Trusted Firmware - M

Handle Management Mechanism Enhancement

Precondition for fast RoT Services API call

Ken Liu

2020-Apr-13

Content

- Background – To improve the performance of RoT Service API
- Design – Usage analysis and the proposal.
- Discussions – Related topics: Memory usage and performance.

Background

- In a Tech Forum so far (Jan 23rd), partners comment that psa_connect()/psa_call()/psa_close() cost much for one-shot service call.
- Here is how we encapsulate a RoT service API today:

```
int32_t RoTService(void)
{
    handle = psa_connect(SID, VERSION);
    if (!PSA_HANDLE_IS_VALID(handle) {
        return PSA_HANDLE_TO_ERROR(handle);
    }
    status = psa_call(handle, PSA_IPC_CALL, NULL, 0, NULL, 0);
    psa_close(handle);

    return status;
}
```

- Then some investigation happened to see if we can enhance this part.

Assumptions before going

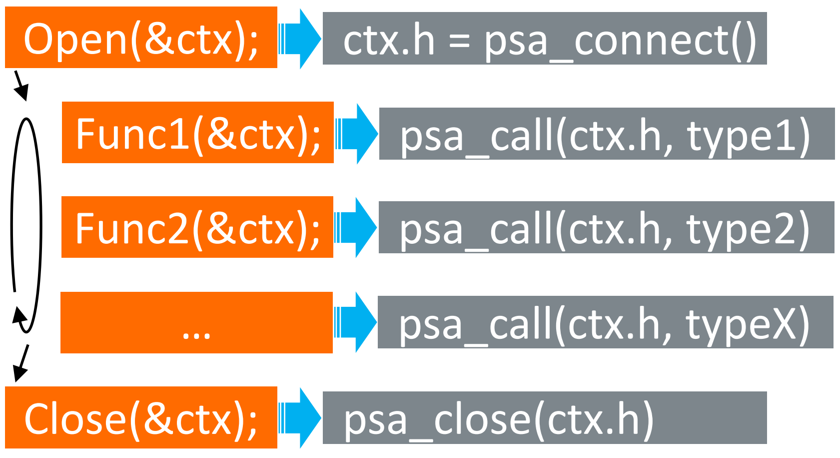
- Avoid significant changes in PSA FF - Be simple.

- Security consideration
  - Connection-based mechanism is necessary – SPM and services could identify clients by connection.
  - Connecting process is known by services.

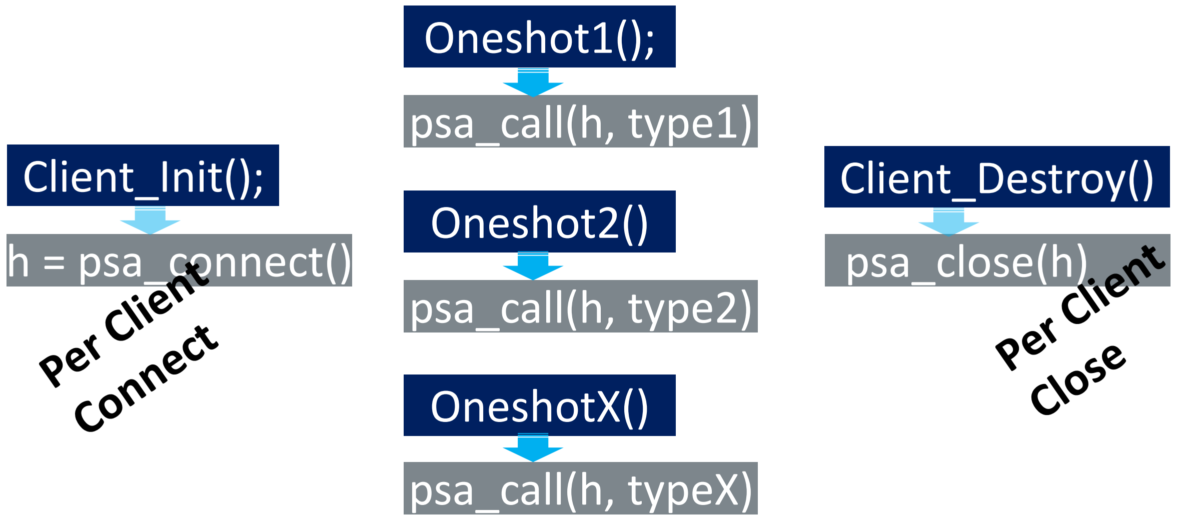
- Let's go through the analysis and possible implementations…

Thoughts – When to call psa_connect()?

