

Object manager server (om_server) library for sos

Generated by Doxygen 1.7.1

Tue Aug 2 2011 14:11:30

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	2
2.1	File List	2
3	Data Structure Documentation	2
3.1	om_address_space_t Struct Reference	2
3.1.1	Detailed Description	2
3.2	om_async_endpoint_t Struct Reference	2
3.2.1	Detailed Description	3
3.3	om_contiguous_mem_t Struct Reference	3
3.3.1	Detailed Description	3
3.4	om_device_frame_t Struct Reference	3
3.4.1	Detailed Description	3
3.5	om_endpoint_t Struct Reference	3
3.5.1	Detailed Description	3
3.6	om_frame_t Struct Reference	4
3.6.1	Detailed Description	4
3.7	om_tcb_t Struct Reference	4
3.7.1	Detailed Description	4
4	File Documentation	4
4.1	om_server.h File Reference	4
4.1.1	Detailed Description	8
4.1.2	Define Documentation	9
4.1.3	Enumeration Type Documentation	9
4.1.4	Function Documentation	9
4.1.5	Variable Documentation	23
4.2	om_serverAPI.h File Reference	23
4.2.1	Detailed Description	23

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

om_address_space_t (An <code>om_address_space_t</code> corresponds with an seL4 Page Directory, the top level of a two-level page table)	2
om_async_endpoint_t (An <code>om_async_endpoint_t</code> represents a seL4 asynchronous endpoint)	2
om_contiguous_mem_t (An <code>om_contiguous_mem_t</code> object represents a region of contiguous physical memory)	3
om_device_frame_t (An <code>om_device_frame_t</code> represents one physical frame of a device)	3
om_endpoint_t (An <code>om_endpoint_t</code> represents a seL4 synchronous endpoint)	3
om_frame_t (An <code>om_frame_t</code> represents one frame of physical memory)	4
om_tcb_t (An <code>om_tcb_t</code> corresponds with an seL4 thread control block (TCB))	4

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

om_server.h (Interface for accessing the object manager server)	4
om_serverAPI.h (Constants to identify API IPCs to the <code>om_server</code>)	23

3 Data Structure Documentation

3.1 `om_address_space_t` Struct Reference

An `om_address_space_t` corresponds with an seL4 Page Directory, the top level of a two-level page table.

3.1.1 Detailed Description

An `om_address_space_t` corresponds with an seL4 Page Directory, the top level of a two-level page table. Each thread that you create is mapped into an `om_address_space_t`. Each process or server will have its own `om_address_space_t`. Map `om_frame_t` and `om_device_frame_t` objects into address spaces to allow threads in that address space to access those objects.

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

- [om_server.h](#)

3.2 `om_async_endpoint_t` Struct Reference

An `om_async_endpoint_t` represents a seL4 asynchronous endpoint.

3.2.1 Detailed Description

An `om_async_endpoint_t` represents a seL4 asynchronous endpoint. `om_async_endpoints` are used for interrupt handling and can also be used for locking. `libsel4sync` provides a basic locking library that uses asynchronous endpoints to implement some lock primitives.

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

- [om_server.h](#)

3.3 `om_contiguous_mem_t` Struct Reference

An `om_contiguous_mem_t` object represents a region of contiguous physical memory.

3.3.1 Detailed Description

An `om_contiguous_mem_t` object represents a region of contiguous physical memory. These objects are required for DMA on regions larger than one frame.

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

- [om_server.h](#)

3.4 `om_device_frame_t` Struct Reference

An `om_device_frame_t` represents one physical frame of a device.

3.4.1 Detailed Description

An `om_device_frame_t` represents one physical frame of a device. These frames are set up for un-cached access.

Device frames correspond to the physical address of that device on the hardware. Depending on the size of a device, one `device_frame_t` may contain several devices, or one device may be made up of many `device_frame_t` objects.

Map these into an `om_address_space_t` to access the memory from that a thread mapped into that address space.

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

- [om_server.h](#)

3.5 `om_endpoint_t` Struct Reference

An `om_endpoint_t` represents a seL4 synchronous endpoint.

3.5.1 Detailed Description

An `om_endpoint_t` represents a seL4 synchronous endpoint. When creating a thread, you need to pass in a synchronous endpoint to act as the fault endpoint for that thread.

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

- [om_server.h](#)

3.6 `om_frame_t` Struct Reference

An `om_frame_t` represents one frame of physical memory.

3.6.1 Detailed Description

An `om_frame_t` represents one frame of physical memory. However, the `.address` attribute of the frame does not correspond to the physical address of the frame - it is just a reference for the `om_server` to use in its own book keeping.

Map `om_frame_t` objects into an `om_address_space_t` to be able to access them from a thread within that address space.

