Разработать на языке C для ОС Linux:

программу, позволяющую выполнять рекурсивный поиск файлов, начиная с

указанного каталога, с помощью динамических (разделяемых) библиотек-плагинов

(использовать в качестве основы программу, #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <unistd.h>

#include <getopt.h>

#define MAX\_PATH\_LENGTH 1024

// Function to check if a file contains a specific byte sequence

int file\_contains(const char \*filename, const char \*search\_target) {

FILE \*file = fopen(filename, "rb");

if (!file) {

perror("Error opening file");

return 0;

}

// Get file size

fseek(file, 0, SEEK\_END);

long file\_size = ftell(file);

fseek(file, 0, SEEK\_SET);

// Allocate memory to store file content

char \*file\_content = (char \*)malloc((size\_t)file\_size + 1); // Add 1 for null terminator

if (!file\_content) {

fclose(file);

perror("Error allocating memory");

return 0;

}

// Read file content

if (fread(file\_content, 1, (size\_t)file\_size, file) != (size\_t)file\_size) {

fclose(file);

free(file\_content);

perror("Error reading file");

return 0;

}

file\_content[file\_size] = '\0'; // Null-terminate the string

fclose(file);

// Search for the target sequence

char \*found = strstr(file\_content, search\_target);

free(file\_content);

return (found != NULL);

}

// Function to recursively search files in a directory

void search\_directory(const char \*path, const char \*search\_target) {

DIR \*dir;

struct dirent \*entry;

char full\_path[MAX\_PATH\_LENGTH];

dir = opendir(path);

if (!dir) {

perror("Error opening directory");

return;

}

while ((entry = readdir(dir)) != NULL) {

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

continue;

snprintf(full\_path, MAX\_PATH\_LENGTH, "%s/%s", path, entry->d\_name);

struct stat statbuf;

if (stat(full\_path, &statbuf) == -1) {

perror("Error stating file");

continue;

}

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

search\_directory(full\_path, search\_target);

} else {

if (file\_contains(full\_path, search\_target)) {

printf("%s\n", full\_path);

}

}

}

closedir(dir);

}

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

int opt;

int help\_flag = 0, version\_flag = 0;

// Parse command line options

static struct option long\_options[] = {

{"help", no\_argument, 0, 'h'},

{"version", no\_argument, 0, 'v'},

{0, 0, 0, 0}

};

while ((opt = getopt\_long(argc, argv, "hv", long\_options, NULL)) != -1) {

switch (opt) {

case 'h':

help\_flag = 1;

break;

case 'v':

version\_flag = 1;

break;

default:

fprintf(stderr, "Usage: %s [OPTION]... DIRECTORY SEARCH\_TARGET\n", argv[0]);

exit(EXIT\_FAILURE);

}

}

if (help\_flag) {

printf("Help information...\n");

exit(EXIT\_SUCCESS);

}

if (version\_flag) {

printf("Version information: Lab11cvdN3247\n");

exit(EXIT\_SUCCESS);

}

if (optind >= argc - 1) {

fprintf(stderr, "Usage: %s [OPTION]... DIRECTORY SEARCH\_TARGET\n", argv[0]);

exit(EXIT\_FAILURE);

}

const char \*directory = argv[optind];

const char \*search\_target = argv[optind + 1];

// Perform recursive search in the specified directory

search\_directory(directory, search\_target);

return 0;

}

);

динамическую библиотеку, реализующую заданный вариантом лабораторной работы

из Табл. 4 критерий поиска файлов.

Программа должна представлять собой консольную утилиту, настройка работы

которой осуществляется путем передачи аргументов в строке запуска и/или с помощью

переменных окружения:

lab12cvdN3247 [опции] [каталог]

Программа должна выполнять рекурсивный поиск файлов, отвечающих критериям,

которые задаются опциями в командной строке. Доступные критерии поиска (и,

соответственно, доступные опции) определяются наличием в заданном каталоге

динамических библиотек, расширяющих функциональность программы (далее — плагинов).

