dvdusplit

Lars-Dominik Braun

March 14, 2016

Contents

1. Introduction

dvdusplit extracts video and metadata from a DVD based on information found in the *Program Chain Information (PGCI) Unit Table*. It is able to *rip* a DVD to one or multiple files with chapter metadata preserved.

Its source code is availble at `https://6xq.net/dvdusplit/dvdusplit.tar.bz2`.

Being a literate program the document you are reading right now is source code and documentation at the same time. The program's basic structure looks like this:

1 ⟨*main function* 1⟩≡ (12a)
⟨*parse commandline* 2a⟩
⟨*open dvd* 5a⟩
⟨*read metadata* 6b⟩
⟨*write output files* 7c⟩

⟨*close dvd* 9d⟩
```
return EXIT_SUCCESS;
```

2. Usage

By default all units found in track 1 are written into one big file. This behavior can be modified by the command line switches below.

2a ⟨*parse commandline* 2a⟩≡ (1) 2b▷
```
bool splitUnits = false, splitDiscontinuity = false, splitCells = false,
        verbose = false;
const char *dvdpath = NULL;
int title = 1;
```
Defines:
`dvdpath`, used in chunks 2b, 4b, and 5a.
`splitCells`, used in chunks 3a and 7b.
`splitDiscontinuity`, used in chunks 3b and 7b.
`splitUnits`, used in chunks 2c and 7a.
`title`, used in chunks 3–5 and 8a.
`verbose`, used in chunks 3c and 5b.

They start with a single dash `-` followed by one character for example `-v`. The last argument not starting with a dash is used as the DVD's path. libdvdread accepts devices, images and directories.

2b ⟨*parse commandline* 2a⟩+≡ (1) ◁2a 4b▷
```
for (size_t i = 1; i < argc; i++) {
    switch (argv[i][0]) {
        case '-':
            switch (argv[i][1]) {
                ⟨commandline options 2c⟩
            }
            break;

        default:
            dvdpath = argv[i];
            break;
    }
}
```
Uses `dvdpath` 2a.

Put every unit into one file. A unit is usually equivalent to a DVD track.

2c ⟨*commandline options* 2c⟩≡ (2b) 3a▷
```
case 'u':
    splitUnits = true;
    break;
```
Uses `splitUnits` 2a.

Put each cell into a new file. Units consist of one or more cells. A cell is usually equivalent to a DVD chapter.

3a ⟨*commandline options* 2c⟩+≡ (2b) ◁2c 3b▷

```
case 'c':
    splitCells = true;
    break;
```

Uses `splitCells` 2a.

Split if cell contains discontinuity flag. This flag is set if there is a (logical) gap between two cells, i.e. a new track starts.

3b ⟨*commandline options* 2c⟩+≡ (2b) ◁3a 3c▷

```
case 'd':
    splitDiscontinuity = true;
    break;
```

Uses `splitDiscontinuity` 2a.

Which combination of the three file split conditions above produces the best results depends on the actual DVD and how much effort was put into the mastering process. Increasing verbosity may help to figure out which of those are needed.

3c ⟨*commandline options* 2c⟩+≡ (2b) ◁3b 3d▷

```
case 'v':
    verbose = true;
    break;
```

Uses `verbose` 2a.

Select DVD title.

3d ⟨*commandline options* 2c⟩+≡ (2b) ◁3c 4a▷

```
case 't':
    ++i;
    if (i >= argc) {
        printf ("Missing argument for -t\n");
        help (argv[0]);
        return EXIT_FAILURE;
    }
    title = strtoul (argv[i], NULL, 0);
    break;
```

Uses `help` 4c and `title` 2a.

Unknown options are rejected.

4a ⟨*commandline options* 2c⟩+≡ (2b) ◁3d

```
default:
    printf ("Unknown command line option %s\n", argv[i]);
    help (argv[0]);
    return EXIT_FAILURE;
    break;
```

Uses `help` 4c.