Whilst there are methods for retrieving the actual physical address of an `om_frame_t`, this should only be necessary for DMA operations.

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

- [om_server.h](#)

3.7 `om_tcb_t` Struct Reference

An `om_tcb_t` corresponds with an seL4 thread control block (TCB).

3.7.1 Detailed Description

An `om_tcb_t` corresponds with an seL4 thread control block (TCB). Use `om_tcb_t` objects to create and control individual threads.

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

- [om_server.h](#)

4 File Documentation

4.1 `om_server.h` File Reference

Interface for accessing the object manager server.

Data Structures

- struct [om_address_space_t](#)
An `om_address_space_t` corresponds with an seL4 Page Directory, the top level of a two-level page table.
- struct [om_frame_t](#)

An `om_frame_t` represents one frame of physical memory.

- struct `om_device_frame_t`
An `om_device_frame_t` represents one physical frame of a device.
- struct `om_endpoint_t`
An `om_endpoint_t` represents a seL4 synchronous endpoint.
- struct `om_async_endpoint_t`
An `om_async_endpoint_t` represents a seL4 asynchronous endpoint.
- struct `om_tcb_t`
An `om_tcb_t` corresponds with an seL4 thread control block (TCB).
- struct `om_contiguous_mem_t`
An `om_contiguous_mem_t` object represents a region of contiguous physical memory.

Defines

- #define `PAGESIZE` (1 << (seL4_PageBits))
Size in bytes of pages/frames as understood by the server.
- #define `IS_PAGESIZE_ALIGNED(x)` (((x) & (PAGESIZE - 1)) == 0)
Checks if a given address is aligned to the page size.
- #define `PAGE_ALIGN(x)` ((x) & ~(PAGESIZE - 1))
Rounds a virtual address down to the nearest page boundary.
- #define `MAX_TCB_PRIORITY` 100
The highest priority a thread may be set to.
- #define `USER_ENDPOINT_SLOT` (1 <<< 22)
Default `om_server` endpoint address.

Enumerations

- enum `om_error`
An extended version of seL4 error codes for the `om_server`.

Functions

- seL4_Word `om_available_memory` ()
Retrieves available memory, in bytes.
- `om_address_space_t` `om_new_address_space` ()
Create a new address space.

- seL4_Word [om_free_address_space](#) (om_address_space_t address_space)
Frees an address space and all memory associated with it.
- om_address_space_t [om_self_address_space](#) ()
Return the address space for the current thread.
- om_frame_t [om_new_frame](#) ()
Create a new, unmapped frame.
- seL4_Word [om_map_frame](#) (om_address_space_t address_space, om_frame_t frame, seL4_Word vaddr, seL4_CapRights rights)
Map frame into address space at vaddr with rights.
- seL4_Word [om_unmap_frame](#) (om_address_space_t address_space, om_frame_t frame, seL4_Word vaddr)
Unmap frame that is mapped to vaddr from address space.
- seL4_Word [om_frame_paddr](#) (om_frame_t frame)
Retrieve the hardware physical address of a frame.
- seL4_Word [om_flush_frame](#) (om_frame_t frame)
Flush a frame from the cache.
- seL4_Word [om_free_frame](#) (om_frame_t frame)
Frees a frame and all memory associated with it.
- seL4_Word [om_num_device_frames](#) (seL4_Word paddr)
Return the number of frames in the device at paddr.
- om_device_frame_t [om_get_device_frame](#) (seL4_Word paddr, unsigned int offset)
Return the device frame at paddr + the offset in frames.
- seL4_Word [om_map_device_frame](#) (om_address_space_t address_space, om_device_frame_t device_frame, seL4_Word vaddr, seL4_CapRights rights)
Map device frame into address space at vaddr with rights.
- seL4_Word [om_unmap_device_frame](#) (om_address_space_t address_space, om_device_frame_t device_frame, seL4_Word vaddr)
Unmap device frame that is mapped to vaddr from address space.
- seL4_Word [om_free_device_frame](#) (om_device_frame_t device_frame)
Frees a device frame.
- om_tcb_t [om_new_tcb](#) (om_address_space_t address_space, om_endpoint_t fault_endpoint, seL4_CapData badge, int priority, om_frame_t ipc_buffer_frame, seL4_Word ipc_buffer_frame_vaddr)
Create and initialise a new tcb.
- seL4_Word [om_start_thread](#) (om_tcb_t tcb, seL4_Word stack, seL4_Word entry_point, seL4_Word arg)

Start a thread/tcb running.

