BKK19-415 OP-TEE: Shared memory between TAs

Jens Wiklander

15 Jan 2019

Agenda

- What is shared memory between TAs?
- Shared memory without pager
- Paged memory
- Paged shared memory
- Data structures used by pager
- Example - releasing a physical page
- Sharing read-only pages of TAs
- Lifecycle of struct fobj

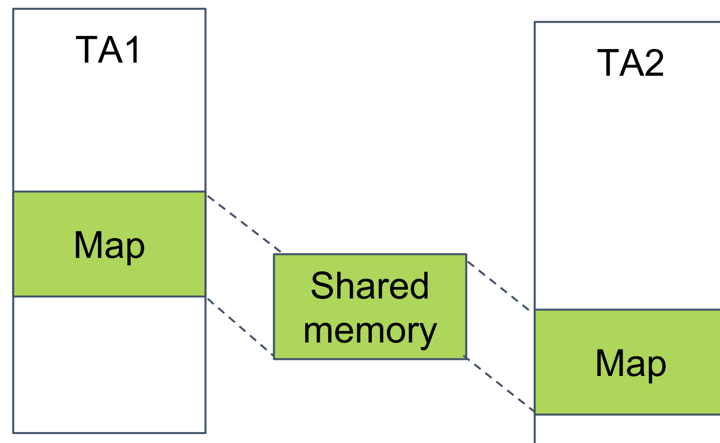
What is shared memory between TAs?

A TA is a Trusted Application with its own context in secure world

Wikipedia defines shared memory as:

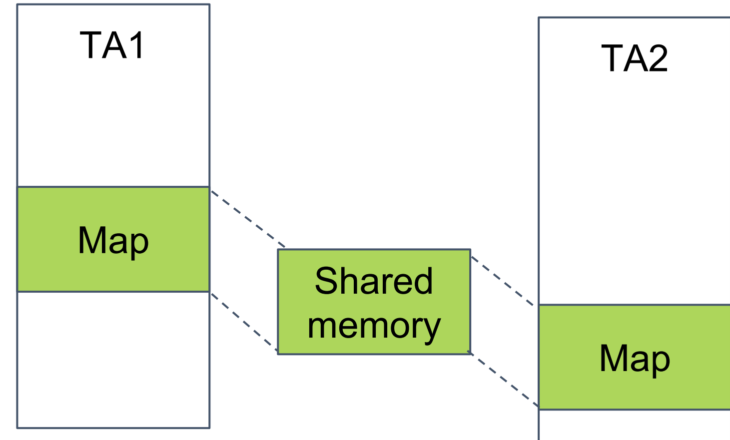
> *Shared memory is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies.*

In OP-TEE shared memory is used for both purposes where saving memory is a priority



