**1. Основы работы в среде Unix. Команды и утилиты. Скрипты shell.**

Цель: Изучение элементов и конструкций скриптов shell: переменных, параметров, ветвлений, циклов, вычислений, команд shell и вызовов внешних команд (программ).

Общая задача:

Написать скрипт для shell, обеспечивающий получение заданным образом организованной выходной информации. Результаты выполнения записываются в файл (наиболее универсальный способ их сохранения).

Скрипт следует сделать исполняемым, выполнить, проанализировать результат.

Используются перенаправление ввода-вывода, внешние утилиты и фильтры, переменные shell.

Для редактирования shell-скрипта рекомендуется использовать консольные редакторы текста, например vim, nano, micro.

Варианты заданий

1) Топ 5 процессов, которые потребляют больше всего памяти и процессорного времени в системе: имя процесса, имя пользователя, PID.

2) Список подкаталогов и файлов в заданном каталоге: имя анализируемого каталога, список подкаталогов, список файлов, общее число файлов и подкаталогов.

Имя анализируемого каталога может быть задано аргументом командной строки.

3) Шестнадцатиричный дамп заданного файла: имя файла, строки дампа, имя пользователя, дата (в качестве подписи).

4) "Самодокументирующийся скрипт": форматирует собственный текст, снабжает заголовком и подписью по аналогии с предыдущим вариантом.

5) Статистика текущего сеанса: имя пользователя, текущее время, дата, текущий каталог, число процессов в системе, время работы.

6) Замер времени выполнения команды: имя команды/процесса, время выполнения. Команда или имя внешней программы задаются в командной строке скрипта. Сам скрипт такой программой быть не может (что сильно упрощает реализацию!)

7) Список файлов в каталоге и подкаталогах, к которым не обращались в течение времени, заданного аргументом командной строки.

8) Список коммитов популярного репозитория (например, https://github.com/ или схожем) в формате дерева, короткого хэша коммита, электронной почты автора и названия коммита.

----

**2. Скрипты shell (углубленная версия)**

Цель: Изучение элементов и конструкций скриптов shell: переменных, параметров, ветвлений, циклов, вычислений, команд shell и вызовов внешних программ (shell, sed, awk, wget, различные фильтры и т.д.) для решения достаточно сложной задачи, имеющей практическое значение, а также принципов интеграции Unix-программ скриптами shell.

Общая задача:

Написать один или несколько скриптов shell в соответствии с вариантами заданий.

Можно считать, что размер окна консоли жёстко задан и не может изменяться во время выполнения.

Кроме bash можно использовать другие варианты shell'ов (zsh, csh и другие).

Варианты заданий:

1) Хранитель экрана (screensaver) для консоли: большие текстовые часы, меняющие своё расположение на экране.

Реализовать свой генератор случайных чисел, необходимый для нахождения нового положения часов на экране. Основой для генератора может послужить мультипликативный алгоритм генерации (можно выбрать другой, желательно тоже целочисленный). Инициализация от текущего времени. Обновление генератора через фиксированные промежутки времени (например, 10 секунд).

2) Клиент для pastebin (или похожего сервиса). Реализовать поддержку следующих функций:

- создание новой текстовой записи из stdin или файла, указанного через аргументы командной строки, автоматического (на основе расширения файла) или ручного (через аргументы командной строки) выбора языка подсветки;

- создание новой текстовой записи должно возвращать id;

- вывод на консоль или в файл (в зависимости от аргументов) текстовой записи по id;

- для создания и скачивания должен использоваться один и тот же скрипт.

Пример использования:

Создать новую запись с типом текст: echo testing | ./pastebin.sh -t text

Вывести на экран запись по id=QEXe0syg: ./pastebin.sh QEXe0syg

3) Консольная версия игры "2048".

Дополнительно: хранение лучших результатов. Для сохранения использовать файл, список результатоы демонстрируется по окончании игры.

4) Консольная версия игры "Крестики-нолики" (для двух игроков).

Дополнительно: создание скриншота - файла, содержащего текст текущего игрового поля.

5) Поиск файлов с обходом дерева каталогов. Над найденными файлами выполняются заданные действия (похоже на утилиту find).