- seL4_Word [om_pause_thread](#) (om_tcb_t tcb)
Pause a thread.
- seL4_Word [om_resume_thread](#) (om_tcb_t tcb)
Resume a thread.
- seL4_Word [om_set_thread_spip](#) (om_tcb_t tcb, seL4_Word sp, seL4_Word ip)
Change the stack pointer (sp) and instruction pointer (ip) of a thread.
- seL4_Word [om_get_thread_spip](#) (om_tcb_t tcb, seL4_Word *sp, seL4_Word *ip)
Read a the stack pointer (sp) and instruction pointer (ip) of a thread.
- seL4_Word [om_free_tcb](#) (om_tcb_t tcb)
Frees a tcb and all memory associated with it.
- om_tcb_t [om_init_thread_tcb](#) ()
Returns the TCB of the initial thread.
- om_endpoint_t [om_new_endpoint](#) ()
Create a new synchronous endpoint.
- seL4_CPtr [om_map_endpoint](#) (om_address_space_t address_space, om_endpoint_t endpoint, seL4_CapRights rights, seL4_CapData badge)
Map an endpoint to an address space.
- seL4_Word [om_unmap_endpoint](#) (om_address_space_t address_space, om_endpoint_t endpoint, seL4_CPtr cptr)
Unmap an endpoint from an address space.
- seL4_Word [om_free_endpoint](#) (om_endpoint_t endpoint)
Free an endpoint and all memory associated with it.
- om_endpoint_t [om_init_thread_endpoint](#) ()
Get the endpoint of the initial thread that om_server listens to.
- om_async_endpoint_t [om_new_async_endpoint](#) ()
Create a new asynchronous endpoint.
- seL4_CPtr [om_map_async_endpoint](#) (om_address_space_t address_space, om_async_endpoint_t endpoint, seL4_CapRights rights, seL4_CapData badge)
Map an endpoint to an address space.
- seL4_Word [om_unmap_async_endpoint](#) (om_address_space_t address_space, om_async_endpoint_t endpoint, seL4_CPtr cptr)
Unmap an endpoint from an address space.
- seL4_Word [om_free_async_endpoint](#) (om_async_endpoint_t endpoint)
Free an endpoint and all memory associated with it.

- seL4_IRQHandler [om_new_interrupt](#) (int irq, om_tcb_t thread)
Register an interrupt to be delivered to the specified thread.
- om_async_endpoint_t [om_get_interrupt_endpoint](#) (om_tcb_t thread)
Retrieve the interrupt handler for the specified thread.
- om_contiguous_mem_t [om_new_contiguous_mem](#) (seL4_Word size)
Create a new contiguous memory region.
- seL4_Word [om_map_contiguous_mem](#) (om_address_space_t address_space, om_contiguous_mem_t mem, seL4_Word vaddr, seL4_CapRights rights)
Map a contiguous region into an address space.
- seL4_Word [om_unmap_contiguous_mem](#) (om_address_space_t address_space, om_contiguous_mem_t mem, seL4_Word vaddr)
Unmap contiguous region that is mapped to vaddr from address space.
- seL4_Word [om_get_contiguous_mem_paddr](#) (om_contiguous_mem_t mem)
Retrieve the hardware physical address of the contiguous region.
- seL4_Word [om_flush_contiguous_mem](#) (om_contiguous_mem_t mem, seL4_Word paddr)
Flush a frame in a contiguous region.
- seL4_Word [om_free_contiguous_mem](#) (om_contiguous_mem_t mem)
Frees a contiguous region and all memory associated with it.
- seL4_CPtr [om_new_cslot](#) (om_address_space_t address_space)
Allocate a new cslot in an address space.
- seL4_Word [om_free_cslot](#) (om_address_space_t address_space, seL4_CPtr cslot)
Free a cslot.
- void [om_save_reply_cap](#) (om_address_space_t as, seL4_CPtr slot)
Save current reply cap in a cslot.

Variables

- seL4_CPtr [__om_server_endpoint](#)
Address of endpoint used by all om_server functions.

4.1.1 Detailed Description

Interface for accessing the object manager server. This server provides object and cspace management through an IPC interface. The functions declared here are small wrapper functions that send their arguments over IPC to the om_server.

The om_server can serve requests to the initial application that it boots up (sos) and any application that has been configured to use the [om_init_thread_endpoint](#) with the badge set to the [om_address_space_t.address](#) of that application.

To allow the type system to catch simple programming errors the structs [om_address_space_t](#), [om_frame_t](#), [om_device_frame_t](#), [om_endpoint_t](#), [om_async_endpoint_t](#), [om_tcb_t](#), [om_contiguous_mem_t](#) are just wrappers around a [seL4_Word](#) that is the identifier for that object.

4.1.2 Define Documentation

4.1.2.1 #define MAX_TCB_PRIORITY 100

The highest priority a thread may be set to.

