# TrustSQLite

2022年4月12日 10:08

### Modifying the OP-TEE

- **添加系统调用**

  **用户空间代码的修改**

  1. 修改optee_os/lib/libutee/arch/arm/utee_syscalls_asm.S文件，添加如下内容：

```
// optee_os/lib/libutee/arch/arm/utee_syscalls_asm.S
        UTEE_SYSCALL utee_tzvfs_open, TEE_SCN_TZVFS_OPEN, 4

        UTEE_SYSCALL utee_tzvfs_close, TEE_SCN_TZVFS_CLOSE, 2

        UTEE_SYSCALL utee_tzvfs_getcwd, TEE_SCN_TZVFS_GETCWD, 3

        UTEE_SYSCALL utee_tzvfs_lstat, TEE_SCN_TZVFS_LSTAT, 3

        UTEE_SYSCALL utee_tzvfs_stat, TEE_SCN_TZVFS_STAT, 3

        UTEE_SYSCALL utee_tzvfs_fstat, TEE_SCN_TZVFS_FSTAT, 3

        UTEE_SYSCALL utee_tzvfs_fcntl, TEE_SCN_TZVFS_FCNTL, 4

        UTEE_SYSCALL utee_tzvfs_read, TEE_SCN_TZVFS_READ, 4

        UTEE_SYSCALL utee_tzvfs_write, TEE_SCN_TZVFS_WRITE, 4

        UTEE_SYSCALL utee_tzvfs_geteuid, TEE_SCN_TZVFS_GETEUID, 1

        UTEE_SYSCALL utee_tzvfs_unlink, TEE_SCN_TZVFS_UNLINK, 2

        UTEE_SYSCALL utee_tzvfs_access, TEE_SCN_TZVFS_ACCESS, 3

        UTEE_SYSCALL utee_tzvfs_mmap, TEE_SCN_TZVFS_MMAP, 7

        UTEE_SYSCALL utee_tzvfs_mremap, TEE_SCN_TZVFS_MREMAP, 5

        UTEE_SYSCALL utee_tzvfs_munmap, TEE_SCN_TZVFS_MUNMAP, 3

        UTEE_SYSCALL utee_tzvfs_strcspn, TEE_SCN_TZVFS_STRCSPN, 3

        UTEE_SYSCALL utee_tzvfs_utimes, TEE_SCN_TZVFS_UTIMES, 3

        UTEE_SYSCALL utee_tzvfs_lseek, TEE_SCN_TZVFS_LSEEK, 4

        UTEE_SYSCALL utee_tzvfs_fsync, TEE_SCN_TZVFS_FSYNC, 2

        UTEE_SYSCALL utee_tzvfs_getenv, TEE_SCN_TZVFS_GETENV, 2

        UTEE_SYSCALL utee_tzvfs_getpid, TEE_SCN_TZVFS_GETPID, 1

        UTEE_SYSCALL utee_tzvfs_time, TEE_SCN_TZVFS_TIME, 2

        UTEE_SYSCALL utee_tzvfs_sleep, TEE_SCN_TZVFS_SLEEP, 2

        UTEE_SYSCALL utee_tzvfs_gettimeofday, TEE_SCN_TZVFS_GETTIMEOFDAY, 3

        UTEE_SYSCALL utee_tzvfs_fchown, TEE_SCN_TZVFS_FCHOWN, 4
```

  1. 修改optee_os/lib/libutee/include/utee_syscalls.h文件，添加如下内容，申明上述函数接口，在TA的源代码中包含该头文件后就可调用该接口，同时添加optee_os/lib/libutee/include/tzvfs_types.h头文件。

```
// optee_os/lib/libutee/include/utee_syscalls.h
#include <tzvfs_types.h>
int utee_tzvfs_open(int *tzvfs_errno, const char *filename, int flags, mode_t mode);
int utee_tzvfs_close(int *tzvfs_errno, int fd);
char *utee_tzvfs_getcwd(int *tzvfs_errno, char *buf, size_t size);
int utee_tzvfs_lstat(int *tzvfs_errno, const char* path, struct tzvfs_stat *buf);
int utee_tzvfs_stat(int *tzvfs_errno, const char *path, struct tzvfs_stat *buf);
int utee_tzvfs_fstat(int *tzvfs_errno, int fd, struct tzvfs_stat *buf);
```

```
int utee_tzvfs_fcntl(int *tzvfs_errno, int fd, int cmd, struct tzvfs_flock *arg);
ssize_t utee_tzvfs_read(int *tzvfs_errno, int fd, void *buf, size_t count);
ssize_t utee_tzvfs_write(int *tzvfs_errno, int fd, const void *buf, size_t count);
uid_t utee_tzvfs_geteuid(int *tzvfs_errno);
int utee_tzvfs_unlink(int *tzvfs_errno, const char *pathname);
int utee_tzvfs_access(int *tzvfs_errno, const char *pathname, int mode);
void *utee_tzvfs_mmap(int *tzvfs_errno, void *addr, size_t len, int prot, int flags, int fildes, off_t
off);
void *utee_tzvfs_mremap(int *tzvfs_errno, void *old_address, size_t old_size, size_t new_size, int
flags);
int utee_tzvfs_munmap(int *tzvfs_errno, void *addr, size_t length);
size_t utee_tzvfs_strcspn(int *tzvfs_errno, const char *str1, const char *str2);
int utee_tzvfs_utimes(int *tzvfs_errno, const char *filename, const struct tzvfs_timeval times[2]);
off_t utee_tzvfs_lseek(int *tzvfs_errno, int fd, off_t offset, int whence);
int utee_tzvfs_fsync(int *tzvfs_errno, int fd);
char* utee_tzvfs_getenv(int *tzvfs_errno, const char *name);
pid_t utee_tzvfs_getpid(int *tzvfs_errno);
time_t utee_tzvfs_time(int *tzvfs_errno, time_t *t);
unsigned int utee_tzvfs_sleep(int *tzvfs_errno, unsigned int seconds);
int utee_tzvfs_gettimeofday(int *tzvfs_errno, struct tzvfs_timeval *tv, struct tzvfs_timezone *tz);
int utee_tzvfs_fchown(int *tzvfs_errno, int fd, uid_t owner, gid_t group);
```

