Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет компьютерных систем и сетей Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ОТЧЕТ
по лабораторной работе №1 на тему
“Знакомство с Linux/Unix и средой программирования. POSIX-совместимая файловая система.”

Выполнил:
студ. гр. 350501 Везенков М. Ю.

Проверил: ст.пр. Поденок Л. П.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ.....3
2 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ.....4
2.1 Описание алгоритма выполнения работы.....4
2.2 Описание основных функций.....5
2.3 Описание порядка сборки и использования.....7
3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ.....8
4 ВЫВОД.....9

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Разработать программу `dirwalk`, сканирующую файловую систему и выводящую в `stdout` информацию в соответствии с опциями программы.

Формат вывода аналогичен формату вывода утилиты `find`.

```
dirwalk [dir] [options]
```

`dir` – начальный каталог. Если опущен, текущий (`./`).

`options` – опции.

`-l` – только символические ссылки (`-type l`)

`-d` – только каталоги (`-type d`)

`-f` – только файлы (`-type f`)

`-s` – сортировать выход в соответствии с `LC_COLLATE`

Опции могут быть указаны как перед каталогом, так и после.

Опции могут быть указаны как раздельно, так и вместе (`-l -d, -ld`).

Если опции `ldf` опущены, выводятся каталоги, файлы и ссылки.

Для обработки опций рекомендуется использовать `getopt(3)`.

Программа должна быть переносимой (возможности `linux` не используются, только `POSIX`).

2 ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

2.1 Описание алгоритма выполнения работы

Программа предназначена для рекурсивного обхода дерева каталогов, начиная с указанного пути (или текущего каталога по умолчанию), с использованием функции `nftw`() стандарта `POSIX`. Её основная задача — вывести на стандартный вывод пути к элементам файловой системы, таким как файлы, каталоги и символические ссылки.

Выполнение программы начинается с разбора аргументов командной строки с помощью функции `getopt`(). В ходе этого процесса определяется начальный каталог для обхода, который по умолчанию является текущим каталогом ".", если пользователь не указал другой. Также определяются флаги фильтрации: `-l` для вывода только символических ссылок, `-d` для каталогов и `-f` для обычных файлов. Дополнительно может быть указан флаг сортировки `-s`, который предписывает сортировать итоговый вывод. Если пользователь не указал ни одного из флагов фильтрации (`-l`, `-d`, `-f`), программа по умолчанию выводит все типы записей, обнаруженные `nftw` (за исключением специальных `FTW_NS` и `FTW_DP`), что соответствует поведению утилиты `find` без опции `-type`.

В случае, если указан флаг сортировки (`-s`), программа инициализирует динамический список (`path_list_s`). Этот список служит для временного хранения найденных путей перед их сортировкой и выводом. Память для этого списка выделяется динамически и может расширяться по мере необходимости во время добавления новых путей.

После настройки опций вызывается основная функция обхода `nftw`(). Ей передаются начальный путь, указатель на функцию обратного вызова `process_entry`, максимальное количество одновременно открытых файловых дескрипторов и флаг `FTW_PHYS`. Этот флаг указывает `nftw` не следовать по символическим ссылкам во время обхода. Так как стандартный интерфейс `nftw` не предусматривает передачу произвольных пользовательских данных в функцию обратного вызова, опции программы становятся доступны для `process_entry` через статическую глобальную переменную `g_callback_options`.

Функция `process_entry` затем вызывается для каждого элемента, найденного в файловой системе во время обхода. Она анализирует тип элемента (`typeflag`) и его метаданные (`struct stat`), особенно используя `S_ISREG` для точной идентификации обычных файлов при активном флаге `-f`. Эта проверка выполняется для определения, соответствует ли элемент критериям фильтрации, заданным пользователем через опции. Если элемент удовлетворяет условиям (или если фильтрация вовсе не активна), выполняется одно из двух действий. При выключенной сортировке путь к

элементу немедленно печатается на стандартный вывод с помощью printf(). Если же сортировка включена, путь копируется с помощью strdup() и добавляется в ранее инициализированный динамический список функцией add_path_to_list().