This is also the priority that sos is started at.

4.1.2.2 #define USER_ENDPOINT_SLOT (1<<22)

Default om_server endpoint address.

When om_server starts the first sos thread, it places the endpoint to be used for IPC to the om_server at this location in the CSpace. This is also the endpoint of the fault handler for the initial thread. To allow additional threads in the sos address space to use this endpoint slot they must set their fault endpoint to [om_init_thread_endpoint](#)

4.1.3 Enumeration Type Documentation

4.1.3.1 enum om_error

An extended version of sel4 error codes for the om_server.

Note that after each om_server call, message register 0 has an error code in it

4.1.4 Function Documentation

4.1.4.1 seL4_Word om_available_memory ()

Retrieves available memory, in bytes.

Queries the om_server for how many bytes of memory it has remaining. This memory pool is the same memory pool needed by the om_server to perform internal book keeping of objects and serves as an upper bound / close approximation of how much memory sos can allocate.

This is merely a total count and ignores fragmentation, just because it reports 4k of available memory does not mean a 4k object (such as a frame) can be allocated as it may be in 4 1k blocks.

Returns

Bytes of free memory available.

4.1.4.2 om_address_space_t om_new_address_space ()

Create a new address space.

Returns

Identifier for new address space. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.3 seL4_Word om_free_address_space (om_address_space_t *address_space*)

Frees an address space and all memory associated with it.

An address space can only be freed if all pages and endpoints have been unmapped from it.

Parameters

address_space Identifier of the address space to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.4 om_address_space_t om_self_address_space ()

Return the address space for the current thread.

Returns

Address space identifier of the current thread.

4.1.4.5 om_frame_t om_new_frame ()

Create a new, unmapped frame.

Returns

Identifier for new frame. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.6 seL4_Word om_map_frame (om_address_space_t *address_space*, om_frame_t *frame*, seL4_Word *vaddr*, seL4_CapRights *rights*)

Map frame into address space at *vaddr* with *rights*.

Parameters

address_space Address space to map the frame into
frame Identifier of the frame to map in
vaddr Virtual address at which to map the frame
rights Access rights to the frame

Returns

[om_error](#) code on failure (0 on success).

4.1.4.7 seL4_Word om_unmap_frame (om_address_space_t *address_space*, om_frame_t *frame*, seL4_Word *vaddr*)

Unmap frame that is mapped to *vaddr* from address space.

A frame may be mapped into the same address space at multiple *vaddr*s so the *vaddr* of the mapping to remove needs to be specified to uniquely identify it.

Parameters

address_space Address space to unmap the frame from
frame Identifier of the frame to be unmapping
vaddr Virtual address of the mapping to remove

Returns

[om_error](#) code on failure (0 on success).

4.1.4.8 seL4_Word om_frame_paddr (om_frame_t *frame*)

Retrieve the hardware physical address of a frame.

A frame can not be directly accessed by physical address, but operations such as DMA require knowing where a frame really is in memory.

Parameters

frame Identifier of the frame to retrieve physical address of

Returns

Physical address of the frame

4.1.4.9 seL4_Word om_flush_frame (om_frame_t *frame*)

Flush a frame from the cache.

Performs a cache clean followed by an invalidate.

Parameters

frame Identifier of the frame to flush

Returns

[om_error](#) code on failure (0 on success).

4.1.4.10 seL4_Word om_free_frame (om_frame_t *frame*)

Frees a frame and all memory associated with it.

Removes any mappings this frame may have and then returns the memory to the available pool.

Parameters

frame Identifier of the frame to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.11 seL4_Word om_num_device_frames (seL4_Word *paddr*)

Return the number of frames in the device at *paddr*.

Devices are identified by the physical address in memory they are located at. This function will return how many frames the om_server thinks the device takes (in general if you know the *paddr* to perform the query you should already know how many frames it takes up).

Parameters

paddr Physical address of the requested device

Returns

Number of frames used by the device.

4.1.4.12 om_device_frame_t om_get_device_frame (seL4_Word *paddr*, unsigned int *offset*)

Return the device frame at *paddr* + the *offset* in frames.

An *offset* of 0 gives the first frame for a device, *offset* of 1 gives the second etc

Parameters

paddr Physical address of the requested device

offset Index of the frame to retrieve

Returns

Identifier of a device frame. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.13 seL4_Word om_map_device_frame (om_address_space_t *address_space*, om_device_frame_t *device_frame*, seL4_Word *vaddr*, seL4_CapRights *rights*)

Map device frame into address space at *vaddr* with rights.

Device frames are mapped in uncached.

Parameters

address_space Address space to map the frame into
device_frame Identifier of the device frame to map in
vaddr Virtual address at which to map the frame
rights Access rights to the frame

Returns

[om_error](#) code on failure (0 on success).