3. 修改optee_os/lib/libutee/include/tee_syscall_numbers.h文件，添加上述系统调用接口的索引值，并修改TEE_SCN_MAX的值，需要修改和添加的内容如下：

```
// optee_os/lib/libutee/include/tee_syscall_numbers.h
#define TEE_SCN_TZVFS_OPEN              73
#define TEE_SCN_TZVFS_CLOSE             74
#define TEE_SCN_TZVFS_GETCWD            75
#define TEE_SCN_TZVFS_LSTAT             76
#define TEE_SCN_TZVFS_STAT              77
#define TEE_SCN_TZVFS_FSTAT             78
#define TEE_SCN_TZVFS_FCNTL             79
#define TEE_SCN_TZVFS_READ              80
#define TEE_SCN_TZVFS_WRITE             81
#define TEE_SCN_TZVFS_GETEUID           82
#define TEE_SCN_TZVFS_UNLINK            83
#define TEE_SCN_TZVFS_ACCESS            84
#define TEE_SCN_TZVFS_MMAP              85
#define TEE_SCN_TZVFS_MREMAP            86
#define TEE_SCN_TZVFS_MUNMAP            87
#define TEE_SCN_TZVFS_STRCSPN           88
#define TEE_SCN_TZVFS_UTIMES            89
#define TEE_SCN_TZVFS_LSEEK             90
#define TEE_SCN_TZVFS_FSYNC             91
#define TEE_SCN_TZVFS_GETENV            92
#define TEE_SCN_TZVFS_GETPID            93
#define TEE_SCN_TZVFS_TIME              94
#define TEE_SCN_TZVFS_SLEEP             95
#define TEE_SCN_TZVFS_GETTIMEOFDAY      96
#define TEE_SCN_TZVFS_FCHOWN            97
#define TEE_SCN_MAX                     97
```

**内核空间代码的修改**

4. 修改optee_os/core/arch/arm/tee/arch_svc.c文件中系统调用数组变量tee_svc_syscall_table的内容，将上述系统调用对应的内核层接口添加到该数组中，并包含申明该接口的头文件，在该文件中添加的内容如下：

```
// optee_os/core/arch/arm/tee/arch_svc.c
#include <tee/tee_tzvfs.h>
static const struct syscall_entry tee_svc_syscall_table[] = {
    ......
    SYSCALL_ENTRY(syscall_tzvfs_open),
    SYSCALL_ENTRY(syscall_tzvfs_close),
    SYSCALL_ENTRY(syscall_tzvfs_getcwd),
    SYSCALL_ENTRY(syscall_tzvfs_lstat),
    SYSCALL_ENTRY(syscall_tzvfs_stat),
    SYSCALL_ENTRY(syscall_tzvfs_fstat),
    SYSCALL_ENTRY(syscall_tzvfs_fcntl),
    SYSCALL_ENTRY(syscall_tzvfs_read),
    SYSCALL_ENTRY(syscall_tzvfs_write),
    SYSCALL_ENTRY(syscall_tzvfs_geteuid),
    SYSCALL_ENTRY(syscall_tzvfs_unlink),
    SYSCALL_ENTRY(syscall_tzvfs_access),
    SYSCALL_ENTRY(syscall_tzvfs_mmap),
    SYSCALL_ENTRY(syscall_tzvfs_mremap),
```

```
    SYSCALL_ENTRY(syscall_tzvfs_munmap),
    SYSCALL_ENTRY(syscall_tzvfs_strcspn),
    SYSCALL_ENTRY(syscall_tzvfs_utimes),
    SYSCALL_ENTRY(syscall_tzvfs_lseek),
    SYSCALL_ENTRY(syscall_tzvfs_fsync),
    SYSCALL_ENTRY(syscall_tzvfs_getenv),
    SYSCALL_ENTRY(syscall_tzvfs_getpid),
    SYSCALL_ENTRY(syscall_tzvfs_time),
    SYSCALL_ENTRY(syscall_tzvfs_sleep),
    SYSCALL_ENTRY(syscall_tzvfs_gettimeofday),
    SYSCALL_ENTRY(syscall_tzvfs_fchown),
};
```

