# ManBearPig: Final Report

spiros thanasoulas st19@illinois.edu

December 15, 2020

# Description

The manbearpig project is an attempt to understand parts of the mandoc system and to lay the foundations for full text search in it. Mandoc is a set of tools that display and index *man* or *mdoc* files to users of UNIX systems. The main purpose of these files are to provide documentation for commands, APIs, system components etc in a consistent way to the users.

## The Mdoc language and its history

UNIX manual pages originally were written in a language called *roff* which itself was a descendant of an even earlier system called *RUNOFF* originally written by Jerry Saltzer for the *Compatible Time Sharing System* around 1964. That language provided macros that controlled the typesetting of text, like `.CENTER` for centering or `.br` for line breaking. Around 1970 it was rewritten as *roff* by doug mcIllroy and bob morris , and then it was ported to the UNIX system by ken thompson. There it was first used for documenting the aspects of the system and since then it has remained the preferred way for performing these tasks. The GNU system has also brought forward the info system which is similar, but it hasn't gained widespread adoption.

Mdoc originally appeared as a *troff* macro package in 4.4BSD. It was then significantly updated by Werner Lemberg and Ruslan Ermilov in *groff-1.17*. The standalone implementation that is part of the mandoc utility we use for this project written by Kristaps Dzonsons and it appeared in OpenBSD 4.6.

Mdoc allows the semantic annotation of words and phrases , and also supports document hyperlinking. In an mdoc document, lines beginning with the control character '.' are called "macro lines". The first word is the macro name. It consists of two or three letters. Most macro names begin with a capital letter. The words following the macro name are arguments to the macro, optionally including the names of other, callable macros. Lines not beginning with the control character are called "text lines". They provide free-form text to be printed the formatting of the text depends on the respective processing context which is controlled by the parent macro. An example some mdoc test for a fictional utlity called "progname" could be as follows

```
.Dd Mdocdate
.Dt PROGNAME section
.Os
.Sh NAME
.Nm progname
.Nd one line about what it does
.∵Sh LIBRARY
.F̈or sections 2, 3, and 9 only.
.N̈ot used in OpenBSD.
.Sh SYNOPSIS
.Nm progname
.Op Fl options
.Ar
.Sh DESCRIPTION
The
.Nm
utility processes files ...
```

## Searching mdocs

Traditionally the manual pages can be searched with a command names `apropos` (originating from the french expression à propos, which means "about"). Also to make things even more confusing, another way to search manual pages has been invoking the man command with a -k (keyword) flag. In the mandoc system although this syntax exists for compatibility, it just invokes the apropos command on the backend. This is a point where across UNIX systems, things can begin to diverge greatly. In the mandoc system that we are examining the apropos and whatis utilities query manual page databases generated by the makewhatis command, By default, apropos searches for makewhatis databases using case-insensitive extended regular expression matching over manual names and descriptions (the `.Nm` and `.Nd` macro keys). In the mandoc system these databases are basically hashtables based on the ohash open hashing helper functions written originally by Marc Espie for OpenBSD. On other implementations though like the GNU ones in some linux distributions (again, this means not mandoc, but completely different manual page systems, which are presented here just for reference) the databases are implemented differently. Below are the options for the system shipped with debian linux of the mandb database formats and how it compares with mandoc in terms of async access, database naming and backend.

| Name | Type | Async | Filename |
|---|---|---|---|
| mandoc db | Hashed (ohash) | Yes | section/$arch/title.secion |
| debian man/Berkeley db | Binary tree | Yes | index.bt |
| debian man/GNU gdbm | Hashed | Yes | index.db |
| debian man/UNIX ndbm | Hashed | No | index.(dir—pag) |

As we have seen the current search functionality for all systems allows the query of very specific keywords in very specific parts of the document. Mandoc greatly improved the state of the art when it appeared because it also allowed certain semantic search capabilities. For example you could perform and/or operations on different macros to refine your search result. But still the result would be just the manual page name and section, and the keyword would have to be in an easily indexable part of the mdoc, because as it can be seen from the example above, the free text is intermixed with typesetting information.

## Small steps forward

### Extracting text

One good addition to the system would be to enable full text search. To do so we first have to extract the text in as a pure form as we can, and then somehow index it. For this task a small utlity was written (which borrows heavily from the demandoc command) to extract relevant text from an mdoc page. The current result is far from perfect as it needs to make decisions about , spaces, linebrakes , capitalization etc, but it still achieves the goal for the most part. It is able to extract text without formatting macros. The relevant code for this lives under `code/extract_text.c`, and it works by recursively parsing the mdoc structures (they can be embedded) in order to output only the words that are not language tokens.

### Efficient free text indexing

Since the makewhatis database is already in a hashtable format, it would make sense to choose a representation that maps well to that backend if it is to ever be merged in the main codebase. We want the user to be able to enter a small set of words and to fetch the results of the manual page this text sequence exists in. Also we would like for the user to have the ability to find a matching line of text midsentence. Consider for example a part of a manual page stating that `"A manual page consists of several sections."`. If we only kept an inverted index of words and our user wanted to look for the string "age consist", our system could lead him quite astray since none of the words page and consists match the query properly. Therefore as discussed before in the proposal documents, we will follow the approach that Russ Cox used while implementing the backend for google code search, which consists of splitting the text in tri-grams and storing their occurences. Under the trigram transformation the word "word" creates the set of the following trigrams `__w, _wo, wor, ord, rd_, d__`. An example program that perfors this transformation on its arguments can be found in code/words_to_trigrams.c.

## Running the example code

Under the directory `code/` there is a Makefile that builds the two binaries. Under the directory `code/input/` exist some sample files for input to the test programs. For the code to be compiled the mandoc source code should be compiled and existing at the same level of directory as the courseproject code. Below are some sample runs of the two provided binaries.

```
>cd code;

>make;
cc -c -I ../../mandoc/  extract_text.c
cc extract_text.o -L ../../mandoc/ -L /lib/x86_64-linux-gnu/  -lmandoc -lz -o extract_text
cc -o words_to_trigrams words_to_trigrams.c

>./extract_text input/apropos.1;
operating on file input/apropos.1
October APROPOS NAME apropos whatis search manual page databases SYNOPSIS apropos afk file
each file in each database By default they display the names section numbers and descriptio

>./words_to_trigrams the quick brown fox jumped over the lazy dog
arg: the trigrams  [_ _ t]  [_ t h]  [t h e]  [h e _]  [e _ _]
arg: quick trigrams  [_ _ q]  [_ q u]  [q u i]  [u i c]  [i c k]  [c k _]  [k _ _]
arg: brown trigrams  [_ _ b]  [_ b r]  [b r o]  [r o w]  [o w n]  [w n _]  [n _ _]
arg: fox trigrams  [_ _ f]  [_ f o]  [f o x]  [o x _]  [x _ _]
arg: jumped trigrams  [_ _ j]  [_ j u]  [j u m]  [u m p]  [m p e]  [p e d]  [e d _]
[d _ _]
arg: over trigrams  [_ _ o]  [_ o v]  [o v e]  [v e r]  [e r _]  [r _ _]
arg: the trigrams  [_ _ t]  [_ t h]  [t h e]  [h e _]  [e _ _]
arg: lazy trigrams  [_ _ l]  [_ l a]  [l a z]  [a z y]  [z y _]  [y _ _]
arg: dog trigrams  [_ _ d]  [_ d o]  [d o g]  [o g _]  [g _ _]
```