_max_reduce(void) {
00314   int npes = p_shmem_n_pes();
00315   int mype = p_shmem_my_pe();
00316
00317   long *src = (long *)p_shmem_malloc(sizeof(long));
00318   long *dest = (long *)p_shmem_malloc(sizeof(long));
00319
00320   *src = mype;
00321
00322   p_shmem_long_max_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00323
00324   bool success = (*dest == npes - 1);
00325
00326   p_shmem_free(src);
00327   p_shmem_free(dest);
00328
00329   return success;
00330 }
```

4.15 src/tests/collectives/collectives_tests.hpp File Reference

Contains function declarations for the OpenSHMEM collectives tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <cstring>
```

```
#include <stdbool.h>
#include <stdio.h>
```

Functions

- bool test_shmem_sync (void)

  Tests the shmem_sync() routine.
- bool test_shmem_sync_all (void)

  Tests the shmem_sync_all() routine.
- bool test_shmem_alltoall (void)

  Tests the shmem_alltoall() routine.
- bool test_shmem_alltoalls (void)

  Tests the shmem_alltoalls() routine.
- bool test_shmem_broadcast (void)

  Tests the shmem_broadcast() routine.
- bool test_shmem_collect (void)

  Tests the shmem_collect() routine.
- bool test_shmem_fcollect (void)

  Tests the shmem_fcollect() routine.
- bool test_shmem_and_reduce (void)

  Tests the shmem_and_reduce() routine.
- bool test_shmem_max_reduce (void)

  Tests the shmem_max_reduce() routine.
- bool test_shmem_min_reduce (void)

  Tests the shmem_min_reduce() routine.
- bool test_shmem_sum_reduce (void)

  Tests the shmem_sum_reduce() routine.
- bool test_shmem_prod_reduce (void)

  Tests the shmem_prod_reduce() routine.

4.15.1 Detailed Description

Contains function declarations for the OpenSHMEM collectives tests.

Definition in file collectives_tests.hpp.

4.15.2 Function Documentation

4.15.2.1 test_shmem_alltoall()

```
bool test_shmem_alltoall (
            void )
```

Tests the shmem_alltoall() routine.

This test verifies that the shmem_alltoall() routine correctly performs an all-to-all data exchange among all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 45 of file collectives_tests.cpp.

```
00045                                {
00046   int npes = p_shmem_n_pes();
00047   int mype = p_shmem_my_pe();
00048
00049   long *src = (long *)p_shmem_malloc(npes * sizeof(long));
00050   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00051
00052   for (int i = 0; i < npes; ++i) {
00053     src[i] = mype + i;
00054   }
00055
00056   p_shmem_long_alltoall(SHMEM_TEAM_WORLD, dest, src, 1);
00057
00058   bool success = true;
00059   for (int i = 0; i < npes; ++i) {
00060     if (dest[i] != mype + i) {
00061       success = false;
00062       break;
00063     }
00064   }
00065
00066   p_shmem_free(src);
00067   p_shmem_free(dest);
00068
00069   return success;
00070 }
```

References p_shmem_free, p_shmem_long_alltoall, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.2 test_shmem_alltoalls()

```
bool test_shmem_alltoalls (
            void )
```

Tests the shmem_alltoalls() routine.

This test verifies that the shmem_alltoalls() routine correctly performs a strided all-to-all data exchange among all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 80 of file collectives_tests.cpp.

```
00080                                {
00081   int npes = p_shmem_n_pes();
00082   int mype = p_shmem_my_pe();
00083
00084   long *src = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00085   long *dest = (long *)p_shmem_malloc(npes * npes * sizeof(long));
00086
00087   for (int i = 0; i < npes; ++i) {
00088     src[i] = mype + i * npes;
00089   }
00090
00091   p_shmem_long_alltoalls(SHMEM_TEAM_WORLD, dest, src, 1, 1, npes);
00092
00093   bool success = true;
00094   for (int i = 0; i < npes; ++i) {
00095     if (dest[i] != i * npes + mype) {
00096       success = false;
00097       break;
00098     }
00099   }
00100
00101   p_shmem_free(src);
00102   p_shmem_free(dest);
00103
00104   return success;
00105 }
```

References p_shmem_free, p_shmem_long_alltoalls, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.3 test_shmem_and_reduce()

```
bool test_shmem_and_reduce (
            void )
```

Tests the shmem_and_reduce() routine.

This test verifies that the shmem_and_reduce() routine correctly performs a bitwise AND reduction operation across all PEs.

Returns

True if the test is successful, false otherwise.

4.15.2.4 test_shmem_broadcast()

```
bool test_shmem_broadcast (
            void )
```

Tests the shmem_broadcast() routine.

This test verifies that the shmem_broadcast() routine correctly broadcasts data from the root PE to all other PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 115 of file collectives_tests.cpp.

```
00115                                 {
00116   int npes = p_shmem_n_pes();
00117   int mype = p_shmem_my_pe();
00118
00119   long *src = (long *)p_shmem_malloc(4 * sizeof(long));
00120   long *dest = (long *)p_shmem_malloc(4 * sizeof(long));
00121
00122   if (mype == 0) {
00123     for (int i = 0; i < 4; ++i) {
00124       src[i] = i + 1;
00125     }
00126   }
00127
00128   for (int i = 0; i < 4; ++i) {
00129     dest[i] = -1;
00130   }
00131
00132   p_shmem_barrier_all();
00133
00134   p_shmem_long_broadcast(SHMEM_TEAM_WORLD, dest, src, 4, 0);
00135
00136   p_shmem_barrier_all();
00137
00138   bool success = true;
00139   for (int i = 0; i < 4; ++i) {
00140     if (dest[i] != i + 1) {
00141       success = false;
00142       break;
00143     }
00144   }
00145
00146   p_shmem_free(src);
00147   p_shmem_free(dest);
00148
00149   return success;
00150 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_broadcast, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.5 test_shmem_collect()

```
bool test_shmem_collect (
            void )
```

Tests the shmem_collect() routine.

This test verifies that the shmem_collect() routine correctly collects data from all PEs to a single PE.

Returns

True if the test is successful, false otherwise.

Definition at line 160 of file collectives_tests.cpp.

```
00160                                   {
00161   int npes = p_shmem_n_pes();
00162   int mype = p_shmem_my_pe();
00163
00164   long *src = (long *)p_shmem_malloc(sizeof(long));
00165   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00166
00167   src[0] = mype;
00168
00169   p_shmem_long_collect(SHMEM_TEAM_WORLD, dest, src, 1);
00170
00171   bool success = true;
00172   for (int i = 0; i < npes; ++i) {
00173     if (dest[i] != i) {
00174       success = false;
00175       break;
00176     }
00177   }
00178
00179   p_shmem_free(src);
00180   p_shmem_free(dest);
00181
00182   return success;
00183 }
```

References p_shmem_free, p_shmem_long_collect, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.6 test_shmem_fcollect()

```
bool test_shmem_fcollect (
            void )
```

Tests the shmem_fcollect() routine.

This test verifies that the shmem_fcollect() routine correctly collects data from all PEs to a single PE in a more efficient manner than shmem_collect().

Returns

True if the test is successful, false otherwise.

Definition at line 193 of file collectives_tests.cpp.

```
00193                                   {
00194   int npes = p_shmem_n_pes();
00195   int mype = p_shmem_my_pe();
00196
00197   long *src = (long *)p_shmem_malloc(sizeof(long));
00198   long *dest = (long *)p_shmem_malloc(npes * sizeof(long));
00199
00200   src[0] = mype;
00201
00202   p_shmem_long_fcollect(SHMEM_TEAM_WORLD, dest, src, 1);
```

```
00203
00204   bool success = true;
00205   for (int i = 0; i < npes; ++i) {
00206     if (dest[i] != i) {
00207       success = false;
00208       break;
00209     }
00210   }
00211
00212   p_shmem_free(src);
00213   p_shmem_free(dest);
00214
00215   return success;
00216 }
```

References p_shmem_free, p_shmem_long_fcollect, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.7 test_shmem_max_reduce()

```
bool test_shmem_max_reduce (
             void )
```

Tests the shmem_max_reduce() routine.

This test verifies that the shmem_max_reduce() routine correctly computes the maximum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 313 of file collectives_tests.cpp.

```
00313                                   {
00314   int npes = p_shmem_n_pes();
00315   int mype = p_shmem_my_pe();
00316
00317   long *src = (long *)p_shmem_malloc(sizeof(long));
00318   long *dest = (long *)p_shmem_malloc(sizeof(long));
00319
00320   *src = mype;
00321
00322   p_shmem_long_max_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00323
00324   bool success = (*dest == npes - 1);
00325
00326   p_shmem_free(src);
00327   p_shmem_free(dest);
00328
00329   return success;
00330 }
```

References p_shmem_free, p_shmem_long_max_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.8 test_shmem_min_reduce()

```
bool test_shmem_min_reduce (
             void )
```

Tests the shmem_min_reduce() routine.

This test verifies that the shmem_min_reduce() routine correctly computes the minimum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 286 of file collectives_tests.cpp.

```
00286                                    {
00287   int npes = p_shmem_n_pes();
00288   int mype = p_shmem_my_pe();
00289
00290   long *src = (long *)p_shmem_malloc(sizeof(long));
00291   long *dest = (long *)p_shmem_malloc(sizeof(long));
00292
00293   *src = mype;
00294
00295   p_shmem_long_min_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00296
00297   bool success = (*dest == 0);
00298
00299   p_shmem_free(src);
00300   p_shmem_free(dest);
00301
00302   return success;
00303 }
```

References p_shmem_free, p_shmem_long_min_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.9 test_shmem_prod_reduce()

```
bool test_shmem_prod_reduce (
             void )
```

Tests the shmem_prod_reduce() routine.

This test verifies that the shmem_prod_reduce() routine correctly computes the product of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 254 of file collectives_tests.cpp.

```
00254                                    {
00255   int npes = p_shmem_n_pes();
00256   int mype = p_shmem_my_pe();
00257
00258   long *src = (long *)p_shmem_malloc(sizeof(long));
00259   long *dest = (long *)p_shmem_malloc(sizeof(long));
00260
00261   *src = mype + 1;
00262
00263   p_shmem_long_prod_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00264
00265   long expected_prod = 1;
00266   for (int i = 1; i <= npes; i++) {
00267     expected_prod *= i;
00268   }
00269
00270   bool success = (*dest == expected_prod);
00271
00272   p_shmem_free(src);
00273   p_shmem_free(dest);
00274
00275   return success;
00276 }
```

References p_shmem_free, p_shmem_long_prod_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.10 test_shmem_sum_reduce()

```
bool test_shmem_sum_reduce (
            void )
```

Tests the shmem_sum_reduce() routine.

This test verifies that the shmem_sum_reduce() routine correctly computes the sum of data from all PEs and stores it on the root PE.

Returns

True if the test is successful, false otherwise.

Definition at line 226 of file collectives_tests.cpp.

```
00226                                   {
00227   int npes = p_shmem_n_pes();
00228   int mype = p_shmem_my_pe();
00229
00230   long *src = (long *)p_shmem_malloc(sizeof(long));
00231   long *dest = (long *)p_shmem_malloc(sizeof(long));
00232
00233   *src = mype;
00234
00235   p_shmem_long_sum_reduce(SHMEM_TEAM_WORLD, dest, src, 1);
00236
00237   long expected_sum = npes * (npes - 1) / 2;
00238   bool success = (*dest == expected_sum);
00239
00240   p_shmem_free(src);
00241   p_shmem_free(dest);
00242
00243   return success;
00244 }
```

References p_shmem_free, p_shmem_long_sum_reduce, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.15.2.11 test_shmem_sync()

```
bool test_shmem_sync (
            void )
```

Tests the shmem_sync() routine.

This test verifies that the shmem_sync() routine correctly synchronizes all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file collectives_tests.cpp.

```
00015                               {
00016   static long pSync[SHMEM_SYNC_SIZE];
00017   for (int i = 0; i < SHMEM_SYNC_SIZE; i++) {
00018     pSync[i] = SHMEM_SYNC_VALUE;
00019   }
00020   p_shmem_barrier_all();
00021   p_shmem_sync(0, 0, p_shmem_n_pes(), pSync);
00022   return true;
00023 }
```

References p_shmem_barrier_all, p_shmem_n_pes, and p_shmem_sync.

4.15.2.12 test_shmem_sync_all()

```
bool test_shmem_sync_all (
             void )
```

Tests the shmem_sync_all() routine.

This test verifies that the shmem_sync_all() routine correctly synchronizes all PEs.

Returns

True if the test is successful, false otherwise.

Definition at line 32 of file collectives_tests.cpp.

```
00032                                {
00033   p_shmem_sync_all();
00034   return true;
00035 }
```

References p_shmem_sync_all.

4.16 collectives_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef COLLECTIVES_TESTS_HPP
00007 #define COLLECTIVES_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <cstring>
00012 #include <stdbool.h>
00013 #include <stdio.h>
00014
00022 bool test_shmem_sync(void);
00023
00031 bool test_shmem_sync_all(void);
00032
00041 bool test_shmem_alltoall(void);
00042
00051 bool test_shmem_alltoalls(void);
00052
00061 bool test_shmem_broadcast(void);
00062
00071 bool test_shmem_collect(void);
00072
00081 bool test_shmem_fcollect(void);
00082
00091 bool test_shmem_and_reduce(void);
00092
00101 bool test_shmem_max_reduce(void);
00102
00111 bool test_shmem_min_reduce(void);
00112
00121 bool test_shmem_sum_reduce(void);
00122
00131 bool test_shmem_prod_reduce(void);
00132
00133 #endif /* COLLECTIVES_TESTS_HPP */
```

4.17 src/tests/comms/comms_tests.cpp File Reference

Contains OpenSHMEM communication/context tests.

```
#include "comms_tests.hpp"
```

Functions

- bool test_shmem_ctx_create (void)

  Tests the shmem_ctx_create() function.
- bool test_shmem_team_create_ctx (void)

  Tests the shmem_team_create_ctx() function.
- bool test_shmem_ctx_destroy (void)

  Tests the shmem_ctx_destroy() function.
- bool test_shmem_ctx_get_team (void)

  Tests the shmem_ctx_get_team() function.

4.17.1 Detailed Description

Contains OpenSHMEM communication/context tests.

Definition in file comms_tests.cpp.

4.17.2 Function Documentation

4.17.2.1 test_shmem_ctx_create()

```
bool test_shmem_ctx_create (
            void )
```

Tests the shmem_ctx_create() function.

This test verifies that the shmem_ctx_create() function correctly creates a communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file comms_tests.cpp.

```
00015                                   {
00016   shmem_ctx_t ctx;
00017   int ret = p_shmem_ctx_create(0, &ctx);
00018   if (ret != 0) {
00019     return false;
00020   }
00021   p_shmem_ctx_destroy(ctx);
00022   return true;
00023 }
```

References p_shmem_ctx_create, and p_shmem_ctx_destroy.

4.17.2.2 test_shmem_ctx_destroy()

```
bool test_shmem_ctx_destroy (
            void )
```

Tests the shmem_ctx_destroy() function.

This test verifies that the shmem_ctx_destroy() function correctly destroys a communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 53 of file comms_tests.cpp.

```
00053                                       {
00054   shmem_ctx_t ctx;
00055   p_shmem_ctx_create(0, &ctx);
00056   p_shmem_ctx_destroy(ctx);
00057   return true;
00058 }
```

References p_shmem_ctx_create, and p_shmem_ctx_destroy.

4.17.2.3 test_shmem_ctx_get_team()

```
bool test_shmem_ctx_get_team (
            void )
```

Tests the shmem_ctx_get_team() function.

This test verifies that the shmem_ctx_get_team() function correctly retrieves the team associated with a given communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 68 of file comms_tests.cpp.

```
00068                                       {
00069   shmem_ctx_t ctx;
00070   shmem_team_t team;
00071   p_shmem_ctx_create(0, &ctx);
00072   int ret = p_shmem_ctx_get_team(ctx, &team);
00073   p_shmem_ctx_destroy(ctx);
00074   return (ret == 0 && team == SHMEM_TEAM_WORLD);
00075 }
```

References p_shmem_ctx_create, p_shmem_ctx_destroy, and p_shmem_ctx_get_team.

4.17.2.4 test_shmem_team_create_ctx()

```
bool test_shmem_team_create_ctx (
            void )
```

Tests the shmem_team_create_ctx() function.

This test verifies that the shmem_team_create_ctx() function correctly creates a context for a specified team.

Returns

True if the test is successful, false otherwise.

Definition at line 33 of file comms_tests.cpp.

```
00033                                       {
00034   shmem_team_t team;
00035   shmem_ctx_t ctx;
00036   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00037   int ret = shmem_team_create_ctx(team, 0, &ctx);
00038   if (ret != 0) {
00039     return false;
00040   }
00041   p_shmem_ctx_destroy(ctx);
00042   p_shmem_team_destroy(team);
00043   return true;
00044 }
```

References p_shmem_ctx_destroy, p_shmem_n_pes, p_shmem_team_destroy, and p_shmem_team_split_strided.

4.18 comms_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "comms_tests.hpp"
00007
00015 bool test_shmem_ctx_create(void) {
00016   shmem_ctx_t ctx;
00017   int ret = p_shmem_ctx_create(0, &ctx);
00018   if (ret != 0) {
00019     return false;
00020   }
00021   p_shmem_ctx_destroy(ctx);
00022   return true;
00023 }
00024
00033 bool test_shmem_team_create_ctx(void) {
00034   shmem_team_t team;
00035   shmem_ctx_t ctx;
00036   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00037   int ret = shmem_team_create_ctx(team, 0, &ctx);
00038   if (ret != 0) {
00039     return false;
00040   }
00041   p_shmem_ctx_destroy(ctx);
00042   p_shmem_team_destroy(team);
00043   return true;
00044 }
00045
00053 bool test_shmem_ctx_destroy(void) {
00054   shmem_ctx_t ctx;
00055   p_shmem_ctx_create(0, &ctx);
00056   p_shmem_ctx_destroy(ctx);
00057   return true;
00058 }
00059
00068 bool test_shmem_ctx_get_team(void) {
00069   shmem_ctx_t ctx;
00070   shmem_team_t team;
00071   p_shmem_ctx_create(0, &ctx);
00072   int ret = p_shmem_ctx_get_team(ctx, &team);
00073   p_shmem_ctx_destroy(ctx);
00074   return (ret == 0 && team == SHMEM_TEAM_WORLD);
00075 }
```

4.19 src/tests/comms/comms_tests.hpp File Reference

Contains function declarations for the OpenSHMEM communication/context tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
```

Functions

- bool test_shmem_ctx_create (void)

  Tests the shmem_ctx_create() function.
- bool test_shmem_team_create_ctx (void)

  Tests the shmem_team_create_ctx() function.
- bool test_shmem_ctx_destroy (void)

  Tests the shmem_ctx_destroy() function.
- bool test_shmem_ctx_get_team (void)

  Tests the shmem_ctx_get_team() function.

4.19.1 Detailed Description

Contains function declarations for the OpenSHMEM communication/context tests.

Definition in file comms_tests.hpp.

4.19.2 Function Documentation

4.19.2.1 test_shmem_ctx_create()

```
bool test_shmem_ctx_create (
            void )
```

Tests the shmem_ctx_create() function.

This test verifies that the shmem_ctx_create() function correctly creates a communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file comms_tests.cpp.

```
00015                                   {
00016   shmem_ctx_t ctx;
00017   int ret = p_shmem_ctx_create(0, &ctx);
00018   if (ret != 0) {
00019     return false;
00020   }
00021   p_shmem_ctx_destroy(ctx);
00022   return true;
00023 }
```

References p_shmem_ctx_create, and p_shmem_ctx_destroy.

4.19.2.2 test_shmem_ctx_destroy()

```
bool test_shmem_ctx_destroy (
            void )
```

Tests the shmem_ctx_destroy() function.

This test verifies that the shmem_ctx_destroy() function correctly destroys a communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 53 of file comms_tests.cpp.

```
00053                                    {
00054   shmem_ctx_t ctx;
00055   p_shmem_ctx_create(0, &ctx);
00056   p_shmem_ctx_destroy(ctx);
00057   return true;
00058 }
```

References p_shmem_ctx_create, and p_shmem_ctx_destroy.

4.19.2.3 test_shmem_ctx_get_team()

```
bool test_shmem_ctx_get_team (
            void )
```

Tests the shmem_ctx_get_team() function.

This test verifies that the shmem_ctx_get_team() function correctly retrieves the team associated with a given communication context.

Returns

True if the test is successful, false otherwise.

Definition at line 68 of file comms_tests.cpp.

```
00068                                    {
00069   shmem_ctx_t ctx;
00070   shmem_team_t team;
00071   p_shmem_ctx_create(0, &ctx);
00072   int ret = p_shmem_ctx_get_team(ctx, &team);
00073   p_shmem_ctx_destroy(ctx);
00074   return (ret == 0 && team == SHMEM_TEAM_WORLD);
00075 }
```

References p_shmem_ctx_create, p_shmem_ctx_destroy, and p_shmem_ctx_get_team.

4.19.2.4 test_shmem_team_create_ctx()

```
bool test_shmem_team_create_ctx (
             void )
```

Tests the shmem_team_create_ctx() function.

This test verifies that the shmem_team_create_ctx() function correctly creates a context for a specified team.

Returns

True if the test is successful, false otherwise.

Definition at line 33 of file comms_tests.cpp.

```
00033                                          {
00034   shmem_team_t team;
00035   shmem_ctx_t ctx;
00036   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00037   int ret = shmem_team_create_ctx(team, 0, &ctx);
00038   if (ret != 0) {
00039     return false;
00040   }
00041   p_shmem_ctx_destroy(ctx);
00042   p_shmem_team_destroy(team);
00043   return true;
00044 }
```

References p_shmem_ctx_destroy, p_shmem_n_pes, p_shmem_team_destroy, and p_shmem_team_split_strided.

4.20 comms_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef COMMS_TESTS_HPP
00007 #define COMMS_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012
00020 bool test_shmem_ctx_create(void);
00021
00030 bool test_shmem_team_create_ctx(void);
00031
00039 bool test_shmem_ctx_destroy(void);
00040
00049 bool test_shmem_ctx_get_team(void);
00050
00051 #endif /* COMMS_TESTS_HPP */
```

4.21 src/tests/locking/locking_tests.cpp File Reference

Contains OpenSHMEM distributed locking tests.

```
#include "locking_tests.hpp"
```

Functions

- bool test_shmem_lock_unlock (void)

  Tests the shmem_set_lock() and shmem_clear_lock() routines.

4.21.1 Detailed Description

Contains OpenSHMEM distributed locking tests.

Definition in file locking_tests.cpp.

4.21.2 Function Documentation

4.21.2.1 test_shmem_lock_unlock()

```
bool test_shmem_lock_unlock (
            void )
```

Tests the shmem_set_lock() and shmem_clear_lock() routines.

This test verifies that the shmem_set_lock() and shmem_clear_lock() routines correctly set and clear a distributed lock. It ensures that the lock is properly set by one PE and the state is correctly observed by another PE.

Returns

True if the test is successful, false otherwise.

Definition at line 17 of file locking_tests.cpp.

