NVS

Non-Volatile Storage - Энергонезависимое хранилище.

NVS оперирует:

- страницами
- записями

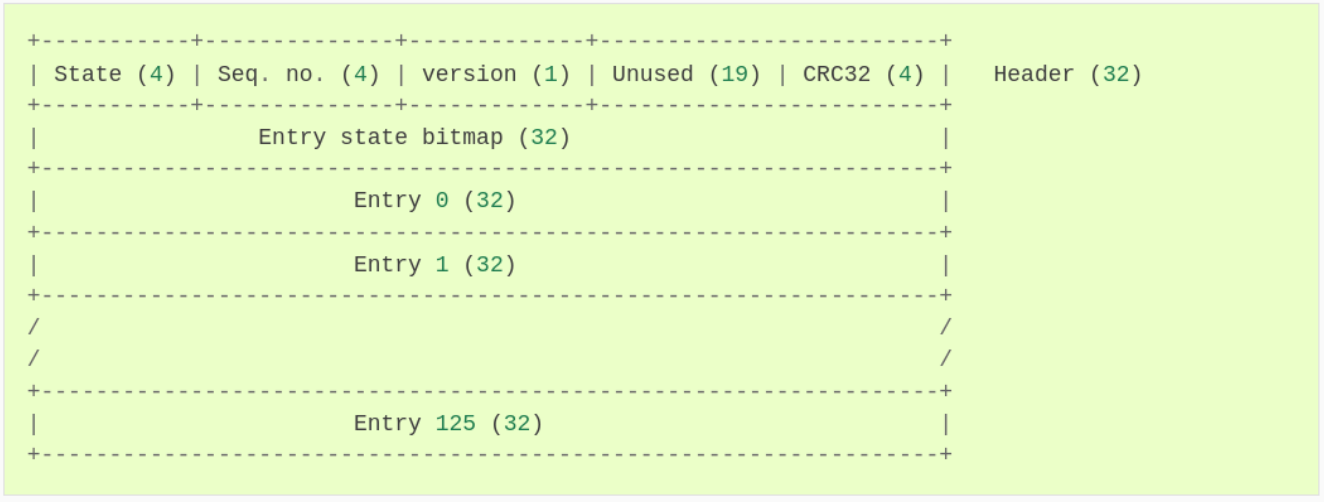
Страница - это логическая структура в которой хранится часть общего набора данных. Занимает один сектор - 4096 байт.

Запись - одна пара ключ-значение. Размер 32 байта, но данных в нем - 8 байт. Если данные занимают больше 8 байт, то распределяются по нескольким записям.

Типы данных:

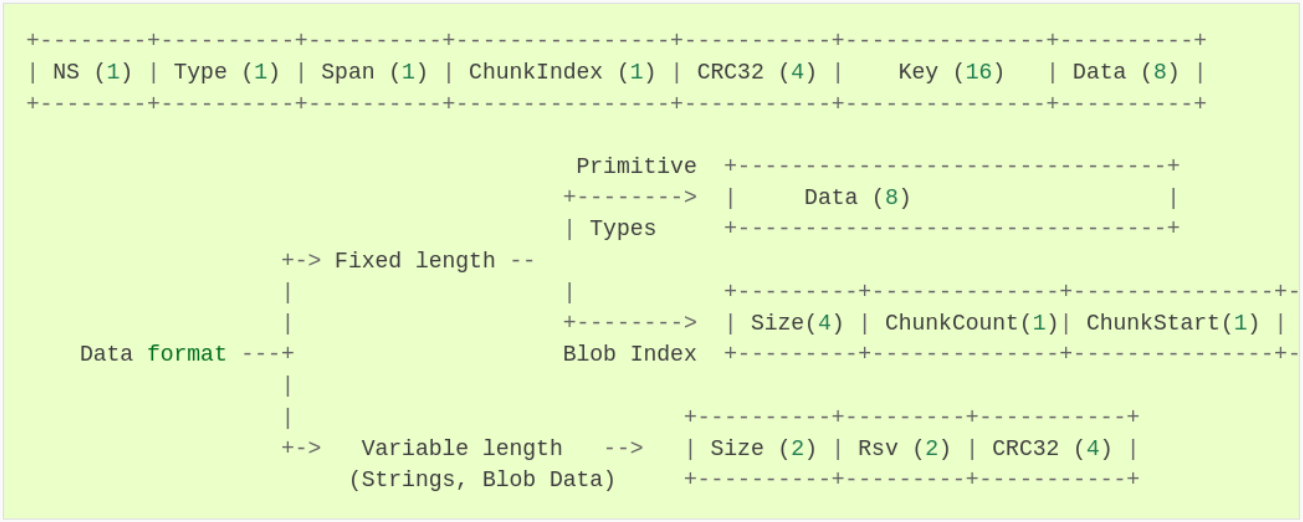
- uint8_t - 64_t.
- Строки меньше 4000 байт
- Двоичные данные (blob)

