

Отчет по лабораторной работе № 14

**Программирование в командном процессоре ОС Unix. Расширенное
программирование**

Скобеева Алиса Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	11
5	Ответы на контрольные вопросы	12

Список иллюстраций

3.1	Пишем программу	7
3.2	Проверка работы программы	8
3.3	Пишем программу	8
3.4	Программа работает корректно	9
3.5	Пишем программу	10
3.6	Все работает корректно	10

Список таблиц

1 Цель работы

Изучить ОП в оболочке ОС Unix. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

Написать три командных файла с использованием логических управляющих конструкций и циклов.

3 Выполнение лабораторной работы

Пишем командный файл, реализующий упрощенный механизм семафоров. Также дорабатываем программу так, чтобы имела возможность взаимодействия трёх и более процессов.



```
foot
GNU nano 8.1 semaphore1.sh
#!/bin/bash

# Имя файла-семафора
semaphore_file="/tmp/my_semaphore"

# Время ожидания (t1) в секундах
wait_time=5

# Время удержания (t2) в секундах
hold_time=3

# Функция для захвата семафора
acquire_semaphore() {
    if [ -e "$semaphore_file" ]; then
        echo "Процесс $$ ждет освобождения ресурса..."
        start_time=$(date +%s)
        while [ -e "$semaphore_file" ]; do
            sleep 1
            current_time=$(date +%s)
            elapsed_time=$((current_time - start_time))
            if [ "$elapsed_time" -ge "$wait_time" ]; then
                echo "Процесс $$ превысил время ожидания."
                return 1 # Возвращаем код ошибки, если не удалось захватить семафор
            fi
        done
    fi
    touch "$semaphore_file"
    echo "Процесс $$ захватил ресурс."
}
```

Рис. 3.1: Пишем программу

Запускаем файл и открываем еще два терминала, смотрим на работу файла также и в них

```

1 2
foot
[askobeeva@fedora ~]$ nano semafor1.sh
[askobeeva@fedora ~]$ ./semafor1.sh > /dev/tty2 &
[1] 4484
[askobeeva@fedora ~]$ ./semafor1.sh > /dev/tty3 &
[2] 4493
bash: /dev/tty3: Отказано в доступе
[1] Завершён ./semafor1.sh > /dev/tty2
[askobeeva@fedora ~]$ sudo ./semafor1.sh > /dev/tty2 &
[3] 4665
[askobeeva@fedora ~]$ [sudo] пароль для askobeeva:
123
bash: 123: команда не найдена
[2] Выход 1 ./semafor1.sh > /dev/tty3
[3]+ Остановлен sudo ./semafor1.sh > /dev/tty2
[askobeeva@fedora ~]$ ./semafor1.sh > /dev/pts/2 &
[4] 4722
[askobeeva@fedora ~]$ ./semafor1.sh > /dev/pts/3 &
[5] 4756
[4] Завершён ./semafor1.sh > /dev/pts/2
[askobeeva@fedora ~]$
foot
[askobeeva@fedora ~]$ tty
/dev/pts/3
[askobeeva@fedora ~]$

```

Рис. 3.2: Проверка работы программы

Реализовываем команду man с помощью командного файла

```

foot
GNU nano 8.1 mantask.sh
#!/bin/bash

# Каталог с файлами справки
man_dir="/usr/share/man/man1"

# Команда, для которой нужно получить справку
command="$1"

# Проверка наличия аргумента
if [ -z "$command" ]; then
    echo "Использование: ./my_man <команда>"
    exit 1
fi

# Имя файла справки
man_file="$man_dir/$command.1.gz"

# Проверка существования файла
if [ ! -f "$man_file" ]; then
    echo "Справка для команды '$command' не найдена."
    exit 1
fi

# Открываем файл с помощью less
zless "$man_file"
exit 0

```

Рис. 3.3: Пишем программу

Запускаем файл

```
foot
.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH LS "1" "November 2024" "GNU coreutils 9.5" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[\fI,OPTION\fR]... [\fI,FILE\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of \fB-\fR-cftuvSUX\fR nor \fB-\fR-sort\fR is specified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB-\fR-a\fR, \fB-\fR-all\fR
do not ignore entries starting with .
.TP
\fB-\fR-A\fR, \fB-\fR-almost-all\fR
do not list implied . and ..
.TP
\fB-\fR-author\fR
with \fB-\fR-l\fR, print the author of each file
.TP
\fB-\fR-b\fR, \fB-\fR-escape\fR
print C-style escapes for nongraphic characters
.TP
\fB-\fR-block-size=\fI,SIZE\fR \fI
with \fB-\fR-l\fR scale sizes by SIZE when printing them:
```

Рис. 3.4: Программа работает корректно

Пишем командный файл, который используя встроенную переменную \$RANDOM генерирует случайную последовательность букв латинского алфавита

```
foot
GNU nano 8.1 random.sh
#!/bin/bash

# Длина последовательности
length=10

# Алфавит
alphabet="abcdefghijklmnopqrstuvwxyz"

# Функция для генерации случайной буквы
generate_random_letter() {
    random_index=$((RANDOM % ${#alphabet}))
    echo "${alphabet:$random_index:1}"
}

# Генерация последовательности
random_string=""
for i in $(seq 1 $length); do
    random_string+=$(generate_random_letter)
done

echo "$random_string"

exit 0
```

Рис. 3.5: Пишем программу

Запускаем файл и проверяем корректность работы программы

```
[aaskobeeva@fedora ~]$ touch random.sh
[aaskobeeva@fedora ~]$ chmod +x random.sh
[aaskobeeva@fedora ~]$ nano random.sh
[aaskobeeva@fedora ~]$ ./random.sh
ncztugwknc
[aaskobeeva@fedora ~]$ ./random.sh
brhlotmebj
[aaskobeeva@fedora ~]$
```

Рис. 3.6: Все работает корректно

4 Выводы

Мы успешно написали 3 командных файла с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы

1. Синтаксическая ошибка:

```
Bash while [ "$1" != "exit" ]
```

Необходимы пробелы вокруг [и], а также кавычки вокруг \$1, чтобы избежать ошибок, если \$1 не определена. 2. Конкатенация строк:

```
Bash result="string1string2$string3"
```

3. Утилита seq: Генерирует последовательность чисел. Другие способы:

- Цикл for: `for i in $(eval echo {1..10}); do ... done`
- while: `i=1; while [i -le 10]; do ...; i=$((i+1)); done`

4. Результат ((10/3)): 3 (целочисленное деление).

5. zsh vs. bash:

- Расширенная автодополнение
- Темы оформления и плагины
- Более гибкая настройка

6. Синтаксис for ((a=1; a <= LIMIT; a++)): Синтаксис верен.

7. bash vs. другие ЯП:

- Преимущества: Прямой доступ к командам ОС, удобство управления процессами, хорошо подходит для автоматизации задач.
- Недостатки: Ограниченные возможности в обработке сложных данных, менее эффективен для вычислительных задач, сложнее в поддержке

крупных проектов. Bash - скриптовый язык, другие языки могут быть скомпилированы.