4.1.4.14 seL4_Word om_unmap_device_frame (om_address_space_t *address_space*, om_device_frame_t *device_frame*, seL4_Word *vaddr*)

Unmap device frame that is mapped to *vaddr* from address space.

A device frame may be mapped into the same address space at multiple *vaddrs* so the *vaddr* of the mapping to remove needs to be specified to uniquely identify it.

Parameters

address_space Address space to unmap the frame from
device_frame Identifier of the device frame to be unmapping
vaddr Virtual address of the mapping to remove

Returns

[om_error](#) code on failure (0 on success).

4.1.4.15 seL4_Word om_free_device_frame (om_device_frame_t *device_frame*)

Frees a device frame.

Removes any mappings this device frame may have. While no memory is actually freed the device frame identifier is no longer valid after calling this function.

Parameters

device_frame Identifier of the device frame to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.16 `om_tcb_t om_new_tcb (om_address_space_t address_space, om_endpoint_t fault_endpoint, seL4_CapData badge, int priority, om_frame_t ipc_buffer_frame, seL4_Word ipc_buffer_frame_vaddr)`

Create and initialise a new tcb.

The priority of any created thread is limited to [MAX_TCB_PRIORITY](#).

Parameters

address_space Address space for the tcb to reside in

fault_endpoint The endpoint that faults for this tcb will be sent to. The cptr to this fault endpoint in the specified *address_space* is [USER_ENDPOINT_SLOT](#).

badge The badge that will be assigned to the fault endpoint

priority The priority for the thread

ipc_buffer_frame A frame that has been mapped to *ipc_buffer_frame_vaddr* in the specified *address_space*

ipc_buffer_frame_vaddr The virtual address of the ipc buffer frame in the context of the specified *address_space*

Returns

Identifier of the new tcb. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.17 `seL4_Word om_start_thread (om_tcb_t tcb, seL4_Word stack, seL4_Word entry_point, seL4_Word arg)`

Start a thread/tcb running.

A thread is assumed to have an entry point of the form `void entry_point(seL4_Word arg);`

Parameters

tcb Identifier of the tcb to start

stack Address of the stack

entry_point Entry point of the thread

arg Argument to be passed to new thread (on arm this is merely placed in r0)

Returns

[om_error](#) code on failure (0 on success).

4.1.4.18 `seL4_Word om_pause_thread (om_tcb_t tcb)`

Pause a thread.

Parameters

tcb Identifier of the tcb to pause

Returns

[om_error](#) code on failure (0 on success).

4.1.4.19 seL4_Word om_resume_thread (om_tcb_t tcb)

Resume a thread.

Parameters

tcb Identifier of the tcb to resume

Returns

[om_error](#) code on failure (0 on success).

4.1.4.20 seL4_Word om_set_thread_spip (om_tcb_t tcb, seL4_Word sp, seL4_Word ip)

Change the stack pointer (sp) and instruction pointer (ip) of a thread.

Parameters

tcb Identifier of the tcb to modify.

sp New stack pointer

ip New instruction pointer

Returns

[om_error](#) code on failure (0 on success).

4.1.4.21 seL4_Word om_get_thread_spip (om_tcb_t tcb, seL4_Word * sp, seL4_Word * ip)

Read a the stack pointer (sp) and instruction pointer (ip) of a thread.

Parameters

tcb Identifier of the tcb to read from

sp Address to store the read stack pointer

ip Address to store the read instruction pointer

Returns

[om_error](#) code on failure (0 on success).

4.1.4.22 seL4_Word om_free_tcb (om_tcb_t tcb)

Frees a tcb and all memory associated with it.

Parameters

tcb Identifier of the tcb to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.23 om_tcb_t om_init_thread_tcb ()

Returns the TCB of the initial thread.

This is the thread started at main() in sos and is located inside the [om_init_thread_address_space](#) address space

Returns

Identifier of the initial sos tcb

4.1.4.24 om_endpoint_t om_new_endpoint ()

Create a new synchronous endpoint.

Returns

Identifier of new endpoint. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.25 seL4_CPtr om_map_endpoint (om_address_space_t address_space, om_endpoint_t endpoint, seL4_CapRights rights, seL4_CapData badge)

Map an endpoint to an address space.

Once mapped in the returned cptr can be used by any thread in the specified address_space in any of the seL4 IPC system calls (seL4_Call, seL4_Wait, etc) as determined by the given access rights.

