

Отчёт по лабораторной работе №14

Программирование в командном процессоре ОС UNIX.

Фатима Халилова

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
3 Вывод	9
4 Контрольные вопросы	10

Список иллюстраций

2.1 Задание 1	6
2.2 Задание 2	7
2.3 Задание 3	8

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Написали командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени t_1 дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использует его в течение некоторого времени $t_2 < t_1$, 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 ($> /dev/ttyn$, где n – 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.

The screenshot shows two terminal windows. The left window, titled 'lab14_1.sh', contains the command `./lab14_1.sh` and its output, which consists of repeated lines: 'Пишу в файл...', 'Жду разблокировки файла', and 'Пишу в файл...'. The right window, titled 'lab14_3.sh', contains the command `#!/bin/bash` followed by a script. The script uses a lockfile named 'lockfile' located at `/tmp/lockfile`. It checks if the file exists, creates it if it doesn't, then enters a loop where it sleeps for 1 second, prints 'Жду разблокировки файла', and then writes 'Записываем в файл...' to the lockfile. The script ends with a cleanup section that removes the lockfile and exits.

```
#!/bin/bash
while test -f lockfile
do
    sleep 1
    echo "Жду разблокировки файла"
done
touch lockfile
let c=10
while ((c--))
do
    sleep 1
    echo "Пишу в файл..."
    echo "Записываем в файл...">>lockfile
done
rm lockfile
```

Рисунок 2.1: Задание 1

2. 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 получает в виде аргумента командной строки название команды и в виде результата выдает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

```

frhalova@frhalilova:~/work/study/2025-2026/Операционные системы/os-mirovats/lab14
ESC[4mDUESC[24m(1)
ESC[4mDUESC[24m(1)

ESC[1mNAME ESC[0m
du - estimate file space usage

ESC[1mSYNOPSIS ESC[0m
ESC[1mdu [ESC[22m[ESC[4mOPTION ESC[24m]... [ESC[4mFILE ESC[24m]...
ESC[1mdu [ESC[22m[ESC[4mOPTION ESC[24m]... [ESC[4m--files0-from=FILE] [ESC[0m

ESC[1mDESCRIPTION ESC[0m
Summarize device usage of the set of FILEs, recursively for directories.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m--bytes ESC[22m, ESC[1m--null ESC[0m
end each output line with NUL, not newline

ESC[1m--all ESC[22m, ESC[1m--all ESC[0m
write counts for all files, not just directories

ESC[1m--apparent-size ESC[0m
print apparent sizes rather than device usage; although the apparent
size is usually smaller, it may be larger due to holes in ('sparse')
files, internal fragmentation, indirect blocks, and the like

ESC[1m-B ESC[22m, ESC[1m--block-size=SIZE ESC[22m[ESC[4mSIZE ESC[0m
scale sizes by SIZE before printing them; e.g., '-B8M' prints sizes in
units of 1,048,576 bytes; see SIZE format below

ESC[1m-b ESC[22m, ESC[1m--bytes ESC[0m
equivalent to '--apparent-size=1'

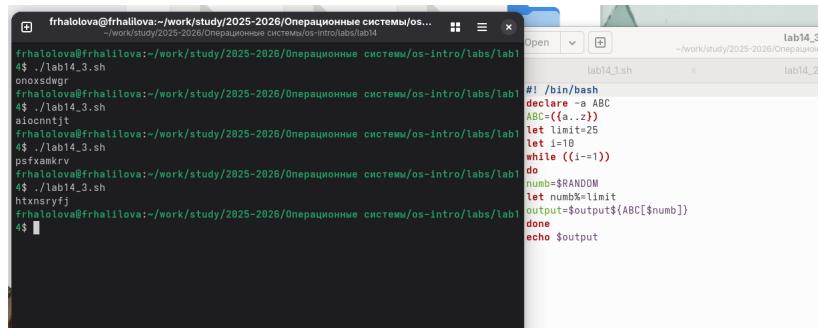
ESC[1m-c ESC[22m, ESC[1m--total ESC[0m
produce a grand total
m

/usr/share/man/man1/du.1.gz

```

Рисунок 2.2: Задание 2

3. Используя встроенную переменную \$RANDOM , написали командный файл, генерирующий случайную последовательность букв латинского алфавита