При запуске без имени каталога для поиска программа выводит справочную

информацию по опциям и доступным в момент запуска плагинам. Поддерживаемые

программой опции перечислены в Табл. 2. Выводятся все опции, поддерживаемые

плагинами, и их описание. По умолчанию плагины ищутся в том же каталоге, где

расположен исполняемый файл программы, а если задана опция -P, то в каталоге, указанном

в этой опции. В случае запуска программы с несколькими опциями, задающими критерии

поиска, эти критерии объединяются логической операцией «И» (то же самое, если задана

опция -A) или логической операцией «ИЛИ» (если задана опция -O). Если задана опция -N,

то после объединения всех условий поиска по «И» или «ИЛИ», оно меняется на

противоположное.

Плагины представляют собой динамические библиотеки в формате ELF с

произвольным именем и расширением .so и интерфейсными функциями, перечисленными в

Табл. 1. Подробное описание API плагинов содержится в файле plugin\_api.h.

При обнаружении файла, отвечающего заданным критериям поиска, в стандартный

поток вывода выводится полный путь к этому файлу. При определении переменной

окружения LAB1DEBUG в стандартный поток ошибок должна выводиться информация о том,

что и в каком месте файла нашлось (чтобы было легче понять, почему файл отвечает

критериям поиска), а также может выводиться любая дополнительная отладочная

информация. Переменные окружения, которые должны поддерживаться программой и

библиотекой, приведены в Табл. 3.

Имя программы должно начинаться на lab1, далее должен следовать уникальный для

варианта суффикс. Имя библиотеки (плагина) должно начинаться на lib, далее должен

следовать уникальный для варианта суффикс и расширение .so. Уникальный суффикс

составляется из первых букв имени, отчества (если есть) и фамилии студента, выполняющего

лабораторную работу. Далее следует номер группы студента. Используются строчные

латинские буквы и арабские (в традиционном понимании, т. е. 0..9) цифры. Например, если

студента, выполняющего лабораторную, зовут Петр Сергеевич Иванов, его группа — N3245,

то имя программы должно быть lab1psiN3245, а имя библиотеки — libpsiN3245.so.

Проект (исходные коды, заголовочные файлы, Makefile и прочие файлы, необходимые

для сборки) должен содержаться в отдельном каталоге с именем, совпадающим с названием

программы (lab12cvdN3247) и собираться с помощью стандартной утилиты make. Makefile

должен поддерживать как минимум цели all и clean. Если для сборки проекта требуется что-

то большее, чем make all, или для запуска и проверки проекта требуются какие-либо

нетривиальные или неочевидные действия, то инструкции по сборке и запуску проекта

следует добавить в файл README.txt в формате plain text и разместить его в каталоге

проекта.

Замечание 1. При выполнении лабораторной работы следует использовать функции

стандартной библиотеки С и системные вызовы операционной системы. Использовать ввод-

вывод в стиле С++ (классы ifstream/ofstream/...) запрещено. Использовать контейнеры и

алгоритмы STL (<string>, <vector>, <map>, ...) запрещено.

Замечание 2. В программах должна присутствовать обработка ошибок: в случаях, если

пользователь задал неверную комбинацию опций, указал файлы, которые невозможно

открыть, и т.д. программа должна выдавать диагностическое сообщение на консоль (в

стандартный поток ошибок и/или лог-файл), прежде чем завершиться.

Замечание 3. При обходе дерева каталогов нужно учитывать, что доступ к некоторым

файлам и каталогам может завершиться с ошибкой (например, по причине отсутствия прав

доступа и т.д.). В таком случае следует вывести сообщение об ошибке в стандартный поток

ошибок и продолжить обход. Для упрощения реализации обхода символические ссылки (и

аналогичные средства — жесткие ссылки, bind mount'ы и т.д.) можно игнорировать.

Замечание 4. Категорически запрещается использовать статические массивы (с размерами,

заданными на этапе компиляции) для любых данных, размер которых зависит от входных

данных или условий запуска. Для хранения таких данных необходимо использовать

динамическую память и определять объем необходимой памяти в зависимости от ситуации.

Статические массивы можно использовать в тех ситуациях, когда известен максимальный

размер обрабатываемых данных (и он не превышает размеров стека или максимального

размера статических массивов, допускаемого компилятором).

