Memory Management Function Replacements of the C and C++ Standard Libraries: Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

Document serial number: 000004892117

C Language Library Replacement Functions

user_malloc_init

Initialize the memory management functions

Definition

```
void user_malloc_init(void)
```

Arguments

None

Return Values

None

Description

This function initializes the memory management functions.

Include the allocation of a heap area, its initialization processing, etc.

This function will be called within the `module_start()` of the C and C++ standard library module. For details on the `module_start()`, refer to the "Kernel Overview" document.

If the initialization fails, call `sceLibcSetHeapInitError()` in <stdlib.h> with a non-zero argument, and return from this function. Then a string like below is output to TTY, and `module_start()` fails:

```
Initialize malloc : Failed (err=0x01234567)
```

Because the initialization of the C and C++ standard libraries will not be completed yet when this function is called, `malloc()` or functions that use the mutex cannot be used.

Only functions of a library module started before the C and C++ standard library module can be used within this function.

TLS variables cannot be accessed from this function.

Notes

The following functions cannot be used.

`atexit()`, file stream functions, `exit()`, `_Exit()`, `abort()`, `assert()`, `sceKernelExitProcess()`, `sceKernelLoadStartModule()`, `sceKernelStopUnloadModule()`

user_malloc_finalize

Terminate the memory management functions

Definition

```
void user_malloc_finalize(void)
```

Arguments

None

Return Values

None

Description

This function terminates the memory management functions.

Include the deletion of the heap area, its termination processing, etc.

This function will be called within the `module_exit()` of the C and C++ standard library module. For details on the `module_exit()`, refer to the "Kernel Overview" document.

Because termination of the C and C++ standard libraries will be ongoing when this function is called, `malloc()` or functions that use the mutex cannot be used.

Only functions of a library module started before the C and C++ standard library module can be used within this function.

Notes

The following functions cannot be used.

`atexit()`, file stream functions, `exit()`, `_Exit()`, `abort()`, `assert()`, `sceKernelExitProcess()`

user_malloc

Allocate area

Definition

```
void *user_malloc(
        size_t size
)
```

Arguments

size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from `malloc()`.

The area to be allocated must be aligned to a 16-byte boundary or more.

If the size is 0, the operation must be the same as when another size other than 0 is requested.

Implement so that this function will be multithread safe.

Notes

A function that calls `malloc()` cannot be used within this function.

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_free

Free area

Definition

```
void user_free(
        void *ptr
)
```

Arguments

ptr Allocated area

Return Values

None

Description

When replacement succeeds, this function will be called from `free()`.

Implement so that this function will be multithread safe.

Notes

A function that calls `free()` cannot be used within this function.

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_calloc

Allocate and zero-initialize area

Definition

```
void *user_calloc(
        size_t nelem,
        size_t size
)
```

Arguments

nelem	Number of elements
size	Size of element

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from `calloc()`.

The area to be allocated must be aligned to a 16-byte boundary or more.

If the size is 0, the operation must be the same as when another size other than 0 is requested.

Implement so that this function will be multithread safe.

Notes

A function that calls `calloc()` cannot be used within this function.

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_realloc

Reallocate area

Definition

```
void *user_realloc(
        void *ptr,
        size_t size
)
```

Arguments

ptr	Allocated area
size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from `realloc()`.

The area to be allocated must be aligned to a 16-byte boundary or more.

If the size is 0, the operation must be the same as when another size other than 0 is requested.

Implement so that this function will be multithread safe.

Notes

A function that calls `realloc()` cannot be used within this function.

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_memalign

Allocate area with alignment specified

Definition

```
void *user_memalign(
        size_t boundary,
        size_t size
)
```

Arguments

boundary	Area's alignment
size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from `memalign()`.
If the boundary is a power of 2, align the area to a multiple of the boundary.
If the boundary is not a power of 2, use the nearest greater power value.
If the boundary is 16 or less, make the boundary 16.
If the size is 0, the operation must be the same as when another size other than 0 is requested.
Implement so that this function will be multithread safe.