По завершении работы nftw и, следовательно, всего обхода дерева каталогов, программа переходит к обработке результатов. Если была включена сортировка, собранные в списке пути упорядочиваются с помощью стандартной функции qsort(). В качестве компаратора используется функция compare_paths, которая применяет strcoll() для сравнения строк с учётом текущей локали (LC_COLLATE), обеспечивая правильный алфавитный порядок. После сортировки весь список путей последовательно выводится на стандартный вывод. Затем вся динамически выделенная память, использовавшаяся для хранения списка и копий строк путей, освобождается вызовом free_path_list(). Если же сортировка не была запрошена, то вывод уже произошел во время работы process_entry, и никаких дополнительных действий по выводу не требуется. Программа завершает свою работу, возвращая EXIT_SUCCESS в случае успешного выполнения или EXIT_FAILURE, если в процессе возникли ошибки, например, связанные с выделением памяти, чтением каталога или записью в стандартный вывод.

2.2 Описание основных функций

Программа состоит из нескольких ключевых функций, каждая из которых выполняет свою часть общей задачи по обходу каталогов и выводу результатов.

Функция main() является главной точкой входа. Она инициализирует выполнение, начиная с установки локали через setlocale() для обеспечения корректной сортировки строк. Затем она разбирает аргументы командной строки с помощью getopt(), определяя начальный каталог и активированные опции (-l, -d, -f, -s). На основе этих опций инициализируется структура options_s. Если пользователь не указал конкретный каталог, используется текущий каталог ".". Если не указаны флаги фильтрации типов (-l, -d, -f), то по умолчанию включаются все три типа для вывода. Если запрошена сортировка (-s), main() инициализирует динамический список path_list_s и выделяет для него начальную память. Перед вызовом nftw она устанавливает глобальный указатель g_callback_options, чтобы сделать опции доступными для функции обратного вызова. После завершения nftw, если была включена сортировка, main() вызывает qsort() для сортировки списка путей с использованием compare_paths, печатает результат и освобождает память через free_path_list(). Функция также отвечает за обработку ошибок на различных этапах и завершает программу с соответствующим кодом возврата (EXIT_SUCCESS или EXIT_FAILURE).

Функция process_entry() — это функция обратного вызова, которую

nftw() вызывает для каждого обнаруженного элемента файловой системы.

Она получает путь к элементу (fpath), указатель на его структуру stat (sb), флаг типа (typeflag) и структуру FTW (ftwbuf). Доступ к опциям программы осуществляется через глобальный указатель g_callback_options. Функция сначала отбрасывает нерелевантные типы FTW_NS (ошибка получения статуса) и FTW_DP (повторное посещение каталога после обработки его содержимого). Затем она применяет логику фильтрации, сверяя typeflag (и sb->st_mode с S_ISREG для -f) с флагами show_links, show_dirs, show_files из структуры опций. Она также корректно обрабатывает нечитаемые каталоги (FTW_DNR) при активной опции -d или при отсутствии фильтров. Если фильтры не заданы, любой действительный элемент считается совпадением.

В зависимости от того, включена ли сортировка, process_entry либо немедленно печатает путь с помощью printf(), либо вызывает add_path_to_list() для добавления копии пути в динамический список. Функция возвращает 0 для продолжения обхода или -1 при возникновении ошибки (например, при неудачном выделении памяти или ошибке записи), что приводит к остановке nftw.

Функция add_path_to_list() является вспомогательной и используется только при включенной сортировке. Её задача — добавить копию строки пути в динамический список path_list_s. Перед добавлением она проверяет, достаточно ли места в текущем массиве paths. Если массив заполнен, его емкость удваивается с помощью realloc(). Функция обрабатывает возможные ошибки выделения памяти при расширении массива. После обеспечения достаточного места она создает копию переданной строки пути с помощью strdup(), которая также выделяет память под копию. Указатель на эту копию сохраняется в массиве paths, и счетчик элементов count увеличивается. Функция возвращает 0 в случае успеха или -1, если произошла ошибка выделения памяти (realloc или strdup), сигнализируя process_entry о необходимости остановить обход.

