Avionics Systems

AVS-DOC-TN-0000

PAFFS Documentation

Issue:	1.1
Date:	2016-08-31

DLR

Contents

List of Figures

List of Tables

Documents

Reference Documents

[RD1] Artem B. Bityuckiy. *JFFS3 design issues*. Standard. 2005.

[RD2] Ferenc Havasi et al. *JFFS3 plan extension*. Standard. 2006.

1 Introduction

PAFFS stands for "Protective Aeronautics Flash FS" and aims to use a minimum RAM footprint with the ability to manage multiple flashes. Most of the design ideas are inspired by the file system JFFS3 [RD1], which was later extended ([RD2]) and discontinued as predecessor of UBIFS.
The main differences to usual disk file systems are `out-of-place-writing` and high deletion costs in terms of durability and speed. Keeping track of the files data chunks has to be different than just maintaining a big table on a disk, because the high frequented lookup table would be worn out earlier while other parts of flash will be nearly untouched. The standard approach of flash aware file systems (such as YAFFS) is to maintain a table-like structure in RAM and committing every chunk of new data to flash with an unique, increasing number, just like a list. This adds far more complexity to the file system as it has to scan the whole flash when mounting, but increases lifetime of the flash enormously. However, this RAM table grows linear with flash size, and thus does not scale for modern (> 2GB) flash chips. To solve this, the information has to be on flash. B+-Tree -> Section 1.1.2. Another big challenge is to reduce wear. Every change to data has to be written to another place and the old location has to be invalidated somehow. This is because the smallest unit of deletion is bigger (usually around 512 - 4086 times) than the smallest unit of a write operation. The common approach is to give every logical chunk an increasing version number. Because of the disadvantages pointed out earlier, this is not applicable to PAFFS. Areas and address mapping -> Section 1.1.1

1.1 Fundamental structures

1.1.1 Areas

Overview

- To separate between different Types of Data
- `AreaType` can be one of Superblock, Index, Data, (Journaling)
- Address split in `logical area n°` and `page n°`, to give way for an easy garbage collector (see fig. 1.1)
- `AreaMap` held in FLASH (but cached in RAM) translates between `logical area n°` and `physical area n°`
- `AreaMap` also keeps record of corresponding types and usage statistics for garbage collection

AreaTypes

Superblock Superblock is automatically first[1] area of flash. It contains

1.1.2 Tree Indexing

Overview B+Tree, minimum height. Is convenient when having to update whole path to root after a single node was changed. RAM Cache with parent pointers. Variable size, minimum Treeheight + 1[2].

1.2 Images

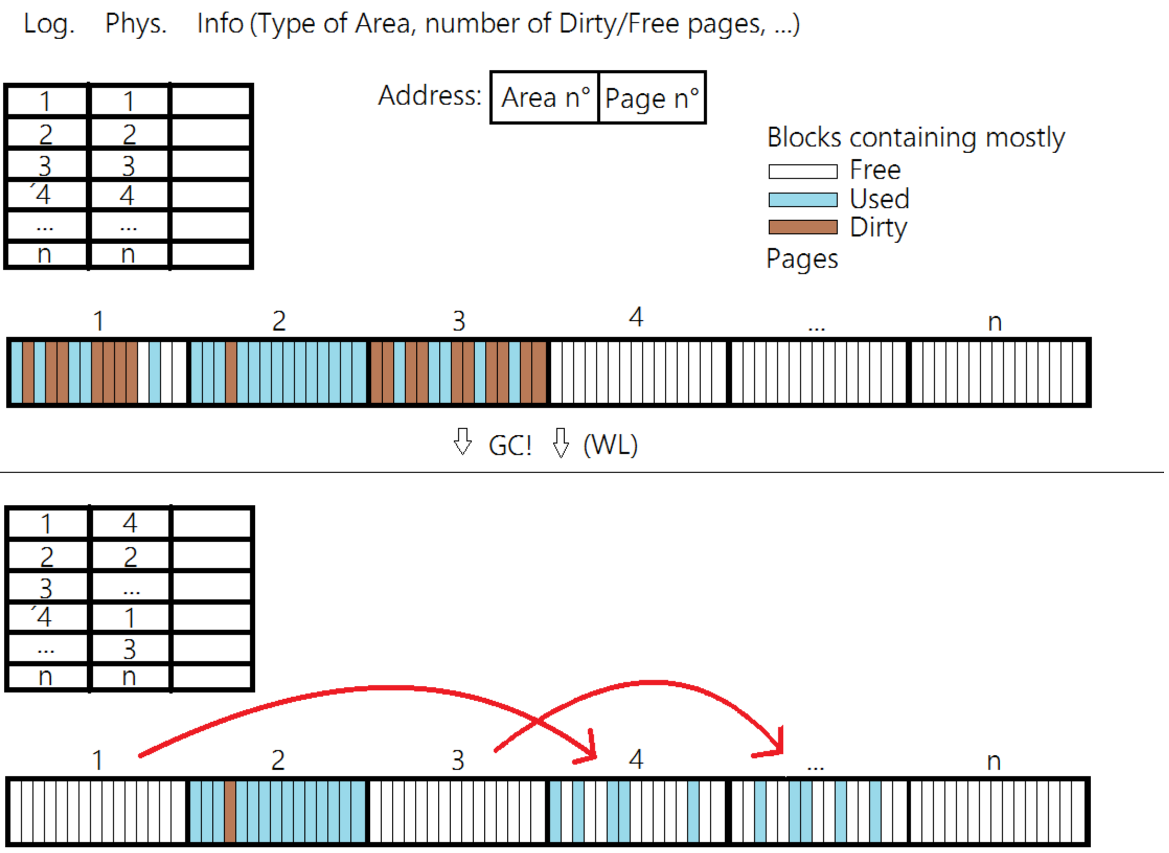

Figure 1.1: Basic process of garbage collection with areas

[1] or first two, just to be sure?
[2] Check if true.

A Stuff

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

optionEndOfDocument

- END OF DOCUMENT -