Структура страницы



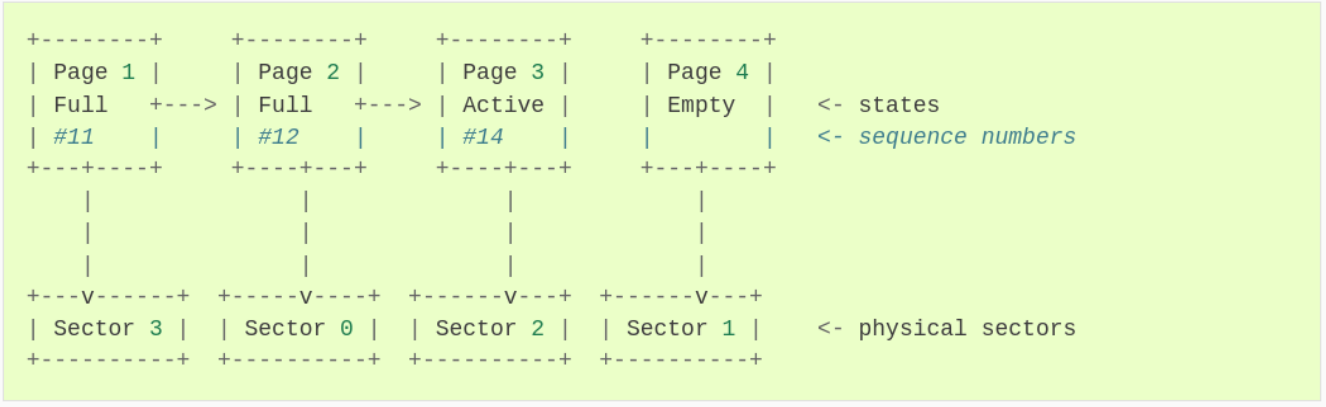
Страница хранит в голове размером 32 байта: состояние, версию, порядковый номер и контрольную сумму.
Далее идет 32 байт состояний записей.
Все оставшееся пространство страницы, а это 4096 - 64 = 4032 используются для самих записей.
126 записей.

Структура хранения записей:



По схеме смотрим переменные фиксированной длинны или изменяемой.
В записи, как уже говорилось, 32 байта. На схеме подробно описано поле дата, 8 байт.
Если данные фиксированной длины и это примитив, то занимается все 8 байт данных.
В случае фиксированной длины, но blob объекта - записывается размер 4 байта, количество чанков, начало чанка и 2 байта зерезервированны.
Если это переменная не фиксированной длины, а то есть строка или blob записывается 2 байта размер, 2 байта зарезервированы и 4 байта хеш суммы.

Структура физического расположения страниц



Каждая страница имеет свое состояние и порядковый номер, а так же может иметь указатель на следующую страницу.
Из схемы видно, что секторы связаны не по порядку со страницами.