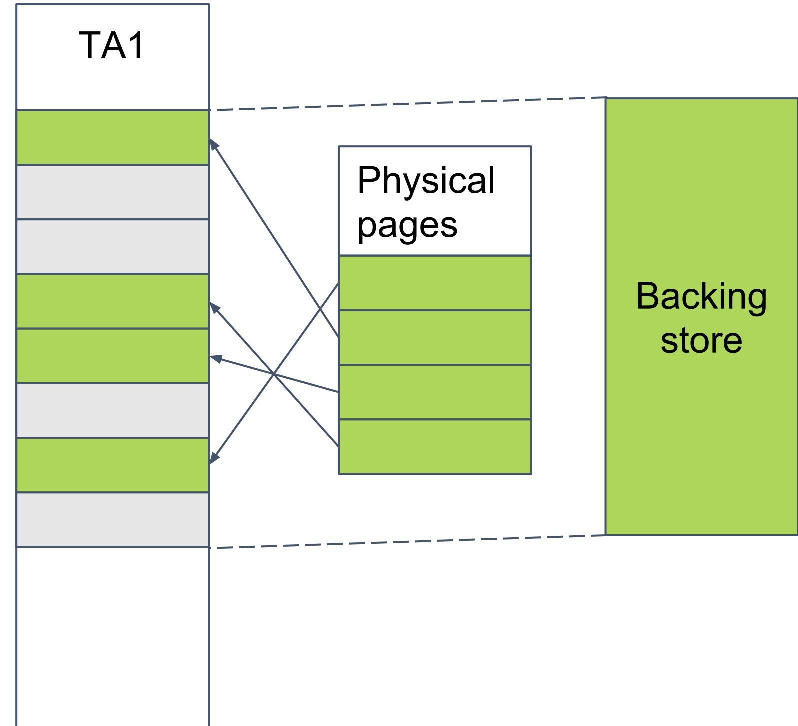
Shared memory without pager

- Without pager shared memory is achieved by mapping the same physical memory in the different TA contexts



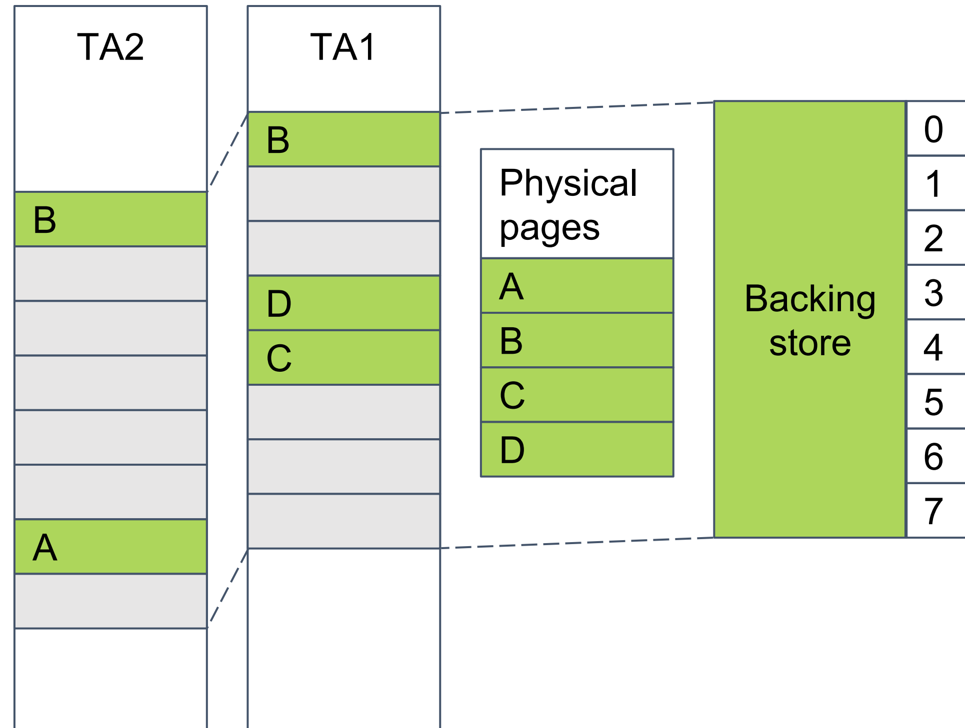
Paged memory

- Paged memory has a backing store
- Remapped on demand to maintain an illusion that the entire memory region is mapped



Paged shared memory

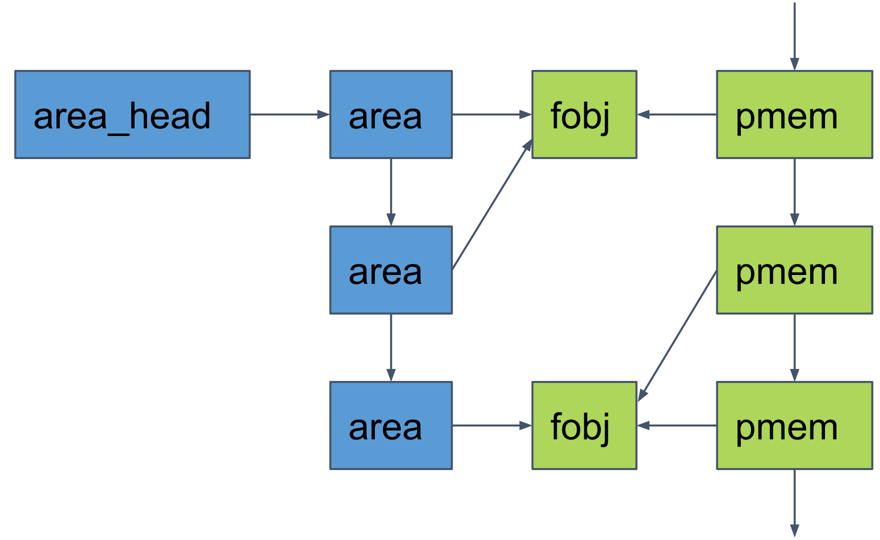
- Paged shared memory has a backing store
- Remapped on demand to maintain an illusion that the entire memory region is mapped
- Only referenced pages are mapped in each TA
- To ensure a coherent view of the memory, a backing store index can only be represented by one physical page at a time
- Shared memory is achieved by sharing physical pages representing the backing store