Parameters

address_space Identifier of the address space to map the endpoint into

endpoint Identifier of the endpoint to map in

rights Access rights to the endpoint mapping

badge Badge for the new endpoint mapping

Returns

A capability that the endpoint can be referenced by. Returns 0 on error - the [om_error](#) code can be read from message register 0

4.1.4.26 `seL4_Word om_unmap_endpoint (om_address_space_t address_space, om_endpoint_t endpoint, seL4_CPtr cptr)`

Unmap an endpoint from an address space.

An endpoint may be mapped multiple times into a single address space so the cptr as given by [om_map_endpoint](#) must also be given to uniquely identify the endpoint mapping to remove.

Parameters

address_space Identifier of the address space to remove the mapping from

endpoint Identifier of the endpoint to unmap

cptr Address of the mapping to remove

Returns

[om_error](#) code on failure (0 on success).

4.1.4.27 `seL4_Word om_free_endpoint (om_endpoint_t endpoint)`

Free an endpoint and all memory associated with it.

Removes all mappings that may have been made.

Parameters

endpoint Identifier of the endpoint to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.28 `om_endpoint_t om_init_thread_endpoint ()`

Get the endpoint of the initial thread that om_server listens to.

This is the endpoint that is located at [USER_ENDPOINT_SLOT](#) in the initial sos thread and used for fault handling by that thread.

If mapped into any other address space the badge should be set to the [om_address_space.address](#) of that address space.

It is recommended that every thread created in sos (that is the [om_init_thread_address_space](#) use this as their fault handler otherwise they will not be able to use functions in this library to access the om_server.

You should **not** call `om_free_endpoint` on the returned endpoint.

If sos wishes to give another process access to the om_server then when performing `om_map_endpoint` the badge needs to be set to identify this process to the om_server. You **MUST** set the badge to the `om_address_space.address` of the process that you are giving access to the om_server, or the om_server functions will not work.

Returns

Identifier of the initial thread's endpoint. .address is 0 on failure - the `om_error` code can be read from message register 0

4.1.4.29 om_async_endpoint_t om_new_async_endpoint ()

Create a new asynchronous endpoint.

Returns

Identifier of new async endpoint. .address is 0 on failure - the `om_error` code can be read from message register 0

4.1.4.30 seL4_CPtr om_map_async_endpoint (om_address_space_t address_space, om_async_endpoint_t endpoint, seL4_CapRights rights, seL4_CapData badge)

Map an endpoint to an address space.

Once mapped in the returned cptr can be used by any thread in the specified address_space in any of the seL4 IPC system calls (seL4_Call, seL4_Wait, etc) as determined by the given access rights.

Parameters

address_space Identifier of the address space to map the endpoint into
endpoint Identifier of the endpoint to map in
rights Access rights to the endpoint mapping
badge Badge for the new endpoint mapping

Returns

A capability that the endpoint can be referenced by. Returns 0 on error - the `om_error` code can be read from message register 0

4.1.4.31 seL4_Word om_unmap_async_endpoint (om_address_space_t address_space, om_async_endpoint_t endpoint, seL4_CPtr cptr)

Unmap an endpoint from an address space.

An endpoint may be mapped multiple times into a single address space so the cptr as given by `om_map_endpoint` must also be given to uniquely identify the endpoint mapping to remove.

Parameters

address_space Identifier of the address space to remove the mapping from

endpoint Identifier of the endpoint to unmap

ptr Address of the mapping to remove

Returns

`om_error` code on failure (0 on success).

4.1.4.32 `seL4_Word om_free_async_endpoint (om_async_endpoint_t endpoint)`

Free an endpoint and all memory associated with it.

Removes all mappings that may have been made.

Parameters

endpoint Identifier of the endpoint to free

Returns

`om_error` code on failure (0 on success).

4.1.4.33 `seL4_IRQHandler om_new_interrupt (int irq, om_tcb_t thread)`

Register an interrupt to be delivered to the specified thread.

An async endpoint is created behind the scenes to bind the interrupt to. This endpoint is bound to the thread using the `seL4_TCB_BindAEP` such that any wait on a synchronous endpoint by the specified thread could result in the interrupt being delivered.

It is recommended that an IRQ be acked after binding it to ensure any missed interrupts are acked.

Parameters

irq IRQ number of the interrupt to receive

thread Thread to bind the interrupt to

Returns

A `seL4_IRQHandler` that can be used in calls to `seL4_IRQHandler_Ack`. Use of any other `seL4` syscalls that use `seL4_IRQHandler` objects is possible but will confuse `om_server` and aren't recommended. 0 on failure.

4.1.4.34 `om_async_endpoint_t om_get_interrupt_endpoint (om_tcb_t thread)`

Retrieve the interrupt handler for the specified thread.

Every thread that receives interrupts has a single async endpoint that interrupts are delivered to. This async endpoint can only be used in `seL4` IPC calls by the thread that owns it.