Требования к поиску:

- поиск файла по имени, образец поиска может быть регулярным выражением (имена проверяются на соответствие регулярному выражению). Образец передается как аргумент командной строки. Поиск начинается с текущей директории;

- поиск файла по имени, образец поиска задан списком (имена проверяются на совпадение с любым из этого списка). Способ передачи списка -- ряд аргументов в командной строке. Поиск начинается с текущей директории.

Действия над найденными файлами:

- для файлов с заданным заголовком (первой строкой) вывод их содержимого в виде листинга: построчно, строки пронумерованы. Заголовок считается фиксированным.

6) Поиск файлов аналогично предыдущему заданию, отличаются действия над найденными файлами:

- подсчет суммы значений байтов каждого файла и общей по всем файлам (подсчет контрольной суммы файлов). Вместо суммирования можно использовать другую доступную функцию.

----

**3. Обработка текстовой информации. Регулярные выражения.**

Цель: Изучение регулярных выражений и использующих их утилит.

Общая задача:

Написать скрипт (скрипты) для sed, awk и т.д., либо скрипт shell, обращающийся к необходимым программам, для обработки входных данных (файлов) в соответствии с вариантом задания.

Необходимо предусмтреть поведение скрпта (скриптов) при ошибочных или "неочищенных" входных данных.

Варианты зданий:

1) Замена строчных букв на заглавные в начале предложений, т.е. в начале документа и после точки, не находящейся внутри, например, числа, а также после знаков "!", "?".

Предложение может начинаться с новой строки (т.е. предыдущая точка может находиться в одной строке, а заменяемая строчная буква - в следующей).

2) Извлечение из текста подстрок с характерным форматом (например, адреса, web-ссылки, наименования книг с авторами и т.п.)

В сложных случаях допустимо вводить искусственное определение формата, например для литературного источника - "Фамилия\_автора запятая Инициалы точка Название в кавычках тире Год\_издания Необязательная\_интернет-ссылка".

3) "Табличный калькулятор"

Входные данные - таблицы (матрицы) в файле (файлах) в формате CSV ("comma-separated") и признак выполняемой над ними операции.

Нераспознанные (нечисловые) элементы заменяются нейтральными/незначащими значениями (например, нулями), аналогично дополняются недостающие.

Примечание: Признак операции и дополнительные сведения о данных, например размерность, можно включить во входные файлы - это может упростить логику обработки (при использовании awk)

4) Входные данные - файл (файлы) со списками товаров в формате: наименование, количество, цена. Одно и то же наименование может встречаться многократно.

Выходные - список товаров без повторов наименований с вычисленными для них суммарным количеством, средней ценой, общей стоиомстью.

5) Входные данные - таблица, содержащая среди прочих колонки начала и окончания интервала в виде временнЫх меток (дата и время).

Выходные - таблица, строки которой отсортированы по началу, окончанию, длительности интервала.

Даты и время в различных строках таблицы могут быть записаны в разном формате (но с однозначным критерием распознавания).

6) Импорт данных

Входные данные - файл CSV. Выходные - SQL-запросы INSERT для переноса их в базу данных.

Первая строка входного файла содержит имя таблицы БД, вторая - имена соотвествующих полей в этой таблице.

----

**4. Основы программирования на C под Unix. Инструментарий программиста в Unix.**

Цель: Изучить основные инструменты (программы) для работы с программным проектос: cc (gcc, c++, g++), cpp, make и др.

Общая задача:

Написать программу в соответствии с вариантом задания, скомпилировать её в выходной исполняемый файл, преобразовать проект в многомодульнй (например, головной модуль и "прикладные" функции), создать makefile для управления обработкой проекта и проверить выполнение описанный в нем целей.

Разбить программу, разработанную в предыдущей л/р на несколько модулей (как минимум на два: main.c и algorithm.c).

1) Программа-фильтр -- инверсия порядка байт в потоке (первый -- последний, второй -- предпоследний, и т.д.).

2) Программа-фильтр -- инверсия порядка строк потока (предполагается, что поток -- текст, состоящий из отдельных строк). Длину строк можно считать ограниченной некоторой константой.