- **添加系统服务**
  1. 在本示例中建立的系统服务的源代码为`tee_tzvfs.c`文件，需将该文件保存到`optee_os/core/tee`目录中。

```
// optee_os/core/tee/tee_tzvfs.c
#include <assert.h>
#include <string.h>
#include <optee_rpc_cmd.h>
#include <kernel/thread.h>
#include <kernel/msg_param.h>
#include <tee/tee_svc.h>
#include <mm/tee_mm.h>
#include <mm/mobj.h>
#include <tee/tee_tzvfs.h>
// 打开文件. 调用成功时返回一个文件描述符fd，调用失败时返回-1，并修改errno
int syscall_tzvfs_open(int *tzvfs_errno, const char *filename, int flags, mode_t mode){
  int ret = -1;

  size_t size = TZVFS_FS_NAME_MAX;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, filename, strlen(filename)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_OPEN, flags, mode);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
// 若文件顺利关闭则返回0，发生错误时返回-1
int syscall_tzvfs_close(int *tzvfs_errno, int fd){
  int ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_CLOSE, fd, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
    if (ret == -1) *tzvfs_errno = params[1].u.value.b;
  }
  return ret;
}
// 获取当前工作目录，成功则返回当前工作目录；如失败返回NULL，错误代码存于errno
```

```c
char *syscall_tzvfs_getcwd(int *tzvfs_errno, char *buf, size_t size){
  char* ret = NULL;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return NULL;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return NULL;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_GETCWD, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(OUT, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    memcpy(buf, va, size);
    ret = (char*)((int)params[2].u.value.a);
    if (ret == NULL) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
// 获取一些文件相关的信息，成功执行时，返回0。失败返回-1，errno
// lstat函数是不穿透（不追踪）函数，对软链接文件进行操作时，操作的是软链接文件本身
int syscall_tzvfs_lstat(int *tzvfs_errno, const char* path, struct tzvfs_stat *buf) {
  int ret = -1;
  size_t size1 = TZVFS_FS_NAME_MAX;
  size_t size2 = sizeof(struct tzvfs_stat);
  struct thread_param params[4];
  struct mobj *mobj1 = NULL, *mobj2 = NULL;
  void *va1, *va2;
  // 分配共享内存
  mobj1 = thread_rpc_alloc_payload(size1);
  if (!mobj1) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj1->size < size1) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj1);
    return -1;
  }
  mobj2 = thread_rpc_alloc_payload(size2);
  if (!mobj2) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj2->size < size2) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj2);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va1 = mobj_get_va(mobj1, 0);
  va2 = mobj_get_va(mobj2, 0);
  memcpy(va1, path, strlen(path)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_LSTAT, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj1, 0, size1);
  params[2] = THREAD_PARAM_MEMREF(OUT, mobj2, 0, size2);
  params[3] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 4, params)) {
    memcpy(buf, va2, size2);
```

```
    ret = params[3].u.value.a;
    if (ret == -1) *tzvfs_errno = params[3].u.value.b;
  }
  thread_rpc_free_payload(mobj1);
  thread_rpc_free_payload(mobj2);

  return ret;
}
// 获取一些文件相关的信息，成功执行时，返回0。失败返回-1，errno
// stat函数是穿透（追踪）函数，即对软链接文件进行操作时，操作的是链接到的那一个文件，不是软链接文件本身
int syscall_tzvfs_stat(int *tzvfs_errno, const char *path, struct tzvfs_stat *buf){
  int ret = -1;
  size_t size1 = TZVFS_FS_NAME_MAX;
  size_t size2 = sizeof(struct tzvfs_stat);
  struct thread_param params[4];
  struct mobj *mobj1 = NULL, *mobj2 = NULL;
  void *va1, *va2;
  // 分配共享内存
  mobj1 = thread_rpc_alloc_payload(size1);
  if (!mobj1) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj1->size < size1) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj1);
    return -1;
  }
  mobj2 = thread_rpc_alloc_payload(size2);
  if (!mobj2) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj2->size < size2) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj2);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va1 = mobj_get_va(mobj1, 0);
  va2 = mobj_get_va(mobj2, 0);
  memcpy(va1, path, strlen(path)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_STAT, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj1, 0, size1);
  params[2] = THREAD_PARAM_MEMREF(OUT, mobj2, 0, size2);
  params[3] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 4, params)) {
    memcpy(buf, va2, size2);
    ret = params[3].u.value.a;
    if (ret == -1) *tzvfs_errno = params[3].u.value.b;
  }
  thread_rpc_free_payload(mobj1);
  thread_rpc_free_payload(mobj2);

  return ret;
}
// fstat函数与stat函数的功能一样，只是第一个形参是文件描述符
int syscall_tzvfs_fstat(int *tzvfs_errno, int fd, struct tzvfs_stat *buf){
  int ret = -1;
  size_t size = sizeof(struct tzvfs_stat);
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
```

```
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_FSTAT, fd, 0);
  params[1] = THREAD_PARAM_MEMREF(OUT, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    memcpy(buf, va, size);
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
// 通过fcntl可以改变已打开的文件性质，F_SETLK 设置文件锁定的状态
// fcntl的返回值与命令有关。如果出错，所有命令都返回 - 1，如果成功则返回某个其他值。
int syscall_tzvfs_fcntl(int *tzvfs_errno, int fd, int cmd, struct tzvfs_flock *arg){
  int ret = -1;

  size_t size = sizeof(struct tzvfs_flock);
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, arg, size);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_FCNTL, fd, cmd);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
// read会把参数fd所指的文件传送count个字节到buf指针所指的内存中
// 返回值为实际读取到的字节数，如果返回0，表示已到达文件尾或是无可读取的数据
// 当有错误发生时则返回-1，错误代码存入errno 中
ssize_t syscall_tzvfs_read(int *tzvfs_errno, int fd, void *buf, size_t count){
  ssize_t ret = -1;
  size_t size = count;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
```

```
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_READ, fd, 0);
  params[1] = THREAD_PARAM_MEMREF(OUT, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    memcpy(buf, va, size);
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);

  return ret;
}
// write函数把buf中nbyte写入文件描述符handle所指的文档
// 成功时返回写的字节数，错误时返回-1
ssize_t syscall_tzvfs_write(int *tzvfs_errno, int fd, const void *buf, size_t count){
  ssize_t ret = -1;
  size_t size = count;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, buf, size);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_WRITE, fd, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);

  return ret;
}
// geteuid()用来取得执行目前进程有效的用户识别码
// 返回有效的用户识别码
uid_t syscall_tzvfs_geteuid(int *tzvfs_errno){
  uid_t ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_GETEUID, 0, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
    if (ret == (unsigned int)-1) *tzvfs_errno = params[1].u.value.b;
  }

  return ret;
}
int syscall_tzvfs_unlink(int *tzvfs_errno, const char *pathname){
  int ret = -1;

  size_t size = TZVFS_FS_NAME_MAX;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
```

```
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, pathname, strlen(pathname)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_UNLINK, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
int syscall_tzvfs_access(int *tzvfs_errno, const char *pathname, int mode){
  int ret = -1;
  size_t size = TZVFS_FS_NAME_MAX;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, pathname, strlen(pathname)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_ACCESS, mode, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
void *syscall_tzvfs_mmap(int *tzvfs_errno, void *addr, size_t len, int prot, int flags, int fildes,
off_t off){
  void* ret = (void *)-1;
  struct thread_param params[4];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_MMAP, 0, 0);
  params[1] = THREAD_PARAM_VALUE(IN, (int)addr, len, prot);
  params[2] = THREAD_PARAM_VALUE(IN, flags, fildes, off);