Замечание 5. Плагинами поддерживаются только длинные опции. Плагин может

поддерживать несколько опций (например, в целях отладки). Совпадение имен опций в

разных плагинах считается ошибкой.

Замечание 6. Информационные сообщения выводятся программой в стандартный поток

вывода, сообщения об ошибках — в стандартный поток ошибок. С помощью определения

переменной окружения LAB1DEBUG можно включить вывод отладочных сообщений

программой и плагинами в стандартный поток ошибок. Рекомендуется предварять все

сообщения плагина его названием (чтобы было легче разбираться, каким плагином были

выведены сообщения).

Замечание 7. Плагины (в качестве потенциальных плагинов можно рассматривать любые

файлы с расширением .so) по умолчанию ищутся в том же каталоге, где расположен

исполняемый файл. Обратите внимание, что этот каталог может не совпадать с текущим

рабочим каталогом. Опция -P позволяет задать каталог с плагинами явно.

Замечание 8. Несмотря на то, что для компиляции программ необходимо использовать

компилятор gcc, использования расширений GNU C желательно по возможности избегать и

ориентироваться на использование стандарта C11 или более позднего.

Замечание 9. В вариантах, предусматривающих поиск многобайтовых значений (бинарных,

«упакованных» данных) нужно предусмотреть поиск вариантов хранения данных и в little-

endian, и в big-endian порядке.

Замечание 10. В вариантах, где необходимо сравнивать некоторые величины с заданными

значениями, операторы сравнения задаются следующим образом: eq означает «равно»

(equal), ne означает «не равно» (not equal), gt означает «строго больше» (greater than), lt

означает «строго меньше» (less than), ge означает «больше или равно» (greater or equal), le

означает «меньше или равно» (less or equal).

Замечание 11. Опции -A, -O и -N должны работать следующим образом. Предположим,

программе доступно два плагина. Первый поддерживает опцию --exe, позволяющую найти

любые исполняемые файлы, второй — опцию --file-len, позволяющую найти файлы

заданной длины. При запуске

lab12cvdN3247 --exe --file-len 1000 /tmp

программа должна искать в каталоге /tmp и вложенных в него исполняемые файлы длиной

1000 байтов. При запуске

lab12cvdN3247 -O --exe --file-len 1000 /tmp

программа должна искать в каталоге /tmp и вложенных в него или исполняемые файлы, или

файлы длиной 1000 (то есть совпадение должно быть хотя бы по одному из критериев). В

случае запуска

lab12cvdN3247 -O -N --exe --file-len 1000 /tmp

программа должна искать файлы, которые не являются исполняемыми и не имеют длину

1000 байтов.

Таблица 1. Интерфейсные функции библиотеки-плагина

| Функция | Назначение |
| --- | --- |
| plugin\_get\_info | Получение информации о поддерживаемых плагином опциях. |
| plugin\_process\_file | Проверка соответствия файла заданным критериям. |

Таблица 2. Опции командной строки, поддерживаемые программой

| Опция | Назначение |
| --- | --- |
| -P dir | Каталог с плагинами. |
| -A | Объединение опций плагинов с помощью операции «И»  (действует по умолчанию). |
| -O | Объединение опций плагинов с помощью операции «ИЛИ». |
| -N | Инвертирование условия поиска (после объединения опций  плагинов с помощью -A или -O). |
| -v | Вывод версии программы и информации о программе (ФИО  исполнителя, номер группы, номер варианта лабораторной). |
| -h | Вывод справки по опциям. |

Таблица 3. Переменные окружения, поддерживаемые программой и динамической библиотекой

| Переменная | Назначение |
| --- | --- |
| LAB1DEBUG | Включение вывода отладочной информации. |

Таблица 4. Опции, поддерживаемые библиотекой-плагином

| **Описание поддерживаемых плагином опций** |
| --- |
| **Опция: --crc16 <значение>**  **Назначение: поиск файлов, контрольная сумма которых, рассчитанная с помощью**  **алгоритма CRC-16-CCITT, равна заданному значению. Значение суммы задается**  **строкой, содержащей запись числа либо в двоичной (0b...), либо в десятичной,**  **либо шестнадцатеричной (0x...) системах.**  **Пример: --crc16 0xbabe** |