3) Программа-фильтр -- инверсия порядка символов в каждой строке потока, порядок самих строк не изменяется. Длину строк можно считать ограниченной некоторой константой.

4) Упрощённый аналог утилиты ps. Запуск программы с аргументом -l выводит список процессов (pid и имя исполняемого файла). Запуск с аргументом -p <pid> выводит дополнительную информацию о процессе.

5) Проверка и коррекция текста по словарю. Проверка производится на совпадение, без анализа словоформ; корректировать достаточно одиночные ошибки. Программа оформляется как фильтр, словарь -- внешний файл, его размер заранее не известен.

6) Проверка и "цензура" текста: удаление (замена фиксированным шаблоном) "запрещенных" слов, заданных словарем. Проверка на совпадение, без анализа словоформ. Программа оформляется как фильтр, словарь -- внешний файл, его размер заранее не известен.

7) Программа -- "записная книжка". Режим работы -- командная строка (программа вызывается на 1 операцию, задаваемую аргументами командной строки, после его исполнения управление возвращается в shell. База данных -- текстовый файл, записи включают несколько полей (формат выбирается произвольно). Операции: добавление, удаление, поиск (по различным полям) и отображение, изменение записи.

8) Программа -- "записная книжка". Аналогично, но режим работы интерактивный (диалоговый): ввод команды, исполнение, ожидание следующей.

9) Программа-фильтр -- преобразование символов в комбинации азбуки Морзе. Непреобразуемые символы отбрасываются.

10) Программа-фильтр -- преобразование комбинаций азбуки Морзе в печатные символы.

11) Программа -- криптографический фильтр: шифрование и дешифрование потока. Криптоалгоритм выбирается произвольно (можно простейшие). Способ передачи пароля и других параметров шифрования выбирается произвольно. (В случае достаточно сложного криптоалгоритма задание раздивается на 2: прямое и обратное преобразование.).

Примечания:

1) Список заданий может быть модифицирован и расширен.

2) Задания выполняются с использованием как библиотек, так и, возможно, вызовов внешних программ. Специальных ограничений на создание временных файлов не налагается, но желательно этого избегать.

3) В makefile должны быть следующие цели:

1. Цель по сборке основной программы (цель по умолчанию)

2. Цели по сборке отдельных модулей

3. Цель для очистки проекта (clean)

4. Цель для тестирования проекта. Нужно, чтобы прогонялось некоторое количество тестов. Каждый тест, например, состоит из входных данных (in.txt) и ожидаемых выходных данных (out-expected.txt). Запускается программа на входных данных; ее выход сохраняется в некотором файле (out.txt). Затем эмпирический полученный результат сравнивается (out.txt) с ожидаемым (out-expected.txt). Если результы совпадают, тест прошёл. Программа считается готовой, когда она проходит все тесты.

----

**5. Файловая система, сигналы, потоки, другие ресурсы.**

Цель: Знакомство с возможностями операционной системы по работе с файловой системой, многопоточные приложения, взаимодействию между потоками.

Общая задача

Консольное приложение, использующее возможности операционной системы по обеспечению многопоточности и корректно закрывать потоки, освобождать ресурсы, как минимум, при получении сигнала SIGINT.

Требования к сборке проекта аналогичны предыдущей лабораторной работе.

Варианты заданий:

1) Упрощенный "пакетный калькулятор" с параллельной обработкой данных.

Многопоточное приложение, которое выполнит требуемые действия над числами и запишет сумму результатов в отдельный файл. Название рабочего каталога и файла с результатами передаются в виде аргументов командной строки.

В рабочем каталоге лежат входные текстовые файлы. Каждый файл состоит из двух строк. В первой строке - число, обозначающее действие, а во второй - числа с плавающей точкой, разделенные пробелом.

Действия могут быть следующими:

"1" - сложение

"2" - умножение

"3" - сумма квадратов

(список операций может быть изменен и/или дополнен)

2) Демнстрация многопоточности и управления потоками.

После запуска программы один поток начинает писать в файл, заданный через аргумент командной строки, некоторый текст в вечном цикле.

По нажатию клавиши <+> создаётся ещё один поток, по нажатию клавиши <-> удаляется последний добавленный.

