Code of HelloEnclave

App

App.h

```
#ifndef _APP_H_
#define _APP_H_

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

#include "sgx_error.h"       /* sgx_status_t */
#include "sgx_eid.h"     /* sgx_enclave_id_t */

#ifndef TRUE
# define TRUE 1
#endif

#ifndef FALSE
# define FALSE 0
#endif

# define TOKEN_FILENAME   "enclave.token"
# define ENCLAVE_FILENAME "enclave.signed.so"

extern sgx_enclave_id_t global_eid;    /* global enclave id */

#if defined(__cplusplus)
extern "C" {
#endif

#if defined(__cplusplus)
}
#endif

#endif /* !_APP_H_ */
```

App.cpp

```
#include <stdio.h>
#include <string.h>

# include <unistd.h>
# include <pwd.h>
```

```
# define MAX_PATH FILENAME_MAX

#include "sgx_urts.h"
#include "App.h"
#include "Enclave_u.h"

/* Global EID shared by multiple threads */
sgx_enclave_id_t global_eid = 0;

typedef struct _sgx_errlist_t {
    sgx_status_t err;
    const char *msg;
    const char *sug; /* Suggestion */
} sgx_errlist_t;

/* Error code returned by sgx_create_enclave */
static sgx_errlist_t sgx_errlist[] = {
        {
                SGX_ERROR_UNEXPECTED,
                "Unexpected error occurred.",
                NULL
        },
        {
                SGX_ERROR_INVALID_PARAMETER,
                "Invalid parameter.",
                NULL
        },
        {
                SGX_ERROR_OUT_OF_MEMORY,
                "Out of memory.",
                NULL
        },
        {
                SGX_ERROR_ENCLAVE_LOST,
                "Power transition occurred.",
                "Please refer to the sample \"PowerTransition\" for details."
        },
        {
                SGX_ERROR_INVALID_ENCLAVE,
                "Invalid enclave image.",
                NULL
        },
        {
                SGX_ERROR_INVALID_ENCLAVE_ID,
                "Invalid enclave identification.",
                NULL
        },
        {
                SGX_ERROR_INVALID_SIGNATURE,
                "Invalid enclave signature.",
                NULL
        },
        {
                SGX_ERROR_OUT_OF_EPC,
                "Out of EPC memory.",
                NULL
        },
```

```
        {
                SGX_ERROR_NO_DEVICE,
                "Invalid SGX device.",
                "Please make sure SGX module is enabled in the BIOS, and install
SGX driver afterwards."
        },
        {
                SGX_ERROR_MEMORY_MAP_CONFLICT,
                "Memory map conflicted.",
                NULL
        },
        {
                SGX_ERROR_INVALID_METADATA,
                "Invalid enclave metadata.",
                NULL
        },
        {
                SGX_ERROR_DEVICE_BUSY,
                "SGX device was busy.",
                NULL
        },
        {
                SGX_ERROR_INVALID_VERSION,
                "Enclave version was invalid.",
                NULL
        },
        {
                SGX_ERROR_INVALID_ATTRIBUTE,
                "Enclave was not authorized.",
                NULL
        },
        {
                SGX_ERROR_ENCLAVE_FILE_ACCESS,
                "Can't open enclave file.",
                NULL
        },
};

/* Check error conditions for loading enclave */
void print_error_message(sgx_status_t ret) {
    size_t idx = 0;
    size_t ttl = sizeof sgx_errlist / sizeof sgx_errlist[0];

    for (idx = 0; idx < ttl; idx++) {
        if (ret == sgx_errlist[idx].err) {
            if (NULL != sgx_errlist[idx].sug)
                printf("Info: %s\n", sgx_errlist[idx].sug);
            printf("Error: %s\n", sgx_errlist[idx].msg);
            break;
        }
    }

    if (idx == ttl)
        printf("Error code is 0x%X. Please refer to the \"Intel SGX SDK Developer
Reference\" for more details.\n",
               ret);
}
```

```
/* Initialize the enclave:
 *   Step 1: try to retrieve the launch token saved by last transaction
 *   Step 2: call sgx_create_enclave to initialize an enclave instance
 *   Step 3: save the launch token if it is updated
 */
int initialize_enclave(void) {
    char token_path[MAX_PATH] = {'\0'};
    sgx_launch_token_t token = {0};
    sgx_status_t ret = SGX_ERROR_UNEXPECTED;
    int updated = 0;

    /* Step 1: try to retrieve the launch token saved by last transaction
     *         if there is no token, then create a new one.
     */
    /* try to get the token saved in $HOME */
    const char *home_dir = getpwuid(getuid())->pw_dir;

    if (home_dir != NULL &&
        (strlen(home_dir) + strlen("/") + sizeof(TOKEN_FILENAME) + 1) <=
MAX_PATH) {
        /* compose the token path */
        strncpy(token_path, home_dir, strlen(home_dir));
        strncat(token_path, "/", strlen("/"));
        strncat(token_path, TOKEN_FILENAME, sizeof(TOKEN_FILENAME) + 1);
    } else {
        /* if token path is too long or $HOME is NULL */
        strncpy(token_path, TOKEN_FILENAME, sizeof(TOKEN_FILENAME));
    }

    FILE *fp = fopen(token_path, "rb");
    if (fp == NULL && (fp = fopen(token_path, "wb")) == NULL) {
        printf("Warning: Failed to create/open the launch token file \"%s\".\n",
token_path);
    }

    if (fp != NULL) {
        /* read the token from saved file */
        size_t read_num = fread(token, 1, sizeof(sgx_launch_token_t), fp);
        if (read_num != 0 && read_num != sizeof(sgx_launch_token_t)) {
            /* if token is invalid, clear the buffer */
            memset(&token, 0x0, sizeof(sgx_launch_token_t));
            printf("Warning: Invalid launch token read from \"%s\".\n",
token_path);
        }
    }
    /* Step 2: call sgx_create_enclave to initialize an enclave instance */
    /* Debug Support: set 2nd parameter to 1 */
    ret = sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG, &token, &updated,
&global_eid, NULL);
    if (ret != SGX_SUCCESS) {
        print_error_message(ret);
        if (fp != NULL) fclose(fp);
        return -1;
    }

    /* Step 3: save the launch token if it is updated */
    if (updated == FALSE || fp == NULL) {
```

```
        /* if the token is not updated, or file handler is invalid, do not
perform saving */
        if (fp != NULL) fclose(fp);
        return 0;
    }

    /* reopen the file with write capablity */
    fp = freopen(token_path, "wb", fp);
    if (fp == NULL) return 0;
    size_t write_num = fwrite(token, 1, sizeof(sgx_launch_token_t), fp);
    if (write_num != sizeof(sgx_launch_token_t))
        printf("Warning: Failed to save launch token to \"%s\".\n", token_path);
    fclose(fp);
    return 0;
}

/* OCall functions */
void ocall_print_string(const char *str) {
    /* Proxy/Bridge will check the length and null-terminate
     * the input string to prevent buffer overflow.
     */
    printf("%s", str);
}


/* Application entry */
int SGX_CDECL main(int argc, char *argv[]) {
    (void) (argc);
    (void) (argv);


    /* Initialize the enclave */
    if (initialize_enclave() < 0) {
        printf("Enter a character before exit ...\n");
        getchar();
        return -1;
    }

    printf_helloworld(global_eid);

    /* Destroy the enclave */
    sgx_destroy_enclave(global_eid);

    return 0;
}
```