Parameters

thread Identifier of the thread to retrieve interrupt endpoint for

Returns

Identifier of interrupt async endpoint. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.35 `om_contiguous_mem_t om_new_contiguous_mem (seL4_Word size)`

Create a new contiguous memory region.

Contiguous memory regions are guaranteed to occupy a contiguous region of physical memory. This is necessary if you are DMA transfers > 1 page in size

Parameters

size Size in bytes of contiguous region. Rounded up to nearest [PAGESIZE](#)

Returns

Identifier of contiguous memory region. .address is 0 on failure - the [om_error](#) code can be read from message register 0

4.1.4.36 `seL4_Word om_map_contiguous_mem (om_address_space_t address_space, om_contiguous_mem_t mem, seL4_Word vaddr, seL4_CapRights rights)`

Map a contiguous region into an address space.

Parameters

address_space Identifier to the address space to map into

mem Identifier of the contiguous memory region to map in

vaddr Virtual address to start mapping memory in at

rights Access rights of the mapping

Returns

[om_error](#) code on failure (0 on success).

4.1.4.37 `seL4_Word om_unmap_contiguous_mem (om_address_space_t address_space, om_contiguous_mem_t mem, seL4_Word vaddr)`

Unmap contiguous region that is mapped to vaddr from address space.

A contiguous region may be mapped into the same address space at multiple vaddrs so the vaddr of the mapping to remove needs to be specified to uniquely identify it.

Parameters

address_space Address space to unmap the contiguous region from

mem Identifier of the contiguous region to be unmapping

vaddr Virtual address of the mapping to remove

Returns

[om_error](#) code on failure (0 on success).

4.1.4.38 seL4_Word om_get_contiguous_mem_paddr (om_contiguous_mem_t mem)

Retrieve the hardware physical address of the contiguous region.

A frame can not be directly accessed by physical address, but operations such as DMA require knowing where a frame really is in memory.

Parameters

mem Identifier of the contiguous region to retrieve physical address of

Returns

Physical address of the start of the contiguous memory region

4.1.4.39 seL4_Word om_flush_contiguous_mem (om_contiguous_mem_t mem, seL4_Word paddr)

Flush a frame in a contiguous region.

Performs a cache clean followed by an invalidate on the frame containing the specified physical address

Parameters

mem Identifier of the contiguous region to flush in

paddr Physical address of the frame to flush

Returns

[om_error](#) code on failure (0 on success).

4.1.4.40 seL4_Word om_free_contiguous_mem (om_contiguous_mem_t mem)

Frees a contiguous region and all memory associated with it.

Removes any mappings this contiguous region may have and then returns the memory to the available pool.

Parameters

mem Identifier of the contiguous region to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.41 seL4_CPtr om_new_cslot (om_address_space_t address_space)

Allocate a new cslot in an address space.

This function may be called from an endpoint that does not have a badge of 0. If done so the address_space is ignored and the slot is created in the callers address space.

Parameters

address_space Identifier of the address space to allocate cslot in

Returns

CPtr of the allocated slot in the specified address space

4.1.4.42 seL4_Word om_free_cslot (om_address_space_t address_space, seL4_CPtr cslot)

Free a cslot.

This function may be called from an endpoint that does not have a badge of 0. If done so the address_space is ignored and the slot being returned is assumed to reside in the callers cspace.

Parameters

address_space Identifier of the address space to free the cslot from

cslot Slot to free

Returns

[om_error](#) code on failure (0 on success).

4.1.4.43 void om_save_reply_cap (om_address_space_t as, seL4_CPtr slot)

Save current reply cap in a cslot.

This is purely a helper function and uses the [om_new_cslot](#) and [om_free_cslot](#) to perform the actual work. As such this can be called from any process.

Parameters

as the address space to save the reply cap to. Note that this will generally be the address space that the IPC was made to, but not necessarily.

slot the cslot to store the reply cap in. You can allocate these with [om_new_cslot](#).

4.1.5 Variable Documentation

4.1.5.1 seL4_CPtr __om_server_endpoint

Address of endpoint used by all om_server functions.

Initially this is set to [USER_ENDPOINT_SLOT](#), however this is only correct in the CSpace of sos. If another process has been given access to om_server this value needs to be changed to reflect the location that the endpoint was mapped at.

4.2 om_serverAPI.h File Reference

constants to identify API IPCs to the om_server.

4.2.1 Detailed Description

constants to identify API IPCs to the om_server.