Linaro

Data structures used by pager

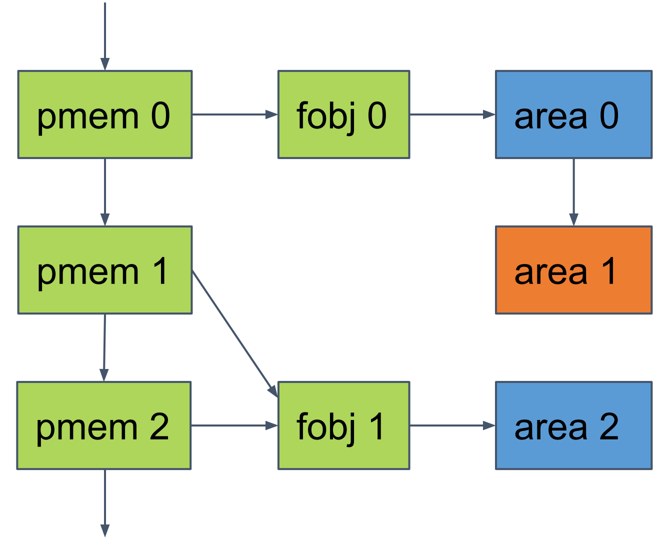
area_head	Represents the combined VM space for one TA
area	Virtual memory range of a partial or a complete fobj
fobj	Backing store for one secure shared memory object
pmem	Physical page



Linaro

Example - releasing a physical page

- A physical page may be mapped at several places
- Pager must find owners of shared page when releasing a page
- Areas are present in two different linked lists:
  - By virtual memory context as in the previous slide
  - By **fobj** to be able to find users of the same physical page
- To the right area 0 and 2 belongs to the same VM context while area 1 belongs to a different VM context
- pmem 0 might be mapped by area 0 and area 1



Linaro

Sharing read-only pages of TAs

- Read-only sections (slices) of a Trusted application are registered with file_new()
- Read-write sections are not saved since it costs memory
- The **tag** is a hash of the entire file from which the TA is loaded from

```
struct file_slice {
        struct fobj *fobj;
        unsigned int page_offset;
};

struct file *
file_new(uint8_t *tag,
         unsigned int taglen,
         struct file_slice *slices,
         unsigned int num_slices);
```

Sharing read-only pages of TAs, continued

- file_get_by_tag() is used while a TA is loaded to find out if there are any sharable sections available
- file_find_slice() is used to find a specific section which can be shared
- The **fobj** in struct file_slice is a pointer to a **fobj** which backs a read-only section of a TA

```
struct file_slice {
        struct fobj *fobj;
        unsigned int page_offset;
};

struct file *
file_get_by_tag(uint8_t *tag,
                unsigned int len);

struct file_slice *
file_find_slice(struct file *f,
                unsigned int offs);
```

Lifecycle of struct fobj

- struct fobj is reference counted
- fobj_put() is called each time a reference to a **fobj** is released
- When the reference count reaches 0 the **fobj** is freed
- The same principle applies for struct file

```
struct fobj {
        ...
        struct refcount refc;
        ...
};

Void fobj_put(struct fobj *fobj);
```

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contact@linaro.org