```
00017                                   {
00018   long *lock = (long *)p_shmem_malloc(sizeof(long));
00019   *lock = 0;
00020   int mype = p_shmem_my_pe();
00021   bool result = true;
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_set_lock(lock);
00027     *lock = 1;
00028     p_shmem_clear_lock(lock);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     p_shmem_set_lock(lock);
00035     if (*lock != 1) {
00036       result = false;
00037     }
00038     p_shmem_clear_lock(lock);
00039   }
00040
00041   p_shmem_barrier_all();
00042
00043   p_shmem_free(lock);
00044   return result;
00045 }
```

References p_shmem_barrier_all, p_shmem_clear_lock, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_set_lock.

4.22 locking_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "locking_tests.hpp"
00007
00017 bool test_shmem_lock_unlock(void) {
00018   long *lock = (long *)p_shmem_malloc(sizeof(long));
00019   *lock = 0;
00020   int mype = p_shmem_my_pe();
00021   bool result = true;
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_set_lock(lock);
00027     *lock = 1;
00028     p_shmem_clear_lock(lock);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     p_shmem_set_lock(lock);
00035     if (*lock != 1) {
00036       result = false;
00037     }
00038     p_shmem_clear_lock(lock);
00039   }
00040
00041   p_shmem_barrier_all();
00042
00043   p_shmem_free(lock);
00044   return result;
00045 }
```

4.23 src/tests/locking/locking_tests.hpp File Reference

Contains function declarations for the OpenSHMEM distributed locking tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
#include <cstdlib>
#include <cstring>
```

Functions

- bool test_shmem_lock_unlock (void)

  Tests the shmem_set_lock() and shmem_clear_lock() routines.

4.23.1 Detailed Description

Contains function declarations for the OpenSHMEM distributed locking tests.

Definition in file locking_tests.hpp.

4.23.2 Function Documentation

4.23.2.1 test_shmem_lock_unlock()

```
bool test_shmem_lock_unlock (
            void )
```

Tests the shmem_set_lock() and shmem_clear_lock() routines.

This test verifies that the shmem_set_lock() and shmem_clear_lock() routines correctly set and clear a distributed lock. It ensures that the lock is properly set by one PE and the state is correctly observed by another PE.

Returns

True if the test is successful, false otherwise.

Definition at line 17 of file locking_tests.cpp.

```
00017                                   {
00018   long *lock = (long *)p_shmem_malloc(sizeof(long));
00019   *lock = 0;
00020   int mype = p_shmem_my_pe();
00021   bool result = true;
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_set_lock(lock);
00027     *lock = 1;
00028     p_shmem_clear_lock(lock);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     p_shmem_set_lock(lock);
00035     if (*lock != 1) {
00036       result = false;
00037     }
00038     p_shmem_clear_lock(lock);
00039   }
00040
00041   p_shmem_barrier_all();
00042
00043   p_shmem_free(lock);
00044   return result;
00045 }
```

References p_shmem_barrier_all, p_shmem_clear_lock, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_set_lock.

4.24 locking_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef LOCKING_TESTS_HPP
00007 #define LOCKING_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012 #include <cstdlib>
00013 #include <cstring>
00014
00024 bool test_shmem_lock_unlock(void);
00025
00026 #endif /* LOCKING_TESTS_HPP */
```

4.25 src/tests/mem/mem_tests.cpp File Reference

Contains OpenSHMEM memory management tests.

```
#include "mem_tests.hpp"
```

Functions

- bool test_shmem_malloc_free (void)

  Tests the shmem_malloc() and shmem_free() functions.
- bool test_shmem_ptr ()

  Tests the shmem_ptr() routine.
- bool test_shmem_addr_accessible ()

  Tests the shmem_addr_accessible() routine.
- bool test_shmem_realloc (void)

  Tests the shmem_realloc() routine.
- bool test_shmem_align (void)

  Tests the shmem_align() routine.
- bool test_shmem_malloc_with_hints (void)

  Tests the shmem_malloc_with_hints() routine.
- bool test_shmem_calloc (void)

  Tests the shmem_calloc() routine.

4.25.1 Detailed Description

Contains OpenSHMEM memory management tests.

Definition in file mem_tests.cpp.

4.25.2 Function Documentation

4.25.2.1 test_shmem_addr_accessible()

```
bool test_shmem_addr_accessible (
            void )
```

Tests the shmem_addr_accessible() routine.

This test verifies that the shmem_addr_accessible() function correctly checks whether a memory address is accessible from all PEs.

Returns

True if the address is accessible from all PEs, false otherwise.

Definition at line 74 of file mem_tests.cpp.

```
00074                                   {
00075   int mype = p_shmem_my_pe();
00076   int npes = p_shmem_n_pes();
00077   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00078
00079   if (ptr == nullptr) {
00080     return false;
00081   }
00082
00083   *ptr = mype;
00084
00085   p_shmem_barrier_all();
00086
00087   bool test_passed = true;
00088
00089   for (int pe = 0; pe < npes; ++pe) {
00090     if (p_shmem_addr_accessible(ptr, pe) != 1) {
00091       test_passed = false;
00092     }
00093   }
00094
00095   p_shmem_free(ptr);
00096   return test_passed;
00097 }
```

References p_shmem_addr_accessible, p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.25.2.2 test_shmem_align()

```
bool test_shmem_align (
             void )
```

Tests the shmem_align() routine.

This test verifies that the shmem_align() function correctly allocates memory with the specified alignment.

Returns

True if the test is successful, false otherwise.

Definition at line 131 of file mem_tests.cpp.

```
00131                                   {
00132   size_t alignment = 64;
00133   size_t size = 1024;
00134   void *ptr = p_shmem_align(alignment, size);
00135   if (ptr == nullptr) {
00136     return false;
00137   }
00138   p_shmem_free(ptr);
00139   return true;
00140 }
```

References p_shmem_align, and p_shmem_free.

4.25.2.3 test_shmem_calloc()

```
bool test_shmem_calloc (
            void )
```

Tests the shmem_calloc() routine.

This test verifies that the shmem_calloc() function correctly allocates and initializes memory to zero.

Returns

True if the test is successful, false otherwise.

Definition at line 169 of file mem_tests.cpp.

```
00169                                {
00170   size_t count = 256;
00171   size_t size = sizeof(int);
00172   int *ptr = (int *)p_shmem_calloc(count, size);
00173   if (ptr == nullptr) {
00174     return false;
00175   }
00176   for (size_t i = 0; i < count; ++i) {
00177     if (ptr[i] != 0) {
00178       p_shmem_free(ptr);
00179       return false;
00180     }
00181   }
00182   p_shmem_free(ptr);
00183   return true;
00184 }
```

References p_shmem_calloc, and p_shmem_free.

4.25.2.4 test_shmem_malloc_free()

```
bool test_shmem_malloc_free (
            void )
```

Tests the shmem_malloc() and shmem_free() functions.

This test verifies that the shmem_malloc() function allocates memory correctly and that the shmem_free() function deallocates the memory correctly.

Returns

True if the tests are successful, false otherwise.

Definition at line 16 of file mem_tests.cpp.

```
00016                                    {
00017   size_t size = 1024;
00018   void *ptr = p_shmem_malloc(size);
00019   if (ptr == nullptr) {
00020     return false;
00021   }
00022   p_shmem_free(ptr);
00023   return true;
00024 }
```

References p_shmem_free, and p_shmem_malloc.

4.25.2.5 test_shmem_malloc_with_hints()

```
bool test_shmem_malloc_with_hints (
            void )
```

Tests the shmem_malloc_with_hints() routine.

This test verifies that the shmem_malloc_with_hints() function correctly allocates memory with the specified hints.

Returns

True if the test is successful, false otherwise.

Definition at line 150 of file mem_tests.cpp.

```
00150                                          {
00151   size_t size = 1024;
00152   long hints = SHMEM_MALLOC_ATOMICS_REMOTE;
00153   void *ptr = p_shmem_malloc_with_hints(size, hints);
00154   if (ptr == nullptr) {
00155     return false;
00156   }
00157   p_shmem_free(ptr);
00158   return true;
00159 }
```

References p_shmem_free, and p_shmem_malloc_with_hints.

4.25.2.6 test_shmem_ptr()

```
bool test_shmem_ptr (
            void )
```

Tests the shmem_ptr() routine.

This test verifies that the shmem_ptr() function correctly provides access to the memory of another PE and that the memory content is accessible and correct.

Returns

True if the pointer is accessible, false otherwise.

Definition at line 34 of file mem_tests.cpp.

```
00034                       {
00035   int mype = p_shmem_my_pe();
00036   int npes = p_shmem_n_pes();
00037   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00038
00039   if (ptr == nullptr) {
00040     return false;
00041   }
00042
00043   *ptr = mype;
00044
00045   p_shmem_barrier_all();
00046
00047   bool test_passed = true;
00048
00049   for (int pe = 0; pe < npes; ++pe) {
00050     int *remote_ptr = (int *)p_shmem_ptr(ptr, pe);
00051
00052     if (remote_ptr != nullptr) {
00053       int remote_val = *remote_ptr;
00054       if (remote_val != pe) {
00055         test_passed = false;
00056       }
00057     } else if (pe == mype) {
00058       test_passed = false;
00059     }
00060   }
00061
00062   p_shmem_free(ptr);
00063   return test_passed;
00064 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_ptr.

4.25.2.7 test_shmem_realloc()

```
bool test_shmem_realloc (
             void )
```

Tests the shmem_realloc() routine.

This test verifies that the shmem_realloc() function correctly reallocates memory and that the new memory block is usable.

Returns

True if the test is successful, false otherwise.

Definition at line 107 of file mem_tests.cpp.

```
00107                                {
00108   size_t size = 1024;
00109   void *ptr = p_shmem_malloc(size);
00110   if (ptr == nullptr) {
00111     return false;
00112   }
00113   size_t new_size = 2048;
00114   void *new_ptr = p_shmem_realloc(ptr, new_size);
00115   if (new_ptr == nullptr) {
00116     p_shmem_free(ptr);
00117     return false;
00118   }
00119   p_shmem_free(new_ptr);
00120   return true;
00121 }
```

References p_shmem_free, p_shmem_malloc, and p_shmem_realloc.

4.26 mem_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "mem_tests.hpp"
00007
00016 bool test_shmem_malloc_free(void) {
00017   size_t size = 1024;
00018   void *ptr = p_shmem_malloc(size);
00019   if (ptr == nullptr) {
00020     return false;
00021   }
00022   p_shmem_free(ptr);
00023   return true;
00024 }
00025
00034 bool test_shmem_ptr() {
00035   int mype = p_shmem_my_pe();
00036   int npes = p_shmem_n_pes();
00037   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00038
00039   if (ptr == nullptr) {
00040     return false;
00041   }
00042
00043   *ptr = mype;
00044
00045   p_shmem_barrier_all();
00046
00047   bool test_passed = true;
00048
00049   for (int pe = 0; pe < npes; ++pe) {
00050     int *remote_ptr = (int *)p_shmem_ptr(ptr, pe);
00051
00052     if (remote_ptr != nullptr) {
00053       int remote_val = *remote_ptr;
00054       if (remote_val != pe) {
00055         test_passed = false;
00056       }
```

```
00057     } else if (pe == mype) {
00058       test_passed = false;
00059     }
00060   }
00061
00062   p_shmem_free(ptr);
00063   return test_passed;
00064 }
00065
00074 bool test_shmem_addr_accessible() {
00075   int mype = p_shmem_my_pe();
00076   int npes = p_shmem_n_pes();
00077   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00078
00079   if (ptr == nullptr) {
00080     return false;
00081   }
00082
00083   *ptr = mype;
00084
00085   p_shmem_barrier_all();
00086
00087   bool test_passed = true;
00088
00089   for (int pe = 0; pe < npes; ++pe) {
00090     if (p_shmem_addr_accessible(ptr, pe) != 1) {
00091       test_passed = false;
00092     }
00093   }
00094
00095   p_shmem_free(ptr);
00096   return test_passed;
00097 }
00098
00107 bool test_shmem_realloc(void) {
00108   size_t size = 1024;
00109   void *ptr = p_shmem_malloc(size);
00110   if (ptr == nullptr) {
00111     return false;
00112   }
00113   size_t new_size = 2048;
00114   void *new_ptr = p_shmem_realloc(ptr, new_size);
00115   if (new_ptr == nullptr) {
00116     p_shmem_free(ptr);
00117     return false;
00118   }
00119   p_shmem_free(new_ptr);
00120   return true;
00121 }
00122
00131 bool test_shmem_align(void) {
00132   size_t alignment = 64;
00133   size_t size = 1024;
00134   void *ptr = p_shmem_align(alignment, size);
00135   if (ptr == nullptr) {
00136     return false;
00137   }
00138   p_shmem_free(ptr);
00139   return true;
00140 }
00141
00150 bool test_shmem_malloc_with_hints(void) {
00151   size_t size = 1024;
00152   long hints = SHMEM_MALLOC_ATOMICS_REMOTE;
00153   void *ptr = p_shmem_malloc_with_hints(size, hints);
00154   if (ptr == nullptr) {
00155     return false;
00156   }
00157   p_shmem_free(ptr);
00158   return true;
00159 }
00160
00169 bool test_shmem_calloc(void) {
00170   size_t count = 256;
00171   size_t size = sizeof(int);
00172   int *ptr = (int *)p_shmem_calloc(count, size);
00173   if (ptr == nullptr) {
00174     return false;
00175   }
00176   for (size_t i = 0; i < count; ++i) {
00177     if (ptr[i] != 0) {
00178       p_shmem_free(ptr);
00179       return false;
00180     }
00181   }
00182   p_shmem_free(ptr);
00183   return true;
```

```
00184 }
```

4.27 src/tests/mem/mem_tests.hpp File Reference

Contains function declarations for the OpenSHMEM memory management tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
#include <cstdlib>
#include <cstring>
```

Functions

- bool test_shmem_ptr (void)

  Tests the shmem_ptr() routine.
- bool test_shmem_malloc_free (void)

  Tests the shmem_malloc() and shmem_free() functions.
- bool test_shmem_addr_accessible (void)

  Tests the shmem_addr_accessible() routine.
- bool test_shmem_realloc (void)

  Tests the shmem_realloc() routine.
- bool test_shmem_align (void)

  Tests the shmem_align() routine.
- bool test_shmem_malloc_with_hints (void)

  Tests the shmem_malloc_with_hints() routine.
- bool test_shmem_calloc (void)

  Tests the shmem_calloc() routine.

4.27.1 Detailed Description

Contains function declarations for the OpenSHMEM memory management tests.

Definition in file mem_tests.hpp.

4.27.2 Function Documentation

4.27.2.1 test_shmem_addr_accessible()

```
bool test_shmem_addr_accessible (
            void )
```

Tests the shmem_addr_accessible() routine.

This test verifies that the shmem_addr_accessible() function correctly checks whether a memory address is accessible from all PEs.

Returns

True if the address is accessible from all PEs, false otherwise.

Definition at line 74 of file mem_tests.cpp.

```
00074                                   {
00075   int mype = p_shmem_my_pe();
00076   int npes = p_shmem_n_pes();
00077   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00078
00079   if (ptr == nullptr) {
00080     return false;
00081   }
00082
00083   *ptr = mype;
00084
00085   p_shmem_barrier_all();
00086
00087   bool test_passed = true;
00088
00089   for (int pe = 0; pe < npes; ++pe) {
00090     if (p_shmem_addr_accessible(ptr, pe) != 1) {
00091       test_passed = false;
00092     }
00093   }
00094
00095   p_shmem_free(ptr);
00096   return test_passed;
00097 }
```

References p_shmem_addr_accessible, p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, and p_shmem_n_pes.

4.27.2.2 test_shmem_align()

```
bool test_shmem_align (
             void )
```

Tests the shmem_align() routine.

This test verifies that the shmem_align() function correctly allocates memory with the specified alignment.

Returns

True if the test is successful, false otherwise.

Definition at line 131 of file mem_tests.cpp.

```
00131                                   {
00132   size_t alignment = 64;
00133   size_t size = 1024;
00134   void *ptr = p_shmem_align(alignment, size);
00135   if (ptr == nullptr) {
00136     return false;
00137   }
00138   p_shmem_free(ptr);
00139   return true;
00140 }
```

References p_shmem_align, and p_shmem_free.

4.27.2.3 test_shmem_calloc()

```
bool test_shmem_calloc (
            void )
```

Tests the shmem_calloc() routine.

This test verifies that the shmem_calloc() function correctly allocates and initializes memory to zero.

Returns

True if the test is successful, false otherwise.

Definition at line 169 of file mem_tests.cpp.

```
00169                                {
00170   size_t count = 256;
00171   size_t size = sizeof(int);
00172   int *ptr = (int *)p_shmem_calloc(count, size);
00173   if (ptr == nullptr) {
00174     return false;
00175   }
00176   for (size_t i = 0; i < count; ++i) {
00177     if (ptr[i] != 0) {
00178       p_shmem_free(ptr);
00179       return false;
00180     }
00181   }
00182   p_shmem_free(ptr);
00183   return true;
00184 }
```

References p_shmem_calloc, and p_shmem_free.

4.27.2.4 test_shmem_malloc_free()

```
bool test_shmem_malloc_free (
            void )
```

Tests the shmem_malloc() and shmem_free() functions.

This test verifies that the shmem_malloc() function allocates memory correctly and that the shmem_free() function deallocates the memory correctly.

Returns

True if the tests are successful, false otherwise.

Definition at line 16 of file mem_tests.cpp.

```
00016                                  {
00017   size_t size = 1024;
00018   void *ptr = p_shmem_malloc(size);
00019   if (ptr == nullptr) {
00020     return false;
00021   }
00022   p_shmem_free(ptr);
00023   return true;
00024 }
```

References p_shmem_free, and p_shmem_malloc.

4.27.2.5 test_shmem_malloc_with_hints()

```
bool test_shmem_malloc_with_hints (
            void )
```

Tests the shmem_malloc_with_hints() routine.

This test verifies that the shmem_malloc_with_hints() function correctly allocates memory with the specified hints.

Returns

True if the test is successful, false otherwise.

Definition at line 150 of file mem_tests.cpp.

```
00150                                          {
00151   size_t size = 1024;
00152   long hints = SHMEM_MALLOC_ATOMICS_REMOTE;
00153   void *ptr = p_shmem_malloc_with_hints(size, hints);
00154   if (ptr == nullptr) {
00155     return false;
00156   }
00157   p_shmem_free(ptr);
00158   return true;
00159 }
```

References p_shmem_free, and p_shmem_malloc_with_hints.

4.27.2.6 test_shmem_ptr()

```
bool test_shmem_ptr (
            void )
```

Tests the shmem_ptr() routine.

This test verifies that the shmem_ptr() function correctly provides access to the memory of another PE and that the memory content is accessible and correct.

Returns

True if the pointer is accessible, false otherwise.

Definition at line 34 of file mem_tests.cpp.

```
00034                       {
00035   int mype = p_shmem_my_pe();
00036   int npes = p_shmem_n_pes();
00037   int *ptr = (int *)p_shmem_malloc(sizeof(int));
00038
00039   if (ptr == nullptr) {
00040     return false;
00041   }
00042
00043   *ptr = mype;
00044
00045   p_shmem_barrier_all();
00046
00047   bool test_passed = true;
00048
00049   for (int pe = 0; pe < npes; ++pe) {
00050     int *remote_ptr = (int *)p_shmem_ptr(ptr, pe);
00051
00052     if (remote_ptr != nullptr) {
00053       int remote_val = *remote_ptr;
00054       if (remote_val != pe) {
00055         test_passed = false;
00056       }
00057     } else if (pe == mype) {
00058       test_passed = false;
00059     }
00060   }
00061
00062   p_shmem_free(ptr);
00063   return test_passed;
00064 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_ptr.

4.27.2.7 test_shmem_realloc()

```
bool test_shmem_realloc (
             void )
```

Tests the shmem_realloc() routine.

This test verifies that the shmem_realloc() function correctly reallocates memory and that the new memory block is usable.

Returns

True if the test is successful, false otherwise.

Definition at line 107 of file mem_tests.cpp.

```
00107                                {
00108   size_t size = 1024;
00109   void *ptr = p_shmem_malloc(size);
00110   if (ptr == nullptr) {
00111     return false;
00112   }
00113   size_t new_size = 2048;
00114   void *new_ptr = p_shmem_realloc(ptr, new_size);
00115   if (new_ptr == nullptr) {
00116     p_shmem_free(ptr);
00117     return false;
00118   }
00119   p_shmem_free(new_ptr);
00120   return true;
00121 }
```

References p_shmem_free, p_shmem_malloc, and p_shmem_realloc.

4.28 mem_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef MEM_TESTS_HPP
00007 #define MEM_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012 #include <cstdlib>
00013 #include <cstring>
00014
00023 bool test_shmem_ptr(void);
00024
00033 bool test_shmem_malloc_free(void);
00034
00043 bool test_shmem_addr_accessible(void);
00044
00053 bool test_shmem_realloc(void);
00054
00063 bool test_shmem_align(void);
00064
00073 bool test_shmem_malloc_with_hints(void);
00074
00083 bool test_shmem_calloc(void);
00084
00085 #endif /* MEM_TESTS_HPP */
```

4.29 src/tests/mem_ordering/mem_ordering_tests.cpp File Reference

Contains OpenSHMEM memory ordering tests.

```
#include "mem_ordering_tests.hpp"
```

Functions

- bool test_shmem_fence (void)

  Tests the shmem_fence() routine.
- bool test_shmem_quiet (void)

  Tests the shmem_quiet() routine.

4.29.1 Detailed Description

Contains OpenSHMEM memory ordering tests.

Definition in file mem_ordering_tests.cpp.

4.29.2 Function Documentation

4.29.2.1 test_shmem_fence()

```
bool test_shmem_fence (
            void )
```

Tests the shmem_fence() routine.

This test verifies that the shmem_fence() routine correctly ensures the ordering of memory operations by checking that the memory store operation on one PE is seen by another PE in the correct order.

Returns

True if the test is successful, false otherwise.

Definition at line 17 of file mem_ordering_tests.cpp.

```
00017                                {
00018   long *flag = (long *)p_shmem_malloc(sizeof(long));
00019   *flag = 0;
00020   int mype = p_shmem_my_pe();
00021
00022   p_shmem_barrier_all();
00023
00024   if (mype == 0) {
00025     p_shmem_long_p(flag, 1, 1);
00026     p_shmem_fence();
00027     *flag = 2;
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   bool result = true;
00033   if (mype == 1) {
00034     if (*flag != 1) {
00035       result = false;
00036     }
00037   }
00038
00039   p_shmem_free(flag);
00040   return result;
00041 }
```

References p_shmem_barrier_all, p_shmem_fence, p_shmem_free, p_shmem_long_p, p_shmem_malloc, and p_shmem_my_pe.

4.29.2.2 test_shmem_quiet()

```
bool test_shmem_quiet (
             void )
```

Tests the shmem_quiet() routine.

This test verifies that the shmem_quiet() routine correctly ensures the completion of all outstanding memory operations by checking that a memory store operation on one PE is completed before proceeding.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file mem_ordering_tests.cpp.

```
00052                                {
00053   long *flag = (long *)p_shmem_malloc(sizeof(long));
00054   *flag = 0;
00055   int mype = p_shmem_my_pe();
00056
00057   p_shmem_barrier_all();
00058
00059   if (mype == 0) {
00060     p_shmem_long_p(flag, 1, 1);
00061     p_shmem_quiet();
00062   }
00063
00064   p_shmem_barrier_all();
00065
00066   bool result = true;
00067   if (mype == 1) {
00068     if (*flag != 1) {
00069       result = false;
00070     }
00071   }
00072
00073   p_shmem_free(flag);
00074   return result;
00075 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.30 mem_ordering_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "mem_ordering_tests.hpp"
00007
00017 bool test_shmem_fence(void) {
00018   long *flag = (long *)p_shmem_malloc(sizeof(long));
00019   *flag = 0;
00020   int mype = p_shmem_my_pe();
00021
00022   p_shmem_barrier_all();
00023
00024   if (mype == 0) {
00025     p_shmem_long_p(flag, 1, 1);
00026     p_shmem_fence();
00027     *flag = 2;
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   bool result = true;
00033   if (mype == 1) {
00034     if (*flag != 1) {
00035       result = false;
00036     }
00037   }
00038
```

```
00039   p_shmem_free(flag);
00040   return result;
00041 }
00042
00052 bool test_shmem_quiet(void) {
00053   long *flag = (long *)p_shmem_malloc(sizeof(long));
00054   *flag = 0;
00055   int mype = p_shmem_my_pe();
00056
00057   p_shmem_barrier_all();
00058
00059   if (mype == 0) {
00060     p_shmem_long_p(flag, 1, 1);
00061     p_shmem_quiet();
00062   }
00063
00064   p_shmem_barrier_all();
00065
00066   bool result = true;
00067   if (mype == 1) {
00068     if (*flag != 1) {
00069       result = false;
00070     }
00071   }
00072
00073   p_shmem_free(flag);
00074   return result;
00075 }
```

4.31 src/tests/mem_ordering/mem_ordering_tests.hpp File Reference

Contains function declarations for the OpenSHMEM memory ordering tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
#include <cstdlib>
#include <cstring>
```

Functions

- bool test_shmem_fence (void)

  Tests the shmem_fence() routine.
- bool test_shmem_quiet (void)

  Tests the shmem_quiet() routine.

