30 Ноября 2013 года в 13:50

1. Введение.

MKDISK

Утилита создания образов дисков MKDOS
(Версия 0.1)

Yellow Rabbit

Создает образ диска в формате MKDOS помещая в него указанные файлы. Объем образа 800KiB. Русские имена файлов кодируются в КОИ8.

2. Общая схема программы.

```
⟨ Включение заголовочных файлов 22 ⟩
⟨ Директивы препроцессора ⟩
⟨ Константы 15 ⟩
⟨ Собственные типы данных 10 ⟩
⟨ Глобальные переменные 3 ⟩
int main(int argc, char *argv[])
{
  ⟨ Данные программы 4 ⟩
  const char *srcname;
  int i;
  ⟨ Разобрать командную строку 21 ⟩
  ⟨ Подготовить перекодировку 7 ⟩
  ⟨ Создать файл образа 6 ⟩
  total_size = 0;
  cur_src = 0;      /* Поочередно обрабатываем все заданные файлы */
  while ((srcname = config.srcnames[cur_src]) ≠ ~) {
    ⟨ Открыть исходный файл 5 ⟩
    handleOneFile(fsrc, fresult);
    fclose(fsrc);
    if (total_size ≥ DISK_SIZE) {
      PRINTERR("Files␣are␣too␣big\n");
      return (ERR_TOO_BIG);
    }
    ++cur_src;
  }     /* Создать корневой каталог диска */
  if (createDir(fresult) ≠ 0) {
    return (ERR_CREATE_DIR);
  }
  fclose(fresult);
  ⟨ Очистить перекодировку 8 ⟩
  return (0);
}
```

3.

```
⟨ Глобальные переменные 3 ⟩ ≡
  static int cur_src;
```

Смотри также секции 9, 13, 16, 17, 19, и 23.

Этот код используется в секции 2.

4.

```
⟨ Данные программы 4 ⟩ ≡
  FILE *fsrc, *fresult;
```

Этот код используется в секции 2.

5.

```
⟨ Открыть исходный файл 5 ⟩ ≡
  fsrc = fopen(srcname, "r");
  if (fsrc ≡ ~) {
    PRINTERR("Can't␣open␣%s\n", srcname);
    return (ERR_CANTOPEN);
  }
```

Этот код используется в секции 2.

6. ⟨ Создать файл образа 6 ⟩ ≡

```
fresult = fopen(config.output_filename, "w");
if (fresult ≡ ~) {
  PRINTERR("Can't␣create␣%s\n", config.output_filename);
  return (ERR_CANTOPEN);
}
memset(buf, 0, BLOCK_SIZE);
for (i = 0; i < DISK_CATALOG_SIZE; ++i) {
  fwrite(buf, BLOCK_SIZE, 1, fresult);
}     /* инициализация записей каталога */
memset(dir, 0, sizeof (dir));
```

Этот код используется в секции 2.

7. Требуются переколировка имени файлов в КОИ8.

⟨ Подготовить перекодировку 7 ⟩ ≡

```
setlocale(LC_ALL, "");
cd = iconv_open("KOI8-R", nl_langinfo(CODESET));
if (cd ≡ (iconv_t) − 1) {
  PRINTERR("Can't␣open␣encoding␣converter\n");
  return (ERR_ENCODING);
}
```

Этот код используется в секции 2.

8. ⟨ Очистить перекодировку 8 ⟩ ≡

```
iconv_close(cd);
```

Этот код используется в секции 2.

9. ⟨ Глобальные переменные 3 ⟩ +≡

```
static iconv_t cd;
```

10. Сбор информации о файлах и запись каталога.

```
#define MKDOS_ID °123456
#define MKDOS_DIR_ID °51414
#define MKDOS_FILE_STATUS_NORMAL °0
#define MKDOS_FILE_STATUS_PROTECTED °1
#define MKDOS_FILE_STATUS_LDISK °2      /* логический диск */
#define MKDOS_FILE_STATUS_BAD °200      /* плохой блок */
#define MKDOS_FILE_STATUS_DELETED °377      /* удаленный/свободный */
#define DISK_SIZE 1600      /* 1600 512-байтных блоков */
#define DISK_CATALOG_SIZE °24      /* число блоков, занятых каталогом */
#define MKDOS_NUM_FILES 172
#define MKDOS_MAX_NAME_LEN 14
```

⟨ Собственные типы данных 10 ⟩ ≡

```
typedef struct _DiskHeader {
  uint8_t dummy0[°30];
  uint16_t num_files;      /* количество файлов */
  uint16_t num_used_blocks;      /* количество занятых блоков */
  uint8_t dummy1[°400 − 4 − °30];
  uint16_t mkdos_id;      /* метка принадлежности к формату MKDOS */
  uint16_t dir_id;      /* метка формата каталога MKDOS */
  uint8_t dummy2[°466 − 4 − °400];
  uint16_t num_blocks;      /* емкость диска в блоках */
  uint16_t first_block;      /* номер первого блока первого файла */
  uint8_t dummy3[°500 − 4 − °466];
} DiskHeader;
typedef struct _DirRecord {
  uint8_t status;      /* статус файла */
  uint8_t subdir_num;      /* номер подкаталога (0 — корень) */
  char name[MKDOS_MAX_NAME_LEN];
    /* если [0]==0117, то это подкаталог, а в поле статуса указан номер этого подкаталога */
  uint16_t block;      /* номер блока */
  uint16_t block_len;      /* длина в блоках */
  uint16_t addr;      /* адрес */
  uint16_t len;      /* длина */
} DirRecord;
```

Смотри также секцию 18.

Этот код используется в секции 2.

11. Собираем информацию о входных файлах и создаҷм каталог диска.

```
static int createDir(FILE *fresult)
{
  DiskHeader *hdr;
  int i;      /* пишем 0-ой блок */
  rewind(fresult);
  memset(buf, 0, BLOCK_SIZE);
  hdr = (DiskHeader *) buf;
  hdr→mkdos_id = MKDOS_ID;
  hdr→dir_id = MKDOS_DIR_ID;
  hdr→num_files = config.num_src;
  hdr→num_used_blocks = DISK_CATALOG_SIZE + total_size;
    /* это блоки под каталог диска + размер файлов */
  hdr→num_blocks = DISK_SIZE;
  hdr→first_block = DISK_CATALOG_SIZE;      /* первый блок сразу после каталога */
  fwrite(buf, sizeof(DiskHeader), 1, fresult);
  for (i = 0; i < config.num_src; ++i) {
    fwrite(dir + i, sizeof(DirRecord), 1, fresult);
  }
  return (0);
}
```

12. Обработать один входной файл.

```
#define BLOCK_SIZE °1000
  static void handleOneFile(FILE *fsrc, FILE *fresult)
  {
    char name[MKDOS_MAX_NAME_LEN + 1], *pname;
    const char *sname;
    size_t slen, dlen;
    int block_len, size, start_block;
    size = 0;
    start_block = total_size + DISK_CATALOG_SIZE;
    while (¬feof(fsrc)) {
      block_len = fread(buf, 1, BLOCK_SIZE, fsrc);
      if (block_len ≡ 0) {
        break;
      }
      ++total_size;
      size += block_len;
      fwrite(buf, BLOCK_SIZE, 1, fresult);
    }
    PRINTVERB(1, "File: %s, length: %d, total_blocks: %d.\n", config.srcnames[cur_src], size,
        total_size);
    dir[cur_src].block = start_block;
    dir[cur_src].block_len = total_size − start_block + DISK_CATALOG_SIZE;
    dir[cur_src].addr = °1000;
    dir[cur_src].len = size;      /* Конвертировать имя файла в КОИ8 */
    pname = name;
    sname = basename(config.srcnames[cur_src]);
    slen = MKDOS_MAX_NAME_LEN;
    dlen = slen;
    PRINTVERB(2, "Src name:%s, slen:%ld, dst name:%s, dlen:%ld.\n", sname, slen, pname, dlen);
    iconv(cd, &sname, &slen, &pname, &dlen);
    PRINTVERB(2, "PSrc:%s, slen:%ld, PDst:%s, dlen:%ld.\n", sname, slen, pname, dlen);
    strncpy(dir[cur_src].name, name, MKDOS_MAX_NAME_LEN);
  }
```

13.

```
⟨ Глобальные переменные 3 ⟩ +≡
  static void handleOneFile(FILE *, FILE *);
  static int createDir(FILE *);
  static uint8_t buf[BLOCK_SIZE];
  static DirRecord dir[MKDOS_NUM_FILES];      /* Каталог диска */
  static unsigned int total_size;      /* Общее число блоков в файлах */
```

14. Разбор параметров командной строки.
Для этой цели используется достаточно удобная свободная библиотека *argp*.

```
#define VERSION "0.9"
```

15. ⟨ Константы 15 ⟩ ≡

```
const char *argp_program_version = "mkdisk, "VERSION;
const char *argp_program_bug_address = "<yellowrabbit@bk.ru>";
```

Этот код используется в секции 2.

16. ⟨ Глобальные переменные 3 ⟩ +≡

```
static char argp_program_doc[] = "Make MKDOS disk image";
```

17. Распознаются следующие опции:

`-o` — имя выходного файла;
`-v` — вывод дополнительной информации.

⟨ Глобальные переменные 3 ⟩ +≡

```
static struct argp_option options[] = {
    {"output", 'o', "FILENAME", 0, "Output filename"},
    {"verbose", 'v', ~, 0, "Verbose output"}, {0}
};
static error_t parse_opt(int, char *, struct argp_state *);
static struct argp argp = {options, parse_opt, ~, argp_program_doc};
```

18. Эта структура используется для получения результатов разбора параметров командной строки.

⟨ Собственные типы данных 10 ⟩ +≡

```
typedef struct _Arguments {
  int verbosity;
  char output_filename[FILENAME_MAX];      /* Имя файла с текстом */
  int num_src;      /* Количество исходных файлов */
  char **srcnames;      /* Имена исходных файлов srcnames[?] == NULL -> конец имен */
} Arguments;
```

19. ⟨ Глобальные переменные 3 ⟩ +≡

```
static Arguments config = {0, {0}, 0, ~, };
```

20. Задачей данного простого парсера является заполнение структуры **Arguments** из указанных параметров командной строки.

```
static error_t parse_opt(int key, char *arg, struct argp_state *state)
{
  Arguments *arguments;
  arguments = (Arguments *) state->input;
  switch (key) {
  case 'v': ++arguments->verbosity;
    break;
  case 'o':
    if (strlen(arg) ≡ 0) return (ARGP_ERR_UNKNOWN);
    strncpy(arguments->output_filename, arg, FILENAME_MAX − 1);
    break;
  case ARGP_KEY_ARG:      /* Имена исходных файлов */
    arguments->srcnames = &state->argv[state->next − 1];      /* Останавливаем разбор параметров */
    state->next = state->argc;
    break;
  default: break;
    return (ARGP_ERR_UNKNOWN);
  }
  return (0);
}
```

21.

```
#define ERR_SYNTAX 1
#define ERR_CANTOPEN 2
#define ERR_CANTCREATE 3
#define ERR_TOO_MANY_FILES 4
#define ERR_CREATE_DIR 5
#define ERR_ENCODING 6
#define ERR_TOO_BIG 7
```

⟨ Разобрать командную строку 21 ⟩ ≡

```
  argp_parse(&argp, argc, argv, 0, 0, &config);

  if (config.srcnames ≡ ~) {
    PRINTERR("No input filenames specified\n");
    return (ERR_SYNTAX);
  }
  for (config.num_src = 0; config.srcnames[config.num_src] ≠ ~; ++config.num_src) {
    if (config.num_src ≥ MKDOS_NUM_FILES) {
      PRINTERR("Must be <= %d files.\n", MKDOS_NUM_FILES);
      return (ERR_TOO_MANY_FILES);
    }
  }
```

Этот код используется в секции 2.

22. ⟨ Включение заголовочных файлов 22 ⟩ ≡
```
#include <string.h>
#include <stdlib.h>
#include <libgen.h>
#ifdef __linux__
#include <stdint.h>
#endif
#include <locale.h>
#include <langinfo.h>
#include <iconv.h>
#include <argp.h>
```
Этот код используется в секции 2.

23.

⟨ Глобальные переменные 3 ⟩ +≡
```
#define PRINTVERB (level, fmt, a ... ) (((config.verbosity) ≥ level) ? printf((fmt), ##a) : 0)
#define PRINTERR (fmt, a ... ) fprintf(stderr, (fmt), ##a)
```

24. Индекс.

⟨ Включение заголовочных файлов 22 ⟩ Используется в секции 2.
⟨ Глобальные переменные 3, 9, 13, 16, 17, 19, 23 ⟩ Используется в секции 2.
⟨ Данные программы 4 ⟩ Используется в секции 2.
⟨ Константы 15 ⟩ Используется в секции 2.
⟨ Открыть исходный файл 5 ⟩ Используется в секции 2.
⟨ Очистить перекодировку 8 ⟩ Используется в секции 2.
⟨ Подготовить перекодировку 7 ⟩ Используется в секции 2.
⟨ Разобрать командную строку 21 ⟩ Используется в секции 2.
⟨ Собственные типы данных 10, 18 ⟩ Используется в секции 2.
⟨ Создать файл образа 6 ⟩ Используется в секции 2.

MKDISK