Ключ - это строка менее 15 символов, включая `\0`. Он работает в пространстве имен, которых может быть в разделе до 254 шт. Пространства имен имеют те же ограничения на 15 символов, что и ключи. Одинаковые имена namespace приведут к перезаписи. Используйте одинаковые пространства имен только на разных разделах. Вложенность пространств имен не предусмотрена. Пространства имен не ограничены одной страницей в памяти, они распределяются по всему NVS разделу.

Потребление ресурсов

- Каждый 1 мб не зависимо от заполнения потребляет 22 кб ОЗУ, а каждые 1000 ключей 5,5 ОЗУ.
- Продолжительность инициализации пропорциональна количеству существующих ключей. 1000 ключей - 0,5 сек..
- Есть возможно использовать PSRAM, но скорость операций с целыми числами падает примерно в 2,5 раза.

Как устроена NVS

NVS хранит данные последовательно друг за другом, а новые записи добавляются в конец.
Запись при удалении помечается как удаленноая, а не затирается.
При изменении значения, старая ячейка помечается как удаленная, а новая пишется в конец.

Если страница закончилась, то система открывает новую. Страницы имеют порядковые номера. Чем больше, тем новее страница.
Библиотека проверяет порядковые номера страниц найденных в каждом секторе флеш памяти и организует страницы в список на основе этих номеров.

Когда все записи будут помечены как удаленные, страница может быть "отформатирована" для повторного использования. Пока есть свободные страницы - обычно используются новые.

Когда все страницы заполнены, nvs попробует отформатировать страницы, занятые только удаленными записями. Если осталось мало записей, то NVS может принудительно их переместить на новое место для стирания старой таблицы и её повторного использования.
Все это направлено на уменьшение износа FLASH памяти.

Практическая часть

Перейдем в каталог с нашим проектом
`cd ~/esp/led`
Вызываем алиас из прошлым видео
`idf`
Вставляем новый код в наш файл
`vim main/led.c`

```
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "nvs_flash.h"
#include "nvs.h"
#include "esp_log.h"
```

```
void app_main(void)
{
    // инициализируем nvs
    esp_err_t err = nvs_flash_init();
    // если нет свободных страниц или не распознан nvs
    if ((err == ESP_ERR_NVS_NO_FREE_PAGES) || (err ==
ESP_ERR_NVS_NEW_VERSION_FOUND)) {
      ESP_LOGW("NVS", "Очистка NVS раздела...");
      // очищаем nvs
      nvs_flash_erase();
      // инициализируем после очистки
      err = nvs_flash_init();
    };
    if (err == ESP_OK) {
      ESP_LOGI("NVS", "NVS partition initilized");
    } else {
      ESP_LOGE("NVS", "NVS ошибка инициализации раздела: %d (%s)", err,
esp_err_to_name(err));
    };



    // открываем пространство имен
    nvs_handle_t nvs_handle;
    err = nvs_open("testgroup", NVS_READWRITE, &nvs_handle);
    if (err != ESP_OK) {
      if (!(err == ESP_ERR_NVS_NOT_FOUND)) {
        ESP_LOGE("NVS", "Ошибка открытия пространства имен \"%s\": %d
(%s)!", "testgroup", err, esp_err_to_name(err));
      };
      // рестарт при ошибке
      esp_restart();
    };

    uint8_t mynumber = 9;
    // читаем наше число
    err = nvs_get_u8(nvs_handle, "mynumber", &mynumber);
    if (err == ESP_ERR_NVS_NOT_FOUND) {
        // если нечего читать, то пишем
        nvs_set_u8(nvs_handle, "mynumber", mynumber);
    }
```

```
    // увеличиваем на единицу
    mynumber++;
    nvs_set_u8(nvs_handle, "mynumber", mynumber);
    ESP_LOGE("NVS", "My number: %d", mynumber);



    // Запись строки, если её нет в nvs
    char mystra[12] = "test string";
    size_t mystra_size;
    // получаем длину строки
    err = nvs_get_str(nvs_handle, "mystr", NULL, &mystra_size);
    if (err == ESP_ERR_NVS_NOT_FOUND) {
      // если нечего читать, то пишем
      nvs_set_str(nvs_handle, "mystr", mystra);
    }



    // Чтение строки без указания длины
    char *mystr;
    size_t mystr_size;
    // получаем длину строки
    nvs_get_str(nvs_handle, "mystr", NULL, &mystr_size);
    // выделяем память для строки
    mystr = malloc(mystr_size);
    // читаем нашу строку
    nvs_get_str(nvs_handle, "mystr", mystr, &mystr_size);
    // выводим на экран
    ESP_LOGE("NVS", "Length str: %d. My str: %s\n", mystr_size, mystr);

    // выполняем комит
    nvs_commit(nvs_handle);
    // закрываем пространство имен
    nvs_close(nvs_handle);



    // Example of nvs_get_stats() to get the number of used entries and free
entries:
    nvs_stats_t nvs_stats;
    nvs_get_stats(NULL, &nvs_stats);
    printf("Количество: Использовано Записей = (%d), Свободных записей =
```

```
(%d), Всего записей = (%d)\n",
           nvs_stats.used_entries, nvs_stats.free_entries,
nvs_stats.total_entries);


    size_t free_heap = esp_get_free_heap_size();
    printf("Свободная ОЗУ: %d байт\n", free_heap);

    size_t min_free_heap = esp_get_minimum_free_heap_size();
    printf("Минимальная свободная ОЗУ: %d байт\n", min_free_heap);
}
```

Разберем его построчно

Начнем с include

Новых всего три строки:

```
#include "esp_log.h"
#include "nvs_flash.h"
#include "nvs.h"
```

Первое - это `esp_log.h`. Для удобного вывода цветного текста.
Второе это `nvs_flash`, который используется для инициализации и закрытия nvs
Третий этот `nvs.h`, в котором описан основной функционал для работы с nvs.

Код разбит на блоки с помощью множества пробелов, их можно раздельно читать не думая о содержимом оставшегося кода.

1. Первый блок - инициализация nvs с помощью команды `esp_err_t err = nvs_flash_init();`
   Если нет свободных страниц или NVS не найден, то выполняется очистка хранилища.
   Далее снова проверяется получилось ли инициализировать раздел

Первый блок на этом завершен.

2. Во втором блоке мы открываем пространство имен и в случае ошибки перезапускаем esp32.
3. В третем блоке мы объявляем число, читаем его, если оно не было записано, то записываем стандартным значением.
   Далее увеличиваем на единицу и снова записываем в nvs.
   Таким образом после каждой перезагрузки число увеличится на единицу.
4. В четвертом блоке идет запись строки в NVS при её отсутствии.
5. В пятом блоке мы создаем строку и переменную размера строки. Далее есть вариант чтения строки зная размер в одну команду или получая отдельно размер в две команды. Мы используем более сложный вариант.

Передаем третим аргументом NULL, сообщая, что нам нужна только длинна.
Выделяем память под неё и читаем строку.

6. После этого делаем комит для сохранения изменений и закрываем пространство имен.

7.8 Последние два блока мы используем для отладки. В них мы получается по средствам библиотеки nvs количество записей. Всего, свободно и занято. Выводим их на экран.

Далее мы Смотрим на количество совбодной памяти в куче. Запускаем и видим число увеличенное не единицу, строку созданную и отладочную информацию, в которой написано количество записей, а так же свободная озу.
Мы уже слышали о том что примерно 22 кбайта памяти занимает 1 мб NVS. Давайте проверим сколько будет свободной оперативной памяти, если закомментировать весь код связанные с NVS и его include.
У меня получилось разница в 22812 байт, а это примерно 22 килобайта, как и говорилось в документации.

Спасибо за внимание