arm

Trusted Firmware - M

Handle Management Mechanism Enhancement

Precondition for fast RoT Services API call

Ken Liu

2020-May-14

Content

- Background – To improve the performance of RoT Service API
- Design – Usage analysis and the proposal
- Discussions – Related topics: Memory usage and performance
- Updates since initial version of the proposal
  - Including the service version in the default handle
  - Implementing just-in-time auto-connection within the SPM

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;
}
```

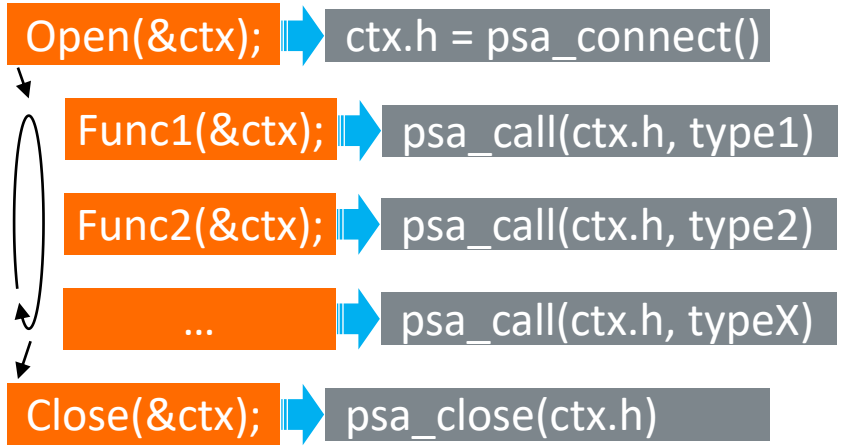
  - Then some investigation happened to see if we can enhance this part.

Assumptions before going

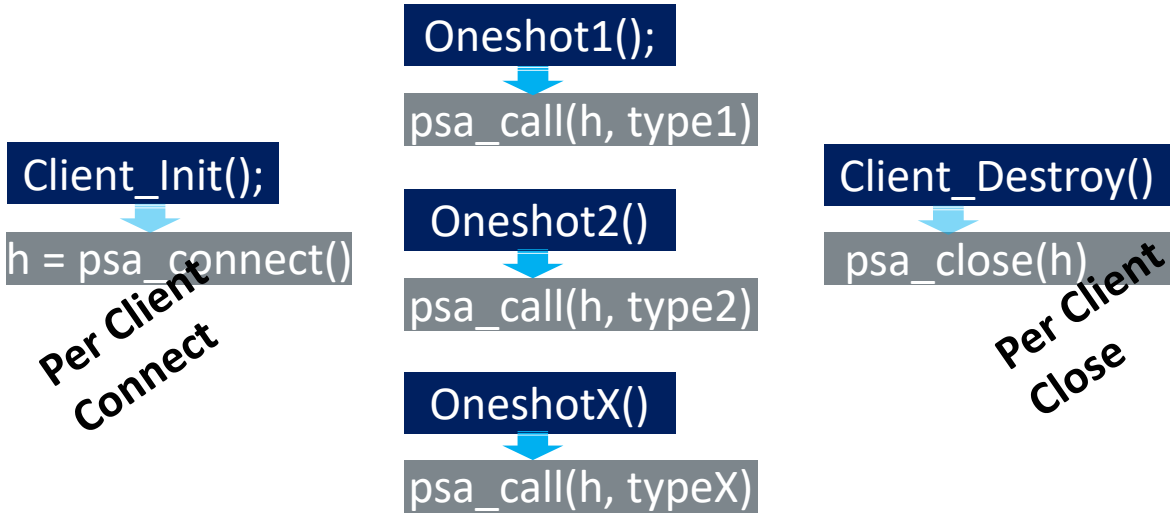
- Avoid significant changes in PSA FF - Be simple.
- Security consideration
  - Connection-based mechanism is necessary – SPM and services could identify clients by connection.
  - Connecting process is known by services.
- Let’s go through the analysis and possible implementations…

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 costs are amortized in the functions that 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 cannot be shared between clients – one client one connection.



Session-based RoT Service API

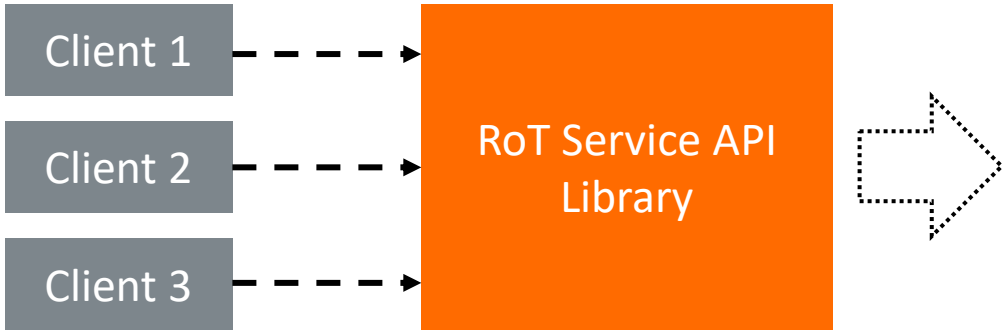


Session-less RoT Service API

arm

A Typical Design Candidate – Store the connected handle

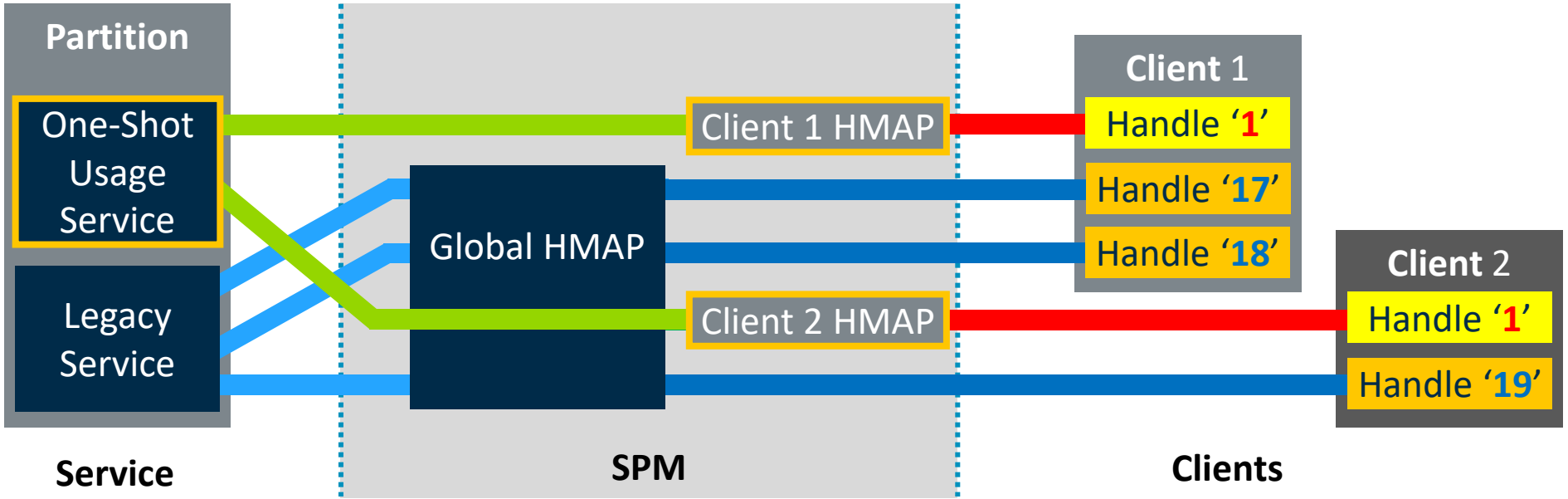
- If stored as global variables:
  - RoT Service API is implemented as a library and being shared by multiple clients, how does this library know how many handle variables it should reserve in static allocation case?
  - All Clients share one saved variable – brings more trouble to systems that implement isolation.
- Could resolve this with an 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.
- **Does not look 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(SERVICE1_HANDLE, type, …)’**
- Need to make different clients can use 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.**

arm

Proposal – PSA-FF-M level details

- A new manifest field for services to indicate a default handle **INDEX** for session-less service API usage:

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

- Default handle **VALUE** is decided by the framework/implementation.
  - **INDEX** and service **VERSION** are encoded into **VALUE** using **Implementation-defined** encoding.
  - Two services cannot have the same default handle **INDEX** and **VALUE**.
- Default handle value is assigned to a macro, just like service SID and VERSION.
- Default handles are connected during the first 'psa_call()' with default handle value.
  - Client **explicit** call to 'psa_connect()' returns non-default handle value.
  - Closing a default handle causes **PROGRAMMER ERROR** – no closing allowed to avoid affecting other code that is working on this default handle.

Proposal – Implementation ideas: Tooling and Coding

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

service_a.json:

```
“name”: "SERVICE_A",
“version”: 1,
“default_handle”: 3
```

psa_manifest/sid.h:

```
/* Auto-Generated file, DO NOT MODIFY! */
#define SERVICE_A_HANDLE    ((psa_handle_t)((1 << 8)|(3 & 0xff)))
```

psa_service_a.c:

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

Proposal – Implementation ideas: Auto-connecting

- **Auto-connecting** during ‘psa_call()’ - this can be handled inside SPM, without needing changes in the client or the SPRTL.

```
spm_psa_call_handler(handle, type, call_params)
{
    if (IS_VALID_HANDLE(handle)) {
        msg = create_call_message(handle, call_params);
        if (IS_DEFAULT_HANDLE(handle) && NOT_CONNECTED(client, handle)) {
            validate_version(SID_LOOKUP(handle), VERSION(handle));
            msg.type = PSA_IPC_CONNECT;
            client->data = type; /* Backup the type for following call message. */
        }
        send_message(msg);
        wait_on(msg.event);
    }
    return;
}
```

Discussions – Related topics

- Memory usage – Should be tiny increase.
  - 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.
  - But reduction in client code to connect and close transient handles, or to store handle variables.

- 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.

arm

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