Лабораторная работа № 14

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Павличенко Родион Андреевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Выполнение лабораторной работы

Написали командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустили командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (> /dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработали программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

```bash
#!/bin/bash

if [ $# -ne 2 ]; then
    echo "Использование: $0 <t1> <t2>"
    exit 1
fi

t1=$1

t2=$2

semaphore_file="semaphore.lock"

touch $semaphore_file

function access_resource {
    while ! ln $semaphore_file $0.lock 2>/dev/null; do
        echo "Ресурс занят, ожидание освобождения..."
        sleep $tl
    done

    echo "Ресурс освобожден, начало использования на $t2 секунд"
    sleep $t2;
    echo "Ресурс освобожден, использование завершено"

    rm $0.lock
}

access_resource
```

Рис. 2.1: 1 задание

Реализовали команду man с помощью командного файла. Изучили содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

```bash
#!/bin/bash

if [ $# -ne 1 ]; then
     echo "Использование: $0 <Название команды>"
     exit 1
fi

command_name=$1

man_directory="/usr/share/man/man1"

if [ -f "$man_directory/$command_name.1.gz" ]; then
    zcat "$man_directory/$command_name.1.gz" | less
else
   echo "Справка для команды '$command_name' не найдена"
fi
```

Рис. 2.2: 2 задание

Используя встроенную переменную $RANDOM, написали командный файл, генерирующий случайную последовательность букв латинского алфавита. Учли, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767

```bash
#!/bin/bash

generate_random_letter() {
    random_number=$((RANDOM % 26))

    letter=$(printf \\$(printf '%03o' $((65 + random_number))))

    echo -n "$letter"
}

random_sequence=""
for ((i=0; i<10; i++)); do
       random_sequence="$random_sequence$
       (generate_random_letter)"
done

echo "Случайная последовательность букв латинского алфавита: $random_sequence"
```

Рис. 2.3: 3 задание

3 Контрольные вопросы

1 Ошибка: отсутствует пробел между [и условием. Правильный синтаксис:

while [“$1” != “exit”]

В Bash важно ставить пробелы вокруг [и]. Также лучше заключать переменные в кавычки, чтобы избежать ошибок при обработке значений с пробелами.

2. Для объединения строк в Bash можно использовать простой подход с помощью оператора конкатенации. Пример:

str1=“Hello” str2=“World” result=“$str1 $str2” echo $result

3.Утилита seq используется для генерации последовательности чисел. Пример использования:

seq 1 5 Результат:

1 2 3 4 5

Альтернативы: Можно реализовать функциональность seq с помощью Bash-цикла:

for i in {1..5}; do

echo $i

done

4. В Bash выражения внутри $(()) выполняются как целочисленные операции. Ожидаемый результат:

Результат: 3. В Bash при целочисленных операциях дробная часть усекается.

5.Основные отличия между zsh и bash:

Автодополнение: Zsh имеет более продвинутую систему автодополнения, включая расширенные возможности автодополнения путей и параметров команд.

Тема и настройка: Zsh предлагает больше возможностей для кастомизации и настройки через темы (например, с помощью oh-my-zsh).

Модальности: Zsh поддерживает дополнительные возможности для работы с массивами и строками.

Поддержка глобальных псевдонимов: в Zsh можно использовать глобальные псевдонимы, которые работают в любом контексте, в отличие от Bash, где они ограничены только оболочкой.

6. Синтаксис верен, но нужно убедиться, что переменная LIMIT определена.

```
LIMIT=10
for ((a=1; a <= LIMIT; a++)); do
echo $a
done
```

7. Преимущества Bash:

Интеграция с ОС: Bash тесно интегрирован с операционной системой, что делает его удобным для работы с файловой системой, процессами и системными командами.

Простота и скорость: скрипты на Bash легко пишутся и быстро выполняются, идеально подходят для автоматизации системных задач.

Низкоуровневая работа с процессами: Bash позволяет запускать процессы, считывать их вывод и управлять ими, что обеспечивает полную гибкость в управлении.

Недостатки Bash:

Ограниченные возможности для сложных программ: Bash не подходит для написания крупных приложений, поскольку это не полноценный язык программирования, а скорее оболочка для скриптов.

Отсутствие объектно-ориентированного подхода: в Bash отсутствуют классы и объекты, что затрудняет организацию сложных программных решений.

Ошибки при работе с переменными: управление типами данных в Bash ограничено

4 Выводы

Изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.