  params[3] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 4, params)) {
    ret = (void*)((int)params[3].u.value.a);
    if (ret == (void *)-1) *tzvfs_errno = params[3].u.value.b;
  }

  return ret;
```

```c
}
void *syscall_tzvfs_mremap(int *tzvfs_errno, void *old_address, size_t old_size, size_t new_size, int
flags){
  void* ret = (void *)-1;
  struct thread_param params[3];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_MREMAP, (int)old_address, old_size);
  params[1] = THREAD_PARAM_VALUE(IN, new_size, flags, 0);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = (void*)((int)params[2].u.value.a);
    if (ret == (void *)-1) *tzvfs_errno = params[2].u.value.b;
  }
  return ret;
}
int syscall_tzvfs_munmap(int *tzvfs_errno, void *addr, size_t length){
  int ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_MUNMAP, (int)addr, length);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
    if (ret == -1) *tzvfs_errno = params[1].u.value.b;
  }
  return ret;
}
size_t syscall_tzvfs_strcspn(int *tzvfs_errno, const char *str1, const char *str2){
  size_t ret = 0;
  tzvfs_errno = tzvfs_errno;
  size_t size1 = TZVFS_FS_NAME_MAX;
  size_t size2 = TZVFS_FS_NAME_MAX;
  struct thread_param params[4];
  struct mobj *mobj1 = NULL, *mobj2 = NULL;
  void *va1, *va2;
  // 分配共享内存
  mobj1 = thread_rpc_alloc_payload(size1);
  if (!mobj1) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj1->size < size1) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj1);
    return -1;
  }
  mobj2 = thread_rpc_alloc_payload(size2);
  if (!mobj2) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj2->size < size2) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj2);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va1 = mobj_get_va(mobj1, 0);
  va2 = mobj_get_va(mobj2, 0);
  memcpy(va1, str1, strlen(str1)+1);
  memcpy(va2, str2, strlen(str2)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_STRCSPN, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj1, 0, size1);
  params[2] = THREAD_PARAM_MEMREF(IN, mobj2, 0, size2);
  params[3] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 4, params)) {
    ret = params[3].u.value.a;
  }
  thread_rpc_free_payload(mobj1);
  thread_rpc_free_payload(mobj2);

  return ret;
}
```

```c
int syscall_tzvfs_utimes(int *tzvfs_errno, const char *filename, const struct tzvfs_timeval times[2]){
  int ret = -1;
  times = times;
  size_t size = TZVFS_FS_NAME_MAX;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, filename, strlen(filename)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_UTIMES, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);

  return ret;
}
off_t syscall_tzvfs_lseek(int *tzvfs_errno, int fd, off_t offset, int whence){
  off_t ret = -1;
  struct thread_param params[3];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_LSEEK, 0, 0);
  params[1] = THREAD_PARAM_VALUE(IN, fd, offset, whence);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  return ret;
}
int syscall_tzvfs_fsync(int *tzvfs_errno, int fd){
  int ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_FSYNC, fd, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
    if (ret == -1) *tzvfs_errno = params[1].u.value.b;
  }
  return ret;
}
char* syscall_tzvfs_getenv(int *tzvfs_errno, const char *name){
  char* ret = NULL;
  tzvfs_errno = tzvfs_errno;
  unsigned int size = TZVFS_FS_NAME_MAX;
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return NULL;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
```

```
    return NULL;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
  memcpy(va, name, strlen(name)+1);
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_GETENV, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(IN, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = (char*)((int)params[2].u.value.a);
  }
  thread_rpc_free_payload(mobj);

  return ret;
}
pid_t syscall_tzvfs_getpid(int *tzvfs_errno){
  pid_t ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_GETPID, 0, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
    if (ret == -1) *tzvfs_errno = params[1].u.value.b;
  }
  return ret;
}
time_t syscall_tzvfs_time(int *tzvfs_errno, time_t *t){
  time_t ret = -1;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_TIME, 0, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    *t = params[1].u.value.a;
    ret = params[1].u.value.a;
    if (ret == -1) *tzvfs_errno = params[1].u.value.b;
  }
  return ret;
}
unsigned int syscall_tzvfs_sleep(int *tzvfs_errno, unsigned int seconds){
  unsigned int ret = 0;
  tzvfs_errno = tzvfs_errno;
  struct thread_param params[2];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_SLEEP, seconds, 0);
  params[1] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 2, params)) {
    ret = params[1].u.value.a;
  }
  return ret;
}
int syscall_tzvfs_gettimeofday(int *tzvfs_errno, struct tzvfs_timeval *tv, struct tzvfs_timezone *tz){
  int ret = -1;
  tz = tz;
  size_t size = sizeof(struct tzvfs_timeval);
  struct thread_param params[3];
  struct mobj *mobj = NULL;
  void *va;
  // 分配共享内存
  mobj = thread_rpc_alloc_payload(size);
  if (!mobj) {
    *tzvfs_errno = TEE_ERROR_OUT_OF_MEMORY;
    return -1;
  }
  if (mobj->size < size) {
    *tzvfs_errno = TEE_ERROR_SHORT_BUFFER;
    thread_rpc_free_payload(mobj);
    return -1;
  }
  // 获取分配的共享内存的虚拟地址
  va = mobj_get_va(mobj, 0);
```

```
  // 初始RPC参数
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_GETTIMEOFDAY, 0, 0);
  params[1] = THREAD_PARAM_MEMREF(OUT, mobj, 0, size);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    memcpy(tv, va, size);
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }
  thread_rpc_free_payload(mobj);
  return ret;
}
int syscall_tzvfs_fchown(int *tzvfs_errno, int fd, uid_t owner, gid_t group){
  int ret = -1;
  struct thread_param params[3];
  params[0] = THREAD_PARAM_VALUE(IN, TZVFS_RPC_FS_FCHOWN, 0, 0);
  params[1] = THREAD_PARAM_VALUE(IN, fd, owner, group);
  params[2] = THREAD_PARAM_VALUE(OUT, 0, 0, 0);
  // 发起RPC调用
  if (TEE_SUCCESS == thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TZVFS, 3, params)) {
    ret = params[2].u.value.a;
    if (ret == -1) *tzvfs_errno = params[2].u.value.b;
  }

  return ret;
}
/*
int tzvfs_ftruncate(int *tzvfs_errno, int fd, off_t length){
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
int tzvfs_fchmod(int *tzvfs_errno, int fd, mode_t mode){
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
void *tzvfs_dlopen(int *tzvfs_errno, const char *filename, int flag){
  DMSG("%s: haven't been realized!\n", __func__);
  return NULL;
}
char *tzvfs_dlerror(int *tzvfs_errno){
  DMSG("%s: haven't been realized!\n", __func__);
  return NULL;
}
void *tzvfs_dlsym(int *tzvfs_errno, void *handle, const char *symbol){
  DMSG("%s: haven't been realized!\n", __func__);
  return NULL;
}
int tzvfs_dlclose(int *tzvfs_errno, void *handle){
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
int tzvfs_mkdir(int *tzvfs_errno, const char *pathname, mode_t mode) {
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
int tzvfs_rmdir(int *tzvfs_errno, const char *pathname){
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
ssize_t tzvfs_readlink(int *tzvfs_errno, const char *path, char *buf, size_t bufsiz){
  DMSG("%s: haven't been realized!\n", __func__);
  return 0;
}
long int tzvfs_sysconf(int *tzvfs_errno, int name){
  long int ret = -1;
  DMSG("%s: haven't been realized!\n", __func__);
  return ret;
}
struct tzvfs_tm *tzvfs_localtime(int *tzvfs_errno, const time_t *timep){
  struct tzvfs_tm * ret = NULL;
  DMSG("%s: haven't been realized!\n", __func__);
  return ret;
}
*/
```

1. 修改optee_os/core/tee目录下的sub.mk文件，将tee_tzvfs.c文件添加编译系统中。

```
// optee_os/core/tee/sub.mk
srcs-y += tee_tzvfs.c
```

2. 同时将tee_tzvfs.h文件保存到optee_os/core/include/tee目录中，以及tzvfs.h文件。

```
// 8optee_os/core/include/tee/tee_tzvfs.h
#ifndef TEE_TZVFS_H
#define TEE_TZVFS_H
#include <tee/tzvfs.h>
int syscall_tzvfs_open(int *tzvfs_errno, const char *filename, int flags, mode_t mode);
int syscall_tzvfs_close(int *tzvfs_errno, int fd);
char *syscall_tzvfs_getcwd(int *tzvfs_errno, char *buf, size_t size);
int syscall_tzvfs_lstat(int *tzvfs_errno, const char* path, struct tzvfs_stat *buf);
int syscall_tzvfs_stat(int *tzvfs_errno, const char *path, struct tzvfs_stat *buf);
int syscall_tzvfs_fstat(int *tzvfs_errno, int fd, struct tzvfs_stat *buf);
int syscall_tzvfs_fcntl(int *tzvfs_errno, int fd, int cmd, struct tzvfs_flock *arg);
ssize_t syscall_tzvfs_read(int *tzvfs_errno, int fd, void *buf, size_t count);
ssize_t syscall_tzvfs_write(int *tzvfs_errno, int fd, const void *buf, size_t count);
uid_t syscall_tzvfs_geteuid(int *tzvfs_errno);
int syscall_tzvfs_unlink(int *tzvfs_errno, const char *pathname);
int syscall_tzvfs_access(int *tzvfs_errno, const char *pathname, int mode);
void *syscall_tzvfs_mmap(int *tzvfs_errno, void *addr, size_t len, int prot, int flags, int fildes,
off_t off);
void *syscall_tzvfs_mremap(int *tzvfs_errno, void *old_address, size_t old_size, size_t new_size, int
flags);
int syscall_tzvfs_munmap(int *tzvfs_errno, void *addr, size_t length);
size_t syscall_tzvfs_strcspn(int *tzvfs_errno, const char *str1, const char *str2);
int syscall_tzvfs_utimes(int *tzvfs_errno, const char *filename, const struct tzvfs_timeval times[2]);
off_t syscall_tzvfs_lseek(int *tzvfs_errno, int fd, off_t offset, int whence);
int syscall_tzvfs_fsync(int *tzvfs_errno, int fd);
char* syscall_tzvfs_getenv(int *tzvfs_errno, const char *name);
pid_t syscall_tzvfs_getpid(int *tzvfs_errno);
time_t syscall_tzvfs_time(int *tzvfs_errno, time_t *t);
unsigned int syscall_tzvfs_sleep(int *tzvfs_errno, unsigned int seconds);
int syscall_tzvfs_gettimeofday(int *tzvfs_errno, struct tzvfs_timeval *tv, struct tzvfs_timezone *tz);
int syscall_tzvfs_fchown(int *tzvfs_errno, int fd, uid_t owner, gid_t group);
#endif
```

3. 修改optee_os/core/include/optee_rpc_cmd.h文件增加OPTEE_MSG_RPC_CMD_TZVFS宏：

```
// optee_os/core/include/optee_rpc_cmd.h
/*
 * TZVFS
 */
#define OPTEE_MSG_RPC_CMD_TZVFS     66
```

- **Updating the OP-TEE**

```
• Building the OP-TEE
  <OP-TEE installation directory>/README.HOW_TO.txt helper file tells the instructions.
  $> export FIP_DEPLOYDIR_ROOT=$PWD/../../FIP_artifacts
  $> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
  $> make -f $PWD/../Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-dk2 all
  The generated FIP images are available in $FIP_DEPLOYDIR_ROOT/fip
• Updating the SDK
  $> cp -r $PWD/../build/stm32mp157c-dk2/export-ta_arm32/* <STM32MP1 SDK PATH>/sysroots/cortexa7t2hf-
  neon-vfpv4-ostl-linux-gnueabi/usr/include/optee/export-user_ta
• Deploying the OP-TEE
  Replace the fip-stm32mp157c-dk2-optee.bin and recreate the image.
  $> cp $FIP_DEPLOYDIR_ROOT/fip/fip-stm32mp157c-dk2-optee.bin <STM32MP1 IMAGE PATH>/stm32mp1/fip
• Create the Image
  $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/images/stm32mp1/scripts/
  $> ./create_sdcard_from_flashlayout.sh ../flashlayout_st-image-
  weston/optee/FlashLayout_sdcard_stm32mp157c-dk2-optee.tsv
• Image flashing
  $> sudo dd if=../flashlayout_st-image-weston/extensible/../../FlashLayout_sdcard_stm32mp157c-dk2-
  optee.raw of=/dev/sdb bs=8M conv=fdatasync status=progress
```

### Modifying the OPTEE-CLIENT

- **添加RPC调用**

1. 修改optee_client/tee-supplicant/src/optee_msg_supplicant.h文件增加OPTEE_MSG_RPC_CMD_TZVFS宏：

```
// optee_client/tee-supplicant/src/optee_msg_supplicant.h
/*
 * TZVFS
 */
#define OPTEE_MSG_RPC_CMD_TZVFS    66
```

2. 修改optee_client/tee-supplicant/src/tee_supplicant.c文件增加tee_supp_tzvfs_process函数：

```
// optee_client/tee-supplicant/src/tee_supplicant.c
#include <tee_supp_tzvfs.h>

static bool process_one_request(struct thread_arg *arg)
{
    ......
    switch (func) {
    case  OPTEE_MSG_RPC_CMD_TZVFS:
        ret = tee_supp_tzvfs_process(num_params, params);
        break;
    ......
    }
}
```

3. 添加optee_client/tee-supplicant/src/tee_supp_tzvfs.c文件处理tzvfs相关功能：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/time.h>  // utimes() and gettimeofday()
#include <fcntl.h>
#include <sys/mman.h>
#include <time.h>  // time() and localtime()
#include <teec_trace.h>
#include <optee_msg_supplicant.h>
#include <tee_supplicant.h>
#include <tee_supp_tzvfs.h>
#ifndef __aligned
#define __aligned(x) __attribute__((__aligned__(x)))
#endif
#include <linux/tee.h>
static TEEC_Result ree_tzvfs_open(size_t num_params,
                  struct tee_ioctl_param *params)
{
    int ret = -1;
    char *filename = NULL;
    int flags = 0;
    mode_t mode = 0;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_INPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    filename = tee_supp_param_to_va(params + 1);
    if (!filename) return TEEC_ERROR_BAD_PARAMETERS;
    flags = params[0].b;
    mode = params[0].c;

    ret = open(filename, flags, mode);
    params[2].a = ret;
    if (ret == -1) params[2].b = errno;
    printf("DMSG: call%s, filename=%s, flags=%d, mode=%d, ret=%d, errno=%s\n", __func__, filename,
flags, mode, ret, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_close(size_t num_params,
                  struct tee_ioctl_param *params)
{
    int ret = -1;
```

```
    int fd = -1;
    if (num_params != 2 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    fd = params[0].b;
    ret = close(fd);
    params[1].a = ret;
    if (ret == -1) params[1].b = errno;
    printf("DMSG: call%s, fd=%d, ret=%d, errno=%s\n", __func__, fd, ret, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_getcwd(size_t num_params,
                  struct tee_ioctl_param *params)
{
    char *ret = NULL;
    char *buf = NULL;
    size_t size = 0;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_OUTPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 1);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    size = MEMREF_SIZE(params + 1);

    ret = getcwd(buf, size);
    params[2].a = ret;
    if (ret == NULL) params[2].b = errno;
    printf("DMSG: call%s, buf=%x, size=%d, ret=%x, errno=%s\n", __func__, buf, size, ret,
strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_lstat(size_t num_params,
                  struct tee_ioctl_param *params)
{
    int ret = -1;
    char *path = NULL;
    struct stat *buf = NULL;
    if (num_params != 4 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_INPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_OUTPUT ||
        (params[3].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    path = tee_supp_param_to_va(params + 1);
    if (!path) return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 2);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    ret = lstat(path, buf);
    params[3].a = ret;
    if (ret == -1) params[3].b = errno;
    printf("DMSG: call%s, path=%s, buf=%x, ret=%d, sizeof(struct flock)=%d, st_uid=%d, errno=%s\n",
__func__, path, buf, ret, sizeof(struct flock), buf->st_uid, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_stat(size_t num_params,
                  struct tee_ioctl_param *params)
{
    int ret = -1;
    char *path = NULL;
    struct stat *buf = NULL;
    if (num_params != 4 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_INPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
```

```
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_OUTPUT ||
        (params[3].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    path = tee_supp_param_to_va(params + 1);
    if (!path) return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 2);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    ret = lstat(path, buf);
    params[3].a = ret;
    if (ret == -1) params[3].b = errno;
    printf("DMSG: call%s, path=%s, buf=%x, ret=%d, sizeof(struct stat)=%d, st_uid=%d, errno=%s\n",
__func__, path, buf, ret, sizeof(struct stat), buf->st_uid, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_fstat(size_t num_params,
                   struct tee_ioctl_param *params)
{
    int ret = -1;
    int fd = -1;
    struct stat *buf = NULL;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_OUTPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 1);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    fd = params[0].b;
    ret = fstat(fd, buf);
    params[2].a = ret;
    if (ret == -1) params[2].b = errno;
    printf("DMSG: call%s, fd=%d, buf=%x, ret=%d, errno=%s\n", __func__, fd, buf, ret,
strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_fcntl(size_t num_params,
                   struct tee_ioctl_param *params)
{
    int ret = -1;
    int fd = -1;
    int cmd = 0;
    struct flock *buf = NULL;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_INPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 1);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    fd = params[0].b;
    cmd = params[0].c;
    ret = fcntl(fd, cmd, buf);
    params[2].a = ret;
    if (ret == -1) params[2].b = errno;
    printf("DMSG: call%s, fd=%d, cmd=%d, buf=%x, ret=%d, sizeof(struct flock)=%d, errno=%s\n",
__func__, fd, cmd, buf, ret, sizeof(struct flock), strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_read(size_t num_params,
                   struct tee_ioctl_param *params)
{
    ssize_t ret = -1;
    int fd = -1;
    void *buf = NULL;
    size_t count = 0;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_MEMREF_OUTPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
```

```
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    buf = tee_supp_param_to_va(params + 1);
    if (!buf) return TEEC_ERROR_BAD_PARAMETERS;
    count = MEMREF_SIZE(params + 1);
    fd = params[0].b;
    ret = read(fd, buf, count);
    params[2].a = ret;
    if (ret == -1) params[2].b = errno;
    printf("DMSG: call%s, fd=%d, buf=%x, count=%d, ret=%d, errno=%s\n", __func__, fd, buf, count, ret,
strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_write(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_geteuid(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_unlink(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_access(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_mmap(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_mremap(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_munmap(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_strcspn(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_utimes(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_lseek(size_t num_params,
                   struct tee_ioctl_param *params)
{
    off_t ret = -1;
    int fd = -1;
    off_t offset = 0;
    int whence = 0;
    if (num_params != 3 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
```

```
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[2].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;
    fd = params[1].a;
    offset = params[1].b;
    whence = params[1].c;
    ret = lseek(fd, offset, whence);
    params[2].a = ret;
    if (ret == -1) params[2].b = errno;
    printf("DMSG: call%s, fd=%d, offset=%d, whence=%d, ret=%d, errno=%s\n", __func__, fd, offset,
whence, ret, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_fsync(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_getenv(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_getpid(size_t num_params,
                   struct tee_ioctl_param *params)
{
    pid_t ret = -1;
    if (num_params != 2 ||
        (params[0].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_INPUT ||
        (params[1].attr & TEE_IOCTL_PARAM_ATTR_TYPE_MASK) !=
            TEE_IOCTL_PARAM_ATTR_TYPE_VALUE_OUTPUT)
        return TEEC_ERROR_BAD_PARAMETERS;

    ret = getpid();
    params[1].a = ret;
    if (ret == -1) params[1].b = errno;
    printf("DMSG: call%s, ret=%d, errno=%s\n", __func__, ret, strerror(errno));
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_time(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_sleep(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_gettimeofday(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
static TEEC_Result ree_tzvfs_fchown(size_t num_params,
                   struct tee_ioctl_param *params)
{
    printf("%s has not been realised!\n", __func__);
    return TEEC_SUCCESS;
}
TEEC_Result tee_supp_tzvfs_process(size_t num_params,
                struct tee_ioctl_param *params)
{
    switch (params->a) {
    case TZVFS_RPC_FS_OPEN:
        return ree_tzvfs_open(num_params, params);
    case TZVFS_RPC_FS_CLOSE:
        return ree_tzvfs_close(num_params, params);
    case TZVFS_RPC_FS_GETCWD:
        return ree_tzvfs_getcwd(num_params, params);
    case TZVFS_RPC_FS_LSTAT:
```

```
        return ree_tzvfs_lstat(num_params, params);
    case TZVFS_RPC_FS_STAT:
        return ree_tzvfs_stat(num_params, params);
    case TZVFS_RPC_FS_FSTAT:
        return ree_tzvfs_fstat(num_params, params);
    case TZVFS_RPC_FS_FCNTL:
        return ree_tzvfs_fcntl(num_params, params);
    case TZVFS_RPC_FS_READ:
        return ree_tzvfs_read(num_params, params);
    case TZVFS_RPC_FS_WRITE:
        return ree_tzvfs_write(num_params, params);
    case TZVFS_RPC_FS_GETEUID:
        return ree_tzvfs_geteuid(num_params, params);
    case TZVFS_RPC_FS_UNLINK:
        return ree_tzvfs_unlink(num_params, params);
    case TZVFS_RPC_FS_ACCESS:
        return ree_tzvfs_access(num_params, params);
    case TZVFS_RPC_FS_MMAP:
        return ree_tzvfs_mmap(num_params, params);
    case TZVFS_RPC_FS_MREMAP:
        return ree_tzvfs_mremap(num_params, params);
    case TZVFS_RPC_FS_MUNMAP:
        return ree_tzvfs_munmap(num_params, params);
    case TZVFS_RPC_FS_STRCSPN:
        return ree_tzvfs_strcspn(num_params, params);
    case TZVFS_RPC_FS_UTIMES:
        return ree_tzvfs_utimes(num_params, params);
    case TZVFS_RPC_FS_LSEEK:
        return ree_tzvfs_lseek(num_params, params);
    case TZVFS_RPC_FS_FSYNC:
        return ree_tzvfs_fsync(num_params, params);
    case TZVFS_RPC_FS_GETENV:
        return ree_tzvfs_getenv(num_params, params);
    case TZVFS_RPC_FS_GETPID:
        return ree_tzvfs_getpid(num_params, params);
    case TZVFS_RPC_FS_TIME:
        return ree_tzvfs_time(num_params, params);
    case TZVFS_RPC_FS_SLEEP:
        return ree_tzvfs_sleep(num_params, params);
    case TZVFS_RPC_FS_GETTIMEOFDAY:
        return ree_tzvfs_gettimeofday(num_params, params);
    case TZVFS_RPC_FS_FCHOWN:
        return ree_tzvfs_fchown(num_params, params);
    default:
        return TEEC_ERROR_BAD_PARAMETERS;
    }
}
```

4. 同时添加optee_client/tee-supplicant/src/tee_supp_tzvfs.h头文件：

```
#ifndef TEE_SUPP_TZVFS_H
#define TEE_SUPP_TZVFS_H
#include <tee_client_api.h>
struct tee_ioctl_param;
TEEC_Result tee_supp_tzvfs_process(size_t num_params,
                struct tee_ioctl_param *params);
// TZVFS define start
#define TZVFS_FS_NAME_MAX 350
/*
 * tzvfs_open
 *
 * [in]     value[0].a      TZVFS_RPC_FS_OPEN
 *          value[0].b      flags
 *          value[0].c      mode
 * [in]     memref[1]         A string holding the file name
 * [out]    value[2].a      File descriptor of open file
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_OPEN       0
/*
 * tzvfs_close
 *
 * [in]     value[0].a      TZVFS_RPC_FS_CLOSE
 *          value[0].b      fd
 * [out]    value[1].a      ret
 *          value[1].b      errno
 */
```