Каждый поток посимвольно пишет в файл свою уникальную строку. При этом операция вывода строки должна быть атомарной, т.е. процесс вывода должен быть синхронизирован таким образом, чтобы строки не перемешивались.

3) Модели взаимодействия параллельных процессов/потоков

Многопоточное (многопроцессное) приложение, реализующее одну из моделей (СМО, "обедающие философы" Дейкстры и т.д.), используя параллельные потоки и средства их синхронизации.

Нужно обеспечить параметризацию модели, протоколирование результатов работы, управление посредством сигналов и отдельного процесса/потока - "консоли оператора".

Сеансы моделирования демонстрируют статистику по модели, возможные тупики, блокировки и т.д.

4) Процессы-демоны

Создание программы-демона, обрабатывающего произвольные заданные сигналы. Обработка заключается в протоколировании сигнала (записи в файл номера сигнала, его символического имени и времени получения).

Список контролируемых сигналов задается файлом конфигурации, который считывается демоном при старте и по сигналу SIGHUP (этот сигнал также может быть среди протоколируемых).

Дополнительно: демон также осуществляет мониторинг изменений файла, заданного при его запуске (например, посредством потока, периодически проверяющего файл). Факт изменения записывается в общий протокол, изменения запоминаются для будущих проверок.

----

**6. Элементы сетевого программирования (сокеты TCP/IP)**

Цель: Изучение стека сетевых протоколов TCP/IP, программного интерфейса сокетов (socket) и приемов программирования с использованием их.

Общая задача:

Создать программу-сервер, реализующие протокол реальной сетевой службы или собственный; как правило, серверы выполняются в режиме демона.

Должны поддерживаться параллельные обращения со стороны нескольких клиентов.

Так как предусмотренный спецификаей порт часто бывает недоступен (требует привилегий администратора), используется доступный непривилегированный порт. Номер порта передается программе при ее запуске.

При необходимости создаются также вспомогательные программы, например, клиент для тестирования и демонстрации работоспособности сервера. В качестве "универсального" клиента для протокола TCP можно использовать программу telnet.

Варианты заданий:

1) "Сервер морзянки" -- прямое преобразование.

Программа-демон, поддерживает соединения TCP, преобразует принимаемый текст в знаки азбуки Морзе и возвращает клиенту, регистр букв игнорируется, непреобразуемые символы отбрасываются либо заменяются пробелами. Номер базового порта TCP задается как параметр командной строки.

2) "Сервер морзянки" -- обратное преобразование.

Аналогично предыдущему, но выполняется преобразование из знаков азбуки Морзе в обычный текст.

Примечание: Представляет интерес исследование способности преобразования к "самосинхронизации" после ошибок в знаках азбуки Морзе.

3) Сервис date (daytime).

Программа-демон, реализующая сервер указанной службы с поддержкой протоколов TCP и UDP (порт номер 13): в ответ на обращение выдает текущие дату и время.

Подробности см. в RFC867.

4) Сервис echo.

Программа-демон, реализующая сервер указанной службы с поддержкой протоколов TCP и UDP (порт номер 7): любое полученное сообщение возвращается отправителю без изменений.

Подробности см. в RFC347.

5) Сервис systat.

Программа-демон, реализующая сервер указанной службы с поддержкой протоколов TCP и UDP (порт номер 11): в ответ на обращение выдает текущий список активных пользователей системы.

Подробности см. в RFC866.

6) Сервис time.

Программа-демон, реализующая сервер указанной службы с поддержкой протоколов TCP и UDP (порт номер 37): в ответ на обращение выдает текущее время в секундах, прошедших с 00:00 1.01.1900.

Подробности см. в RFC868.

7) Упрощенный сервис ftp.

Программа-демон, реализующая сервер указанной службы с ограниченной функциональностью (отклик клиенту, список файлов каталогов, корректные коды состояний, простые варианты передачи файла).

Подробности см. в RFC959.

8) Упрощенный сервис http.

Программа-демон, реализующая HTTP-сервер с минимальной функциональностью.

Подробности см. в RFC2616 либо RFC7230..RFC7237

Примечание: Список реализуемых протоколов может быть пересмотрен и расширен.