The DVD path should be set for every invokation.

4b ⟨*parse commandline* 2a⟩+≡ (1) ◁2b

```
if (dvdpath == NULL) {
    printf ("No DVD path given.\n");
    help (argv[0]);
    return EXIT_FAILURE;
}
```

Uses `dvdpath` 2a and `help` 4c.

In case an option is not recognized a help message can be displayed.

4c ⟨*functions* 4c⟩≡ (12a) 6a▷

```
void help (const char * const progname) {
    printf ("%s [-u] [-d] [-p] [-t title] [-v] <dvd path>\n", progname);
}
```

Defines:
`help`, used in chunks 3 and 4.
Uses `dvd` 5b and `title` 2a.

3. Dependencies

libdvdread is used to access the DVD's video and metadata.

4d ⟨*dvdread headers* 4d⟩≡ (12a)

```
#include <dvdread/ifo_print.h>
#include <dvdread/ifo_read.h>
#include <dvdread/nav_types.h>
#include <dvdread/dvd_reader.h>
```

Building PDF documentation requires GNU make, noweb and LaTeX.

4. Main program

4.1. Read metadata

First open the DVD and obtain a metadata (IFO) handle.

5a ⟨*open dvd* 5a⟩≡ (1) 5b▷

```
dvd_reader_t * const dvd = DVDOpen (dvdpath);
if (dvd == NULL) {
    return EXIT_FAILURE;
}
ifo_handle_t * const ifo = ifoOpen (dvd, title);
if (ifo == NULL) {
    printf ("Selected title does not exist\n");
    return EXIT_FAILURE;
}
```

Uses `dvd` 5b, `dvdpath` 2a, `ifo` 5b, and `title` 2a.

If the user chose to dump the metadata to `stdout`.

5b ⟨*open dvd* 5a⟩+≡ (1) ◁5a

```
if (verbose) {
    ifo_print (dvd, title);
}
```

Defines:
`dvd`, used in chunks 4c, 5a, 8a, and 9d.
`ifo`, used in chunks 5a, 6b, and 9d.
Uses `title` 2a and `verbose` 2a.

While reading DVD metadata `outFile` is filled with references to cells for each output file. They contain the actual start and end sector we're going to read later. `blocks` accumulates the total amount of sectors/blocks to be read/written.

5c ⟨*types* 5c⟩≡ (12a)

```
typedef struct {
    dvd_read_domain_t domain;
    size_t cellsCount;
    cell_playback_t **cells;
    size_t blocks;
} outFile;
```

Defines:
`outFile`, used in chunks 6 and 8a.

If one of the split conditions is encountered the function `next` advances to the output file and returns a pointer to its structure. If no cells belong to the current output file no changes are made. Its two parameters point to the array of output files and its size. They are updated if it had to be resized with realloc.

6a ⟨*functions* 4c⟩+≡ (12a) ◁4c 10b▷

```
outFile *next (outFile ** const retOut, size_t * const retOutCount) {
    outFile *out = *retOut;
    size_t outCount = *retOutCount;

    if (outCount > 0 && out[outCount-1].cells == NULL) {
        assert (out != NULL);
        return &out[outCount-1];
    }

    outCount++;
    out = realloc (out, outCount*sizeof (*out));
    assert (out != NULL);
    outFile * const current = &out[outCount-1];
    memset (current, 0, sizeof (*current));

    *retOut = out;
    *retOutCount = outCount;
    return current;
}
```

Defines:
  `next`, used in chunks 6 and 7.
Uses `outFile` 5c.

Check both, title and menu tables. There's always a file split inbetween, because the domain indicator is per `outFile`, not cell.