- ‘psa_connect’ is **always called** while **session-based service API** setup a session.
  - Session-based API has session maintenance process (setup/process/destroy), PSA API can be called during these process.
  - The connecting cost are diluted in the functions get called.
- One-shot RoT service API is session-less and can **re-use** the connected handles.
  - From security perspective – SPM and services need to identify clients for access control – connected handles can not be shared between clients – one client one connection.



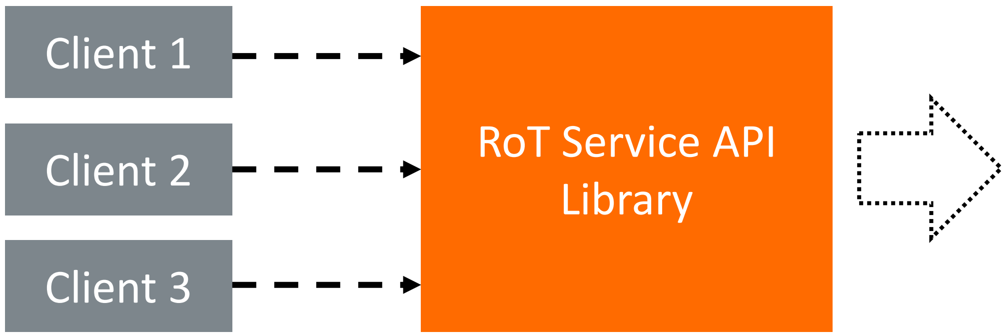
Session-based RoT Service API



Session-less RoT Service API

A Typical Design Candidate – Store the connected handle

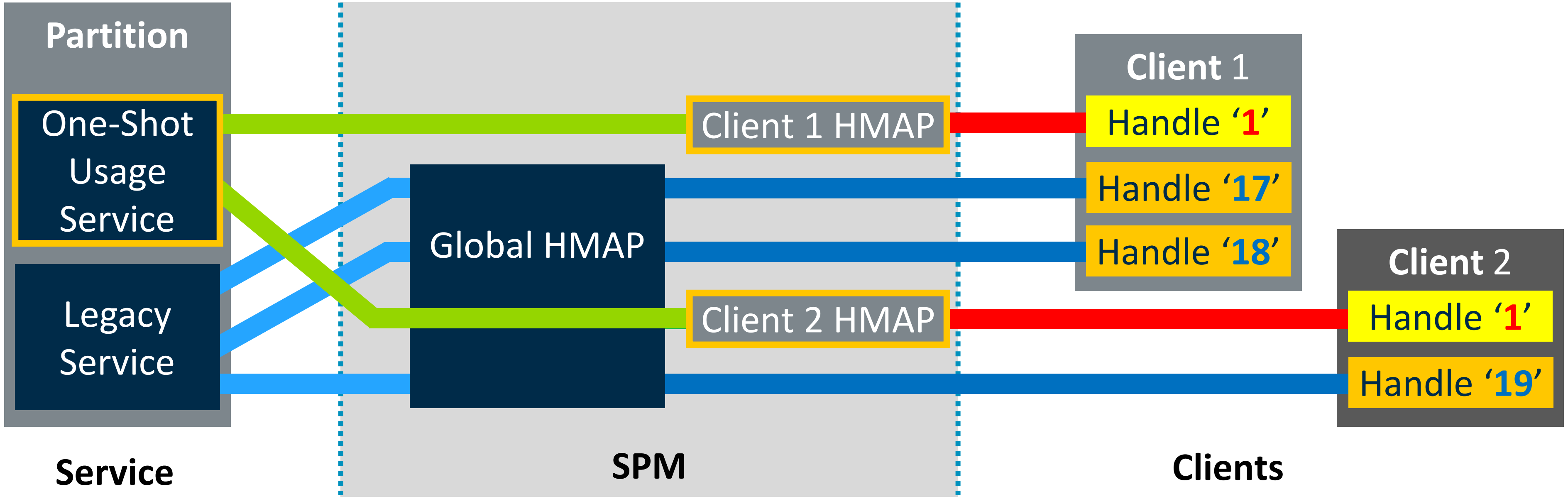
- If stored as global variables:
  - RoT Service API is implemented as a library and being shared by multiple clients, how does this library know how many handle variables it should reserve in static allocation case?
  - All Clients shares one saved variable – bring more trouble to systems support isolation.
- Could save by abstracted allocation API, but:
  - Involves abstraction layer into library – **More Dependencies!**
  - A system without memory management API?
  - Which handle belong to this caller? – Need an ID to represent the caller.
- **Looks not like a nice solution.**




```
handle = GET_SAVED_HANDLE(THIS_CALLER_ID);
if (!PSA_HANDLE_IS_VALID(handle)) {
    handle = psa_connect(SID, VERSION);
    if (!PSA_HANDLE_IS_VALID(handle) {
        return PSA_HANDLE_TO_ERROR(handle);
    }
    SAVE_HANDLE(THIS_CALLER_ID, handle);
}
```