4.31.1 Detailed Description

Contains function declarations for the OpenSHMEM memory ordering tests.

Definition in file mem_ordering_tests.hpp.

4.31.2 Function Documentation

4.31.2.1 test_shmem_fence()

```
bool test_shmem_fence (
            void )
```

Tests the shmem_fence() routine.

This test verifies that the shmem_fence() routine correctly ensures the ordering of memory operations by checking that the memory store operation on one PE is seen by another PE in the correct order.

Returns

True if the test is successful, false otherwise.

Definition at line 17 of file mem_ordering_tests.cpp.

```
00017                                {
00018   long *flag = (long *)p_shmem_malloc(sizeof(long));
00019   *flag = 0;
00020   int mype = p_shmem_my_pe();
00021
00022   p_shmem_barrier_all();
00023
00024   if (mype == 0) {
00025     p_shmem_long_p(flag, 1, 1);
00026     p_shmem_fence();
00027     *flag = 2;
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   bool result = true;
00033   if (mype == 1) {
00034     if (*flag != 1) {
00035       result = false;
00036     }
00037   }
00038
00039   p_shmem_free(flag);
00040   return result;
00041 }
```

References p_shmem_barrier_all, p_shmem_fence, p_shmem_free, p_shmem_long_p, p_shmem_malloc, and p_shmem_my_pe.

4.31.2.2 test_shmem_quiet()

```
bool test_shmem_quiet (
            void )
```

Tests the shmem_quiet() routine.

This test verifies that the shmem_quiet() routine correctly ensures the completion of all outstanding memory operations by checking that a memory store operation on one PE is completed before proceeding.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file mem_ordering_tests.cpp.

```
00052                                {
00053   long *flag = (long *)p_shmem_malloc(sizeof(long));
00054   *flag = 0;
00055   int mype = p_shmem_my_pe();
00056
00057   p_shmem_barrier_all();
00058
00059   if (mype == 0) {
00060     p_shmem_long_p(flag, 1, 1);
00061     p_shmem_quiet();
00062   }
00063
00064   p_shmem_barrier_all();
00065
00066   bool result = true;
00067   if (mype == 1) {
00068     if (*flag != 1) {
00069       result = false;
00070     }
00071   }
00072
00073   p_shmem_free(flag);
00074   return result;
00075 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.32 mem_ordering_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef MEM_ORDERING_TESTS_HPP
00007 #define MEM_ORDERING_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012 #include <cstdlib>
00013 #include <cstring>
00014
00024 bool test_shmem_fence(void);
00025
00035 bool test_shmem_quiet(void);
00036
00037 #endif /* MEM_ORDERING_TESTS_HPP */
```

4.33 src/tests/pt2pt/pt2pt_tests.cpp File Reference

Contains functions definitions with test functions for the point-to-point synchronization routines.

```
#include "pt2pt_tests.hpp"
```

Macros

- #define TIMEOUT 10

Functions

- bool test_shmem_wait_until (void)

  Tests the shmem_wait_until() routine.
- bool test_shmem_wait_until_all (void)

  Tests the shmem_wait_until_all() routine.
- bool test_shmem_wait_until_any (void)

  Tests the shmem_wait_until_any() routine.
- bool test_shmem_wait_until_some (void)

  Tests the shmem_wait_until_some() routine.
- bool test_shmem_test (void)

  Tests the shmem_test() routine.
- bool test_shmem_test_all (void)

  Tests the shmem_test_all() routine.
- bool test_shmem_test_any (void)

  Tests the shmem_test_any() routine.
- bool test_shmem_test_some (void)

  Tests the shmem_test_some() routine.
- bool test_shmem_wait_until_all_vector (void)

  Tests the shmem_wait_until_all_vector() routine.
- bool test_shmem_wait_until_any_vector (void)

  Tests the shmem_wait_until_any_vector() routine.
- bool test_shmem_wait_until_some_vector (void)

  Tests the shmem_wait_until_some_vector() routine.
- bool test_shmem_test_all_vector (void)

  Tests the shmem_test_all_vector() routine.
- bool test_shmem_test_any_vector (void)

  Tests the shmem_test_any_vector() routine.
- bool test_shmem_test_some_vector (void)

  Tests the shmem_test_some_vector() routine.
- bool test_shmem_signal_wait_until (void)

  Tests the shmem_signal_wait_until() routine.

4.33.1 Detailed Description

Contains functions definitions with test functions for the point-to-point synchronization routines.

Definition in file pt2pt_tests.cpp.

4.33.2 Macro Definition Documentation

4.33.2.1 TIMEOUT

```
#define TIMEOUT 10
```

Definition at line 8 of file pt2pt_tests.cpp.

4.33.3 Function Documentation

4.33.3.1 test_shmem_signal_wait_until()

```
bool test_shmem_signal_wait_until (
            void )
```

Tests the shmem_signal_wait_until() routine.

This test verifies that the shmem_signal_wait_until() function correctly waits until a signal on a memory location meets a specified condition.

Returns

True if the test is successful, false otherwise.

Definition at line 648 of file pt2pt_tests.cpp.

```
00648                                          {
00649   uint64_t *flag = (uint64_t *)p_shmem_malloc(sizeof(uint64_t));
00650   if (flag == NULL) {
00651     return false;
00652   }
00653
00654   *flag = 0;
00655   int mype = p_shmem_my_pe();
00656   uint64_t value = 1;
00657
00658   p_shmem_barrier_all();
00659
00660   if (mype == 0) {
00661     shmem_uint64_p(flag, value, 1);
00662     p_shmem_quiet();
00663   }
00664
00665   p_shmem_barrier_all();
00666
00667   if (mype != 0) {
00668     time_t start_time = time(NULL);
00669     while (*flag != value && time(NULL) - start_time < TIMEOUT) {
00670       p_shmem_signal_wait_until(flag, SHMEM_CMP_EQ, value);
00671     }
00672     if (*flag != value) {
00673       p_shmem_free(flag);
00674       return false;
00675     }
00676   }
00677
00678   p_shmem_free(flag);
00679   return true;
00680 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, p_shmem_signal_wait_until, and TIMEOUT.

4.33.3.2 test_shmem_test()

```
bool test_shmem_test (
            void )
```

Tests the shmem_test() routine.

This test verifies that the shmem_test() function correctly tests whether a condition on a memory location is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 174 of file pt2pt_tests.cpp.

```
00174                                {
00175   long *flag = (long *)p_shmem_malloc(sizeof(long));
00176   if (flag == NULL) {
00177     return false;
00178   }
00179
00180   *flag = 0;
00181   int mype = p_shmem_my_pe();
00182
00183   p_shmem_barrier_all();
00184
00185   if (mype == 0) {
00186     *flag = 1;
00187     p_shmem_quiet();
00188   }
00189
00190   p_shmem_barrier_all();
00191
00192   if (mype != 0) {
00193     time_t start_time = time(NULL);
00194     while (!p_shmem_long_test(flag, SHMEM_CMP_EQ, 1)) {
00195       if (time(NULL) - start_time > TIMEOUT) {
00196         break;
00197       }
00198       usleep(1000);
00199     }
00200     if (*flag != 1) {
00201       p_shmem_free(flag);
00202       return false;
00203     }
00204   }
00205
00206   p_shmem_free(flag);
00207   return true;
00208 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.33.3.3 test_shmem_test_all()

```
bool test_shmem_test_all (
             void )
```

Tests the shmem_test_all() routine.

This test verifies that the shmem_test_all() function correctly tests whether all specified conditions on an array of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 218 of file pt2pt_tests.cpp.

```
00218                                    {
00219   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00220   if (flags == NULL) {
00221     return false;
00222   }
00223
00224   for (int i = 0; i < 4; ++i) {
00225     flags[i] = 0;
00226   }
00227   int mype = p_shmem_my_pe();
00228
00229   p_shmem_barrier_all();
00230
```

```
00231   if (mype == 0) {
00232     for (int i = 0; i < 4; ++i) {
00233       flags[i] = 1;
00234     }
00235     p_shmem_quiet();
00236   }
00237
00238   p_shmem_barrier_all();
00239
00240   if (mype != 0) {
00241     time_t start_time = time(NULL);
00242     while (!p_shmem_long_test_all(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00243       if (time(NULL) - start_time > TIMEOUT) {
00244         break;
00245       }
00246       usleep(1000);
00247     }
00248     for (int i = 0; i < 4; ++i) {
00249       if (flags[i] != 1) {
00250         p_shmem_free(flags);
00251         return false;
00252       }
00253     }
00254   }
00255
00256   p_shmem_free(flags);
00257   return true;
00258 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_all, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.33.3.4 test_shmem_test_all_vector()

```
bool test_shmem_test_all_vector (
             void )
```

Tests the shmem_test_all_vector() routine.

This test verifies that the shmem_test_all_vector() function correctly tests whether all specified conditions on a vector of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 492 of file pt2pt_tests.cpp.

```
00492                                       {
00493   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00494   if (flags == NULL) {
00495     return false;
00496   }
00497
00498   for (int i = 0; i < 4; ++i) {
00499     flags[i] = 0;
00500   }
00501   int mype = p_shmem_my_pe();
00502
00503   p_shmem_barrier_all();
00504
00505   if (mype == 0) {
00506     for (int i = 0; i < 4; ++i) {
00507       flags[i] = 1;
00508     }
00509     p_shmem_quiet();
00510   }
00511
00512   p_shmem_barrier_all();
00513
00514   if (mype != 0) {
00515     long cmp_values[4] = {1, 1, 1, 1};
00516     time_t start_time = time(NULL);
00517     while (!p_shmem_long_test_all_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
00518       if (time(NULL) - start_time > TIMEOUT) {
```

```
00519         break;
00520       }
00521       usleep(1000);
00522     }
00523     for (int i = 0; i < 4; ++i) {
00524       if (flags[i] != 1) {
00525         p_shmem_free(flags);
00526         return false;
00527       }
00528     }
00529   }
00530
00531   p_shmem_free(flags);
00532   return true;
00533 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_all_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.33.3.5 test_shmem_test_any()

```
bool test_shmem_test_any (
             void )
```

Tests the shmem_test_any() routine.

This test verifies that the shmem_test_any() function correctly tests whether any one of the specified conditions on an array of memory locations is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 268 of file pt2pt_tests.cpp.

```
00268                                {
00269   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00270   if (flags == NULL) {
00271     return false;
00272   }
00273
00274   for (int i = 0; i < 4; ++i) {
00275     flags[i] = 0;
00276   }
00277   int mype = p_shmem_my_pe();
00278
00279   p_shmem_barrier_all();
00280
00281   if (mype == 0) {
00282     flags[2] = 1;
00283     p_shmem_quiet();
00284   }
00285
00286   p_shmem_barrier_all();
00287
00288   if (mype != 0) {
00289     time_t start_time = time(NULL);
00290     while (!p_shmem_long_test_any(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00291       if (time(NULL) - start_time > TIMEOUT) {
00292         break;
00293       }
00294       usleep(1000);
00295     }
00296     if (flags[2] != 1) {
00297       p_shmem_free(flags);
00298       return false;
00299     }
00300   }
00301
00302   p_shmem_free(flags);
00303   return true;
00304 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_any, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.33.3.6 test_shmem_test_any_vector()

```
bool test_shmem_test_any_vector (
            void )
```

Tests the shmem_test_any_vector() routine.

This test verifies that the shmem_test_any_vector() function correctly tests whether any one of the specified conditions on a vector of memory locations is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 543 of file pt2pt_tests.cpp.

```
00543                                       {
00544   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00545   if (flags == NULL) {
00546     return false;
00547   }
00548
00549   for (int i = 0; i < 4; ++i) {
00550     flags[i] = 0;
00551   }
00552   int mype = p_shmem_my_pe();
00553   int npes = p_shmem_n_pes();
00554
00555   p_shmem_barrier_all();
00556
00557   if (mype == 0) {
00558     long one = 1;
00559     for (int pe = 1; pe < npes; ++pe) {
00560       p_shmem_long_p(&flags[2], one, pe);
00561     }
00562     p_shmem_quiet();
00563   }
00564
00565   p_shmem_barrier_all();
00566
00567   if (mype != 0) {
00568     long cmp_values[4] = {1, 1, 1, 1};
00569     time_t start_time = time(NULL);
00570     while (!p_shmem_long_test_any_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
00571       if (time(NULL) - start_time > TIMEOUT) {
00572         break;
00573       }
00574       usleep(1000);
00575     }
00576     if (flags[2] != 1) {
00577       p_shmem_free(flags);
00578       return false;
00579     }
00580   }
00581
00582   p_shmem_free(flags);
00583   return true;
00584 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_test_any_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, p_shmem_quiet, and TIMEOUT.

4.33.3.7 test_shmem_test_some()

```
bool test_shmem_test_some (
            void )
```

Tests the shmem_test_some() routine.

This test verifies that the shmem_test_some() function correctly tests whether some of the specified conditions on an array of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 314 of file pt2pt_tests.cpp.

```
00314                                    {
00315   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00316   if (flags == NULL) {
00317     return false;
00318   }
00319
00320   for (int i = 0; i < 4; ++i) {
00321     flags[i] = 0;
00322   }
00323   int mype = p_shmem_my_pe();
00324
00325   p_shmem_barrier_all();
00326
00327   if (mype == 0) {
00328     flags[1] = 1;
00329     flags[3] = 1;
00330     p_shmem_quiet();
00331   }
00332
00333   p_shmem_barrier_all();
00334
00335   if (mype != 0) {
00336     size_t indices[4];
00337     time_t start_time = time(NULL);
00338     while (!p_shmem_long_test_some(flags, 4, indices, NULL, SHMEM_CMP_EQ, 1)) {
00339       if (time(NULL) - start_time > TIMEOUT) {
00340         break;
00341       }
00342       usleep(1000);
00343     }
00344     if (flags[1] != 1 || flags[3] != 1) {
00345       p_shmem_free(flags);
00346       return false;
00347     }
00348   }
00349
00350   p_shmem_free(flags);
00351   return true;
00352 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_some, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.33.3.8 test_shmem_test_some_vector()

```
bool test_shmem_test_some_vector (
            void )
```

Tests the shmem_test_some_vector() routine.

This test verifies that the shmem_test_some_vector() function correctly tests whether some of the specified conditions on a vector of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 594 of file pt2pt_tests.cpp.

```
00594                                    {
00595   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00596   if (flags == NULL) {
00597     return false;
00598   }
00599
00600   for (int i = 0; i < 4; ++i) {
00601     flags[i] = 0;
00602   }
```

```
00603   int mype = p_shmem_my_pe();
00604   int npes = p_shmem_n_pes();
00605
00606   p_shmem_barrier_all();
00607
00608   if (mype == 0) {
00609     long one = 1;
00610     for (int pe = 1; pe < npes; ++pe) {
00611       p_shmem_long_p(&flags[1], one, pe);
00612       p_shmem_long_p(&flags[3], one, pe);
00613     }
00614     p_shmem_quiet();
00615   }
00616
00617   p_shmem_barrier_all();
00618
00619   if (mype != 0) {
00620     long cmp_values[4] = {0, 1, 0, 1};
00621     size_t indices[4];
00622     size_t num_indices;
00623     time_t start_time = time(NULL);
00624     while ((num_indices = p_shmem_long_test_some_vector(flags, 4, indices, NULL, SHMEM_CMP_EQ,
      cmp_values)) == 0) {
00625       if (time(NULL) - start_time > TIMEOUT) {
00626         break;
00627       }
00628       usleep(1000);
00629     }
00630     if (flags[1] != 1 || flags[3] != 1) {
00631       p_shmem_free(flags);
00632       return false;
00633     }
00634   }
00635
00636   p_shmem_free(flags);
00637   return true;
00638 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_test_some_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, p_shmem_quiet, and TIMEOUT.

4.33.3.9 test_shmem_wait_until()

```
bool test_shmem_wait_until (
             void )
```

Tests the shmem_wait_until() routine.

This test verifies that the shmem_wait_until() function correctly waits until a condition on a memory location is met.

Returns

True if the test is successful, false otherwise.

Definition at line 18 of file pt2pt_tests.cpp.

```
00018                                   {
00019   long *flag = (long *)p_shmem_malloc(sizeof(long));
00020   *flag = 0;
00021   int mype = p_shmem_my_pe();
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_long_p(flag, 1, 1);
00027     p_shmem_quiet();
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   if (mype != 0) {
00033     p_shmem_long_wait_until(flag, SHMEM_CMP_EQ, 1);
00034     if (*flag != 1) {
00035       p_shmem_free(flag);
00036       return false;
00037     }
```

```
00038   }
00039
00040   p_shmem_free(flag);
00041   return true;
00042 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.10 test_shmem_wait_until_all()

```
bool test_shmem_wait_until_all (
             void )
```

Tests the shmem_wait_until_all() routine.

This test verifies that the shmem_wait_until_all() function correctly waits until all specified conditions on an array of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file pt2pt_tests.cpp.

```
00052                                    {
00053   long *flags = (long *)p_shmem_malloc(2 * sizeof(long));
00054   flags[0] = 0;
00055   flags[1] = 0;
00056   int mype = p_shmem_my_pe();
00057
00058   p_shmem_barrier_all();
00059
00060   if (mype == 0) {
00061     p_shmem_long_p(&flags[0], 1, 1);
00062     p_shmem_long_p(&flags[1], 1, 1);
00063     p_shmem_quiet();
00064   }
00065
00066   p_shmem_barrier_all();
00067
00068   if (mype != 0) {
00069     p_shmem_long_wait_until_all(flags, 2, NULL, SHMEM_CMP_EQ, 1);
00070     if (flags[0] != 1 || flags[1] != 1) {
00071       p_shmem_free(flags);
00072       return false;
00073     }
00074   }
00075
00076   p_shmem_free(flags);
00077   return true;
00078 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_all, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.11 test_shmem_wait_until_all_vector()

```
bool test_shmem_wait_until_all_vector (
             void )
```

Tests the shmem_wait_until_all_vector() routine.

This test verifies that the shmem_wait_until_all_vector() function correctly waits until all specified conditions on a vector of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 362 of file pt2pt_tests.cpp.

```
00362                                          {
00363   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00364   for (int i = 0; i < 4; ++i) {
00365     flags[i] = 0;
00366   }
00367   int mype = p_shmem_my_pe();
00368
00369   p_shmem_barrier_all();
00370
00371   if (mype == 0) {
00372     for (int i = 0; i < 4; ++i) {
00373       p_shmem_long_p(&flags[i], 1, 1);
00374       p_shmem_quiet();
00375     }
00376   }
00377
00378   p_shmem_barrier_all();
00379
00380   if (mype != 0) {
00381     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00382     long cmp_values[4] = {1, 1, 1, 1};
00383     p_shmem_long_wait_until_all_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00384     for (int i = 0; i < 4; ++i) {
00385       if (flags[i] != 1) {
00386         p_shmem_free(flags);
00387         return false;
00388       }
00389     }
00390   }
00391
00392   p_shmem_free(flags);
00393   return true;
00394 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_all_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.12 test_shmem_wait_until_any()

```
bool test_shmem_wait_until_any (
             void )
```

Tests the shmem_wait_until_any() routine.

This test verifies that the shmem_wait_until_any() function correctly waits until any one of the specified conditions on an array of memory locations is met.

Returns

True if the test is successful, false otherwise.

Definition at line 88 of file pt2pt_tests.cpp.

```
00088                                       {
00089   long *flags = (long *)p_shmem_malloc(3 * sizeof(long));
00090   for (int i = 0; i < 3; i++) {
00091     flags[i] = 0;
00092   }
00093   int mype = p_shmem_my_pe();
00094
00095   p_shmem_barrier_all();
00096
00097   if (mype == 0) {
00098     p_shmem_long_p(&flags[2], 1, 1);
00099     p_shmem_quiet();
00100   }
00101
00102   p_shmem_barrier_all();
```

```
00103
00104   if (mype != 0) {
00105     int status[3] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00106     size_t index = p_shmem_long_wait_until_any(flags, 3, status, SHMEM_CMP_EQ, 1);
00107     if (index == SIZE_MAX) {
00108       p_shmem_free(flags);
00109       return false;
00110     }
00111     if (flags[index] != 1) {
00112       p_shmem_free(flags);
00113       return false;
00114     }
00115   }
00116
00117   p_shmem_free(flags);
00118   return true;
00119 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_any, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.13 test_shmem_wait_until_any_vector()

```
bool test_shmem_wait_until_any_vector (
             void )
```

Tests the shmem_wait_until_any_vector() routine.

This test verifies that the shmem_wait_until_any_vector() function correctly waits until any one of the specified conditions on a vector of memory locations is met.

Returns

True if the test is successful, false otherwise.

Definition at line 404 of file pt2pt_tests.cpp.

```
00404                                       {
00405   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00406   for (int i = 0; i < 4; ++i) {
00407     flags[i] = 0;
00408   }
00409   int mype = p_shmem_my_pe();
00410
00411   p_shmem_barrier_all();
00412
00413   if (mype == 0) {
00414     p_shmem_long_p(&flags[2], 1, 1);
00415     p_shmem_quiet();
00416   }
00417
00418   p_shmem_barrier_all();
00419
00420   if (mype != 0) {
00421     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00422     long cmp_values[4] = {1, 1, 1, 1};
00423     size_t index = p_shmem_long_wait_until_any_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00424     if (index == SIZE_MAX) {
00425       p_shmem_free(flags);
00426       return false;
00427     }
00428     if (flags[index] != 1) {
00429       p_shmem_free(flags);
00430       return false;
00431     }
00432   }
00433
00434   p_shmem_free(flags);
00435   return true;
00436 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_any_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.14 test_shmem_wait_until_some()

```
bool test_shmem_wait_until_some (
            void )
```

Tests the shmem_wait_until_some() routine.

This test verifies that the shmem_wait_until_some() function correctly waits until some of the specified conditions on an array of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 129 of file pt2pt_tests.cpp.

```
00129                                       {
00130   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00131   for (int i = 0; i < 4; ++i) {
00132     flags[i] = 0;
00133   }
00134   int mype = p_shmem_my_pe();
00135
00136   p_shmem_barrier_all();
00137
00138   if (mype == 0) {
00139     p_shmem_long_p(&flags[1], 1, 1);
00140     p_shmem_long_p(&flags[3], 1, 1);
00141     p_shmem_quiet();
00142   }
00143
00144   p_shmem_barrier_all();
00145
00146   if (mype != 0) {
00147     size_t indices[4];
00148     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00149     size_t count = p_shmem_long_wait_until_some(flags, 4, indices, status, SHMEM_CMP_EQ, 1);
00150     if (count < 2) {
00151       p_shmem_free(flags);
00152       return false;
00153     }
00154     for (size_t i = 0; i < count; ++i) {
00155       if (flags[indices[i]] != 1) {
00156         p_shmem_free(flags);
00157         return false;
00158       }
00159     }
00160   }
00161
00162   p_shmem_free(flags);
00163   return true;
00164 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_some, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.33.3.15 test_shmem_wait_until_some_vector()

```
bool test_shmem_wait_until_some_vector (
            void )
```

Tests the shmem_wait_until_some_vector() routine.

This test verifies that the shmem_wait_until_some_vector() function correctly waits until some of the specified conditions on a vector of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 446 of file pt2pt_tests.cpp.

```
00446                                                {
00447   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00448   for (int i = 0; i < 4; ++i) {
00449     flags[i] = 0;
00450   }
00451   int mype = p_shmem_my_pe();
00452
00453   p_shmem_barrier_all();
00454
00455   if (mype == 0) {
00456     p_shmem_long_p(&flags[1], 1, 1);
00457     p_shmem_long_p(&flags[3], 1, 1);
00458     p_shmem_quiet();
00459   }
00460
00461   p_shmem_barrier_all();
00462
00463   if (mype != 0) {
00464     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00465     long cmp_values[4] = {1, 1, 1, 1};
00466     size_t indices[4];
00467     size_t num_indices = p_shmem_long_wait_until_some_vector(flags, 4, indices, status, SHMEM_CMP_EQ,
    cmp_values);
00468     if (num_indices < 2) {
00469       p_shmem_free(flags);
00470       return false;
00471     }
00472     for (size_t i = 0; i < num_indices; ++i) {
00473       if (flags[indices[i]] != 1) {
00474         p_shmem_free(flags);
00475         return false;
00476       }
00477     }
00478   }
00479
00480   p_shmem_free(flags);
00481   return true;
00482 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_some_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.34 pt2pt_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "pt2pt_tests.hpp"
00007
00008 #define TIMEOUT 10
00009
00018 bool test_shmem_wait_until(void) {
00019   long *flag = (long *)p_shmem_malloc(sizeof(long));
00020   *flag = 0;
00021   int mype = p_shmem_my_pe();
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_long_p(flag, 1, 1);
00027     p_shmem_quiet();
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   if (mype != 0) {
00033     p_shmem_long_wait_until(flag, SHMEM_CMP_EQ, 1);
00034     if (*flag != 1) {
00035       p_shmem_free(flag);
00036       return false;
00037     }
00038   }
00039
00040   p_shmem_free(flag);
```

```
00041   return true;
00042 }
00043
00052 bool test_shmem_wait_until_all(void) {
00053   long *flags = (long *)p_shmem_malloc(2 * sizeof(long));
00054   flags[0] = 0;
00055   flags[1] = 0;
00056   int mype = p_shmem_my_pe();
00057
00058   p_shmem_barrier_all();
00059
00060   if (mype == 0) {
00061     p_shmem_long_p(&flags[0], 1, 1);
00062     p_shmem_long_p(&flags[1], 1, 1);
00063     p_shmem_quiet();
00064   }
00065
00066   p_shmem_barrier_all();
00067
00068   if (mype != 0) {
00069     p_shmem_long_wait_until_all(flags, 2, NULL, SHMEM_CMP_EQ, 1);
00070     if (flags[0] != 1 || flags[1] != 1) {
00071       p_shmem_free(flags);
00072       return false;
00073     }
00074   }
00075
00076   p_shmem_free(flags);
00077   return true;
00078 }
00079
00088 bool test_shmem_wait_until_any(void) {
00089   long *flags = (long *)p_shmem_malloc(3 * sizeof(long));
00090   for (int i = 0; i < 3; i++) {
00091     flags[i] = 0;
00092   }
00093   int mype = p_shmem_my_pe();
00094
00095   p_shmem_barrier_all();
00096
00097   if (mype == 0) {
00098     p_shmem_long_p(&flags[2], 1, 1);
00099     p_shmem_quiet();
00100   }
00101
00102   p_shmem_barrier_all();
00103
00104   if (mype != 0) {
00105     int status[3] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00106     size_t index = p_shmem_long_wait_until_any(flags, 3, status, SHMEM_CMP_EQ, 1);
00107     if (index == SIZE_MAX) {
00108       p_shmem_free(flags);
00109       return false;
00110     }
00111     if (flags[index] != 1) {
00112       p_shmem_free(flags);
00113       return false;
00114     }
00115   }
00116
00117   p_shmem_free(flags);
00118   return true;
00119 }
00120
00129 bool test_shmem_wait_until_some(void) {
00130   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00131   for (int i = 0; i < 4; ++i) {
00132     flags[i] = 0;
00133   }
00134   int mype = p_shmem_my_pe();
00135
00136   p_shmem_barrier_all();
00137
00138   if (mype == 0) {
00139     p_shmem_long_p(&flags[1], 1, 1);
00140     p_shmem_long_p(&flags[3], 1, 1);
00141     p_shmem_quiet();
00142   }
00143
00144   p_shmem_barrier_all();
00145
00146   if (mype != 0) {
00147     size_t indices[4];
00148     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00149     size_t count = p_shmem_long_wait_until_some(flags, 4, indices, status, SHMEM_CMP_EQ, 1);
00150     if (count < 2) {
00151       p_shmem_free(flags);
```

```
00152       return false;
00153     }
00154     for (size_t i = 0; i < count; ++i) {
00155       if (flags[indices[i]] != 1) {
00156         p_shmem_free(flags);
00157         return false;
00158       }
00159     }
00160   }
00161
00162   p_shmem_free(flags);
00163   return true;
00164 }
00165
00174 bool test_shmem_test(void) {
00175   long *flag = (long *)p_shmem_malloc(sizeof(long));
00176   if (flag == NULL) {
00177     return false;
00178   }
00179
00180   *flag = 0;
00181   int mype = p_shmem_my_pe();
00182
00183   p_shmem_barrier_all();
00184
00185   if (mype == 0) {
00186     *flag = 1;
00187     p_shmem_quiet();
00188   }
00189
00190   p_shmem_barrier_all();
00191
00192   if (mype != 0) {
00193     time_t start_time = time(NULL);
00194     while (!p_shmem_long_test(flag, SHMEM_CMP_EQ, 1)) {
00195       if (time(NULL) - start_time > TIMEOUT) {
00196         break;
00197       }
00198       usleep(1000);
00199     }
00200     if (*flag != 1) {
00201       p_shmem_free(flag);
00202       return false;
00203     }
00204   }
00205
00206   p_shmem_free(flag);
00207   return true;
00208 }
00209
00218 bool test_shmem_test_all(void) {
00219   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00220   if (flags == NULL) {
00221     return false;
00222   }
00223
00224   for (int i = 0; i < 4; ++i) {
00225     flags[i] = 0;
00226   }
00227   int mype = p_shmem_my_pe();
00228
00229   p_shmem_barrier_all();
00230
00231   if (mype == 0) {
00232     for (int i = 0; i < 4; ++i) {
00233       flags[i] = 1;
00234     }
00235     p_shmem_quiet();
00236   }
00237
00238   p_shmem_barrier_all();
00239
00240   if (mype != 0) {
00241     time_t start_time = time(NULL);
00242     while (!p_shmem_long_test_all(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00243       if (time(NULL) - start_time > TIMEOUT) {
00244         break;
00245       }
00246       usleep(1000);
00247     }
00248     for (int i = 0; i < 4; ++i) {
00249       if (flags[i] != 1) {
00250         p_shmem_free(flags);
00251         return false;
00252       }
00253     }
00254   }
```

```
00255
00256   p_shmem_free(flags);
00257   return true;
00258 }
00259
00268 bool test_shmem_test_any(void) {
00269   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00270   if (flags == NULL) {
00271     return false;
00272   }
00273
00274   for (int i = 0; i < 4; ++i) {
00275     flags[i] = 0;
00276   }
00277   int mype = p_shmem_my_pe();
00278
00279   p_shmem_barrier_all();
00280
00281   if (mype == 0) {
00282     flags[2] = 1;
00283     p_shmem_quiet();
00284   }
00285
00286   p_shmem_barrier_all();
00287
00288   if (mype != 0) {
00289     time_t start_time = time(NULL);
00290     while (!p_shmem_long_test_any(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00291       if (time(NULL) - start_time > TIMEOUT) {
00292         break;
00293       }
00294       usleep(1000);
00295     }
00296     if (flags[2] != 1) {
00297       p_shmem_free(flags);
00298       return false;
00299     }
00300   }
00301
00302   p_shmem_free(flags);
00303   return true;
00304 }
00305
00314 bool test_shmem_test_some(void) {
00315   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00316   if (flags == NULL) {
00317     return false;
00318   }
00319
00320   for (int i = 0; i < 4; ++i) {
00321     flags[i] = 0;
00322   }
00323   int mype = p_shmem_my_pe();
00324
00325   p_shmem_barrier_all();
00326
00327   if (mype == 0) {
00328     flags[1] = 1;
00329     flags[3] = 1;
00330     p_shmem_quiet();
00331   }
00332
00333   p_shmem_barrier_all();
00334
00335   if (mype != 0) {
00336     size_t indices[4];
00337     time_t start_time = time(NULL);
00338     while (!p_shmem_long_test_some(flags, 4, indices, NULL, SHMEM_CMP_EQ, 1)) {
00339       if (time(NULL) - start_time > TIMEOUT) {
00340         break;
00341       }
00342       usleep(1000);
00343     }
00344     if (flags[1] != 1 || flags[3] != 1) {
00345       p_shmem_free(flags);
00346       return false;
00347     }
00348   }
00349
00350   p_shmem_free(flags);
00351   return true;
00352 }
00353
00362 bool test_shmem_wait_until_all_vector(void) {
00363   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00364   for (int i = 0; i < 4; ++i) {
00365     flags[i] = 0;
```

```
00366   }
00367   int mype = p_shmem_my_pe();
00368
00369   p_shmem_barrier_all();
00370
00371   if (mype == 0) {
00372     for (int i = 0; i < 4; ++i) {
00373       p_shmem_long_p(&flags[i], 1, 1);
00374       p_shmem_quiet();
00375     }
00376   }
00377
00378   p_shmem_barrier_all();
00379
00380   if (mype != 0) {
00381     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00382     long cmp_values[4] = {1, 1, 1, 1};
00383     p_shmem_long_wait_until_all_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00384     for (int i = 0; i < 4; ++i) {
00385       if (flags[i] != 1) {
00386         p_shmem_free(flags);
00387         return false;
00388       }
00389     }
00390   }
00391
00392   p_shmem_free(flags);
00393   return true;
00394 }
00395
00404 bool test_shmem_wait_until_any_vector(void) {
00405   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00406   for (int i = 0; i < 4; ++i) {
00407     flags[i] = 0;
00408   }
00409   int mype = p_shmem_my_pe();
00410
00411   p_shmem_barrier_all();
00412
00413   if (mype == 0) {
00414     p_shmem_long_p(&flags[2], 1, 1);
00415     p_shmem_quiet();
00416   }
00417
00418   p_shmem_barrier_all();
00419
00420   if (mype != 0) {
00421     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00422     long cmp_values[4] = {1, 1, 1, 1};
00423     size_t index = p_shmem_long_wait_until_any_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00424     if (index == SIZE_MAX) {
00425       p_shmem_free(flags);
00426       return false;
00427     }
00428     if (flags[index] != 1) {
00429       p_shmem_free(flags);
00430       return false;
00431     }
00432   }
00433
00434   p_shmem_free(flags);
00435   return true;
00436 }
00437
00446 bool test_shmem_wait_until_some_vector(void) {
00447   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00448   for (int i = 0; i < 4; ++i) {
00449     flags[i] = 0;
00450   }
00451   int mype = p_shmem_my_pe();
00452
00453   p_shmem_barrier_all();
00454
00455   if (mype == 0) {
00456     p_shmem_long_p(&flags[1], 1, 1);
00457     p_shmem_long_p(&flags[3], 1, 1);
00458     p_shmem_quiet();
00459   }
00460
00461   p_shmem_barrier_all();
00462
00463   if (mype != 0) {
00464     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00465     long cmp_values[4] = {1, 1, 1, 1};
00466     size_t indices[4];
00467     size_t num_indices = p_shmem_long_wait_until_some_vector(flags, 4, indices, status, SHMEM_CMP_EQ,
      cmp_values);
```

```
00468     if (num_indices < 2) {
00469       p_shmem_free(flags);
00470       return false;
00471     }
00472     for (size_t i = 0; i < num_indices; ++i) {
00473       if (flags[indices[i]] != 1) {
00474         p_shmem_free(flags);
00475         return false;
00476       }
00477     }
00478   }
00479
00480   p_shmem_free(flags);
00481   return true;
00482 }
00483
00492 bool test_shmem_test_all_vector(void) {
00493   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00494   if (flags == NULL) {
00495     return false;
00496   }
00497
00498   for (int i = 0; i < 4; ++i) {
00499     flags[i] = 0;
00500   }
00501   int mype = p_shmem_my_pe();
00502
00503   p_shmem_barrier_all();
00504
00505   if (mype == 0) {
00506     for (int i = 0; i < 4; ++i) {
00507       flags[i] = 1;
00508     }
00509     p_shmem_quiet();
00510   }
00511
00512   p_shmem_barrier_all();
00513
00514   if (mype != 0) {
00515     long cmp_values[4] = {1, 1, 1, 1};
00516     time_t start_time = time(NULL);
00517     while (!p_shmem_long_test_all_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
00518       if (time(NULL) - start_time > TIMEOUT) {
00519         break;
00520       }
00521       usleep(1000);
00522     }
00523     for (int i = 0; i < 4; ++i) {
00524       if (flags[i] != 1) {
00525         p_shmem_free(flags);
00526         return false;
00527       }
00528     }
00529   }
00530
00531   p_shmem_free(flags);
00532   return true;
00533 }
00534
00543 bool test_shmem_test_any_vector(void) {
00544   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00545   if (flags == NULL) {
00546     return false;
00547   }
00548
00549   for (int i = 0; i < 4; ++i) {
00550     flags[i] = 0;
00551   }
00552   int mype = p_shmem_my_pe();
00553   int npes = p_shmem_n_pes();
00554
00555   p_shmem_barrier_all();
00556
00557   if (mype == 0) {
00558     long one = 1;
00559     for (int pe = 1; pe < npes; ++pe) {
00560       p_shmem_long_p(&flags[2], one, pe);
00561     }
00562     p_shmem_quiet();
00563   }
00564
00565   p_shmem_barrier_all();
00566
00567   if (mype != 0) {
00568     long cmp_values[4] = {1, 1, 1, 1};
00569     time_t start_time = time(NULL);
00570     while (!p_shmem_long_test_any_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
```

```
00571       if (time(NULL) - start_time > TIMEOUT) {
00572         break;
00573       }
00574       usleep(1000);
00575     }
00576     if (flags[2] != 1) {
00577       p_shmem_free(flags);
00578       return false;
00579     }
00580   }
00581
00582   p_shmem_free(flags);
00583   return true;
00584 }
00585
00594 bool test_shmem_test_some_vector(void) {
00595   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00596   if (flags == NULL) {
00597     return false;
00598   }
00599
00600   for (int i = 0; i < 4; ++i) {
00601     flags[i] = 0;
00602   }
00603   int mype = p_shmem_my_pe();
00604   int npes = p_shmem_n_pes();
00605
00606   p_shmem_barrier_all();
00607
00608   if (mype == 0) {
00609     long one = 1;
00610     for (int pe = 1; pe < npes; ++pe) {
00611       p_shmem_long_p(&flags[1], one, pe);
00612       p_shmem_long_p(&flags[3], one, pe);
00613     }
00614     p_shmem_quiet();
00615   }
00616
00617   p_shmem_barrier_all();
00618
00619   if (mype != 0) {
00620     long cmp_values[4] = {0, 1, 0, 1};
00621     size_t indices[4];
00622     size_t num_indices;
00623     time_t start_time = time(NULL);
00624     while ((num_indices = p_shmem_long_test_some_vector(flags, 4, indices, NULL, SHMEM_CMP_EQ,
      cmp_values)) == 0) {
00625       if (time(NULL) - start_time > TIMEOUT) {
00626         break;
00627       }
00628       usleep(1000);
00629     }
00630     if (flags[1] != 1 || flags[3] != 1) {
00631       p_shmem_free(flags);
00632       return false;
00633     }
00634   }
00635
00636   p_shmem_free(flags);
00637   return true;
00638 }
00639
00648 bool test_shmem_signal_wait_until(void) {
00649   uint64_t *flag = (uint64_t *)p_shmem_malloc(sizeof(uint64_t));
00650   if (flag == NULL) {
00651     return false;
00652   }
00653
00654   *flag = 0;
00655   int mype = p_shmem_my_pe();
00656   uint64_t value = 1;
00657
00658   p_shmem_barrier_all();
00659
00660   if (mype == 0) {
00661     shmem_uint64_p(flag, value, 1);
00662     p_shmem_quiet();
00663   }
00664
00665   p_shmem_barrier_all();
00666
00667   if (mype != 0) {
00668     time_t start_time = time(NULL);
00669     while (*flag != value && time(NULL) - start_time < TIMEOUT) {
00670       p_shmem_signal_wait_until(flag, SHMEM_CMP_EQ, value);
00671     }
00672     if (*flag != value) {
```

```
00673       p_shmem_free(flag);
00674       return false;
00675     }
00676   }
00677
00678   p_shmem_free(flag);
00679   return true;
00680 }
```

4.35 src/tests/pt2pt/pt2pt_tests.hpp File Reference

Contains function declarations for the point-to-point synchronization routines tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
#include <stdio.h>
#include <stdbool.h>
#include <ctime>
#include <unistd.h>
```

Functions

- bool test_shmem_wait_until (void)

  Tests the shmem_wait_until() routine.
- bool test_shmem_wait_until_all (void)

  Tests the shmem_wait_until_all() routine.
- bool test_shmem_wait_until_any (void)

  Tests the shmem_wait_until_any() routine.
- bool test_shmem_wait_until_some (void)

  Tests the shmem_wait_until_some() routine.
- bool test_shmem_wait_until_all_vector (void)

  Tests the shmem_wait_until_all_vector() routine.
- bool test_shmem_wait_until_any_vector (void)

  Tests the shmem_wait_until_any_vector() routine.
- bool test_shmem_wait_until_some_vector (void)

  Tests the shmem_wait_until_some_vector() routine.
- bool test_shmem_test (void)

  Tests the shmem_test() routine.
- bool test_shmem_test_all (void)

  Tests the shmem_test_all() routine.
- bool test_shmem_test_any (void)

  Tests the shmem_test_any() routine.
- bool test_shmem_test_some (void)

  Tests the shmem_test_some() routine.
- bool test_shmem_test_all_vector (void)

  Tests the shmem_test_all_vector() routine.
- bool test_shmem_test_any_vector (void)

  Tests the shmem_test_any_vector() routine.
- bool test_shmem_test_some_vector (void)

  Tests the shmem_test_some_vector() routine.
- bool test_shmem_signal_wait_until (void)

  Tests the shmem_signal_wait_until() routine.

4.35.1 Detailed Description

Contains function declarations for the point-to-point synchronization routines tests.

Definition in file pt2pt_tests.hpp.

4.35.2 Function Documentation

4.35.2.1 test_shmem_signal_wait_until()

```
bool test_shmem_signal_wait_until (
            void )
```

Tests the shmem_signal_wait_until() routine.

This test verifies that the shmem_signal_wait_until() function correctly waits until a signal on a memory location meets a specified condition.

Returns

True if the test is successful, false otherwise.

Definition at line 648 of file pt2pt_tests.cpp.

```
00648                                           {
00649   uint64_t *flag = (uint64_t *)p_shmem_malloc(sizeof(uint64_t));
00650   if (flag == NULL) {
00651     return false;
00652   }
00653
00654   *flag = 0;
00655   int mype = p_shmem_my_pe();
00656   uint64_t value = 1;
00657
00658   p_shmem_barrier_all();
00659
00660   if (mype == 0) {
00661     shmem_uint64_p(flag, value, 1);
00662     p_shmem_quiet();
00663   }
00664
00665   p_shmem_barrier_all();
00666
00667   if (mype != 0) {
00668     time_t start_time = time(NULL);
00669     while (*flag != value && time(NULL) - start_time < TIMEOUT) {
00670       p_shmem_signal_wait_until(flag, SHMEM_CMP_EQ, value);
00671     }
00672     if (*flag != value) {
00673       p_shmem_free(flag);
00674       return false;
00675     }
00676   }
00677
00678   p_shmem_free(flag);
00679   return true;
00680 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, p_shmem_signal_wait_until, and TIMEOUT.

4.35.2.2 test_shmem_test()

```
bool test_shmem_test (
            void )
```

Tests the shmem_test() routine.

This test verifies that the shmem_test() function correctly tests whether a condition on a memory location is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 174 of file pt2pt_tests.cpp.

```
00174                                {
00175   long *flag = (long *)p_shmem_malloc(sizeof(long));
00176   if (flag == NULL) {
00177     return false;
00178   }
00179
00180   *flag = 0;
00181   int mype = p_shmem_my_pe();
00182
00183   p_shmem_barrier_all();
00184
00185   if (mype == 0) {
00186     *flag = 1;
00187     p_shmem_quiet();
00188   }
00189
00190   p_shmem_barrier_all();
00191
00192   if (mype != 0) {
00193     time_t start_time = time(NULL);
00194     while (!p_shmem_long_test(flag, SHMEM_CMP_EQ, 1)) {
00195       if (time(NULL) - start_time > TIMEOUT) {
00196         break;
00197       }
00198       usleep(1000);
00199     }
00200     if (*flag != 1) {
00201       p_shmem_free(flag);
00202       return false;
00203     }
00204   }
00205
00206   p_shmem_free(flag);
00207   return true;
00208 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.35.2.3 test_shmem_test_all()

```
bool test_shmem_test_all (
            void )
```

Tests the shmem_test_all() routine.

This test verifies that the shmem_test_all() function correctly tests whether all specified conditions on an array of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 218 of file pt2pt_tests.cpp.

```
00218                                    {
00219   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00220   if (flags == NULL) {
00221     return false;
00222   }
00223
00224   for (int i = 0; i < 4; ++i) {
00225     flags[i] = 0;
00226   }
00227   int mype = p_shmem_my_pe();
00228
00229   p_shmem_barrier_all();
00230
00231   if (mype == 0) {
00232     for (int i = 0; i < 4; ++i) {
00233       flags[i] = 1;
00234     }
00235     p_shmem_quiet();
00236   }
00237
00238   p_shmem_barrier_all();
00239
00240   if (mype != 0) {
00241     time_t start_time = time(NULL);
00242     while (!p_shmem_long_test_all(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00243       if (time(NULL) - start_time > TIMEOUT) {
00244         break;
00245       }
00246       usleep(1000);
00247     }
00248     for (int i = 0; i < 4; ++i) {
00249       if (flags[i] != 1) {
00250         p_shmem_free(flags);
00251         return false;
00252       }
00253     }
00254   }
00255
00256   p_shmem_free(flags);
00257   return true;
00258 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_all, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.35.2.4 test_shmem_test_all_vector()

```
bool test_shmem_test_all_vector (
             void )
```

Tests the shmem_test_all_vector() routine.