6b ⟨*read metadata* 6b⟩≡ (1)

```
size_t outCount = 0;
outFile *out = NULL, *o = NULL;

const pgcit_t * const tables[2] = {ifo->vts_pgcit, ifo->pgci_ut->lu->pgcit};
const dvd_read_domain_t domains[2] = {DVD_READ_TITLE_VOBS, DVD_READ_MENU_VOBS};
for (size_t i = 0; i < sizeof (tables)/sizeof (*tables); i++) {
    const pgcit_t * const pgcit = tables[i];

    o = next (&out, &outCount);

    ⟨read table srp's 7a⟩
}
```

Uses `ifo` 5b, `next` 6a, and `outFile` 5c.

Iterate over all units in the PGCI unit table. If we're instructed to split on unit boundaries, call `next`.

7a ⟨*read table srp's* 7a⟩≡ (6b)

```
for (size_t j = 0; j < pgcit->nr_of_pgci_srp; j++) {
    const pgci_srp_t * const srp = &pgcit->pgci_srp[j];
    const pgc_t * const pgc = srp->pgc;

    if (splitUnits) {
        o = next (&out, &outCount);
    }

    ⟨process all cells 7b⟩
}
```

Uses `next` 6a and `splitUnits` 2a.

Call `next` again if *a*) we're splitting all cells or *b*) cells with discontinuity should be split. Add each cell to the current output file.

7b ⟨*process all cells* 7b⟩≡ (7a)

```
for (size_t k = 0; k < pgc->nr_of_cells; k++) {
    cell_playback_t * const playback = &pgc->cell_playback[k];

    if (splitCells || splitDiscontinuity && playback->stc_discontinuity) {
        o = next (&out, &outCount);
    }
    ++o->cellsCount;
    o->cells = realloc (o->cells, o->cellsCount * sizeof (*o->cells));
    o->cells[o->cellsCount-1] = playback;
    o->blocks += playback->last_sector - playback->first_sector + 1;
    assert (i < sizeof (domains)/sizeof (*domains));
    o->domain = domains[i];
}
```

Uses `next` 6a, `splitCells` 2a, and `splitDiscontinuity` 2a.

4.2. Write output files

After splitting the DVD into output files we can actually write them to the disk. First allocate a temporary buffer for read/write operations.

7c ⟨*write output files* 7c⟩≡ (1) 8a▷

```
const unsigned int bufBlocks = 16;
unsigned char * const buf = malloc (bufBlocks * DVD_VIDEO_LB_LEN);
```

Defines:
`buf`, used in chunk 9.
`bufBlocks`, used in chunk 9a.

Then iterate over all output files and copy the selected cells.

8a ⟨*write output files* 7c⟩+≡ (1) ◁7c 9c▷

```
for (size_t i = 0; i < outCount; i++) {
    const outFile * const o = &out[i];

    if (o->cells == NULL) {
        continue;
    }

    dvd_file_t * const file = DVDOpenFile (dvd, title, o->domain);
    assert (file != NULL);

    ⟨open and preallocate file 8b⟩
    ⟨copy all cells 9a⟩
    ⟨close file 9b⟩

    DVDCloseFile (file);
    free (o->cells);
}
```

Uses `dvd` 5b, `outFile` 5c, and `title` 2a.

Output file names are not configureable right now and default to `out_i.vob`, with $i \in [0, outCount)$. Preallocate the file to avoid file system fragmentation.

8b ⟨*open and preallocate file* 8b⟩≡ (8a) 10a▷

```
char outfile[32];
snprintf (outfile, sizeof (outfile), "out_%03zu.vob", i);
const int fd = open (outfile, O_WRONLY|O_CREAT|O_TRUNC, 0666);
assert (fd != -1);

int ret = fallocate (fd, 0, 0, o->blocks*DVD_VIDEO_LB_LEN);
assert (ret != -1);
```

Copy all selected cells from `first_sector` to `last_sector` (included) to output file in chunks of `bufBlocks`.