The screenshot shows a terminal window with two tabs open. The left tab contains a command-line session where a user named 'frhalolova' runs a script named 'lab14_3.sh'. The output of the script is displayed in green. The right tab is titled 'lab14_3' and shows the source code of the script. The script is a Bash script that declares an array 'ABC' containing lowercase letters from 'a' to 'z'. It initializes a variable 'limit' to 25 and a loop counter 'i' to 10. Inside the loop, it generates a random number 'numb' between 1 and 25, and prints the corresponding character from the 'ABC' array. The script then increments 'i' by 1 and loops back if 'i' is less than or equal to 1. Finally, it prints the value of '\$output'.

```
frhalolova@frhalilova:~/work/study/2025-2026/Операционные системы/os-intro/labs/lab14_3.sh
#!/bin/bash
declare -a ABC
ABC=(a..z)
let limit=25
let i=10
while ((i>=1))
do
numb=$((RANDOM%$limit))
let numb<limit
output+=${ABC[$numb]}
done
echo $output
```

Рисунок 2.3: Задание 3

3 Вывод

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

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: while [\$1 != "exit"] Ответ: Правильный вариант: while ["\$1" != "exit"]
2. Как объединить (конкатенация) несколько строк в одну? Ответ: Объединение нескольких строк в одну в Bash происходит с помощью символа '\'
3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash? Ответ для примера: В Linux имеется программа seq, которая воспринимает в качестве аргументов два числа и выдает последовательность всех чисел, расположенных между заданными. С помощью этой команды можно заставить for в bash работать точно так же, как аналогичный оператор работает в обычных языках программирования. Для этого достаточно записать цикл for следующим образом:

```
for a in $( seq 1 10 ) ; do  
catfile_$a  
done
```

Эта команда выводит на экран содержимое 10-ти файлов:
«file_1», ..., «file_10».

4. Какой результат даст вычисление выражения \$((10/3))? Ответ: Так как

это целочисленное деление, то произойдет округление в сторону ближайшего числа, и выведется $3.10/3 = 3$.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

Ответ: По размеру Bash больше Zsh. Zsh и Bash предлагают сходный функционал. Обе имеют программируемое дополнение (хотя у Zsh оно появилось раньше), встроенные команды и функции для создания скриптов. У Zsh также в запасе есть несколько собственных хитростей, например, расширенная подстановка имени файла, которая превращает команду поиска find почти что в ненужное излишество. Включение в путь ** означает соответствие любому символу, включая разделитель - слэш, поэтому */*.jpg касается всех файлов *.jpg в текущей директории и в любых поддиректориях. Мало того, сюда также включаются права доступа к файлу, владелец, тип или отметка времени – большинство опций, предусмотренных find. Например, можно использовать ls -l /**/bin/*(s) для вывода списка всех setuid-файлов в /bin, /usr/bin и /usr/local/bin. При наборе имени директории в командной строке Zsh переключается на эту директорию. Выполнение скриптов в Zsh основным быстрее, чем в Bash – по большей части примерно на 20% – однако Zsh разработан для интерактивного пользования. В Zsh расширенная подстановка имени файла и более развитая опция дополнения..

6. Проверьте, верен ли синтаксис данной конструкции for ((a=1; a <= LIMIT; a++)) Ответ: В bash для оператора цикла for существует другая конструкция.

Пример:

```
for A in Ai Bi Ci do
echo A
done
```

на терминал будет выведено :

Ai Bi Ci

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки? Ответ: Вначале был Bourne Shell (sh), его написал Стивен Борн для Bell Labs Research Unix. Bash – это Bourne Again Shell (Снова Оболочка Борна), который, к счастью, редко используется. Почти все современные дистрибутивы Linux используют Bash в качестве оболочки по умолчанию, и это превращает Bash в фактический стандарт, с которым сравниваются все остальные. Дело не в малом размере Bash, и не в скорости. По размеру Bash больше некоторых оболочек, кроме одной: Sash, которая не использует библиотек и имеет несколько дополнительных встроенных команд. Bash также и не самая быстрая оболочка, однако большинству пользователей это неважно, ибо подлинно важна его гибкость. Bash обладает некоторыми функциями, превосходящими стандарт POSIX, хотя при желании можно добиться от него и POSIX-поведения. Если запустить Bash командой sh, с опцией командной строки –posix или при установленной переменной окружения POSIXLY_CORRECT, Bash будет работать как стандартная оболочка POSIX. При запуске через sh, Bash по возможности пытается работать как исходная оболочка Борна, но лишь в тех ситуациях, когда это не вступит в конфликт со стандартом POSIX.