The NTNX HASH Device Library

Last revision: August 2018
Expected completion time: under 60 mins

Introduction

You have been tasked to write a Linux library which will enable applications to use a fictional MD5 hashing device. This is managed by a kernel driver which provides a userspace interface via a character device. Your library will interact solely with this interface and no kernel programming is required. Applications using your library will be able to compute the digest of arbitrarily-sized buffers very efficiently, given the bulk of the work will be offloaded to the device instead of being performed by a CPU.

Library API

The API of this library consists of three functions (which you are expected to implement):

```
ntnx_hash_t *ntnx_hash_setup(void);
char *ntnx_hash_compute(ntnx_hash_t *ctx, void *buf, size_t len);
int ntnx_hash_destroy(ntnx_hash_t *ctx);
```

To use the library, an application must first create a context by calling **`ntnx_hash_setup()`**. Using the created context, it can compute MD5 digests by calling **`ntnx_hash_compute()`**. Apart from the context, **`ntnx_hash_compute()`** takes a **`buf`** of size **`len`** which must contain the data to be hashed. Finally, the application can destroy the context with **`ntnx_hash_destroy()`**. Multi-threaded applications should be able to share a context across threads.

Return values

`ntnx_hash_setup()` returns a pointer to a **`ntnx_hash_t`** context. This should be opaque to the user and you may define it internally in your library as you see fit. In the case of errors, the function must return **`NULL`** and set **`errno`** appropriately.

`ntnx_hash_compute()` returns a pointer to a null-terminated array of 33 chars (including **`NUL`**) which will be allocated in the heap (by your library). It must be later **`free()`**d by the caller. Multi-threaded programs should be able to call this concurrently. Upon successful computation, the array must contain the 32-byte string representation of the MD5 digest for the buffer provided. In the case of errors, the function must return **`NULL`** and set **`errno`** appropriately.

`ntnx_hash_destroy()` should release all resources associated with the context. It returns zero on success or -1 on error, in which case it should set **`errno`** appropriately.

The **`errno`** codes to be used are left at your discretion, as well as the state of the system in the case of errors.

Character Device

Systems with a valid device and a correctly loaded kernel module will provide a character device on **/dev/ntnx_hash**. Your library must open this device upon context creation. On context destruction, it must close the device. While a context is open, it can issue the following **ioctl()**s to operate the device driver:

1) API version retrieval

```
#define NTNX_HASH_GET_API_VERSION  0
```

This takes the address of an **unsigned int** as an "out" parameter. The driver will fill it with its API version. For the purposes of this exercise, only version 1 exists. The **ioctl()** itself will return 0 on success and -1 on error. Your library must check it is compatible with the device driver API.

2) Hash computation

```
#define NTNX_HASH_COMPUTE          1
```

This takes a **struct ntnx_hash_compute** as defined below.

```
struct ntnx_hash_compute {
    void   *buf;     // pointer to the area for hashing
    size_t len;      // length of area for checksumming
    void   *hash;    // pointer to the area for the computed hash
};
```

The driver will offload the area of size **len** pointed to by **buf** to the device. The device will calculate the MD5 hash of the area and write the 32-byte string representation of the digest plus a **NUL** to the address pointed to by **hash**. The **ioctl()** itself will return 0 on success and -1 on error. The device driver expects invocations of this **ioctl()** to be serialised within a context.

Expected solution

Your solution should consist of two files:

1) **ntnx_hash.h**
This file should contain the headers for using your library.

2) **ntnx_hash.c**
This file should contain the implementation of your library.