9a ⟨*copy all cells* 9a⟩≡ (8a)

```
uint64_t totalTime = 0;
for (size_t j = 0; j < o->cellsCount; j++) {
    const cell_playback_t * const playback = o->cells[j];
    printf ("%s, cell %zu/%zu, %x-%x\n", outfile, j+1, o->cellsCount,
            playback->first_sector, playback->last_sector);
    int block = playback->first_sector;
    while (block <= playback->last_sector) {
        const unsigned int remaining = playback->last_sector - block + 1;
        const unsigned int readBlocks = remaining > bufBlocks ? bufBlocks : remaining;
        const ssize_t read = DVDReadBlocks (file, block, readBlocks, buf);
        assert (read == readBlocks);
        write (fd, buf, read*DVD_VIDEO_LB_LEN);
        block += read;
    }

    ⟨write chapter metadata 11b⟩
}
```

Uses `buf` 7c and `bufBlocks` 7c.

When done with all cells, clean up.

9b ⟨*close file* 9b⟩≡ (8a) 11c▷

```
close (fd);
```

Also delete buffer and output file array when done. Not strictly required.

9c ⟨*write output files* 7c⟩+≡ (1) ◁8a

```
free (buf);
free (out);
```

Uses `buf` 7c.

Finally close DVD and IFO handles.

9d ⟨*close dvd* 9d⟩≡ (1)

```
ifoClose (ifo);
DVDClose (dvd);
```

Uses `dvd` 5b and `ifo` 5b.

4.3. Write metadata

In the same step metadata is written to a separate file for each output file. It is called `out_i.txt` and may be used by ffmpeg's metadata (de-)muxer.

10a ⟨*open and preallocate file* 8b⟩+≡ (8a) ◁8b

```
char metaoutfile[32];
snprintf (metaoutfile, sizeof (metaoutfile), "out_%03zu.txt", i);
FILE * const metafd = fopen (metaoutfile, "w");
fputs (";FFMETADATA1\n", metafd);
```

Defines:
`metafd`, used in chunk 11.
`metaoutfile`, never used.

10b ⟨*functions* 4c⟩+≡ (12a) ◁6a 11a▷

```
static uint8_t bcdDecode (const uint8_t v) {
    return (v>>4)*10 + (v&0xf);
}
```

Retrieve time in milliseconds from the DVD's metadata (second argument), add it to the first argument and return the value.

11a ⟨*functions* 4c⟩+≡ (12a) ◁10b

```
static uint64_t addTime (const uint64_t a, const dvd_time_t * const b) {
    uint64_t perframe = 0;
    switch ((b->frame_u >> 6) & 0x3) {
        case 1:
            /* 25 fps */
            perframe = 40;
            break;

        case 3:
            /* 29.97 fps */
            perframe = 33;
            break;

        default:
            assert (0 && "unknown frame rate");
            break;
    }
    return a +
            (bcdDecode (b->hour) * 60*60 +
            bcdDecode (b->minute) * 60 +
            bcdDecode (b->second))*1000 +
            perframe*(bcdDecode (b->frame_u & 0x3f));
}
```

Defines:
`addTime`, used in chunk 11b.

11b ⟨*write chapter metadata* 11b⟩≡ (9a)

```
const uint64_t endTime = addTime (totalTime, &playback->playback_time);
fprintf (metafd, "[CHAPTER]\nTIMEBASE=1/1000\nSTART=%lu\nEND=%lu\n", totalTime,
        endTime);
totalTime = endTime;
```

Uses `addTime` 11a and `metafd` 10a.

11c ⟨*close file* 9b⟩+≡ (8a) ◁9b

```
fclose (metafd);
```

Uses `metafd` 10a.

A. Program skeleton

12a ⟨* 12a⟩≡
⟨*standard headers* 12b⟩
⟨*dvdread headers* 4d⟩

⟨*types* 5c⟩
⟨*functions* 4c⟩

```
int main (int argc, char **argv) {
    ⟨main function 1⟩
}
```

Feature test macro is required for `fallocate`.

12b ⟨*standard headers* 12b⟩≡ (12a)

```
#define _GNU_SOURCE

#include <stdlib.h>
#include <assert.h>
#include <stdint.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
```

B. License

C. Definitions