

Лабораторная работа №14

дисциплина: Архитектура компьютера

Орлов Илья Сергеевич

Содержание

1	Цель	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Контрольные вопросы	12
6	Выводы	15

Список иллюстраций

4.1	1	10
4.2	2	11
4.3	3	11

Список таблиц

1 Цель

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

2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить 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. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите 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`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до

32767.

3 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: — оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; — C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; — оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; — BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

4 Выполнение лабораторной работы

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить 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. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов. (рис. fig. 4.1).

```
GNU nano 0.3 semaphore.sh
#!/bin/bash

# Время ожидания и использования ресурса
WAIT_TIME=${1:-3} # t1 - время ожидания (по умолчанию 3 сек)
USE_TIME=${2:-5}  # t2 - время использования (по умолчанию 5 сек)
SEM_FILE="/tmp/resource.sem"

echo "Процесс $$: Запущен с WAIT_TIME=$WAIT_TIME, USE_TIME=$USE_TIME"

# Функция ожидания ресурса
wait_for_resource() {
    local waited=0
    echo "Процесс $$: Ожидаю освобождения ресурса..."
    while [ $waited -lt $WAIT_TIME ]; do
        if [ ! -f "$SEM_FILE" ]; then
            echo "Процесс $$: Ресурс свободен! Занимаю..."
            touch "$SEM_FILE"
            echo "$$" > "$SEM_FILE"
            return 0
        fi
        echo "Процесс $$: Ресурс занят процессом $(cat "$SEM_FILE" 2>/dev/null || echo 'unknown'), жду..."
        sleep 1
        waited=$((waited + 1))
    done
    echo "Процесс $$: Таймаут ожидания! Ресурс не освободился."
    return 1
}

# Функция использования ресурса
use_resource() {
    echo "Процесс $$: Использую ресурс в течение $USE_TIME секунд..."
    local used=0
    while [ $used -lt $USE_TIME ]; do
        echo "Процесс $$: Работаю с ресурсом... ((${used} + 1)/$USE_TIME)"
        sleep 1
        used=$((used + 1))
    done
}
```

Рис. 4.1: 1

2. Реализовать команду `man` с помощью командного файла. Изучите 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`. (рис. fig. 4.2).

```

GNU nano 8.3                               my_man.sh
# /bin/bash

# Проверяем что указана команда
if [ $# -eq 0 ]; then
    echo "Ошибка: Не указана команда для поиска справки"
    echo "Использование: $0 имя_команды"
    echo "Пример: $0 ls"
    exit 1
fi

COMMAND_NAME="$1"
MAN_DIR="/usr/share/man/man1"
FOUND=0

echo " Поиск справки для команды: $COMMAND_NAME"

# Ищем файлы справки с разными вариантами имен
POSSIBLE_FILES=(
    "${COMMAND_NAME}.1.gz"
    "${COMMAND_NAME}.1"
    "${COMMAND_NAME}.gz"
    "${COMMAND_NAME}"
)

for file in "${POSSIBLE_FILES[@]}; do
    MAN_FILE="${MAN_DIR}/${file}"
    if [ -f "$MAN_FILE" ]; then
        echo " Найдена справка: $MAN_FILE"
        echo "===== "
        # Показываем справку
        if [[ "$MAN_FILE" == *.gz ]]; then
            gunzip -c "$MAN_FILE" | less 2:/dev/null || gunzip -c "$MAN_FILE"
        else
            cat "$MAN_FILE" | less 2:/dev/null || cat "$MAN_FILE"
        fi
        FOUND=1
        break
    fi
done

```

Рис. 4.2: 2

- Используя встроенную переменную \$RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767. (рис. fig. 4.3).

```

GNU nano 8.3                               random_letters.sh
# /bin/bash

# Параметры по умолчанию
COUNT=${1:-20} # Количество последовательностей
LENGTH=${2:-1} # Длина каждой последовательности
CASE=${3:-"both"} # Регистр: lower, upper, both

echo "Генератор случайных букв латинского алфавита"
echo "===== "

# Проверим корректность параметров
if [ "$COUNT" -le 0 ]; then
    echo "Ошибка: Количество должно быть положительным числом"
    exit 1
fi

if [ "$LENGTH" -le 0 ]; then
    echo "Ошибка: Длина должна быть положительным числом"
    exit 1
fi

echo "Генерируем: $COUNT последовательностей"
echo "Длина каждой: $LENGTH букв"
echo "Регистр: $CASE"

# Определяем набор букв в зависимости от регистра
case "$CASE" in
    "lower")
        LETTERS=({a..z})
        echo "Используем только строчные буквы"
        ;;
    "upper")
        LETTERS=({A..Z})
        echo "Используем только заглавные буквы"
        ;;
    "both"*)
        LETTERS=({a..z} {A..Z})
        echo "Используем и строчные и заглавные буквы"
        ;;
esac

```

Рис. 4.3: 3

5 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `1 while [$1 != "exit"]` В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки `[` и перед второй скобкой `]` выражение `$1` необходимо взять в `"`, потому что эта переменная может содержать пробелы Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`
2. Как объединить (конкатенация) несколько строк в одну? Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR1VAR2" echo "$VAR3"` : *Hello, World* : `VAR1 = "Hello,"VAR1+ = "World"echo"VAR1"` Результат: *Hello, World*
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Команда `seq` в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: `seq LAST`: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение `is` не выдает. `seq FIRST LAST`: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. `seq FIRST INCREMENT LAST`: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод. `seq -f «FORMAT» FIRST INCREMENT LAST`: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT

являются необязательными. `seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО`: Эта команда используется для `STRING` для разделения чисел. По умолчанию это значение равно `/n`. `FIRST` и `INCREMENT` являются необязательными. `seq -w FIRST INCREMENT LAST`: эта команда используется для выравнивания ширины путем заполнения начальными нулями. `FIRST` и `INCREMENT` являются необязательными.

4. Какой результат даст вычисление выражения $\$(10/3)$? Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`. Отличия командной оболочки `zsh` от `bash`: В `zsh` более быстрое автодополнение для `cd` с помощью `Tab` В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала В `zsh` поддерживаются числа с плавающей запятой В `zsh` поддерживаются структуры данных «хэш» В `zsh` поддерживается раскрытие полного пути на основе неполных данных В `zsh` поддерживается замена части пути В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`
6. Проверьте, верен ли синтаксис данной конструкции `1 for ((a=1; a <= LIMIT; a++))` `for ((a=1; a <= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` перед переменными `()`.
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки? Преимущества и недостатки скриптового языка `bash`: Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS Удобное перенаправление ввода/вывода Большое количество команд для работы с файловыми системами Linux Можно писать собственные скрипты, упрощающие работу в Linux Недостатки скриптового языка `bash`: Дополнительные

библиотеки других языков позволяют выполнить больше действий Bash не является языком общего назначения Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

6 Выводы

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