```
#define TZVFS_RPC_FS_CLOSE      1
/*
 * tzvfs_getcwd
 *
 * [in]     value[0].a      TZVFS_RPC_FS_GETCWD
 * [out]    memref[1]         A string holding the CWD name
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_GETCWD     2
/*
 * tzvfs_lstat
 *
 * [in]     value[0].a      TZVFS_RPC_FS_LSTAT
 * [in]     memref[1]         A string holding the path name
 * [out]    memref[2]         struct tzvfs_stat
 * [out]    value[3].a      ret
 *          value[3].b      errno
 */
#define TZVFS_RPC_FS_LSTAT      3
/*
 * tzvfs_stat
 *
 * [in]     value[0].a      TZVFS_RPC_FS_STAT
 * [in]     memref[1]         A string holding the path name
 * [out]    memref[2]         struct tzvfs_stat
 * [out]    value[3].a      ret
 *          value[3].b      errno
 */
#define TZVFS_RPC_FS_STAT       4
/*
 * tzvfs_fstat
 *
 * [in]     value[0].a      TZVFS_RPC_FS_FSTAT
 *          value[0].b      fd
 * [out]    memref[1]         struct tzvfs_stat
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_FSTAT      5
/*
 * tzvfs_fcntl
 *
 * [in]     value[0].a      TZVFS_RPC_FS_FCNTL
 *          value[0].b      fd
 *          value[0].c      cmd
 * [out]    memref[1]         struct tzvfs_flock
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_FCNTL      6
/*
 * tzvfs_read
 *
 * [in]     value[0].a      TZVFS_RPC_FS_READ
 *          value[0].b      fd
 * [out]    memref[1]         buf
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_READ       7
/*
 * tzvfs_write
 *
 * [in]     value[0].a      TZVFS_RPC_FS_WRITE
 *          value[0].b      fd
 * [in]     memref[1]         buf
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_WRITE      8
/*
 * tzvfs_geteuid
 *
 * [in]     value[0].a      TZVFS_RPC_FS_GETEUID
 * [out]    value[1].a      ret
 *          value[1].b      errno
 */
```

```
#define TZVFS_RPC_FS_GETEUID        9
/*
 * tzvfs_unlink
 *
 * [in]     value[0].a      TZVFS_RPC_FS_UNLINK
 * [in]     memref[1]         A string holding the pathname
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_UNLINK     10
/*
 * tzvfs_access
 *
 * [in]     value[0].a      TZVFS_RPC_FS_ACCESS
 *          value[0].b      mode
 * [in]     memref[1]         A string holding the pathname
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_ACCESS     11
/*
 * tzvfs_mmap
 *
 * [in]     value[0].a      TZVFS_RPC_FS_MMAP
 * [in]     value[1].a      addr
 *          value[1].b      len
 *          value[1].c      prot
 * [in]     value[2].a      flags
 *          value[2].b      fildes
 *          value[2].c      off
 * [out]    value[3].a      ret
 *          value[3].b      errno
 */
#define TZVFS_RPC_FS_MMAP       12
/*
 * tzvfs_mremap
 *
 * [in]     value[0].a      TZVFS_RPC_FS_MREMAP
 *          value[0].b      old_address
 *          value[0].c      old_size
 * [in]     value[1].a      new_size
 *          value[1].b      flags
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_MREMAP     13
/*
 * tzvfs_munmap
 *
 * [in]     value[0].a      TZVFS_RPC_FS_MUNMAP
 *          value[0].b      addr
 *          value[0].c      length
 * [out]    value[1].a      ret
 *          value[1].b      errno
 */
#define TZVFS_RPC_FS_MUNMAP     14
/*
 * tzvfs_strcspn
 *
 * [in]     value[0].a      TZVFS_RPC_FS_STRCSPN
 * [in]     memref[1]         A string holding the str1
 * [in]     memref[2]         A string holding the str1
 * [out]    value[3].a      ret
 */
#define TZVFS_RPC_FS_STRCSPN        15

/*
 * tzvfs_utimes
 *
 * [in]     value[0].a      TZVFS_RPC_FS_UTIMES
 * [in]     memref[1]         A string holding the filename
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_UTIMES     16
/*
 * tzvfs_lseek
 *
```

```
 * [in]     value[0].a      TZVFS_RPC_FS_LSEEK
 * [in]     value[1].a      fd
 *          value[1].b      offset
 *          value[1].c      whence
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_LSEEK      17
/*
 * tzvfs_fsync
 *
 * [in]     value[0].a      TZVFS_RPC_FS_FSYNC
 * [in]     value[1].a      fd
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_FSYNC      18
/*
 * tzvfs_getenv
 *
 * [in]     value[0].a      TZVFS_RPC_FS_GETENV
 * [in]     memref[1]         A string holding the name
 * [out]    value[2].a      ret
 */
#define TZVFS_RPC_FS_GETENV     20
/*
 * tzvfs_getpid
 *
 * [in]     value[0].a      TZVFS_RPC_FS_GETPID
 * [out]    value[1].a      ret
 *          value[1].b      errno
 */
#define TZVFS_RPC_FS_GETPID     21
/*
 * tzvfs_time
 *
 * [in]     value[0].a      TZVFS_RPC_FS_TIME
 * [out]    value[1].a      ret
 *          value[1].b      errno
 */
#define TZVFS_RPC_FS_TIME       22
/*
 * tzvfs_sleep
 *
 * [in]     value[0].a      TZVFS_RPC_FS_SLEEP
 *          value[0].b      seconds
 * [out]    value[1].a      ret
 */
#define TZVFS_RPC_FS_SLEEP      23

/*
 * tzvfs_gettimeofday
 *
 * [in]     value[0].a      TZVFS_RPC_FS_GETTIMEOFDAY
 * [out]    memref[1]         struct tzvfs_timeval
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_GETTIMEOFDAY       24
/*
 * tzvfs_fchown
 *
 * [in]     value[0].a      TZVFS_RPC_FS_FCHOWN
 * [in]     value[1].a      fd
 *          value[1].b      owner
 *          value[1].c      group
 * [out]    value[2].a      ret
 *          value[2].b      errno
 */
#define TZVFS_RPC_FS_FCHOWN     25
// TZVFS define end
#endif
```

5. 修改optee_client/tee-supplicant/Makefile文件增加需要编译的源文件：

```
TEES_SRCS   := tee_supplicant.c \
          teec_ta_load.c \
```

```
            tee_supp_fs.c \
            rpmb.c \
            handle.c \
            tee_tpm.c \
            smaug_guorui.c \
            sqlite3.c \
            defs.c \
            dbqueue.c \
            mhtdefs.c \
            mhtfile.c \
            tee_supp_tzvfs.c
```

- **Updating the OPTEE-CLIENT**

```
• Building the OPTEE-CLIENT
  $> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
  $> make
• Deploying the OPTEE-CLIENT
  Replace the tee-supplicant in board.
  $> scp out/tee-supplicant/tee-supplicant root@<ip of board>:/usr/bin
```