Notes

A function that calls `memalign()` cannot be used within this function.

user_reallocalign

Reallocate area with alignment specified

Definition

```
void *user_reallocalign(
        void *ptr,
        size_t size,
        size_t boundary
)
```

Arguments

ptr	Allocated area
size	Size of area
boundary	Area's alignment

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from `reallocalign()`.
If the boundary is a power of 2, align the area to a multiple of the boundary.
If the boundary is not a power of 2, use the nearest greater power value.
If the boundary is less than 16, make the boundary 16.
If the size is 0, the operation must be the same as when another size other than 0 is requested.
Implement so that this function will be multithread safe.

Notes

A function that calls `reallocalign()` cannot be used within this function.

user_malloc_stats

Get memory information

Definition

```
int user_malloc_stats(
     struct malloc_managed_size *mmsize
)
```

Arguments

mmsize Memory information structure

Return Values

Value	Description
0	Succeeded
1	Failed

Description

When replacement succeeds, this function will be called from `malloc_stats()`. Implement so that this function will be multithread safe.

Notes

A function that calls `malloc_stats()` cannot be used within this function.

user_malloc_stats_fast

Get memory information

Definition

```
int user_malloc_stats_fast(
     struct malloc_managed_size *mmsize
)
```

Arguments

mmsize Memory information structure

Return Values

Value	Description
0	Succeeded
1	Failed

Description

When replacement succeeds, this function will be called from `malloc_stats_fast()`. Implement so that this function will be multithread safe.

Notes

A function that calls `malloc_stats_fast()` cannot be used within this function.

user_malloc_usable_size

Get area size

Definition

```
size_t user_malloc_usable_size(
     void *ptr
)
```

Arguments

ptr Allocated area

Return Values

Returns the size of the allocated area.

Description

When replacement succeeds, this function will be called from `malloc_usable_size()`. Implement so that this function will be multithread safe.

Notes

A function that calls `malloc_usable_size()` cannot be used within this function.

C Language Library Replacement Functions (for TLS)

user_malloc_for_tls_init

Initialize the memory management functions for TLS

Definition

```
void user_malloc_for_tls_init(void)
```

Arguments

None

Return Values

None

Description

This function initializes the memory management functions for TLS.

Include the allocation of a heap area, its initialization processing, etc.

This function will be called within the `module_start()` of the C and C++ standard library module. For details on the `module_start()`, refer to the "Kernel Overview" document.

If the initialization fails, call `sceLibcSetHeapInitError()` in <stdlib.h> with a non-zero argument, and return from this function. Then a string like below is output to TTY, and `module_start()` fails:

```
Initialize malloc : Failed (err=0x01234567)
```

Because the initialization of the C and C++ standard libraries will not be completed yet when this function is called, `malloc()` or functions that use the mutex cannot be used.

Only functions of a library module started before the C and C++ standard library module can be used within this function.

TLS variables cannot be accessed from this function.

Notes

The following functions cannot be used.

`atexit()`, file stream functions, `exit()`, `_Exit()`, `abort()`, `assert()`, `sceKernelExitProcess()`, `sceKernelLoadStartModule()`, `sceKernelStopUnloadModule()`

user_malloc_for_tls_finalize

Terminate the memory management functions for TLS

Definition

```
void user_malloc_for_tls_finalize(void)
```

Arguments

None

Return Values

None

Description

This function terminates the memory management functions for TLS.

Include the deletion of the heap area, its termination processing, etc.

This function will be called within the `module_exit()` of the C and C++ standard library module. For details on the `module_exit()`, refer to the "Kernel Overview" document.

Because termination of the C and C++ standard libraries will be ongoing when this function is called, `malloc()` or functions that use the mutex cannot be used.

Only functions of a library module started before the C and C++ standard library module can be used within this function.

TLS variables cannot be accessed from this function.

Notes

The following functions cannot be used.

`atexit()`, file stream functions, `exit()`, `_Exit()`, `abort()`, `assert()`, `sceKernelExitProcess()`

user_malloc_for_tls

Allocate area for TLS

Definition

```
void *user_malloc_for_tls(
        size_t size
)
```

Arguments

size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called as a TLS memory allocator.

The area to be allocated must be aligned to a 16-byte boundary or more.

If the size is 0, the operation must be the same as when another size other than 0 is requested.

Implement so that this function will be multithread safe.

TLS variables cannot be accessed from this function.

Notes

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_free_for_tls

Free area for TLS

Definition

```
void user_free_for_tls(
        void *ptr
)
```

Arguments

ptr Allocated area

Return Values

None

Description

When replacement succeeds, this function will be called as a TLS memory deallocator.

Implement so that this function will be multithread safe.

TLS variables cannot be accessed from this function.

Notes

Follow the specifications in "7.20.3 Memory management functions" of "ISO/IEC Standard 9899:1999" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

Document serial number: 000004892117

C++ Language Library Replacement Functions

user_new

Allocate area

Definition

```
void *user_new(
     std::size_t size
) throw(std::bad_alloc)
```

Arguments

size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded

Description

When replacement succeeds, this function will be called from the `new` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `new` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_new(nothrow)

Allocate area

Definition

```
void *user_new(
     std::size_t size,
     const std::nothrow_t& x
) throw()
```

Arguments

size	Size of area
x	ID for `nothrow`

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from the `new(nothrow)` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `new(nothrow)` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_new_array

Allocate area

Definition

```
void *user_new_array(
     std::size_t size
) throw(std::bad_alloc)
```

Arguments

size	Size of area

Return Values

Value	Description
Other than NULL	Succeeded

Description

When replacement succeeds, this function will be called from the `new[]` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `new[]` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_new_array(nothrow)

Allocate area

Definition

```
void *user_new_array(
     std::size_t size,
     const std::nothrow_t& x
) throw()
```

Arguments

size	Size of area
x	ID for `nothrow`

Return Values

Value	Description
Other than NULL	Succeeded
NULL	Failed

Description

When replacement succeeds, this function will be called from the `new[](nothrow)` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `new[](nothrow)` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_delete

Free area

Definition

```
void user_delete(
     void *ptr
) throw()
```

Arguments

ptr Allocated area

Return Values

None

Description

When replacement succeeds, this function will be called from the `delete` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `delete` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_delete(nothrow)

Free area

Definition

```
void user_delete(
     void *ptr,
     const std::nothrow_t& x
) throw()
```

Arguments

ptr	Allocated area
x	ID for `nothrow`

Return Values

None

Description

When replacement succeeds, this function will be called from the `delete(nothrow)` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `delete(nothrow)` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_delete_array

Free area

Definition

```
void user_delete_array(
     void *ptr
) throw()
```

Arguments

ptr Allocated area

Return Values

None

Description

When replacement succeeds, this function will be called from the `delete[]` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `delete[]` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)

user_delete_array(nothrow)

Free area

Definition

```
void user_delete_array(
     void *ptr,
     const std::nothrow_t& x
) throw()
```

Arguments

ptr	Allocated area
x	ID for `nothrow`

Return Values

None

Description

When replacement succeeds, this function will be called from the `delete[](nothrow)` operator.

Implement so that this function will be multithread safe.

Create with both RTTI and exception features enabled.

Notes

A function that calls the `delete[](nothrow)` operator cannot be used within this function.

Follow the specifications in "18.4 Dynamic memory management" of "ISO/IEC Standard 14882:2003" (can be purchased at http://www.iso.org/iso/home.htm).

(The above reference was available on December 5, 2012. Note, however, that it is possible for the applicable page to be moved or modified.)