Index

`__om_server_endpoint`
 [om_server.h, 22](#)

`MAX_TCB_PRIORITY`
 [om_server.h, 8](#)

`om_address_space`, [1](#)

`om_async_endpoint`, [2](#)

`om_available_memory`
 [om_server.h, 8](#)

`om_contiguous_mem`, [2](#)

`om_device_frame`, [2](#)

`om_endpoint`, [2](#)

`om_error`
 [om_server.h, 8](#)

`om_flush_contiguous_mem`
 [om_server.h, 20](#)

`om_flush_frame`
 [om_server.h, 10](#)

`om_frame`, [3](#)

`om_frame_paddr`
 [om_server.h, 10](#)

`om_free_address_space`
 [om_server.h, 9](#)

`om_free_async_endpoint`
 [om_server.h, 18](#)

`om_free_contiguous_mem`
 [om_server.h, 21](#)

`om_free_cslot`
 [om_server.h, 21](#)

`om_free_device_frame`
 [om_server.h, 12](#)

`om_free_endpoint`
 [om_server.h, 16](#)

`om_free_frame`
 [om_server.h, 11](#)

`om_free_tcb`
 [om_server.h, 15](#)

`om_get_contiguous_mem_paddr`
 [om_server.h, 20](#)

`om_get_device_frame`
 [om_server.h, 11](#)

`om_get_interrupt_endpoint`
 [om_server.h, 19](#)

`om_get_thread_spip`
 [om_server.h, 14](#)

`om_init_thread_endpoint`
 [om_server.h, 16](#)

`om_init_thread_tcb`
 [om_server.h, 15](#)

`om_map_async_endpoint`
 [om_server.h, 17](#)

`om_map_contiguous_mem`
 [om_server.h, 19](#)

`om_map_device_frame`
 [om_server.h, 12](#)

`om_map_endpoint`
 [om_server.h, 15](#)

`om_map_frame`
 [om_server.h, 9](#)

`om_new_address_space`
 [om_server.h, 9](#)

`om_new_async_endpoint`
 [om_server.h, 17](#)

`om_new_contiguous_mem`
 [om_server.h, 19](#)

`om_new_cslot`
 [om_server.h, 21](#)

`om_new_endpoint`
 [om_server.h, 15](#)

`om_new_frame`
 [om_server.h, 9](#)

`om_new_interrupt`
 [om_server.h, 18](#)

`om_new_tcb`
 [om_server.h, 13](#)

`om_num_device_frames`
 [om_server.h, 11](#)

`om_pause_thread`
 [om_server.h, 14](#)

`om_resume_thread`
 [om_server.h, 14](#)

`om_save_reply_cap`
 [om_server.h, 21](#)

`om_self_address_space`
 [om_server.h, 9](#)

`om_server.h`, [3](#)

[__om_server_endpoint, 22](#)

[MAX_TCB_PRIORITY, 8](#)

[om_available_memory, 8](#)

[om_error, 8](#)

[om_flush_contiguous_mem, 20](#)

[om_flush_frame, 10](#)

[om_frame_paddr, 10](#)

[om_free_address_space, 9](#)

[om_free_async_endpoint, 18](#)

[om_free_contiguous_mem, 21](#)

[om_free_cslot, 21](#)

[om_free_device_frame, 12](#)

[om_free_endpoint, 16](#)

[om_free_frame, 11](#)

- om_free_tcb, 15
- om_get_contiguous_mem_paddr, 20
- om_get_device_frame, 11
- om_get_interrupt_endpoint, 19
- om_get_thread_spip, 14
- om_init_thread_endpoint, 16
- om_init_thread_tcb, 15
- om_map_async_endpoint, 17
- om_map_contiguous_mem, 19
- om_map_device_frame, 12
- om_map_endpoint, 15
- om_map_frame, 9
- om_new_address_space, 9
- om_new_async_endpoint, 17
- om_new_contiguous_mem, 19
- om_new_cslot, 21
- om_new_endpoint, 15
- om_new_frame, 9
- om_new_interrupt, 18
- om_new_tcb, 13
- om_num_device_frames, 11
- om_pause_thread, 14
- om_resume_thread, 14
- om_save_reply_cap, 21
- om_self_address_space, 9
- om_set_thread_spip, 14
- om_start_thread, 13
- om_unmap_async_endpoint, 18
- om_unmap_contiguous_mem, 20
- om_unmap_device_frame, 12
- om_unmap_endpoint, 16
- om_unmap_frame, 10
- USER_ENDPOINT_SLOT, 8

om_serverAPI.h, 22

om_set_thread_spip

- om_server.h, 14

om_start_thread

- om_server.h, 13

om_tcb, 3

om_unmap_async_endpoint

- om_server.h, 18

om_unmap_contiguous_mem

- om_server.h, 20

om_unmap_device_frame

- om_server.h, 12

om_unmap_endpoint

- om_server.h, 16

om_unmap_frame

- om_server.h, 10

USER_ENDPOINT_SLOT

- om_server.h, 8