Функция free_path_list() также является вспомогательной и используется при сортировке для освобождения ресурсов, занятых списком путей. Она принимает указатель на структуру path_list_s. Сначала она проходит по всем элементам массива paths (от 0 до count - 1) и вызывает free() для каждой строки пути, которая была ранее скопирована с помощью strdup(). После освобождения памяти всех строк она вызывает free() для самого массива указателей paths. Наконец, она сбрасывает поля paths, count и capacity в структуре list в безопасные значения (NULL и 0), чтобы предотвратить использование освобожденной памяти.

Функция compare_paths() предназначена для использования со стандартной функцией qsort(). Она выполняет сравнение двух строк путей, необходимых для сортировки. Функция принимает два аргумента типа const void *, которые являются указателями на элементы сортируемого массива (т.е.

указателями на char *). Для самого сравнения она использует функцию strcoll(), которая сравнивает строки с учетом текущих языковых настроек системы (LC_COLLATE). Это гарантирует правильный алфавитный порядок сортировки, соответствующий ожиданиям пользователя в его локали.

Функция возвращает результат strcoll(), который является отрицательным числом, нулем или положительным числом, указывая порядок сравниваемых строк.

Функция print_usage() отвечает за вывод справочной информации пользователю. Она печатает в стандартный поток ошибок (stderr) сообщение о правильном использовании программы, включая ожидаемый формат командной строки и описание доступных опций (-l, -d, -f, -s). Эта функция вызывается, если пользователь допустил ошибку при вводе аргументов командной строки (например, указал недопустимую опцию или слишком много аргументов-каталогов) или если программа была запущена без каких-либо аргументов.

2.3 Описание порядка сборки и использования

Для сборки проекта используется утилита make, которая автоматизирует процесс компиляции исходного кода и создания исполняемого файла. Процесс сборки организован так, чтобы исполняемый файл и объектные файлы размещались в соответствующих поддиректориях каталога build. При отсутствии этих директорий они создаются автоматически в процессе сборки, что упрощает организацию проекта и поддержание порядка.

Используется два режима сборки: режим отладки (debug) и релизный режим (release). В режиме отладки (make без указания переменной MODE) компилятор включает флаги для упрощения отладки кода, такие как -g2 и -ggdb. В релизном режиме (make MODE=release) включаются строгие флаги компиляции для обеспечения качества финальной сборки, такие как -Werror, запрещающий игнорирование предупреждений компилятора.

Для удаления всех скомпилированных файлов, включая объектные файлы и исполняемый файл, а также для очистки директории build предусмотрена команда make clean. Это гарантирует, что последующие сборки будут выполнены с нуля и не будут зависеть от остаточных артефактов предыдущих процессов.

Таким образом, структура проекта поддерживает четкое разделение этапов сборки, что упрощает переключение между режимами отладки и релиза, а также организацию рабочих файлов.

3 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

```
./build/debug/dirwalk /etc -f | wc
   901     901   34215
./build/debug/dirwalk /etc -d | wc
   288     288    6887
./build/debug/dirwalk /etc -l | wc
   869     869   34620
./build/debug/dirwalk /etc  | wc
  2064    2064   75928
```

Рисунок 1 – пример работы `dirwalk`

```
find /etc -type f | wc
   901     901   34215
find /etc -type d | wc
   288     288    6887
find /etc -type l | wc
   869     869   34620
find /etc | wc
  2064    2064   75928
```

Рисунок 2 – пример работы утилиты `find`

4 ВЫВОД

При выполнении лабораторной работы была изучена работа с файлами в операционной системе `Linux`.

В ходе выполнения работы была разработана программа, которая способна сканировать файлы и выводить в поток вывода информацию в соответствии с опциями программы.