This test verifies that the shmem_test_all_vector() function correctly tests whether all specified conditions on a vector of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 492 of file pt2pt_tests.cpp.

```
00492                                          {
00493   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00494   if (flags == NULL) {
00495     return false;
00496   }
00497
00498   for (int i = 0; i < 4; ++i) {
```

```
00499     flags[i] = 0;
00500   }
00501   int mype = p_shmem_my_pe();
00502
00503   p_shmem_barrier_all();
00504
00505   if (mype == 0) {
00506     for (int i = 0; i < 4; ++i) {
00507       flags[i] = 1;
00508     }
00509     p_shmem_quiet();
00510   }
00511
00512   p_shmem_barrier_all();
00513
00514   if (mype != 0) {
00515     long cmp_values[4] = {1, 1, 1, 1};
00516     time_t start_time = time(NULL);
00517     while (!p_shmem_long_test_all_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
00518       if (time(NULL) - start_time > TIMEOUT) {
00519         break;
00520       }
00521       usleep(1000);
00522     }
00523     for (int i = 0; i < 4; ++i) {
00524       if (flags[i] != 1) {
00525         p_shmem_free(flags);
00526         return false;
00527       }
00528     }
00529   }
00530
00531   p_shmem_free(flags);
00532   return true;
00533 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_all_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.35.2.5 test_shmem_test_any()

```
bool test_shmem_test_any (
             void )
```

Tests the shmem_test_any() routine.

This test verifies that the shmem_test_any() function correctly tests whether any one of the specified conditions on an array of memory locations is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 268 of file pt2pt_tests.cpp.

```
00268                                {
00269   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00270   if (flags == NULL) {
00271     return false;
00272   }
00273
00274   for (int i = 0; i < 4; ++i) {
00275     flags[i] = 0;
00276   }
00277   int mype = p_shmem_my_pe();
00278
00279   p_shmem_barrier_all();
00280
00281   if (mype == 0) {
00282     flags[2] = 1;
00283     p_shmem_quiet();
00284   }
00285
00286   p_shmem_barrier_all();
00287
```

```
00288   if (mype != 0) {
00289     time_t start_time = time(NULL);
00290     while (!p_shmem_long_test_any(flags, 4, NULL, SHMEM_CMP_EQ, 1)) {
00291       if (time(NULL) - start_time > TIMEOUT) {
00292         break;
00293       }
00294       usleep(1000);
00295     }
00296     if (flags[2] != 1) {
00297       p_shmem_free(flags);
00298       return false;
00299     }
00300   }
00301
00302   p_shmem_free(flags);
00303   return true;
00304 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_any, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.35.2.6 test_shmem_test_any_vector()

```
bool test_shmem_test_any_vector (
             void )
```

Tests the shmem_test_any_vector() routine.

This test verifies that the shmem_test_any_vector() function correctly tests whether any one of the specified conditions on a vector of memory locations is met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 543 of file pt2pt_tests.cpp.

```
00543                                       {
00544   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00545   if (flags == NULL) {
00546     return false;
00547   }
00548
00549   for (int i = 0; i < 4; ++i) {
00550     flags[i] = 0;
00551   }
00552   int mype = p_shmem_my_pe();
00553   int npes = p_shmem_n_pes();
00554
00555   p_shmem_barrier_all();
00556
00557   if (mype == 0) {
00558     long one = 1;
00559     for (int pe = 1; pe < npes; ++pe) {
00560       p_shmem_long_p(&flags[2], one, pe);
00561     }
00562     p_shmem_quiet();
00563   }
00564
00565   p_shmem_barrier_all();
00566
00567   if (mype != 0) {
00568     long cmp_values[4] = {1, 1, 1, 1};
00569     time_t start_time = time(NULL);
00570     while (!p_shmem_long_test_any_vector(flags, 4, NULL, SHMEM_CMP_EQ, cmp_values)) {
00571       if (time(NULL) - start_time > TIMEOUT) {
00572         break;
00573       }
00574       usleep(1000);
00575     }
00576     if (flags[2] != 1) {
00577       p_shmem_free(flags);
00578       return false;
00579     }
00580   }
00581
00582   p_shmem_free(flags);
00583   return true;
00584 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_test_any_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, p_shmem_quiet, and TIMEOUT.

4.35.2.7 test_shmem_test_some()

```
bool test_shmem_test_some (
            void )
```

Tests the shmem_test_some() routine.

This test verifies that the shmem_test_some() function correctly tests whether some of the specified conditions on an array of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 314 of file pt2pt_tests.cpp.

```
00314                                {
00315   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00316   if (flags == NULL) {
00317     return false;
00318   }
00319
00320   for (int i = 0; i < 4; ++i) {
00321     flags[i] = 0;
00322   }
00323   int mype = p_shmem_my_pe();
00324
00325   p_shmem_barrier_all();
00326
00327   if (mype == 0) {
00328     flags[1] = 1;
00329     flags[3] = 1;
00330     p_shmem_quiet();
00331   }
00332
00333   p_shmem_barrier_all();
00334
00335   if (mype != 0) {
00336     size_t indices[4];
00337     time_t start_time = time(NULL);
00338     while (!p_shmem_long_test_some(flags, 4, indices, NULL, SHMEM_CMP_EQ, 1)) {
00339       if (time(NULL) - start_time > TIMEOUT) {
00340         break;
00341       }
00342       usleep(1000);
00343     }
00344     if (flags[1] != 1 || flags[3] != 1) {
00345       p_shmem_free(flags);
00346       return false;
00347     }
00348   }
00349
00350   p_shmem_free(flags);
00351   return true;
00352 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_test_some, p_shmem_malloc, p_shmem_my_pe, p_shmem_quiet, and TIMEOUT.

4.35.2.8 test_shmem_test_some_vector()

```
bool test_shmem_test_some_vector (
            void )
```

Tests the shmem_test_some_vector() routine.

This test verifies that the shmem_test_some_vector() function correctly tests whether some of the specified conditions on a vector of memory locations are met without blocking.

Returns

True if the test is successful, false otherwise.

Definition at line 594 of file pt2pt_tests.cpp.

```
00594                                       {
00595   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00596   if (flags == NULL) {
00597     return false;
00598   }
00599
00600   for (int i = 0; i < 4; ++i) {
00601     flags[i] = 0;
00602   }
00603   int mype = p_shmem_my_pe();
00604   int npes = p_shmem_n_pes();
00605
00606   p_shmem_barrier_all();
00607
00608   if (mype == 0) {
00609     long one = 1;
00610     for (int pe = 1; pe < npes; ++pe) {
00611       p_shmem_long_p(&flags[1], one, pe);
00612       p_shmem_long_p(&flags[3], one, pe);
00613     }
00614     p_shmem_quiet();
00615   }
00616
00617   p_shmem_barrier_all();
00618
00619   if (mype != 0) {
00620     long cmp_values[4] = {0, 1, 0, 1};
00621     size_t indices[4];
00622     size_t num_indices;
00623     time_t start_time = time(NULL);
00624     while ((num_indices = p_shmem_long_test_some_vector(flags, 4, indices, NULL, SHMEM_CMP_EQ,
      cmp_values)) == 0) {
00625       if (time(NULL) - start_time > TIMEOUT) {
00626         break;
00627       }
00628       usleep(1000);
00629     }
00630     if (flags[1] != 1 || flags[3] != 1) {
00631       p_shmem_free(flags);
00632       return false;
00633     }
00634   }
00635
00636   p_shmem_free(flags);
00637   return true;
00638 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_test_some_vector, p_shmem_malloc, p_shmem_my_pe, p_shmem_n_pes, p_shmem_quiet, and TIMEOUT.

4.35.2.9 test_shmem_wait_until()

```
bool test_shmem_wait_until (
            void )
```

Tests the shmem_wait_until() routine.

This test verifies that the shmem_wait_until() function correctly waits until a condition on a memory location is met.

Returns

True if the test is successful, false otherwise.

Definition at line 18 of file pt2pt_tests.cpp.

```
00018                                {
00019   long *flag = (long *)p_shmem_malloc(sizeof(long));
00020   *flag = 0;
00021   int mype = p_shmem_my_pe();
00022
00023   p_shmem_barrier_all();
00024
00025   if (mype == 0) {
00026     p_shmem_long_p(flag, 1, 1);
00027     p_shmem_quiet();
00028   }
00029
00030   p_shmem_barrier_all();
00031
00032   if (mype != 0) {
00033     p_shmem_long_wait_until(flag, SHMEM_CMP_EQ, 1);
00034     if (*flag != 1) {
00035       p_shmem_free(flag);
00036       return false;
00037     }
00038   }
00039
00040   p_shmem_free(flag);
00041   return true;
00042 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.10 test_shmem_wait_until_all()

```
bool test_shmem_wait_until_all (
             void )
```

Tests the shmem_wait_until_all() routine.

This test verifies that the shmem_wait_until_all() function correctly waits until all specified conditions on an array of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file pt2pt_tests.cpp.

```
00052                                      {
00053   long *flags = (long *)p_shmem_malloc(2 * sizeof(long));
00054   flags[0] = 0;
00055   flags[1] = 0;
00056   int mype = p_shmem_my_pe();
00057
00058   p_shmem_barrier_all();
00059
00060   if (mype == 0) {
00061     p_shmem_long_p(&flags[0], 1, 1);
00062     p_shmem_long_p(&flags[1], 1, 1);
00063     p_shmem_quiet();
00064   }
00065
00066   p_shmem_barrier_all();
00067
00068   if (mype != 0) {
00069     p_shmem_long_wait_until_all(flags, 2, NULL, SHMEM_CMP_EQ, 1);
00070     if (flags[0] != 1 || flags[1] != 1) {
00071       p_shmem_free(flags);
00072       return false;
00073     }
00074   }
00075
00076   p_shmem_free(flags);
00077   return true;
00078 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_all, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.11 test_shmem_wait_until_all_vector()

```
bool test_shmem_wait_until_all_vector (
            void )
```

Tests the shmem_wait_until_all_vector() routine.

This test verifies that the shmem_wait_until_all_vector() function correctly waits until all specified conditions on a vector of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 362 of file pt2pt_tests.cpp.

```
00362                                                {
00363   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00364   for (int i = 0; i < 4; ++i) {
00365     flags[i] = 0;
00366   }
00367   int mype = p_shmem_my_pe();
00368
00369   p_shmem_barrier_all();
00370
00371   if (mype == 0) {
00372     for (int i = 0; i < 4; ++i) {
00373       p_shmem_long_p(&flags[i], 1, 1);
00374       p_shmem_quiet();
00375     }
00376   }
00377
00378   p_shmem_barrier_all();
00379
00380   if (mype != 0) {
00381     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00382     long cmp_values[4] = {1, 1, 1, 1};
00383     p_shmem_long_wait_until_all_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00384     for (int i = 0; i < 4; ++i) {
00385       if (flags[i] != 1) {
00386         p_shmem_free(flags);
00387         return false;
00388       }
00389     }
00390   }
00391
00392   p_shmem_free(flags);
00393   return true;
00394 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_all_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.12 test_shmem_wait_until_any()

```
bool test_shmem_wait_until_any (
            void )
```

Tests the shmem_wait_until_any() routine.

This test verifies that the shmem_wait_until_any() function correctly waits until any one of the specified conditions on an array of memory locations is met.

Returns

True if the test is successful, false otherwise.

Definition at line 88 of file pt2pt_tests.cpp.

```
00088                                       {
00089   long *flags = (long *)p_shmem_malloc(3 * sizeof(long));
00090   for (int i = 0; i < 3; i++) {
00091     flags[i] = 0;
00092   }
00093   int mype = p_shmem_my_pe();
00094
00095   p_shmem_barrier_all();
00096
00097   if (mype == 0) {
00098     p_shmem_long_p(&flags[2], 1, 1);
00099     p_shmem_quiet();
00100   }
00101
00102   p_shmem_barrier_all();
00103
00104   if (mype != 0) {
00105     int status[3] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00106     size_t index = p_shmem_long_wait_until_any(flags, 3, status, SHMEM_CMP_EQ, 1);
00107     if (index == SIZE_MAX) {
00108       p_shmem_free(flags);
00109       return false;
00110     }
00111     if (flags[index] != 1) {
00112       p_shmem_free(flags);
00113       return false;
00114     }
00115   }
00116
00117   p_shmem_free(flags);
00118   return true;
00119 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_any, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.13 test_shmem_wait_until_any_vector()

```
bool test_shmem_wait_until_any_vector (
            void )
```

Tests the shmem_wait_until_any_vector() routine.

This test verifies that the shmem_wait_until_any_vector() function correctly waits until any one of the specified conditions on a vector of memory locations is met.

Returns

True if the test is successful, false otherwise.

Definition at line 404 of file pt2pt_tests.cpp.

```
00404                                       {
00405   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00406   for (int i = 0; i < 4; ++i) {
00407     flags[i] = 0;
00408   }
00409   int mype = p_shmem_my_pe();
00410
00411   p_shmem_barrier_all();
00412
00413   if (mype == 0) {
00414     p_shmem_long_p(&flags[2], 1, 1);
00415     p_shmem_quiet();
00416   }
00417
00418   p_shmem_barrier_all();
00419
```

```
00420   if (mype != 0) {
00421     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00422     long cmp_values[4] = {1, 1, 1, 1};
00423     size_t index = p_shmem_long_wait_until_any_vector(flags, 4, status, SHMEM_CMP_EQ, cmp_values);
00424     if (index == SIZE_MAX) {
00425       p_shmem_free(flags);
00426       return false;
00427     }
00428     if (flags[index] != 1) {
00429       p_shmem_free(flags);
00430       return false;
00431     }
00432   }
00433
00434   p_shmem_free(flags);
00435   return true;
00436 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_any_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.14 test_shmem_wait_until_some()

```
bool test_shmem_wait_until_some (
             void )
```

Tests the shmem_wait_until_some() routine.

This test verifies that the shmem_wait_until_some() function correctly waits until some of the specified conditions on an array of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 129 of file pt2pt_tests.cpp.

```
00129                                       {
00130   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00131   for (int i = 0; i < 4; ++i) {
00132     flags[i] = 0;
00133   }
00134   int mype = p_shmem_my_pe();
00135
00136   p_shmem_barrier_all();
00137
00138   if (mype == 0) {
00139     p_shmem_long_p(&flags[1], 1, 1);
00140     p_shmem_long_p(&flags[3], 1, 1);
00141     p_shmem_quiet();
00142   }
00143
00144   p_shmem_barrier_all();
00145
00146   if (mype != 0) {
00147     size_t indices[4];
00148     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00149     size_t count = p_shmem_long_wait_until_some(flags, 4, indices, status, SHMEM_CMP_EQ, 1);
00150     if (count < 2) {
00151       p_shmem_free(flags);
00152       return false;
00153     }
00154     for (size_t i = 0; i < count; ++i) {
00155       if (flags[indices[i]] != 1) {
00156         p_shmem_free(flags);
00157         return false;
00158       }
00159     }
00160   }
00161
00162   p_shmem_free(flags);
00163   return true;
00164 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_some, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.35.2.15 test_shmem_wait_until_some_vector()

```
bool test_shmem_wait_until_some_vector (
            void )
```

Tests the shmem_wait_until_some_vector() routine.

This test verifies that the shmem_wait_until_some_vector() function correctly waits until some of the specified conditions on a vector of memory locations are met.

Returns

True if the test is successful, false otherwise.

Definition at line 446 of file pt2pt_tests.cpp.

```
00446                                           {
00447   long *flags = (long *)p_shmem_malloc(4 * sizeof(long));
00448   for (int i = 0; i < 4; ++i) {
00449     flags[i] = 0;
00450   }
00451   int mype = p_shmem_my_pe();
00452
00453   p_shmem_barrier_all();
00454
00455   if (mype == 0) {
00456     p_shmem_long_p(&flags[1], 1, 1);
00457     p_shmem_long_p(&flags[3], 1, 1);
00458     p_shmem_quiet();
00459   }
00460
00461   p_shmem_barrier_all();
00462
00463   if (mype != 0) {
00464     int status[4] = {SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ, SHMEM_CMP_EQ};
00465     long cmp_values[4] = {1, 1, 1, 1};
00466     size_t indices[4];
00467     size_t num_indices = p_shmem_long_wait_until_some_vector(flags, 4, indices, status, SHMEM_CMP_EQ,
      cmp_values);
00468     if (num_indices < 2) {
00469       p_shmem_free(flags);
00470       return false;
00471     }
00472     for (size_t i = 0; i < num_indices; ++i) {
00473       if (flags[indices[i]] != 1) {
00474         p_shmem_free(flags);
00475         return false;
00476       }
00477     }
00478   }
00479
00480   p_shmem_free(flags);
00481   return true;
00482 }
```

References p_shmem_barrier_all, p_shmem_free, p_shmem_long_p, p_shmem_long_wait_until_some_vector, p_shmem_malloc, p_shmem_my_pe, and p_shmem_quiet.

4.36 pt2pt_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef PT2PT_TESTS_HPP
00007 #define PT2PT_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012 #include <stdio.h>
00013 #include <stdbool.h>
00014 #include <ctime>
00015 #include <unistd.h>
```

```
00016
00017 /* Function Prototypes */
00026 bool test_shmem_wait_until(void);
00027
00036 bool test_shmem_wait_until_all(void);
00037
00046 bool test_shmem_wait_until_any(void);
00047
00056 bool test_shmem_wait_until_some(void);
00057
00066 bool test_shmem_wait_until_all_vector(void);
00067
00076 bool test_shmem_wait_until_any_vector(void);
00077
00086 bool test_shmem_wait_until_some_vector(void);
00087
00096 bool test_shmem_test(void);
00097
00106 bool test_shmem_test_all(void);
00107
00116 bool test_shmem_test_any(void);
00117
00126 bool test_shmem_test_some(void);
00127
00136 bool test_shmem_test_all_vector(void);
00137
00146 bool test_shmem_test_any_vector(void);
00147
00156 bool test_shmem_test_some_vector(void);
00157
00166 bool test_shmem_signal_wait_until(void);
00167
00168 #endif /* PT2PT_TESTS_HPP */
```

4.37 src/tests/remote/remote_tests.cpp File Reference

Contains OpenSHMEM remote memory access tests.

```
#include "remote_tests.hpp"
```

Functions

- bool test_shmem_put (void)

  Tests the shmem_put() function.
- bool test_shmem_p (void)

  Tests the shmem_p() function.
- bool test_shmem_iput (void)

  Tests the shmem_iput() function.
- bool test_shmem_get (void)

  Tests the shmem_get() function.
- bool test_shmem_g (void)

  Tests the shmem_g() function.
- bool test_shmem_iget (void)

  Tests the shmem_iget() function.
- bool test_shmem_put_nbi (void)

  Tests the shmem_put_nbi() function.
- bool test_shmem_get_nbi (void)

  Tests the shmem_get_nbi() function.

4.37.1 Detailed Description

Contains OpenSHMEM remote memory access tests.

Definition in file remote_tests.cpp.

4.37.2 Function Documentation

4.37.2.1 test_shmem_g()

```
bool test_shmem_g (
             void )
```

Tests the shmem_g() function.

Tests the shmem_g() routine.

This test verifies that the shmem_g() function correctly retrieves a single data element from PE 0 to PE 1.

PE 1 gets a single data element from PE 0.

Returns

True if the test is successful, false otherwise.

Definition at line 158 of file remote_tests.cpp.

```
00158                           {
00159   static long src, dest;
00160   int mype = p_shmem_my_pe();
00161   int npes = p_shmem_n_pes();
00162
00163   if (mype == 0) {
00164     src = 10;
00165   }
00166
00167   p_shmem_barrier_all();
00168
00169   if (mype == 1) {
00170     dest = p_shmem_long_g(&src, 0);
00171   }
00172
00173   p_shmem_barrier_all();
00174
00175   if (mype == 1) {
00176     if (dest != 10) {
00177       return false;
00178     }
00179   }
00180
00181   return true;
00182 }
```

References p_shmem_barrier_all, p_shmem_long_g, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.2 test_shmem_get()

```
bool test_shmem_get (
             void )
```

Tests the shmem_get() function.

Tests the shmem_get() routine.

This test verifies that the shmem_get() function correctly retrieves data from PE 0 to PE 1.

PE 1 gets data from an array on PE 0.

Returns

True if the test is successful, false otherwise.

Definition at line 120 of file remote_tests.cpp.

```
00120                                {
00121   static long src[10], dest[10];
00122   int mype = p_shmem_my_pe();
00123   int npes = p_shmem_n_pes();
00124
00125   if (mype == 0) {
00126     for (int i = 0; i < 10; i++) {
00127       src[i] = i;
00128     }
00129   }
00130
00131   p_shmem_barrier_all();
00132
00133   if (mype == 1) {
00134     p_shmem_long_get(dest, src, 10, 0);
00135   }
00136
00137   p_shmem_barrier_all();
00138
00139   if (mype == 1) {
00140     for (int i = 0; i < 10; i++) {
00141       if (dest[i] != i) {
00142         return false;
00143       }
00144     }
00145   }
00146
00147   return true;
00148 }
```

References p_shmem_barrier_all, p_shmem_long_get, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.3 test_shmem_get_nbi()

```
bool test_shmem_get_nbi (
             void )
```

Tests the shmem_get_nbi() function.

Tests the shmem_get_nbi() routine.

This test verifies that the shmem_get_nbi() function correctly retrieves data from PE 0 to PE 1 using non-blocking operations.

PE 1 gets data from an array on PE 0 using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 267 of file remote_tests.cpp.

```
00267                                {
00268   static long src[10], dest[10];
00269   int mype = p_shmem_my_pe();
00270   int npes = p_shmem_n_pes();
00271
00272   if (mype == 0) {
00273     for (int i = 0; i < 10; i++) {
00274       src[i] = i;
00275     }
00276   }
00277
00278   p_shmem_barrier_all();
00279
00280   if (mype == 1) {
00281     p_shmem_long_get_nbi(dest, src, 10, 0);
00282     p_shmem_quiet();
00283   }
00284
00285   p_shmem_barrier_all();
00286
00287   if (mype == 1) {
00288     for (int i = 0; i < 10; i++) {
00289       if (dest[i] != i) {
00290         return false;
00291       }
00292     }
00293   }
00294
00295   return true;
00296 }
```

References p_shmem_barrier_all, p_shmem_long_get_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.37.2.4 test_shmem_iget()

```
bool test_shmem_iget (
              void )
```

Tests the shmem_iget() function.

Tests the shmem_iget() routine.

This test verifies that the shmem_iget() function correctly retrieves data from PE 0 to PE 1 using an indirect stride.

PE 1 gets data from an array on PE 0 using an indirect stride.

Returns

True if the test is successful, false otherwise.

Definition at line 192 of file remote_tests.cpp.

```
00192                                {
00193   static long src[10], dest[10];
00194   int mype = p_shmem_my_pe();
00195   int npes = p_shmem_n_pes();
00196
00197   if (mype == 0) {
00198     for (int i = 0; i < 10; i++) {
00199       src[i] = i;
00200     }
00201   }
00202
00203   p_shmem_barrier_all();
00204
00205   if (mype == 1) {
00206     p_shmem_long_iget(dest, src, 2, 2, 5, 0);
```

```
00207   }
00208
00209   p_shmem_barrier_all();
00210
00211   if (mype == 1) {
00212     for (int i = 0; i < 10; i += 2) {
00213       if (dest[i] != i / 2) {
00214         return false;
00215       }
00216     }
00217   }
00218
00219   return true;
00220 }
```

References p_shmem_barrier_all, p_shmem_long_iget, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.5 test_shmem_iput()

```
bool test_shmem_iput (
             void )
```

Tests the shmem_iput() function.

Tests the shmem_iput() routine.

This test verifies that the shmem_iput() function correctly transfers data from PE 0 to PE 1 using an indirect stride.

PE 0 puts data into an array on PE 1 using an indirect stride.

Returns

True if the test is successful, false otherwise.

Definition at line 84 of file remote_tests.cpp.

```
00084                                {
00085   static long src[10], dest[10];
00086   int mype = p_shmem_my_pe();
00087   int npes = p_shmem_n_pes();
00088
00089   for (int i = 0; i < 10; i++) {
00090     src[i] = i + mype;
00091   }
00092
00093   p_shmem_barrier_all();
00094
00095   if (mype == 0) {
00096     p_shmem_long_iput(dest, src, 2, 2, 5, 1);
00097   }
00098
00099   p_shmem_barrier_all();
00100
00101   if (mype == 1) {
00102     for (int i = 0; i < 10; i += 2) {
00103       if (dest[i] != i / 2) {
00104         return false;
00105       }
00106     }
00107   }
00108
00109   return true;
00110 }
```

References p_shmem_barrier_all, p_shmem_long_iput, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.6 test_shmem_p()

```
bool test_shmem_p (
             void )
```

Tests the shmem_p() function.

Tests the shmem_p() routine.

This test verifies that the shmem_p() function correctly transfers a single data element from PE 0 to PE 1.

PE 0 puts a single data element into PE 1.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file remote_tests.cpp.

```
00052                           {
00053   static long src, dest;
00054   int mype = p_shmem_my_pe();
00055   int npes = p_shmem_n_pes();
00056
00057   src = mype;
00058
00059   p_shmem_barrier_all();
00060
00061   if (mype == 0) {
00062     p_shmem_long_p(&dest, src, 1);
00063   }
00064
00065   p_shmem_barrier_all();
00066
00067   if (mype == 1) {
00068     if (dest != 0) {
00069       return false;
00070     }
00071   }
00072
00073   return true;
00074 }
```

References p_shmem_barrier_all, p_shmem_long_p, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.7 test_shmem_put()

```
bool test_shmem_put (
             void )
```

Tests the shmem_put() function.

Tests the shmem_put() routine.

This test verifies that the shmem_put() function correctly transfers data from PE 0 to PE 1.

PE 0 puts data into an array on PE 1.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file remote_tests.cpp.

```
00016                           {
00017   static long src[10], dest[10];
00018   int mype = p_shmem_my_pe();
00019   int npes = p_shmem_n_pes();
00020
00021   for (int i = 0; i < 10; i++) {
00022     src[i] = i + mype;
00023   }
00024
00025   p_shmem_barrier_all();
00026
00027   if (mype == 0) {
00028     p_shmem_long_put(dest, src, 10, 1);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     for (int i = 0; i < 10; i++) {
00035       if (dest[i] != i) {
00036         return false;
00037       }
00038     }
00039   }
00040
00041   return true;
00042 }
```

References p_shmem_barrier_all, p_shmem_long_put, p_shmem_my_pe, and p_shmem_n_pes.

4.37.2.8 test_shmem_put_nbi()

```
bool test_shmem_put_nbi (
             void )
```

Tests the shmem_put_nbi() function.

Tests the shmem_put_nbi() routine.

This test verifies that the shmem_put_nbi() function correctly transfers data from PE 0 to PE 1 using non-blocking operations.

PE 0 puts data into an array on PE 1 using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 230 of file remote_tests.cpp.

```
00230                                {
00231   static long src[10], dest[10];
00232   int mype = p_shmem_my_pe();
00233   int npes = p_shmem_n_pes();
00234
00235   for (int i = 0; i < 10; i++) {
00236     src[i] = i + mype;
00237   }
00238
00239   p_shmem_barrier_all();
00240
00241   if (mype == 0) {
00242     p_shmem_long_put_nbi(dest, src, 10, 1);
00243     p_shmem_quiet();
00244   }
00245
00246   p_shmem_barrier_all();
```

```
00247
00248   if (mype == 1) {
00249     for (int i = 0; i < 10; i++) {
00250       if (dest[i] != i) {
00251         return false;
00252       }
00253     }
00254   }
00255
00256   return true;
00257 }
```

References p_shmem_barrier_all, p_shmem_long_put_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.38 remote_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "remote_tests.hpp"
00007
00016 bool test_shmem_put(void) {
00017   static long src[10], dest[10];
00018   int mype = p_shmem_my_pe();
00019   int npes = p_shmem_n_pes();
00020
00021   for (int i = 0; i < 10; i++) {
00022     src[i] = i + mype;
00023   }
00024
00025   p_shmem_barrier_all();
00026
00027   if (mype == 0) {
00028     p_shmem_long_put(dest, src, 10, 1);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     for (int i = 0; i < 10; i++) {
00035       if (dest[i] != i) {
00036         return false;
00037       }
00038     }
00039   }
00040
00041   return true;
00042 }
00043
00052 bool test_shmem_p(void) {
00053   static long src, dest;
00054   int mype = p_shmem_my_pe();
00055   int npes = p_shmem_n_pes();
00056
00057   src = mype;
00058
00059   p_shmem_barrier_all();
00060
00061   if (mype == 0) {
00062     p_shmem_long_p(&dest, src, 1);
00063   }
00064
00065   p_shmem_barrier_all();
00066
00067   if (mype == 1) {
00068     if (dest != 0) {
00069       return false;
00070     }
00071   }
00072
00073   return true;
00074 }
00075
00084 bool test_shmem_iput(void) {
00085   static long src[10], dest[10];
00086   int mype = p_shmem_my_pe();
00087   int npes = p_shmem_n_pes();
00088
00089   for (int i = 0; i < 10; i++) {
00090     src[i] = i + mype;
00091   }
```

```
00092
00093   p_shmem_barrier_all();
00094
00095   if (mype == 0) {
00096     p_shmem_long_iput(dest, src, 2, 2, 5, 1);
00097   }
00098
00099   p_shmem_barrier_all();
00100
00101   if (mype == 1) {
00102     for (int i = 0; i < 10; i += 2) {
00103       if (dest[i] != i / 2) {
00104         return false;
00105       }
00106     }
00107   }
00108
00109   return true;
00110 }
00111
00120 bool test_shmem_get(void) {
00121   static long src[10], dest[10];
00122   int mype = p_shmem_my_pe();
00123   int npes = p_shmem_n_pes();
00124
00125   if (mype == 0) {
00126     for (int i = 0; i < 10; i++) {
00127       src[i] = i;
00128     }
00129   }
00130
00131   p_shmem_barrier_all();
00132
00133   if (mype == 1) {
00134     p_shmem_long_get(dest, src, 10, 0);
00135   }
00136
00137   p_shmem_barrier_all();
00138
00139   if (mype == 1) {
00140     for (int i = 0; i < 10; i++) {
00141       if (dest[i] != i) {
00142         return false;
00143       }
00144     }
00145   }
00146
00147   return true;
00148 }
00149
00158 bool test_shmem_g(void) {
00159   static long src, dest;
00160   int mype = p_shmem_my_pe();
00161   int npes = p_shmem_n_pes();
00162
00163   if (mype == 0) {
00164     src = 10;
00165   }
00166
00167   p_shmem_barrier_all();
00168
00169   if (mype == 1) {
00170     dest = p_shmem_long_g(&src, 0);
00171   }
00172
00173   p_shmem_barrier_all();
00174
00175   if (mype == 1) {
00176     if (dest != 10) {
00177       return false;
00178     }
00179   }
00180
00181   return true;
00182 }
00183
00192 bool test_shmem_iget(void) {
00193   static long src[10], dest[10];
00194   int mype = p_shmem_my_pe();
00195   int npes = p_shmem_n_pes();
00196
00197   if (mype == 0) {
00198     for (int i = 0; i < 10; i++) {
00199       src[i] = i;
00200     }
00201   }
00202
```

```
00203   p_shmem_barrier_all();
00204
00205   if (mype == 1) {
00206     p_shmem_long_iget(dest, src, 2, 2, 5, 0);
00207   }
00208
00209   p_shmem_barrier_all();
00210
00211   if (mype == 1) {
00212     for (int i = 0; i < 10; i += 2) {
00213       if (dest[i] != i / 2) {
00214         return false;
00215       }
00216     }
00217   }
00218
00219   return true;
00220 }
00221
00230 bool test_shmem_put_nbi(void) {
00231   static long src[10], dest[10];
00232   int mype = p_shmem_my_pe();
00233   int npes = p_shmem_n_pes();
00234
00235   for (int i = 0; i < 10; i++) {
00236     src[i] = i + mype;
00237   }
00238
00239   p_shmem_barrier_all();
00240
00241   if (mype == 0) {
00242     p_shmem_long_put_nbi(dest, src, 10, 1);
00243     p_shmem_quiet();
00244   }
00245
00246   p_shmem_barrier_all();
00247
00248   if (mype == 1) {
00249     for (int i = 0; i < 10; i++) {
00250       if (dest[i] != i) {
00251         return false;
00252       }
00253     }
00254   }
00255
00256   return true;
00257 }
00258
00267 bool test_shmem_get_nbi(void) {
00268   static long src[10], dest[10];
00269   int mype = p_shmem_my_pe();
00270   int npes = p_shmem_n_pes();
00271
00272   if (mype == 0) {
00273     for (int i = 0; i < 10; i++) {
00274       src[i] = i;
00275     }
00276   }
00277
00278   p_shmem_barrier_all();
00279
00280   if (mype == 1) {
00281     p_shmem_long_get_nbi(dest, src, 10, 0);
00282     p_shmem_quiet();
00283   }
00284
00285   p_shmem_barrier_all();
00286
00287   if (mype == 1) {
00288     for (int i = 0; i < 10; i++) {
00289       if (dest[i] != i) {
00290         return false;
00291       }
00292     }
00293   }
00294
00295   return true;
00296 }
```

4.39 src/tests/remote/remote_tests.hpp File Reference

Contains function declarations for the OpenSHMEM remote memory access tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
```

Functions

- bool test_shmem_put (void)
  Tests the shmem_put() routine.
- bool test_shmem_p (void)
  Tests the shmem_p() routine.
- bool test_shmem_iput (void)
  Tests the shmem_iput() routine.
- bool test_shmem_get (void)
  Tests the shmem_get() routine.
- bool test_shmem_g (void)
  Tests the shmem_g() routine.
- bool test_shmem_iget (void)
  Tests the shmem_iget() routine.
- bool test_shmem_put_nbi (void)
  Tests the shmem_put_nbi() routine.
- bool test_shmem_get_nbi (void)
  Tests the shmem_get_nbi() routine.

4.39.1 Detailed Description

Contains function declarations for the OpenSHMEM remote memory access tests.

Definition in file remote_tests.hpp.

4.39.2 Function Documentation

4.39.2.1 test_shmem_g()

```
bool test_shmem_g (
            void )
```

Tests the shmem_g() routine.

This test verifies that the shmem_g() function correctly retrieves a single data element from one PE to another.

Returns

True if the test is successful, false otherwise.

Tests the shmem_g() routine.

This test verifies that the shmem_g() function correctly retrieves a single data element from PE 0 to PE 1.

PE 1 gets a single data element from PE 0.

Returns

True if the test is successful, false otherwise.

Definition at line 158 of file remote_tests.cpp.

```
00158                           {
00159   static long src, dest;
00160   int mype = p_shmem_my_pe();
00161   int npes = p_shmem_n_pes();
00162
00163   if (mype == 0) {
00164     src = 10;
00165   }
00166
00167   p_shmem_barrier_all();
00168
00169   if (mype == 1) {
00170     dest = p_shmem_long_g(&src, 0);
00171   }
00172
00173   p_shmem_barrier_all();
00174
00175   if (mype == 1) {
00176     if (dest != 10) {
00177       return false;
00178     }
00179   }
00180
00181   return true;
00182 }
```

References p_shmem_barrier_all, p_shmem_long_g, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.2 test_shmem_get()

```
bool test_shmem_get (
             void )
```

Tests the shmem_get() routine.

This test verifies that the shmem_get() function correctly retrieves data from one PE to another.

Returns

True if the test is successful, false otherwise.

Tests the shmem_get() routine.

This test verifies that the shmem_get() function correctly retrieves data from PE 0 to PE 1.

PE 1 gets data from an array on PE 0.

Returns

True if the test is successful, false otherwise.

Definition at line 120 of file remote_tests.cpp.

```
00120                                {
00121   static long src[10], dest[10];
00122   int mype = p_shmem_my_pe();
00123   int npes = p_shmem_n_pes();
00124
00125   if (mype == 0) {
00126     for (int i = 0; i < 10; i++) {
00127       src[i] = i;
00128     }
00129   }
00130
00131   p_shmem_barrier_all();
00132
00133   if (mype == 1) {
00134     p_shmem_long_get(dest, src, 10, 0);
00135   }
00136
00137   p_shmem_barrier_all();
00138
00139   if (mype == 1) {
00140     for (int i = 0; i < 10; i++) {
00141       if (dest[i] != i) {
00142         return false;
00143       }
00144     }
00145   }
00146
00147   return true;
00148 }
```

References p_shmem_barrier_all, p_shmem_long_get, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.3 test_shmem_get_nbi()

```
bool test_shmem_get_nbi (
             void )
```

Tests the shmem_get_nbi() routine.

This test verifies that the shmem_get_nbi() function correctly retrieves data from one PE to another using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Tests the shmem_get_nbi() routine.

This test verifies that the shmem_get_nbi() function correctly retrieves data from PE 0 to PE 1 using non-blocking operations.

PE 1 gets data from an array on PE 0 using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 267 of file remote_tests.cpp.

```
00267                                 {
00268   static long src[10], dest[10];
00269   int mype = p_shmem_my_pe();
00270   int npes = p_shmem_n_pes();
00271
00272   if (mype == 0) {
00273     for (int i = 0; i < 10; i++) {
00274       src[i] = i;
00275     }
00276   }
00277
00278   p_shmem_barrier_all();
00279
00280   if (mype == 1) {
00281     p_shmem_long_get_nbi(dest, src, 10, 0);
00282     p_shmem_quiet();
00283   }
00284
00285   p_shmem_barrier_all();
00286
00287   if (mype == 1) {
00288     for (int i = 0; i < 10; i++) {
00289       if (dest[i] != i) {
00290         return false;
00291       }
00292     }
00293   }
00294
00295   return true;
00296 }
```

References p_shmem_barrier_all, p_shmem_long_get_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.39.2.4 test_shmem_iget()

```
bool test_shmem_iget (
             void )
```

Tests the shmem_iget() routine.

This test verifies that the shmem_iget() function correctly retrieves data from one PE to another using an indirect stride.

Returns

True if the test is successful, false otherwise.

Tests the shmem_iget() routine.

This test verifies that the shmem_iget() function correctly retrieves data from PE 0 to PE 1 using an indirect stride.

PE 1 gets data from an array on PE 0 using an indirect stride.

Returns

True if the test is successful, false otherwise.

Definition at line 192 of file remote_tests.cpp.

```
00192                                {
00193   static long src[10], dest[10];
00194   int mype = p_shmem_my_pe();
00195   int npes = p_shmem_n_pes();
00196
00197   if (mype == 0) {
00198     for (int i = 0; i < 10; i++) {
00199       src[i] = i;
00200     }
00201   }
00202
00203   p_shmem_barrier_all();
00204
00205   if (mype == 1) {
00206     p_shmem_long_iget(dest, src, 2, 2, 5, 0);
00207   }
00208
00209   p_shmem_barrier_all();
00210
00211   if (mype == 1) {
00212     for (int i = 0; i < 10; i += 2) {
00213       if (dest[i] != i / 2) {
00214         return false;
00215       }
00216     }
00217   }
00218
00219   return true;
00220 }
```

References p_shmem_barrier_all, p_shmem_long_iget, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.5 test_shmem_iput()

```
bool test_shmem_iput (
             void )
```

Tests the shmem_iput() routine.

This test verifies that the shmem_iput() function correctly transfers data from one PE to another using an indirect stride.

Returns

True if the test is successful, false otherwise.

Tests the shmem_iput() routine.

This test verifies that the shmem_iput() function correctly transfers data from PE 0 to PE 1 using an indirect stride.

PE 0 puts data into an array on PE 1 using an indirect stride.

Returns

True if the test is successful, false otherwise.

Definition at line 84 of file remote_tests.cpp.

```
00084                                {
00085   static long src[10], dest[10];
00086   int mype = p_shmem_my_pe();
00087   int npes = p_shmem_n_pes();
00088
00089   for (int i = 0; i < 10; i++) {
00090     src[i] = i + mype;
00091   }
00092
00093   p_shmem_barrier_all();
00094
00095   if (mype == 0) {
00096     p_shmem_long_iput(dest, src, 2, 2, 5, 1);
00097   }
00098
00099   p_shmem_barrier_all();
00100
00101   if (mype == 1) {
00102     for (int i = 0; i < 10; i += 2) {
00103       if (dest[i] != i / 2) {
00104         return false;
00105       }
00106     }
00107   }
00108
00109   return true;
00110 }
```

References p_shmem_barrier_all, p_shmem_long_iput, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.6 test_shmem_p()

```
bool test_shmem_p (
             void )
```

Tests the shmem_p() routine.

This test verifies that the shmem_p() function correctly transfers a single data element from one PE to another.

Returns

True if the test is successful, false otherwise.

Tests the shmem_p() routine.

This test verifies that the shmem_p() function correctly transfers a single data element from PE 0 to PE 1.

PE 0 puts a single data element into PE 1.

Returns

True if the test is successful, false otherwise.

Definition at line 52 of file remote_tests.cpp.

```
00052                          {
00053   static long src, dest;
00054   int mype = p_shmem_my_pe();
00055   int npes = p_shmem_n_pes();
00056
00057   src = mype;
00058
00059   p_shmem_barrier_all();
00060
00061   if (mype == 0) {
00062     p_shmem_long_p(&dest, src, 1);
00063   }
00064
00065   p_shmem_barrier_all();
00066
00067   if (mype == 1) {
00068     if (dest != 0) {
00069       return false;
00070     }
00071   }
00072
00073   return true;
00074 }
```

References p_shmem_barrier_all, p_shmem_long_p, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.7 test_shmem_put()

```
bool test_shmem_put (
             void )
```

Tests the shmem_put() routine.

This test verifies that the shmem_put() function correctly transfers data from one PE to another.

Returns

True if the test is successful, false otherwise.

Tests the shmem_put() routine.

This test verifies that the shmem_put() function correctly transfers data from PE 0 to PE 1.

PE 0 puts data into an array on PE 1.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file remote_tests.cpp.

```
00016                           {
00017   static long src[10], dest[10];
00018   int mype = p_shmem_my_pe();
00019   int npes = p_shmem_n_pes();
00020
00021   for (int i = 0; i < 10; i++) {
00022     src[i] = i + mype;
00023   }
00024
00025   p_shmem_barrier_all();
00026
00027   if (mype == 0) {
```

```
00028     p_shmem_long_put(dest, src, 10, 1);
00029   }
00030
00031   p_shmem_barrier_all();
00032
00033   if (mype == 1) {
00034     for (int i = 0; i < 10; i++) {
00035       if (dest[i] != i) {
00036         return false;
00037       }
00038     }
00039   }
00040
00041   return true;
00042 }
```

References p_shmem_barrier_all, p_shmem_long_put, p_shmem_my_pe, and p_shmem_n_pes.

4.39.2.8 test_shmem_put_nbi()

```
bool test_shmem_put_nbi (
             void )
```

Tests the shmem_put_nbi() routine.

This test verifies that the shmem_put_nbi() function correctly transfers data from one PE to another using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Tests the shmem_put_nbi() routine.

This test verifies that the shmem_put_nbi() function correctly transfers data from PE 0 to PE 1 using non-blocking operations.

PE 0 puts data into an array on PE 1 using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 230 of file remote_tests.cpp.

```
00230                                {
00231   static long src[10], dest[10];
00232   int mype = p_shmem_my_pe();
00233   int npes = p_shmem_n_pes();
00234
00235   for (int i = 0; i < 10; i++) {
00236     src[i] = i + mype;
00237   }
00238
00239   p_shmem_barrier_all();
00240
00241   if (mype == 0) {
00242     p_shmem_long_put_nbi(dest, src, 10, 1);
00243     p_shmem_quiet();
00244   }
00245
00246   p_shmem_barrier_all();
00247
00248   if (mype == 1) {
00249     for (int i = 0; i < 10; i++) {
00250       if (dest[i] != i) {
00251         return false;
00252       }
00253     }
00254   }
00255
00256   return true;
00257 }
```

References p_shmem_barrier_all, p_shmem_long_put_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.40 remote_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef REMOTE_TESTS_HPP
00007 #define REMOTE_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012
00020 bool test_shmem_put(void);
00021
00029 bool test_shmem_p(void);
00030
00038 bool test_shmem_iput(void);
00039
00047 bool test_shmem_get(void);
00048
00056 bool test_shmem_g(void);
00057
00065 bool test_shmem_iget(void);
00066
00074 bool test_shmem_put_nbi(void);
00075
00083 bool test_shmem_get_nbi(void);
00084
00085 #endif /* REMOTE_TESTS_HPP */
```

4.41 src/tests/setup/setup_tests.cpp File Reference

Contains OpenSHMEM setup tests.

```
#include "setup_tests.hpp"
```

Functions

- bool test_shmem_fake_routine (void)

  Tests the presence of a fake routine for demonstration purposes.
- bool test_shmem_init ()

  Tests the initialization of OpenSHMEM.
- bool test_shmem_barrier_all ()

  Tests the barrier synchronization across all PEs.
- bool test_shmem_barrier (void)

  Tests the shmem_barrier() routine.
- int test_shmem_my_pe ()

  Tests retrieving the PE number of the calling PE.
- int test_shmem_n_pes ()

  Tests retrieving the number of PEs.
- bool test_shmem_pe_accessible ()

  Tests if a PE is accessible from the calling PE.
- std::string test_shmem_info_get_version ()

  Tests retrieving the OpenSHMEM library version.
- std::string test_shmem_info_get_name ()

  Tests retrieving the name of the OpenSHMEM library.
- bool test_shmem_finalize ()

  Tests the finalization of OpenSHMEM.
- bool test_shmem_global_exit ()

  Tests the global exit functionality of OpenSHMEM.

4.41.1 Detailed Description

Contains OpenSHMEM setup tests.

Definition in file setup_tests.cpp.

4.41.2 Function Documentation

4.41.2.1 test_shmem_barrier()

```
bool test_shmem_barrier (
            void )
```

Tests the shmem_barrier() routine.

This test verifies that the `shmem_barrier` routine functions correctly.

Returns

True if the test is successful, false otherwise.

Definition at line 70 of file setup_tests.cpp.

```
00070                                {
00071   static long pSync[SHMEM_BARRIER_SYNC_SIZE];
00072   for (int i = 0; i < SHMEM_BARRIER_SYNC_SIZE; i++) {
00073     pSync[i] = SHMEM_SYNC_VALUE;
00074   }
00075
00076   p_shmem_barrier(0, 0, p_shmem_n_pes(), pSync);
00077   return true;
00078 }
```

References p_shmem_barrier, and p_shmem_n_pes.

4.41.2.2 test_shmem_barrier_all()

```
bool test_shmem_barrier_all (
            void )
```

Tests the barrier synchronization across all PEs.

This test verifies that the barrier synchronization across all PEs is successful.

Returns

True if the barrier synchronization is successful, false otherwise.

Definition at line 45 of file setup_tests.cpp.

```
00045                                {
00046   int mype = p_shmem_my_pe();
00047   int npes = p_shmem_n_pes();
00048
00049   static long sync;
00050   sync = mype;
00051
00052   p_shmem_barrier_all();
00053
00054   bool test_passed = true;
00055
00056   if (sync != mype) {
00057     test_passed = false;
00058   }
00059
00060   return test_passed;
00061 }
```

References p_shmem_barrier_all, p_shmem_my_pe, and p_shmem_n_pes.

4.41.2.3 test_shmem_fake_routine()

```
bool test_shmem_fake_routine (
             void )
```

Tests the presence of a fake routine for demonstration purposes.

Tests to make sure routines that aren't implemented in the tested OpenSHMEM library don't throw compiler errors.

This test checks if the `p_shmem_fake_routine` function is available and calls it if so.

Returns

True if the routine is available and called, false otherwise.

Definition at line 15 of file setup_tests.cpp.

```
00015                                       {
00016   if (p_shmem_fake_routine) {
00017     p_shmem_fake_routine();
00018     return true;
00019   }
00020   else {
00021     std::cerr « "shmem_fake_routine is not available." « std::endl;
00022     return false;
00023   }
00024 }
```

References p_shmem_fake_routine.

4.41.2.4 test_shmem_finalize()

```
bool test_shmem_finalize (
             void )
```

Tests the finalization of OpenSHMEM.

This test verifies that the OpenSHMEM library finalizes successfully.

Returns

True if the finalization is successful, false otherwise.

Definition at line 171 of file setup_tests.cpp.

```
00171                                 {
00172   p_shmem_finalize();
00173   return true;
00174 }
```

References p_shmem_finalize.

4.41.2.5 test_shmem_global_exit()

```
bool test_shmem_global_exit (
            void )
```

Tests the global exit functionality of OpenSHMEM.

This test verifies that the `shmem_global_exit` function successfully performs a global exit.

Returns

True if the global exit is successful, false otherwise.

Definition at line 183 of file setup_tests.cpp.

```
00183                                {
00184   p_shmem_global_exit(0);
00185   return true;
00186 }
```

References p_shmem_global_exit.

4.41.2.6 test_shmem_info_get_name()

```
std::string test_shmem_info_get_name (
            void )
```

Tests retrieving the name of the OpenSHMEM library.

This test verifies that the `shmem_info_get_name` function returns the correct name of the OpenSHMEM library.

Returns

The name of the library as a string if successful, otherwise an empty string.

Definition at line 153 of file setup_tests.cpp.

```
00153                                      {
00154   char name[SHMEM_MAX_NAME_LEN];
00155   p_shmem_info_get_name(name);
00156   if (strlen(name) > 0) {
00157     return std::string(name);
00158   }
00159   else {
00160     return "";
00161   }
00162 }
```

References p_shmem_info_get_name.

4.41.2.7 test_shmem_info_get_version()

```
std::string test_shmem_info_get_version (
            void )
```

Tests retrieving the OpenSHMEM library version.

This test verifies that the `shmem_info_get_version` function returns the correct version of the OpenSHMEM library.

Returns

The version as a string in the format "major.minor".

Definition at line 138 of file setup_tests.cpp.

```
00138                                       {
00139   int major, minor;
00140   p_shmem_info_get_version(&major, &minor);
00141
00142   std::string version = std::to_string(major) + "." + std::to_string(minor);
00143   return version;
00144 }
```

References p_shmem_info_get_version.

4.41.2.8 test_shmem_init()

```
bool test_shmem_init (
            void )
```

Tests the initialization of OpenSHMEM.

This test verifies that the OpenSHMEM library initializes successfully.

Returns

True if the initialization is successful, false otherwise.

Definition at line 33 of file setup_tests.cpp.

```
00033                           {
00034   p_shmem_init();
00035   return true;
00036 }
```

References p_shmem_init.

4.41.2.9 test_shmem_my_pe()

```
int test_shmem_my_pe (
            void )
```

Tests retrieving the PE number of the calling PE.

This test verifies that the `shmem_my_pe` function returns a valid PE number.

Returns

The PE number on success, -1 on failure.

Definition at line 87 of file setup_tests.cpp.

```
00087                           {
00088   int mype = p_shmem_my_pe();
00089   if (mype >= 0) {
00090     return mype;
00091   }
00092   else {
00093     return -1;
00094   }
00095 }
```

References p_shmem_my_pe.

4.41.2.10 test_shmem_n_pes()

```
int test_shmem_n_pes (
            void )
```

Tests retrieving the number of PEs.

This test verifies that the `shmem_n_pes` function returns a valid number of PEs.

Returns

The number of PEs if greater than 0, otherwise 0.

Definition at line 104 of file setup_tests.cpp.

```
00104                           {
00105   int npes = p_shmem_n_pes();
00106   if (!(npes > 0)) {
00107     return 0;
00108   }
00109   else {
00110     return npes;
00111   }
00112 }
```

References p_shmem_n_pes.

4.41.2.11 test_shmem_pe_accessible()

```
bool test_shmem_pe_accessible (
            void )
```

Tests if a PE is accessible from the calling PE.

This test verifies that the `shmem_pe_accessible` function correctly reports accessibility of all PEs.

Returns

True if all PEs are accessible, false otherwise.

Definition at line 121 of file setup_tests.cpp.

```
00121                                    {
00122   int npes = p_shmem_n_pes();
00123   for (int pe = 0; pe < npes; ++pe) {
00124     if (!p_shmem_pe_accessible(pe)) {
00125       return false;
00126     }
00127   }
00128   return true;
00129 }
```

References p_shmem_n_pes, and p_shmem_pe_accessible.

4.42 setup_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "setup_tests.hpp"
00007
00015 bool test_shmem_fake_routine(void) {
00016   if (p_shmem_fake_routine) {
00017     p_shmem_fake_routine();
00018     return true;
00019   }
00020   else {
00021     std::cerr « "shmem_fake_routine is not available." « std::endl;
00022     return false;
00023   }
00024 }
00025
00033 bool test_shmem_init() {
00034   p_shmem_init();
00035   return true;
00036 }
00037
00045 bool test_shmem_barrier_all() {
00046   int mype = p_shmem_my_pe();
00047   int npes = p_shmem_n_pes();
00048
00049   static long sync;
00050   sync = mype;
00051
00052   p_shmem_barrier_all();
00053
00054   bool test_passed = true;
00055
00056   if (sync != mype) {
00057     test_passed = false;
00058   }
00059
00060   return test_passed;
00061 }
00062
00070 bool test_shmem_barrier(void) {
00071   static long pSync[SHMEM_BARRIER_SYNC_SIZE];
00072   for (int i = 0; i < SHMEM_BARRIER_SYNC_SIZE; i++) {
00073     pSync[i] = SHMEM_SYNC_VALUE;
00074   }
00075
```

```
00076   p_shmem_barrier(0, 0, p_shmem_n_pes(), pSync);
00077   return true;
00078 }
00079
00087 int test_shmem_my_pe() {
00088   int mype = p_shmem_my_pe();
00089   if (mype >= 0) {
00090     return mype;
00091   }
00092   else {
00093     return -1;
00094   }
00095 }
00096
00104 int test_shmem_n_pes() {
00105   int npes = p_shmem_n_pes();
00106   if (!(npes > 0)) {
00107     return 0;
00108   }
00109   else {
00110     return npes;
00111   }
00112 }
00113
00121 bool test_shmem_pe_accessible() {
00122   int npes = p_shmem_n_pes();
00123   for (int pe = 0; pe < npes; ++pe) {
00124     if (!p_shmem_pe_accessible(pe)) {
00125       return false;
00126     }
00127   }
00128   return true;
00129 }
00130
00138 std::string test_shmem_info_get_version() {
00139   int major, minor;
00140   p_shmem_info_get_version(&major, &minor);
00141
00142   std::string version = std::to_string(major) + "." + std::to_string(minor);
00143   return version;
00144 }
00145
00153 std::string test_shmem_info_get_name() {
00154   char name[SHMEM_MAX_NAME_LEN];
00155   p_shmem_info_get_name(name);
00156   if (strlen(name) > 0) {
00157     return std::string(name);
00158   }
00159   else {
00160     return "";
00161   }
00162 }
00163
00171 bool test_shmem_finalize() {
00172   p_shmem_finalize();
00173   return true;
00174 }
00175
00183 bool test_shmem_global_exit() {
00184   p_shmem_global_exit(0);
00185   return true;
00186 }
```

4.43 src/tests/setup/setup_tests.hpp File Reference

Contains function declarations for the OpenSHMEM setup tests.

```
#include "routines.hpp"
#include "shmemvv.hpp"
#include <shmem.h>
#include <iostream>
#include <cstring>
#include <string>
```

Functions

- bool test_shmem_fake_routine (void)

  Tests to make sure routines that aren't implemented in the tested OpenSHMEM library don't throw compiler errors.
- bool test_shmem_init (void)

  Tests the initialization of OpenSHMEM.
- bool test_shmem_barrier_all (void)

  Tests the barrier synchronization across all PEs.
- bool test_shmem_barrier (void)

  Tests the shmem_barrier() routine.
- int test_shmem_my_pe (void)

  Tests retrieving the PE number of the calling PE.
- int test_shmem_n_pes (void)

  Tests retrieving the number of PEs.
- bool test_shmem_pe_accessible (void)

  Tests if a PE is accessible from the calling PE.
- std::string test_shmem_info_get_version (void)

  Tests retrieving the OpenSHMEM library version.
- std::string test_shmem_info_get_name (void)

  Tests retrieving the name of the OpenSHMEM library.
- bool test_shmem_finalize (void)

  Tests the finalization of OpenSHMEM.
- bool test_shmem_global_exit (void)

  Tests the global exit functionality of OpenSHMEM.

4.43.1 Detailed Description

Contains function declarations for the OpenSHMEM setup tests.

Definition in file setup_tests.hpp.

4.43.2 Function Documentation

4.43.2.1 test_shmem_barrier()

```
bool test_shmem_barrier (
            void )
```

Tests the shmem_barrier() routine.

This test verifies that the shmem_barrier() routine functions correctly.

Returns

True if the test is successful, false otherwise.

This test verifies that the `shmem_barrier` routine functions correctly.

Returns

True if the test is successful, false otherwise.

Definition at line 70 of file setup_tests.cpp.

```
00070                                {
00071   static long pSync[SHMEM_BARRIER_SYNC_SIZE];
00072   for (int i = 0; i < SHMEM_BARRIER_SYNC_SIZE; i++) {
00073     pSync[i] = SHMEM_SYNC_VALUE;
00074   }
00075
00076   p_shmem_barrier(0, 0, p_shmem_n_pes(), pSync);
00077   return true;
00078 }
```

References p_shmem_barrier, and p_shmem_n_pes.

4.43.2.2 test_shmem_barrier_all()

```
bool test_shmem_barrier_all (
            void )
```

Tests the barrier synchronization across all PEs.

This test verifies that the barrier synchronization across all PEs is successful.

Returns

True if the barrier synchronization is successful, false otherwise.

Definition at line 45 of file setup_tests.cpp.

```
00045                                {
00046   int mype = p_shmem_my_pe();
00047   int npes = p_shmem_n_pes();
00048
00049   static long sync;
00050   sync = mype;
00051
00052   p_shmem_barrier_all();
00053
00054   bool test_passed = true;
00055
00056   if (sync != mype) {
00057     test_passed = false;
00058   }
00059
00060   return test_passed;
00061 }
```

References p_shmem_barrier_all, p_shmem_my_pe, and p_shmem_n_pes.

4.43.2.3 test_shmem_fake_routine()

```
bool test_shmem_fake_routine (
            void )
```

Tests to make sure routines that aren't implemented in the tested OpenSHMEM library don't throw compiler errors.

This test verifies that calling an unimplemented routine in the OpenSHMEM library does not result in compiler errors.

Returns

True if the routine does not throw compiler errors, false otherwise.

Tests to make sure routines that aren't implemented in the tested OpenSHMEM library don't throw compiler errors.

This test checks if the `p_shmem_fake_routine` function is available and calls it if so.

Returns

True if the routine is available and called, false otherwise.

Definition at line 15 of file setup_tests.cpp.

```
00015                                    {
00016   if (p_shmem_fake_routine) {
00017     p_shmem_fake_routine();
00018     return true;
00019   }
00020   else {
00021     std::cerr « "shmem_fake_routine is not available." « std::endl;
00022     return false;
00023   }
00024 }
```

References p_shmem_fake_routine.

4.43.2.4 test_shmem_finalize()

```
bool test_shmem_finalize (
             void )
```

Tests the finalization of OpenSHMEM.

This test verifies that the OpenSHMEM library finalizes successfully.

Returns

True if the finalization is successful, false otherwise.

Definition at line 171 of file setup_tests.cpp.

```
00171                                {
00172   p_shmem_finalize();
00173   return true;
00174 }
```

References p_shmem_finalize.

4.43.2.5 test_shmem_global_exit()

```
bool test_shmem_global_exit (
             void )
```

Tests the global exit functionality of OpenSHMEM.

This test verifies that the shmem_global_exit() function successfully performs a global exit.

Returns

True if the global exit is successful, false otherwise.

This test verifies that the `shmem_global_exit` function successfully performs a global exit.

Returns

True if the global exit is successful, false otherwise.

Definition at line 183 of file setup_tests.cpp.

```
00183                                    {
00184   p_shmem_global_exit(0);
00185   return true;
00186 }
```

References p_shmem_global_exit.

4.43.2.6 test_shmem_info_get_name()

```
std::string test_shmem_info_get_name (
            void )
```

Tests retrieving the name of the OpenSHMEM library.

This test verifies that the shmem_info_get_name() function returns the correct name of the OpenSHMEM library.

Returns

The name of the library as a string if successful, otherwise an empty string.

This test verifies that the `shmem_info_get_name` function returns the correct name of the OpenSHMEM library.

Returns

The name of the library as a string if successful, otherwise an empty string.

Definition at line 153 of file setup_tests.cpp.

```
00153                                       {
00154   char name[SHMEM_MAX_NAME_LEN];
00155   p_shmem_info_get_name(name);
00156   if (strlen(name) > 0) {
00157     return std::string(name);
00158   }
00159   else {
00160     return "";
00161   }
00162 }
```

References p_shmem_info_get_name.

4.43.2.7 test_shmem_info_get_version()

```
std::string test_shmem_info_get_version (
            void )
```

Tests retrieving the OpenSHMEM library version.

This test verifies that the shmem_info_get_version() function returns the correct version of the OpenSHMEM library.

Returns

The version as a string in the format "major.minor".

This test verifies that the `shmem_info_get_version` function returns the correct version of the OpenSHMEM library.

Returns

The version as a string in the format "major.minor".

Definition at line 138 of file setup_tests.cpp.

```
00138                                          {
00139   int major, minor;
00140   p_shmem_info_get_version(&major, &minor);
00141
00142   std::string version = std::to_string(major) + "." + std::to_string(minor);
00143   return version;
00144 }
```

References p_shmem_info_get_version.

4.43.2.8 test_shmem_init()

```
bool test_shmem_init (
             void )
```

Tests the initialization of OpenSHMEM.

This test verifies that the OpenSHMEM library initializes successfully.

Returns

True if the initialization is successful, false otherwise.

Definition at line 33 of file setup_tests.cpp.

```
00033                          {
00034   p_shmem_init();
00035   return true;
00036 }
```

References p_shmem_init.

4.43.2.9 test_shmem_my_pe()

```
int test_shmem_my_pe (
             void )
```

Tests retrieving the PE number of the calling PE.

This test verifies that the shmem_my_pe() function returns a valid PE number.

Returns

The PE number on success, -1 on failure.

This test verifies that the `shmem_my_pe` function returns a valid PE number.

Returns

The PE number on success, -1 on failure.

Definition at line 87 of file setup_tests.cpp.

```
00087                          {
00088   int mype = p_shmem_my_pe();
00089   if (mype >= 0) {
00090     return mype;
00091   }
00092   else {
00093     return -1;
00094   }
00095 }
```

References p_shmem_my_pe.

4.43.2.10 test_shmem_n_pes()

```
int test_shmem_n_pes (
            void )
```

Tests retrieving the number of PEs.

This test verifies that the shmem_n_pes() function returns a valid number of PEs.

Returns

The number of PEs if greater than 0, otherwise 0.

This test verifies that the `shmem_n_pes` function returns a valid number of PEs.

Returns

The number of PEs if greater than 0, otherwise 0.

Definition at line 104 of file setup_tests.cpp.

```
00104                           {
00105   int npes = p_shmem_n_pes();
00106   if (!(npes > 0)) {
00107     return 0;
00108   }
00109   else {
00110     return npes;
00111   }
00112 }
```

References p_shmem_n_pes.

4.43.2.11 test_shmem_pe_accessible()

```
bool test_shmem_pe_accessible (
            void )
```

Tests if a PE is accessible from the calling PE.

This test verifies that the shmem_pe_accessible() function correctly reports the accessibility of all PEs.

Returns

True if all PEs are accessible, false otherwise.

This test verifies that the `shmem_pe_accessible` function correctly reports accessibility of all PEs.

Returns

True if all PEs are accessible, false otherwise.

Definition at line 121 of file setup_tests.cpp.

```
00121                                   {
00122   int npes = p_shmem_n_pes();
00123   for (int pe = 0; pe < npes; ++pe) {
00124     if (!p_shmem_pe_accessible(pe)) {
00125       return false;
00126     }
00127   }
00128   return true;
00129 }
```

References p_shmem_n_pes, and p_shmem_pe_accessible.

4.44 setup_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef SETUP_TESTS_HPP
00007 #define SETUP_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include "shmemvv.hpp"
00011
00012 #include <shmem.h>
00013 #include <iostream>
00014 #include <cstring>
00015 #include <string>
00016
00027 bool test_shmem_fake_routine(void);
00028
00036 bool test_shmem_init(void);
00037
00045 bool test_shmem_barrier_all(void);
00046
00054 bool test_shmem_barrier(void);
00055
00063 int test_shmem_my_pe(void);
00064
00072 int test_shmem_n_pes(void);
00073
00081 bool test_shmem_pe_accessible(void);
00082
00090 std::string test_shmem_info_get_version(void);
00091
00099 std::string test_shmem_info_get_name(void);
00100
00108 bool test_shmem_finalize(void);
00109
00117 bool test_shmem_global_exit(void);
00118
00119 #endif /* SETUP_TESTS_HPP */
```

4.45 src/tests/signaling/signaling_tests.cpp File Reference

Contains OpenSHMEM signaling tests.

```
#include "signaling_tests.hpp"
```

Functions

- bool test_shmem_put_signal (void)

  Tests the shmem_put_signal() routine.
- bool test_shmem_put_signal_nbi (void)

  Tests the shmem_put_signal_nbi() routine.
- bool test_shmem_signal_fetch (void)

  Tests the shmem_signal_fetch() routine.

4.45.1 Detailed Description

Contains OpenSHMEM signaling tests.

Definition in file signaling_tests.cpp.

4.45.2 Function Documentation

4.45.2.1 test_shmem_put_signal()

```
bool test_shmem_put_signal (
            void )
```

Tests the shmem_put_signal() routine.

This test verifies that the shmem_put_signal() function correctly transfers a value and sets a signal on the target PE.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file signaling_tests.cpp.

```
00016                                   {
00017   static long dest = 0;
00018   static long value = 12345;
00019   static uint64_t signal = 0;
00020   int mype = p_shmem_my_pe();
00021   int npes = p_shmem_n_pes();
00022
00023   if (npes < 2) {
00024     return false;
00025   }
00026
00027   int target_pe = (mype + 1) % npes;
00028
00029   p_shmem_barrier_all();
00030
00031   if (mype == 0) {
00032     p_shmem_long_put_signal(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00033   }
00034
00035   p_shmem_barrier_all();
00036
00037   if (mype == 1) {
00038     if (dest != 12345 || signal != 1) {
00039       return false;
00040     }
00041   }
00042
00043   return true;
00044 }
```

References p_shmem_barrier_all, p_shmem_long_put_signal, p_shmem_my_pe, and p_shmem_n_pes.

4.45.2.2 test_shmem_put_signal_nbi()

```
bool test_shmem_put_signal_nbi (
            void )
```

Tests the shmem_put_signal_nbi() routine.

This test verifies that the shmem_put_signal_nbi() function correctly transfers a value and sets a signal on the target PE using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 54 of file signaling_tests.cpp.

```
00054                                    {
00055   static long dest = 0;
00056   static long value = 67890;
00057   static uint64_t signal = 0;
00058   int mype = p_shmem_my_pe();
00059   int npes = p_shmem_n_pes();
00060
00061   if (npes < 2) {
00062     return false;
00063   }
00064
00065   int target_pe = (mype + 1) % npes;
00066
00067   p_shmem_barrier_all();
00068
00069   if (mype == 0) {
00070     p_shmem_long_put_signal_nbi(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00071     p_shmem_quiet();
00072   }
00073
00074   p_shmem_barrier_all();
00075
00076   if (mype == 1) {
00077     if (dest != 67890 || signal != 1) {
00078       return false;
00079     }
00080   }
00081
00082   return true;
00083 }
```

References p_shmem_barrier_all, p_shmem_long_put_signal_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.45.2.3 test_shmem_signal_fetch()

```
bool test_shmem_signal_fetch (
             void )
```

Tests the shmem_signal_fetch() routine.

This test verifies that the shmem_signal_fetch() function correctly fetches the signal value from the target PE.

Returns

True if the test is successful, false otherwise.

Definition at line 93 of file signaling_tests.cpp.

```
00093                                    {
00094   static uint64_t signal = 1;
00095   uint64_t fetched_signal = 0;
00096   int mype = p_shmem_my_pe();
00097   int npes = p_shmem_n_pes();
00098
00099   if (npes < 2) {
00100     return false;
00101   }
00102
00103   p_shmem_barrier_all();
00104
00105   if (mype == 1) {
00106     fetched_signal = p_shmem_signal_fetch(&signal);
00107     if (fetched_signal != 1) {
00108       return false;
00109     }
00110   }
00111
00112   return true;
00113 }
```

References p_shmem_barrier_all, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_signal_fetch.

4.46 signaling_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "signaling_tests.hpp"
00007
00016 bool test_shmem_put_signal(void) {
00017   static long dest = 0;
00018   static long value = 12345;
00019   static uint64_t signal = 0;
00020   int mype = p_shmem_my_pe();
00021   int npes = p_shmem_n_pes();
00022
00023   if (npes < 2) {
00024     return false;
00025   }
00026
00027   int target_pe = (mype + 1) % npes;
00028
00029   p_shmem_barrier_all();
00030
00031   if (mype == 0) {
00032     p_shmem_long_put_signal(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00033   }
00034
00035   p_shmem_barrier_all();
00036
00037   if (mype == 1) {
00038     if (dest != 12345 || signal != 1) {
00039       return false;
00040     }
00041   }
00042
00043   return true;
00044 }
00045
00054 bool test_shmem_put_signal_nbi(void) {
00055   static long dest = 0;
00056   static long value = 67890;
00057   static uint64_t signal = 0;
00058   int mype = p_shmem_my_pe();
00059   int npes = p_shmem_n_pes();
00060
00061   if (npes < 2) {
00062     return false;
00063   }
00064
00065   int target_pe = (mype + 1) % npes;
00066
00067   p_shmem_barrier_all();
00068
00069   if (mype == 0) {
00070     p_shmem_long_put_signal_nbi(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00071     p_shmem_quiet();
00072   }
00073
00074   p_shmem_barrier_all();
00075
00076   if (mype == 1) {
00077     if (dest != 67890 || signal != 1) {
00078       return false;
00079     }
00080   }
00081
00082   return true;
00083 }
00084
00093 bool test_shmem_signal_fetch(void) {
00094   static uint64_t signal = 1;
00095   uint64_t fetched_signal = 0;
00096   int mype = p_shmem_my_pe();
00097   int npes = p_shmem_n_pes();
00098
00099   if (npes < 2) {
00100     return false;
00101   }
00102
00103   p_shmem_barrier_all();
00104
00105   if (mype == 1) {
00106     fetched_signal = p_shmem_signal_fetch(&signal);
00107     if (fetched_signal != 1) {
00108       return false;
00109     }
00110   }
```

```
00111
00112   return true;
00113 }
```

4.47 src/tests/signaling/signaling_tests.hpp File Reference

Contains function declarations for the OpenSHMEM signaling tests.

```
#include "routines.hpp"
#include <shmem.h>
```

Functions

- bool test_shmem_put_signal (void)

  Tests the shmem_put_signal() routine.
- bool test_shmem_put_signal_nbi (void)

  Tests the shmem_put_signal_nbi() routine.
- bool test_shmem_signal_fetch (void)

  Tests the shmem_signal_fetch() routine.

4.47.1 Detailed Description

Contains function declarations for the OpenSHMEM signaling tests.

Definition in file signaling_tests.hpp.

4.47.2 Function Documentation

4.47.2.1 test_shmem_put_signal()

```
bool test_shmem_put_signal (
            void )
```

Tests the shmem_put_signal() routine.

This test verifies that the shmem_put_signal() function correctly transfers a value and sets a signal on the target PE.

Returns

True if the test is successful, false otherwise.

Definition at line 16 of file signaling_tests.cpp.

```
00016                                    {
00017   static long dest = 0;
00018   static long value = 12345;
00019   static uint64_t signal = 0;
00020   int mype = p_shmem_my_pe();
00021   int npes = p_shmem_n_pes();
00022
00023   if (npes < 2) {
00024     return false;
00025   }
00026
00027   int target_pe = (mype + 1) % npes;
00028
00029   p_shmem_barrier_all();
00030
00031   if (mype == 0) {
00032     p_shmem_long_put_signal(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00033   }
00034
00035   p_shmem_barrier_all();
00036
00037   if (mype == 1) {
00038     if (dest != 12345 || signal != 1) {
00039       return false;
00040     }
00041   }
00042
00043   return true;
00044 }
```

References p_shmem_barrier_all, p_shmem_long_put_signal, p_shmem_my_pe, and p_shmem_n_pes.

4.47.2.2 test_shmem_put_signal_nbi()

```
bool test_shmem_put_signal_nbi (
             void )
```

Tests the shmem_put_signal_nbi() routine.

This test verifies that the shmem_put_signal_nbi() function correctly transfers a value and sets a signal on the target PE using non-blocking operations.

Returns

True if the test is successful, false otherwise.

Definition at line 54 of file signaling_tests.cpp.

```
00054                                        {
00055   static long dest = 0;
00056   static long value = 67890;
00057   static uint64_t signal = 0;
00058   int mype = p_shmem_my_pe();
00059   int npes = p_shmem_n_pes();
00060
00061   if (npes < 2) {
00062     return false;
00063   }
00064
00065   int target_pe = (mype + 1) % npes;
00066
00067   p_shmem_barrier_all();
00068
00069   if (mype == 0) {
00070     p_shmem_long_put_signal_nbi(&dest, &value, 1, &signal, 1, target_pe, SHMEM_SIGNAL_SET);
00071     p_shmem_quiet();
00072   }
00073
00074   p_shmem_barrier_all();
```

```
00075
00076   if (mype == 1) {
00077     if (dest != 67890 || signal != 1) {
00078       return false;
00079     }
00080   }
00081
00082   return true;
00083 }
```

References p_shmem_barrier_all, p_shmem_long_put_signal_nbi, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_quiet.

4.47.2.3 test_shmem_signal_fetch()

```
bool test_shmem_signal_fetch (
             void )
```

Tests the shmem_signal_fetch() routine.

This test verifies that the shmem_signal_fetch() function correctly fetches the signal value from the target PE.

Returns

True if the test is successful, false otherwise.

Definition at line 93 of file signaling_tests.cpp.

```
00093                                       {
00094   static uint64_t signal = 1;
00095   uint64_t fetched_signal = 0;
00096   int mype = p_shmem_my_pe();
00097   int npes = p_shmem_n_pes();
00098
00099   if (npes < 2) {
00100     return false;
00101   }
00102
00103   p_shmem_barrier_all();
00104
00105   if (mype == 1) {
00106     fetched_signal = p_shmem_signal_fetch(&signal);
00107     if (fetched_signal != 1) {
00108       return false;
00109     }
00110   }
00111
00112   return true;
00113 }
```

References p_shmem_barrier_all, p_shmem_my_pe, p_shmem_n_pes, and p_shmem_signal_fetch.

4.48 signaling_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef SIGNALING_TESTS_HPP
00007 #define SIGNALING_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011
00020 bool test_shmem_put_signal(void);
00021
00030 bool test_shmem_put_signal_nbi(void);
00031
00040 bool test_shmem_signal_fetch(void);
00041
00042 #endif /* SIGNALING_TESTS_HPP */
```

4.49 src/tests/teams/teams_tests.cpp File Reference

Contains OpenSHMEM teams tests.

```
#include "teams_tests.hpp"
```

Functions

- bool test_shmem_team_my_pe (void)
  Tests the shmem_team_my_pe() routine.
- bool test_shmem_team_n_pes (void)
  Tests the shmem_team_n_pes() routine.
- bool test_shmem_team_get_config (void)
  Tests the shmem_team_get_config() routine.
- bool test_shmem_team_translate_pe (void)
  Tests the shmem_team_translate_pe() routine.
- bool test_shmem_team_split_strided (void)
  Tests the shmem_team_split_strided() routine.
- bool test_shmem_team_split_2d (void)
  Tests the shmem_team_split_2d() routine.
- bool test_shmem_team_destroy (void)
  Tests the shmem_team_destroy() routine.

4.49.1 Detailed Description

Contains OpenSHMEM teams tests.

Definition in file teams_tests.cpp.

4.49.2 Function Documentation

4.49.2.1 test_shmem_team_destroy()

```
bool test_shmem_team_destroy (
            void )
```

Tests the shmem_team_destroy() routine.

This test verifies that the shmem_team_destroy() function correctly destroys a team.

Returns

True if the test is successful, false otherwise.

Definition at line 113 of file teams_tests.cpp.

```
00113                                   {
00114   shmem_team_t team;
00115   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00116   p_shmem_team_destroy(team);
00117   if (!(team == SHMEM_TEAM_INVALID)) {
00118     return true;
00119   }
00120   else {
00121     return false;
00122   }
00123 }
```

References p_shmem_n_pes, p_shmem_team_destroy, and p_shmem_team_split_strided.

4.49.2.2 test_shmem_team_get_config()

```
bool test_shmem_team_get_config (
             void )
```

Tests the shmem_team_get_config() routine.

This test verifies that the shmem_team_get_config() function correctly retrieves the team configuration.

Returns

True if the test is successful, false otherwise.

Definition at line 45 of file teams_tests.cpp.

```
00045                                       {
00046   shmem_team_t team;
00047   shmem_team_config_t config;
00048   long config_mask = SHMEM_TEAM_NUM_CONTEXTS;
00049   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00050   if (team == SHMEM_TEAM_INVALID) {
00051     return false;
00052   }
00053   p_shmem_team_get_config(team, config_mask, &config);
00054   bool result = (config.num_contexts >= 0);
00055   p_shmem_team_destroy(team);
00056   return result;
00057 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_get_config, and p_shmem_team_split_strided.

4.49.2.3 test_shmem_team_my_pe()

```
bool test_shmem_team_my_pe (
             void )
```

Tests the shmem_team_my_pe() routine.

This test verifies that the shmem_team_my_pe() function returns a valid PE number within the team.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file teams_tests.cpp.

```
00015                                       {
00016   shmem_team_t team;
00017   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00018   int my_pe = p_shmem_team_my_pe(team);
00019   p_shmem_team_destroy(team);
00020   return (my_pe >= 0);
00021 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_my_pe, and p_shmem_team_split_strided.

4.49.2.4 test_shmem_team_n_pes()

```
bool test_shmem_team_n_pes (
             void )
```

Tests the shmem_team_n_pes() routine.

This test verifies that the shmem_team_n_pes() function returns the correct number of PEs in the team.

Returns

True if the test is successful, false otherwise.

Definition at line 30 of file teams_tests.cpp.

```
00030                                    {
00031    shmem_team_t team;
00032    p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00033    int npes = p_shmem_team_n_pes(team);
00034    p_shmem_team_destroy(team);
00035    return (npes == p_shmem_n_pes());
00036 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_strided.

4.49.2.5 test_shmem_team_split_2d()

```
bool test_shmem_team_split_2d (
             void )
```

Tests the shmem_team_split_2d() routine.

This test verifies that the shmem_team_split_2d() function correctly splits a team into two-dimensional subteams.

Returns

True if the test is successful, false otherwise.

Definition at line 96 of file teams_tests.cpp.

```
00096                                       {
00097    shmem_team_t team_x, team_y;
00098    p_shmem_team_split_2d(SHMEM_TEAM_WORLD, 2, NULL, 0, &team_x, NULL, 0, &team_y);
00099    int npes_x = p_shmem_team_n_pes(team_x);
00100    int npes_y = p_shmem_team_n_pes(team_y);
00101    p_shmem_team_destroy(team_x);
00102    p_shmem_team_destroy(team_y);
00103    return (npes_x > 0 && npes_y > 0);
00104 }
```

References p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_2d.

4.49.2.6 test_shmem_team_split_strided()

```
bool test_shmem_team_split_strided (
            void )
```

Tests the shmem_team_split_strided() routine.

This test verifies that the shmem_team_split_strided() function correctly splits a team into subteams.

Returns

True if the test is successful, false otherwise.

Definition at line 81 of file teams_tests.cpp.

```
00081                                           {
00082   shmem_team_t team;
00083   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00084   int npes = p_shmem_team_n_pes(team);
00085   p_shmem_team_destroy(team);
00086   return (npes == p_shmem_n_pes());
00087 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_strided.

4.49.2.7 test_shmem_team_translate_pe()

```
bool test_shmem_team_translate_pe (
            void )
```

Tests the shmem_team_translate_pe() routine.

This test verifies that the shmem_team_translate_pe() function correctly translates a PE number from one team to another.

Returns

True if the test is successful, false otherwise.

Definition at line 66 of file teams_tests.cpp.

```
00066                                           {
00067   shmem_team_t team;
00068   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00069   int pe_in_team = p_shmem_team_translate_pe(team, 0, SHMEM_TEAM_WORLD);
00070   p_shmem_team_destroy(team);
00071   return (pe_in_team >= 0);
00072 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_split_strided, and p_shmem_team_translate_pe.

4.50 teams_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "teams_tests.hpp"
00007
00015 bool test_shmem_team_my_pe(void) {
00016   shmem_team_t team;
00017   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00018   int my_pe = p_shmem_team_my_pe(team);
00019   p_shmem_team_destroy(team);
00020   return (my_pe >= 0);
00021 }
00022
00030 bool test_shmem_team_n_pes(void) {
00031   shmem_team_t team;
00032   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00033   int npes = p_shmem_team_n_pes(team);
00034   p_shmem_team_destroy(team);
00035   return (npes == p_shmem_n_pes());
00036 }
00037
00045 bool test_shmem_team_get_config(void) {
00046   shmem_team_t team;
00047   shmem_team_config_t config;
00048   long config_mask = SHMEM_TEAM_NUM_CONTEXTS;
00049   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00050   if (team == SHMEM_TEAM_INVALID) {
00051     return false;
00052   }
00053   p_shmem_team_get_config(team, config_mask, &config);
00054   bool result = (config.num_contexts >= 0);
00055   p_shmem_team_destroy(team);
00056   return result;
00057 }
00058
00066 bool test_shmem_team_translate_pe(void) {
00067   shmem_team_t team;
00068   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00069   int pe_in_team = p_shmem_team_translate_pe(team, 0, SHMEM_TEAM_WORLD);
00070   p_shmem_team_destroy(team);
00071   return (pe_in_team >= 0);
00072 }
00073
00081 bool test_shmem_team_split_strided(void) {
00082   shmem_team_t team;
00083   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00084   int npes = p_shmem_team_n_pes(team);
00085   p_shmem_team_destroy(team);
00086   return (npes == p_shmem_n_pes());
00087 }
00088
00096 bool test_shmem_team_split_2d(void) {
00097   shmem_team_t team_x, team_y;
00098   p_shmem_team_split_2d(SHMEM_TEAM_WORLD, 2, NULL, 0, &team_x, NULL, 0, &team_y);
00099   int npes_x = p_shmem_team_n_pes(team_x);
00100   int npes_y = p_shmem_team_n_pes(team_y);
00101   p_shmem_team_destroy(team_x);
00102   p_shmem_team_destroy(team_y);
00103   return (npes_x > 0 && npes_y > 0);
00104 }
00105
00113 bool test_shmem_team_destroy(void) {
00114   shmem_team_t team;
00115   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00116   p_shmem_team_destroy(team);
00117   if (!(team == SHMEM_TEAM_INVALID)) {
00118     return true;
00119   }
00120   else {
00121     return false;
00122   }
00123 }
```

4.51 src/tests/teams/teams_tests.hpp File Reference

Contains function declarations for the OpenSHMEM teams tests.

```
#include "routines.hpp"
#include <shmem.h>
#include <iostream>
```

Functions

- bool test_shmem_team_my_pe (void)

  Tests the shmem_team_my_pe() routine.
- bool test_shmem_team_n_pes (void)

  Tests the shmem_team_n_pes() routine.
- bool test_shmem_team_get_config (void)

  Tests the shmem_team_get_config() routine.
- bool test_shmem_team_translate_pe (void)

  Tests the shmem_team_translate_pe() routine.
- bool test_shmem_team_split_strided (void)

  Tests the shmem_team_split_strided() routine.
- bool test_shmem_team_split_2d (void)

  Tests the shmem_team_split_2d() routine.
- bool test_shmem_team_destroy (void)

  Tests the shmem_team_destroy() routine.

4.51.1 Detailed Description

Contains function declarations for the OpenSHMEM teams tests.

Definition in file teams_tests.hpp.

4.51.2 Function Documentation

4.51.2.1 test_shmem_team_destroy()

```
bool test_shmem_team_destroy (
             void )
```

Tests the shmem_team_destroy() routine.

This test verifies that the shmem_team_destroy() function correctly destroys a team.

Returns

True if the test is successful, false otherwise.

Definition at line 113 of file teams_tests.cpp.

```
00113                                  {
00114   shmem_team_t team;
00115   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00116   p_shmem_team_destroy(team);
00117   if (!(team == SHMEM_TEAM_INVALID)) {
00118     return true;
00119   }
00120   else {
00121     return false;
00122   }
00123 }
```

References p_shmem_n_pes, p_shmem_team_destroy, and p_shmem_team_split_strided.

4.51.2.2 test_shmem_team_get_config()

```
bool test_shmem_team_get_config (
            void )
```

Tests the shmem_team_get_config() routine.

This test verifies that the shmem_team_get_config() function correctly retrieves the team configuration.

Returns

True if the test is successful, false otherwise.

Definition at line 45 of file teams_tests.cpp.

```
00045                                       {
00046   shmem_team_t team;
00047   shmem_team_config_t config;
00048   long config_mask = SHMEM_TEAM_NUM_CONTEXTS;
00049   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00050   if (team == SHMEM_TEAM_INVALID) {
00051     return false;
00052   }
00053   p_shmem_team_get_config(team, config_mask, &config);
00054   bool result = (config.num_contexts >= 0);
00055   p_shmem_team_destroy(team);
00056   return result;
00057 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_get_config, and p_shmem_team_split_strided.

4.51.2.3 test_shmem_team_my_pe()

```
bool test_shmem_team_my_pe (
            void )
```

Tests the shmem_team_my_pe() routine.

This test verifies that the shmem_team_my_pe() function returns a valid PE number within the team.

Returns

True if the test is successful, false otherwise.

Definition at line 15 of file teams_tests.cpp.

```
00015                                       {
00016   shmem_team_t team;
00017   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00018   int my_pe = p_shmem_team_my_pe(team);
00019   p_shmem_team_destroy(team);
00020   return (my_pe >= 0);
00021 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_my_pe, and p_shmem_team_split_strided.

4.51.2.4 test_shmem_team_n_pes()

```
bool test_shmem_team_n_pes (
            void )
```

Tests the shmem_team_n_pes() routine.

This test verifies that the shmem_team_n_pes() function returns the correct number of PEs in the team.

Returns

True if the test is successful, false otherwise.

Definition at line 30 of file teams_tests.cpp.

```
00030                                   {
00031   shmem_team_t team;
00032   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00033   int npes = p_shmem_team_n_pes(team);
00034   p_shmem_team_destroy(team);
00035   return (npes == p_shmem_n_pes());
00036 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_strided.

4.51.2.5 test_shmem_team_split_2d()

```
bool test_shmem_team_split_2d (
            void )
```

Tests the shmem_team_split_2d() routine.

This test verifies that the shmem_team_split_2d() function correctly splits a team into two-dimensional subteams.

Returns

True if the test is successful, false otherwise.

Definition at line 96 of file teams_tests.cpp.

```
00096                                       {
00097   shmem_team_t team_x, team_y;
00098   p_shmem_team_split_2d(SHMEM_TEAM_WORLD, 2, NULL, 0, &team_x, NULL, 0, &team_y);
00099   int npes_x = p_shmem_team_n_pes(team_x);
00100   int npes_y = p_shmem_team_n_pes(team_y);
00101   p_shmem_team_destroy(team_x);
00102   p_shmem_team_destroy(team_y);
00103   return (npes_x > 0 && npes_y > 0);
00104 }
```

References p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_2d.

4.51.2.6 test_shmem_team_split_strided()

```
bool test_shmem_team_split_strided (
            void )
```

Tests the shmem_team_split_strided() routine.

This test verifies that the shmem_team_split_strided() function correctly splits a team into subteams.

Returns

True if the test is successful, false otherwise.

Definition at line 81 of file teams_tests.cpp.

```
00081                                           {
00082   shmem_team_t team;
00083   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00084   int npes = p_shmem_team_n_pes(team);
00085   p_shmem_team_destroy(team);
00086   return (npes == p_shmem_n_pes());
00087 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_n_pes, and p_shmem_team_split_strided.

4.51.2.7 test_shmem_team_translate_pe()

```
bool test_shmem_team_translate_pe (
            void )
```

Tests the shmem_team_translate_pe() routine.

This test verifies that the shmem_team_translate_pe() function correctly translates a PE number from one team to another.

Returns

True if the test is successful, false otherwise.

Definition at line 66 of file teams_tests.cpp.

```
00066                                           {
00067   shmem_team_t team;
00068   p_shmem_team_split_strided(SHMEM_TEAM_WORLD, 0, 1, p_shmem_n_pes(), NULL, 0, &team);
00069   int pe_in_team = p_shmem_team_translate_pe(team, 0, SHMEM_TEAM_WORLD);
00070   p_shmem_team_destroy(team);
00071   return (pe_in_team >= 0);
00072 }
```

References p_shmem_n_pes, p_shmem_team_destroy, p_shmem_team_split_strided, and p_shmem_team_translate_pe.

4.52 teams_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef TEAMS_TESTS_HPP
00007 #define TEAMS_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include <shmem.h>
00011 #include <iostream>
00012
00020 bool test_shmem_team_my_pe(void);
00021
00029 bool test_shmem_team_n_pes(void);
00030
00038 bool test_shmem_team_get_config(void);
00039
00047 bool test_shmem_team_translate_pe(void);
00048
00056 bool test_shmem_team_split_strided(void);
00057
00065 bool test_shmem_team_split_2d(void);
00066
00074 bool test_shmem_team_destroy(void);
00075
00076 #endif /* TEAMS_TESTS_HPP */
```

4.53 src/tests/threads/threads_tests.cpp File Reference

Contains OpenSHMEM threads tests.

```
#include "threads_tests.hpp"
```

Functions

- bool test_shmem_init_thread (void)

  Tests the initialization of OpenSHMEM with threading support.
- bool test_shmem_query_thread (void)

  Tests querying the level of threading support in OpenSHMEM.

4.53.1 Detailed Description

Contains OpenSHMEM threads tests.

Definition in file threads_tests.cpp.

4.53.2 Function Documentation

4.53.2.1 test_shmem_init_thread()

```
bool test_shmem_init_thread (
            void )
```

Tests the initialization of OpenSHMEM with threading support.

This test verifies that OpenSHMEM can be initialized with the specified level of threading support.

Returns

True if the initialization with threading support is successful, false otherwise.

Definition at line 15 of file threads_tests.cpp.

```
00015                                    {
00016   int provided;
00017   p_shmem_init_thread(SHMEM_THREAD_MULTIPLE, &provided);
00018   return (provided == SHMEM_THREAD_MULTIPLE);
00019 }
```

References p_shmem_init_thread.

4.53.2.2 test_shmem_query_thread()

```
bool test_shmem_query_thread (
             void )
```

Tests querying the level of threading support in OpenSHMEM.

This test verifies that the `p_shmem_query_thread` function correctly queries the level of threading support provided by OpenSHMEM.

Returns

True if the query is successful and the level of threading support is one of the valid levels, false otherwise.

Definition at line 29 of file threads_tests.cpp.

```
00029                                    {
00030   int provided;
00031   p_shmem_query_thread(&provided);
00032   bool success = (provided == SHMEM_THREAD_SINGLE ||
00033                   provided == SHMEM_THREAD_FUNNELED ||
00034                   provided == SHMEM_THREAD_SERIALIZED ||
00035                   provided == SHMEM_THREAD_MULTIPLE);
00036   return success;
00037 }
```

References p_shmem_query_thread.

4.54 threads_tests.cpp

Go to the documentation of this file.

```
00001
00006 #include "threads_tests.hpp"
00007
00015 bool test_shmem_init_thread(void) {
00016   int provided;
00017   p_shmem_init_thread(SHMEM_THREAD_MULTIPLE, &provided);
00018   return (provided == SHMEM_THREAD_MULTIPLE);
00019 }
00020
00029 bool test_shmem_query_thread(void) {
00030   int provided;
00031   p_shmem_query_thread(&provided);
00032   bool success = (provided == SHMEM_THREAD_SINGLE ||
00033                   provided == SHMEM_THREAD_FUNNELED ||
00034                   provided == SHMEM_THREAD_SERIALIZED ||
00035                   provided == SHMEM_THREAD_MULTIPLE);
00036   return success;
00037 }
```

4.55 src/tests/threads/threads_tests.hpp File Reference

Contains function declarations for the OpenSHMEM threads tests.

```
#include "routines.hpp"
#include "shmemvv.hpp"
#include <shmem.h>
#include <iostream>
```

Functions

- bool test_shmem_init_thread (void)

  Tests the initialization of OpenSHMEM with threading support.
- bool test_shmem_query_thread (void)

  Tests querying the level of threading support in OpenSHMEM.

4.55.1 Detailed Description

Contains function declarations for the OpenSHMEM threads tests.

Definition in file threads_tests.hpp.

4.55.2 Function Documentation

4.55.2.1 test_shmem_init_thread()

```
bool test_shmem_init_thread (
            void )
```

Tests the initialization of OpenSHMEM with threading support.

This test verifies that OpenSHMEM can be initialized with the specified level of threading support.

Returns

True if the initialization with threading support is successful, false otherwise.

Definition at line 15 of file threads_tests.cpp.

```
00015                                    {
00016   int provided;
00017   p_shmem_init_thread(SHMEM_THREAD_MULTIPLE, &provided);
00018   return (provided == SHMEM_THREAD_MULTIPLE);
00019 }
```

References p_shmem_init_thread.

4.55.2.2 test_shmem_query_thread()

```
bool test_shmem_query_thread (
             void )
```

Tests querying the level of threading support in OpenSHMEM.

This test verifies that the `p_shmem_query_thread` function correctly queries the level of threading support provided by OpenSHMEM.

Returns

True if the query is successful and the level of threading support is one of the valid levels, false otherwise.

Definition at line 29 of file threads_tests.cpp.

```
00029                                    {
00030   int provided;
00031   p_shmem_query_thread(&provided);
00032   bool success = (provided == SHMEM_THREAD_SINGLE ||
00033                   provided == SHMEM_THREAD_FUNNELED ||
00034                   provided == SHMEM_THREAD_SERIALIZED ||
00035                   provided == SHMEM_THREAD_MULTIPLE);
00036   return success;
00037 }
```

References p_shmem_query_thread.

4.56 threads_tests.hpp

Go to the documentation of this file.

```
00001
00006 #ifndef THREADS_TESTS_HPP
00007 #define THREADS_TESTS_HPP
00008
00009 #include "routines.hpp"
00010 #include "shmemvv.hpp"
00011
00012 #include <shmem.h>
00013 #include <iostream>
00014
00022 bool test_shmem_init_thread(void);
00023
00033 bool test_shmem_query_thread(void);
00034
00035 #endif /* THREADS_TESTS_HPP */
```

Index