Enclave

Enclave.edl

```
enclave {

    /* Import ECALL/OCALL from sub-directory EDLs.
     *  [from]: specifies the location of EDL file.
     *  [import]: specifies the functions to import,
     *  [*]: implies to import all functions.
     */

    trusted {
        public void printf_helloworld();
    };

    /*
     * ocall_print_string - invokes OCALL to display string buffer inside the
enclave.
     *  [in]: copy the string buffer to App outside.
     *  [string]: specifies 'str' is a NULL terminated buffer.
     */
    untrusted {
        void ocall_print_string([in, string] const char *str);
    };

};
```

Enclave.h

```
#ifndef _ENCLAVE_H_
#define _ENCLAVE_H_

#include <stdlib.h>
#include <assert.h>

#if defined(__cplusplus)
extern "C" {
#endif

void printf(const char *fmt, ...);
void printf_helloworld();

#if defined(__cplusplus)
}
#endif

#endif /* !_ENCLAVE_H_ */
```

Enclave.cpp

```
#include <stdarg.h>
#include <stdio.h>      /* vsnprintf */

#include "Enclave.h"
#include "Enclave_t.h"  /* print_string */

/*
 * printf:
 *   Invokes OCALL to display the enclave buffer to the terminal.
 */
void printf(const char *fmt, ...)
{
    char buf[BUFSIZ] = {'\0'};
    va_list ap;
    va_start(ap, fmt);
    vsnprintf(buf, BUFSIZ, fmt, ap);
    va_end(ap);
    ocall_print_string(buf);
}

void printf_helloworld()
{
    printf("Hello World\n");
}
```