Thoughts – If a service handle is known already?

- No Handle Storing is needed - the client can ‘psa_call’ on a known handle value:
  - **‘psa_call(HANDLE_SERVICE1, type, …)’**
- Need to make different clients can get the same handle value for the same one-shot service.（An implementation note in PSA-FF-M 3.3.4 now would become a MUST item for one-shot services).



- **Looks like a neat solution.**

Proposal – The PSA-FF-M level details

- A new manifest field in PSA-FF-M for services to indicate if the default handle value for session-less service API usage:

`'default_handle': <number or pattern>`	
'auto'	System allocation.
1 ~ DEFAULT_HANDLE_MAX	Expected handle value.
Field not available	No default handle value for this service.

- Default handle value assigned by the framework/implementation auto-connecting.
  - No handle storage is needed for default handles.
  - Client 'psa_connect' work as usual .
  - Closing a default handle causes panic() – no closing allowed to avoid affecting other code who is working on this default handle.

Proposal – Implementation: Tooling and Coding

- Tooling to generate the default handle value while building if ‘default_handle’ detected
  - Rot Service API implementation references the handle by MACRO.

psa_service_a.h:

```
/* Auto-Generated file, DO NOT MODIFY! */
#define HANDLE_SERVICE_A    ((psa_handle_t)3)
```

psa_service_a.c:

```
/* RoT Service API */
psa_status_t rot_service_a(void)
{
    return psa_call(HANDLE_SERVICE_A, PSA_IPC_CALL, NULL, 0, NULL, 0);
}
```

Proposal – Auto-connecting implementation examples

- **Auto-connecting** during SP launching – Implicit operation in SP runtime, RoT Service Developers do not need to change anything.

```
void sprt::main(dep_t *sp_dep)
{
    while (sp_dep && sp_dep->default_handle) {
        if (sp_dep->default_handle != psa_connect(sp_dep->sid,
                                                  sp_dep->version))
            psa_panic();
        sp_dep++;
    }

    sp_dep->sp_entry();
}
```

Discussions – Related topics

- Memory usage – Should be tiny increasement.
  - Per client dependencies storage – increased storage size.
  - Extra logic to dispatch ‘default_handle’ – code size in SPM.
  - Auto-connecting in SP Runtime – increases SP Runtime code size a bit.

- Performance – Almost the same.
  - A table lookup is needed for session-less services which cost several more lines.

- Will be a TF-M feature initially and working in parallel on an extension of PSA-FF-M specification to include this feature.

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה