# Содержание

[I. Постановка задачи 3](#_Toc515288230)

[II. Алгоритм решения задачи 3](#_Toc515288231)

[III. Текст программы 4](#_Toc515288232)

[IV. Тестовый пример 10](#_Toc515288233)

[V. Результаты работы программы 11](#_Toc515288234)

# I. Постановка задачи

Написать консольную программу для сортировки файлов CSV.

Управление должно осуществляться через аргументы командной строки. Предусмотреть следующие возможности: многопоточная обработка, проверка доступности файла, создание log-файла с описанием произошедших ошибок, использование шаблонных символов в качестве имени файла, возможность обработки файлов как в каталоге, так и всех его подкаталогах. Для работы использовать функции POSIX API.

# II. Алгоритм решения задачи

**CSV** (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Спецификация:

* Каждая строка файла — это одна строка таблицы.
* Разделителем значений колонок является символ запятой (,).
* Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками.

Исходя из этого, можно реализовать следующий алгоритм:

1. Создать очередь для путей к файлам.
2. Рекурсивно обойти каталог и его содержимое, проверяя имена файлов на соответствие регулярному выражению.
3. Если соответствует, то поместить путь к файлу в очередь, иначе – пропустить и перейти к следующему файлу.
4. Параллельно с этим работают потоки, которые извлекают данные из очереди, используя средства синхронизации потоков (мьютексы, семафоры).
5. После извлечения поток открывает файл для чтения и считывает первую строку для того, чтобы посчитать количество столбцов.
6. Далее происходит считывание каждой строки в стек, т.к. заранее неизвестно число записей.
7. Стек переписывается в вектор и сортируется по столбцу, номер которого был введен в аргументах запуска.
8. Наконец, отсортированные данные записываются в тот же файл, откуда были считаны.
9. Повторяются шаги 4-8 пока очередь не опустеет.

Таким образом, после работы программы все нужные файлы отказываются отсортированными.

# III. Текст программы

#include <string.h>

#include <sys/types.h>

#include <regex.h>

#include <dirent.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <stddef.h>

#include <math.h>

#include <time.h>

#include <pthread.h>

#include <semaphore.h>

//-- Структуры очереди и стека -------

typedef struct \_queue\_node {

 char \*dir;

 char \*fname;

 struct \_queue\_node \*next;

} queue\_node;

typedef struct \_queue {

 queue\_node \*head;

 queue\_node \*tail;

} queue;

void queue\_init(queue \*q) {

 q->head = NULL;

 q->tail = NULL;

}

typedef struct \_stack\_node {

 char \*\*str\_vect;

 struct \_stack\_node \*next;

} stack\_node;

void queue\_push(queue \*q, char \*a, char \*b) {

 queue\_node \*node = (queue\_node \*) malloc(sizeof(queue\_node));

 node->dir = (char \*) malloc(256);

 node->fname = (char \*) malloc(256);

 node->next = NULL;

 strcpy(node->dir, a);

 strcpy(node->fname, b);

 if (q->head == NULL) {

 q->head = q->tail = node;

 } else {

 q->tail->next = node;

 q->tail = node;

 }

}

queue\_node \*queue\_pop(queue \*q) {

 queue\_node \*temp = q->head;

 if (temp != NULL) {

 if (q->head->next == NULL) {

 q->head = q->tail = NULL;

 } else {

 q->head = q->head->next;

 }

 }

 return temp;

}

void queue\_print(queue \*q) {

 queue\_node \*cur = q->head;

 while (cur != NULL) {

 printf("%s/%s\n", cur->dir, cur->fname);

 cur = cur->next;

 }

 printf("\n");

}

void stack\_push(stack\_node \*\*s, char \*\*str) {

 stack\_node \*nnode = (stack\_node \*) malloc(sizeof(stack\_node));

 nnode->next = NULL;

 nnode->str\_vect = str;

 if ((\*s) == NULL) {

 (\*s) = nnode;

 } else {

 nnode->next = (\*s);

 (\*s) = nnode;

 }

}

char \*\*stack\_pop(stack\_node \*\*s) {

 if ((\*s) != NULL) {

 char \*\*temp = (\*s)->str\_vect;

 stack\_node \*st\_temp = (\*s);

 (\*s) = (\*s)->next;

 free(st\_temp);

 return temp;

 }

 return 0;

}

//-- Глоабльные Переменные --------------------

sem\_t global\_semaphore;

queue global\_queue;

pthread\_mutex\_t global\_queue\_mutex;

unsigned end\_flag;

FILE \*err\_file;

//-- Печать ошибок в файл ---------------------

void print\_error(char \*msg, char \*dop) {

 if (err\_file == NULL) {

 time\_t rawtime;

 struct tm \*timeinfo;

 char strtime[25] = "";

 rawtime = time(NULL);

 timeinfo = localtime(&rawtime);

 strftime(strtime, 25, "%Y-%m-%d %H:%M:%S.log", timeinfo);

 err\_file = fopen(strtime, "w");

 }

 fprintf(err\_file, "%s %s\n", msg, dop);

}

//-- Здесь реализован обход каталога

int print\_dir(char \*cdir\_name, regex\_t pr) {

 struct dirent \*entry;

 struct stat statbuf;

 DIR \*cur\_dir;

 int rc = 0;

 if ((cur\_dir = opendir(cdir\_name)) == NULL) {

 print\_error("Не удается открыть директорию", cdir\_name);

 return -1;

 }

 // Меняем рабочий каталог программы

 chdir(cdir\_name);

 char cdir[256];

 getcwd(cdir, 256);

 regmatch\_t \*rm = (regmatch\_t \*) malloc (sizeof(regmatch\_t \*) \* (pr.re\_nsub + 1));

 while ((entry = readdir(cur\_dir)) != NULL) {

 // Получаем информацию о файле

 rc = stat(entry->d\_name, &statbuf);

 if (rc != 0) {

 print\_error("Ошибка при вызове stat for", entry->d\_name);

 return -1;

 }

 // Если это директория, рекурсивно вызываем функцию

 if (S\_ISDIR(statbuf.st\_mode)) {

 // Проверяем, является ли она . или ..

 if (strcmp(".", entry->d\_name) == 0 ||

strcmp("..", entry->d\_name) == 0) {

 // Если да, игнорируем

 continue;

 }

 // Выводим имя директории и заходим в нее

 print\_dir(entry->d\_name, pr);

 }

 else {

 if (regexec(&pr, entry->d\_name, pr.re\_nsub + 1, rm, 0) == 0 &&

 rm[0].rm\_so == 0) {

 pthread\_mutex\_lock(&global\_queue\_mutex);

 queue\_push(&global\_queue, cdir, entry->d\_name);

 sem\_post(&global\_semaphore);

 pthread\_mutex\_unlock(&global\_queue\_mutex);

 unsigned sleep\_time = rand() % 200;

 usleep(sleep\_time \* 1000);

 }

 }

 }

 // Закрываем чтение каталога

 closedir(cur\_dir);

 // Поднимаемся на один уровень вверх

 chdir("..");

 return 0;

}

//-- Работа с потоками -------------------------

void \*thread\_func(void \*param) {

 int num\_col = \*((int\*) param);

 queue\_node \*qn;

 while (1) {

 sem\_wait(&global\_semaphore);

 pthread\_mutex\_lock(&global\_queue\_mutex);

 qn = queue\_pop(&global\_queue);

 if (qn == NULL) {

 sem\_post(&global\_semaphore);

 pthread\_mutex\_unlock(&global\_queue\_mutex);

 return 0;

 }

 pthread\_mutex\_unlock(&global\_queue\_mutex);

 strcat(qn->dir, "/");

 strcat(qn->dir, qn->fname);

 FILE \*f = fopen(qn->dir, "r");

 if (f == NULL) {

 print\_error("Error for open file", qn->dir);

 continue;

 }

 int col\_count = 0;

 int c, i, j, k, n, nn, flag, l;

 //-- Чтение первой строки и подсчет столбцов

 while ((c = fgetc(f)) != EOF && c != '\n') {

 if (c == ',')

 ++col\_count;

 }

 ++col\_count;

 fseek(f, 0, SEEK\_SET);

 char \*\*str\_vect = (char \*\*) malloc(sizeof(char \*) \* col\_count);

 for (i = 0; i < col\_count; ++i) {

 str\_vect[i] = (char \*) malloc(256);

 str\_vect[i][0] = '\0';

 }

 stack\_node \*st = NULL;

 //-- Чтение файла ---------------------

 i = j = k = 0;

 flag = 0;

 while ((c = fgetc(f)) != EOF) {

 //printf("%c\n", c);

 if (c == '"')

 flag = !flag;

 if (c == ' ' && !flag)

 continue;

 if (c == '\n') {

 if (str\_vect[0][0] != '\0') {

 str\_vect[i][j] = '\0';

 stack\_push(&st, str\_vect);

 ++k;

 str\_vect = (char \*\*) malloc(sizeof(char \*) \* col\_count);

 for (i = 0; i < col\_count; ++i) {

 str\_vect[i] = (char \*) malloc(256);

 str\_vect[i][0] = '\0';

 }

 }

 i = j = 0;

 } else if (c == ',') {

 str\_vect[i][j] = '\0';

 ++i; j = 0;

 } else {

 str\_vect[i][j++] = (char) c;

 }

 }

 fclose(f);

 //-- Перемещение строк из стека в вектор -----

 if (str\_vect[0][0] == '\0') {

 --k;

 for (i = 0; i < col\_count; ++i)

 free(str\_vect[i]);

 free(str\_vect);

 str\_vect = stack\_pop(&st);

 }

 char \*\*\*str\_mat = (char \*\*\*) malloc(sizeof(char \*\*\*) \* (++k));

 str\_mat[k-1] = str\_vect;

 for (i = 0; i < k - 1; ++i)

 str\_mat[i] = stack\_pop(&st);

 // Sort

 n = k;

 do {

 nn = 0;

 for (j = 1; j < n; ++j)

 if (strcmp(str\_mat[j-1][num\_col], str\_mat[j][num\_col]) > 0){

 char \*\*temp = str\_mat[j-1];

 str\_mat[j-1] = str\_mat[j];

 str\_mat[j] = temp;

 nn = j;

 }

 n = nn;

 } while (n);

//-- Печать в файл -------------------------

 f = fopen(qn->dir, "w");

 for (i = 0; i < k; ++i) {

 for (j = 0; j < col\_count - 1; ++j) {

 fprintf(f, "%s,", str\_mat[i][j]);

 }

 fprintf(f, "%s\n", str\_mat[i][j]);

 }

 fclose(f);

 free(qn);

 unsigned s\_time = rand() % 300;

 usleep(s\_time \* 1000);

 }

}

int main (int argc, char\* argv[]) {

 srand(time(0));

 int sem\_value = 0, new\_elem, sleep\_time, numCPU;

 unsigned count\_elem\_queue = 0, i, j, sum;

 pthread\_t \*threads;

 DIR \*cur\_dir;

 if (argc < 4) {

 printf("Недостаточно аргументов запуска программы!\n");

 return 1;

 }

 int field = atoi(argv[3]);

//-- Инициализация глобальных переменных ----------------

 sem\_init(&global\_semaphore, 0, 0);

 pthread\_mutex\_init(&global\_queue\_mutex, NULL);

 queue\_init(&global\_queue);

 err\_file = NULL;

 // Проверяем, что в параметрах передан каталог попыткой сделать его рабочим каталогом

 if (chdir(argv[1]) == -1) {

 print\_error("Ошибка каталога, возможно его не существует или это файл:", argv[1]);

 return 1;

 }

 chdir("..");

//-- Запуск потоков ---------------------

 numCPU = sysconf(\_SC\_NPROCESSORS\_ONLN);

 printf("%d threads will be started.\n\n", numCPU);

 threads = (pthread\_t \*) malloc(sizeof(pthread\_t) \* numCPU);

 for (i = 0; i < numCPU; ++i) {

 pthread\_create(&threads[i], NULL, &thread\_func, (void\*)&field);

 }

//-- Подготовка регулярного выражения ---------

 regex\_t preg;

 int err;

 if ((err = regcomp(&preg, argv[2], REG\_EXTENDED | REG\_ICASE)) != 0) {

 char buf[512];

 regerror(err, &preg, buf, 512);

 print\_error("Error regular expression:", buf);

 return 1;

 }

 //-- Вызываем рекурсивный обход ------

 print\_dir(argv[1], preg);

 sem\_post(&global\_semaphore);

 printf("exit");

//-- Ожидание завершения потоков ----------

 for (i = 0; i < numCPU; ++i) {

 pthread\_join(threads[i], NULL);

 }

 printf("exit2");

 if (err\_file != NULL)

 fclose(err\_file);

 return 0;

}

# IV. Тестовый пример

Для тестирования программы была создана следующая схема директорий и файлов:

* a:
	+ b:
		- 1.csv
		- 12.csv
		- 123.csv
	+ c:
		- 4.csv
		- 42.csv
		- 423.csv
	+ d:
		- e:
			* 5.csv
			* 52.csv
		- f:
			* 67.csv
		- 7.csv
		- 72.csv
	+ 34.csv
	+ 123.csv
	+ 123.txt

Каждый файл содержит следующий текст:

# V. Результаты работы программы

Программа была запущена следующей командой:

./PractEx a "(.\*2.\*\.csv)" 2

Алгоритм смог найти файлы:

* a/b/12.csv
* a/b/123.csv
* a/c/42.csv
* a/c/423.csv
* a/d/e/52.csv
* a/d/72.csv
* a/123.csv

И изменить их содержимое на:

