

Лабораторная работа №13

Дисциплина: Операционные системы

Егорова Александра

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	7
4	Выводы	13
5	Контрольные вопросы	14

List of Figures

3.1	Создаем файл	7
3.2	Первый скрипт	8
3.3	Проверка скрипта	8
3.4	Изменяем первый скрипт	9
3.5	Изменяем первый скрипт	9
3.6	Проверка скрипта	9
3.7	Содержимое каталога	10
3.8	Второй скрипт	10
3.9	Проверка скрипта	11
3.10	Проверка скрипта	11
3.11	Проверка скрипта	11
3.12	Третий скрипт	12
3.13	Проверка скрипта	12

1 Цель работы

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 Выполнение лабораторной работы

- 1) Создала файл: `pr1.sh` и написала соответствующий скрипт. Написала 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). (рис. -fig. 3.1) (рис. -fig. 3.2)

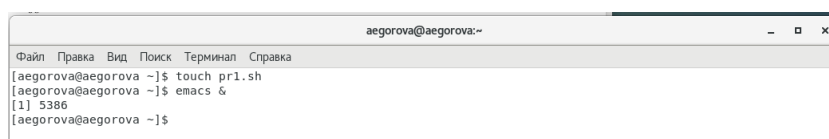
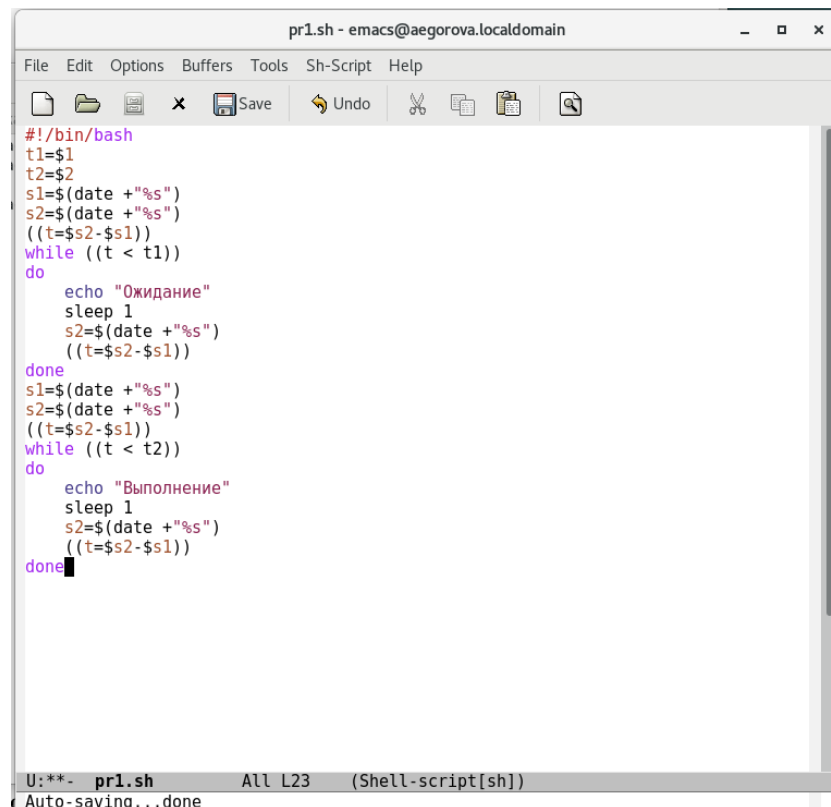


Figure 3.1: Создаем файл



```
pr1.sh - emacs@aegorova.localdomain
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=$s2-$s1))
while ((t < t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=$s2-$s1))
done
done
```

U:**- pr1.sh All L23 (Shell-script[sh])
Auto-saving...done

Figure 3.2: Первый скрипт

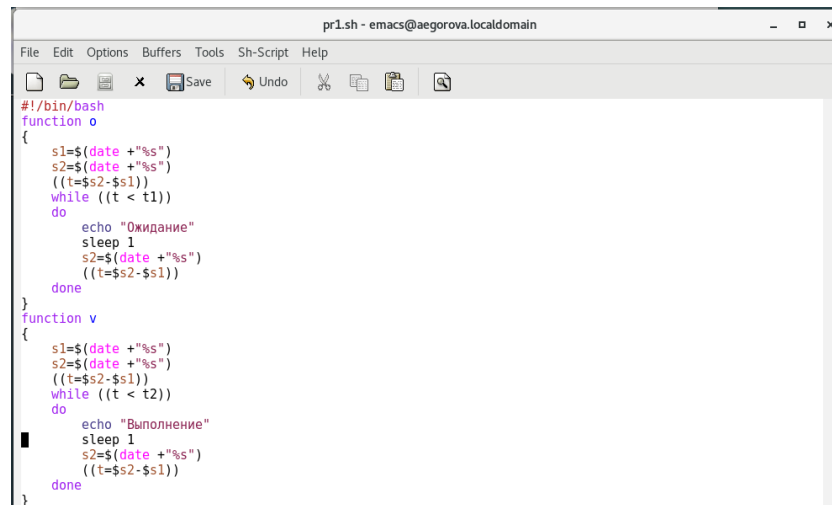
Проверяю работу написанного скрипта (команда «./pr1.sh 3 6»), предварительно добавив право на исполнение файла («chmod +x pr1.sh»). Скрипт работает корректно. (рис. -fig. 3.3)

```
[aegorova@aegorova ~]$ chmod +x pr1.sh
[1]+  Done          emacs
[aegorova@aegorova ~]$ ./pr1.sh 3 6
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
```

Figure 3.3: Проверка скрипта

После этого я изменила скрипт так, чтобы его можно было выполнять в нескольких терминалах и проверила его работу («./pr1.sh 2 3 Ожидание > dev/tty# , где # — номер терминала куда перенаправляется вывод»). Ни одна из команд не

сработала (выводит сообщение “Отказано в доступе”). При этом скрипт работает корректно. (рис. -fig. 3.4) (рис. -fig. 3.5) (рис. -fig. 3.6)



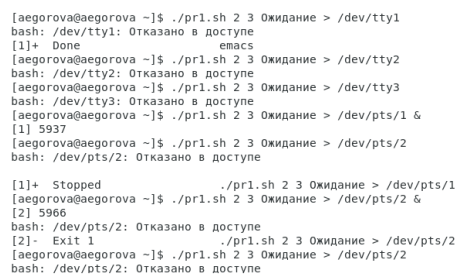
```
#!/bin/bash
function o
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t < t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
function v
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=s2-s1))
    while ((t < t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s")
        ((t=s2-s1))
    done
}
```

Figure 3.4: Изменяем первый скрипт



```
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then
        o
    fi
    if [ "$command" == "Выполнение" ]
    then
        v
    fi
    echo "Следующее действие: "
    read command
done
```

Figure 3.5: Изменяем первый скрипт



```
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/tty1
bash: /dev/tty1: Отказано в доступе
[1]+  Done                  ./pr1.sh 2 3 Ожидание > /dev/tty1
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/tty2
bash: /dev/tty2: Отказано в доступе
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/tty3
bash: /dev/tty3: Отказано в доступе
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/pts/1 &
[1] 5937
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/pts/2
bash: /dev/pts/2: Отказано в доступе
[1]+  Stopped                  ./pr1.sh 2 3 Ожидание > /dev/pts/1
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/pts/2 &
[2] 5966
bash: /dev/pts/2: Отказано в доступе
[2]+  Exit 1                  ./pr1.sh 2 3 Ожидание > /dev/pts/2
[aegorova@aegorova ~]$ ./pr1.sh 2 3 Ожидание > /dev/pts/2
bash: /dev/pts/2: Отказано в доступе
```

Figure 3.6: Проверка скрипта

2) Реализуем команду man с помощью командного файла. Изучила 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. 3.7)

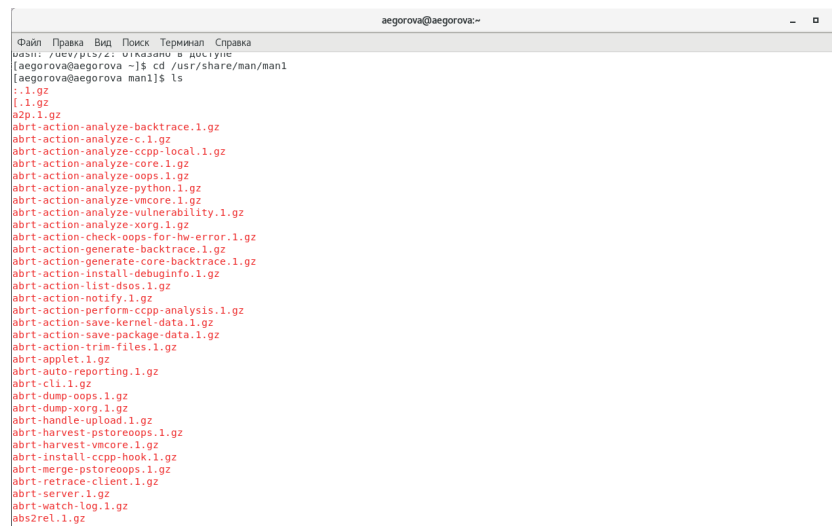


Figure 3.7: Содержимое каталога

Создала файл: `pr2.sh` и написала соответствующий скрипт. Далее я проверила работу написанного скрипта («`./pr2.sh ls`» и «`./pr2.sh mkdir`»), предварительно добавив право на исполнение файла («`chmod +x pr2.sh`»). Скрипт работает корректно. (рис. -fig. 3.8) (рис. -fig. 3.9) (рис. -fig. 3.10) (рис. -fig. 3.11)

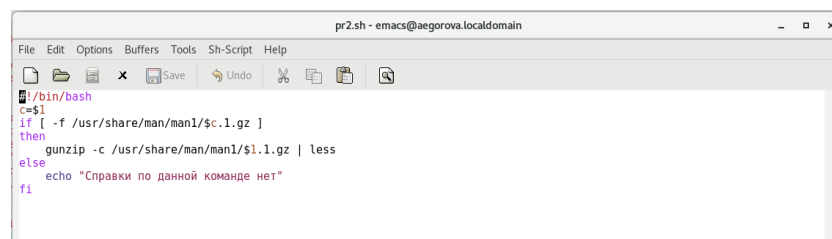


Figure 3.8: Второй скрипт

```
[aegorova@aegorova ~]$ touch pr2.sh
[aegorova@aegorova ~]$ emacs &
[2] 6045
[aegorova@aegorova ~]$ chmod +x pr2.sh
[2]- Done emacs
[aegorova@aegorova ~]$ ./pr2.sh ls
[aegorova@aegorova ~]$ ./pr2.sh mkdir
```

Figure 3.9: Проверка скрипта

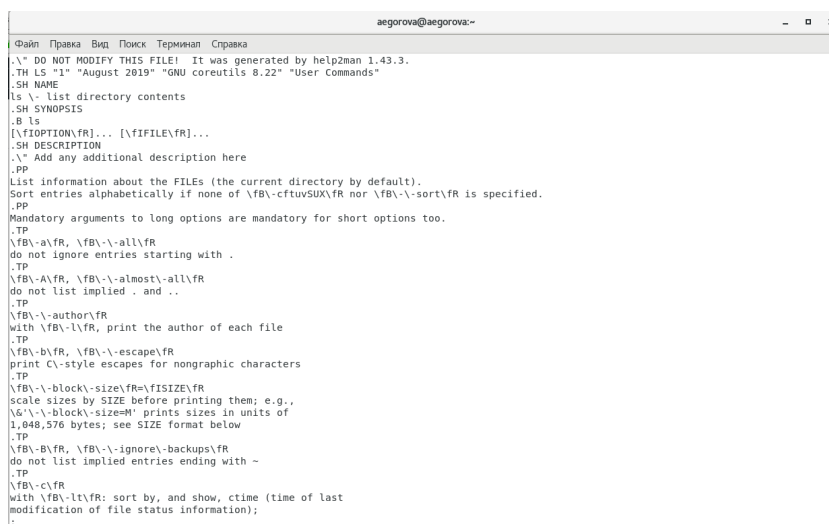


Figure 3.10: Проверка скрипта

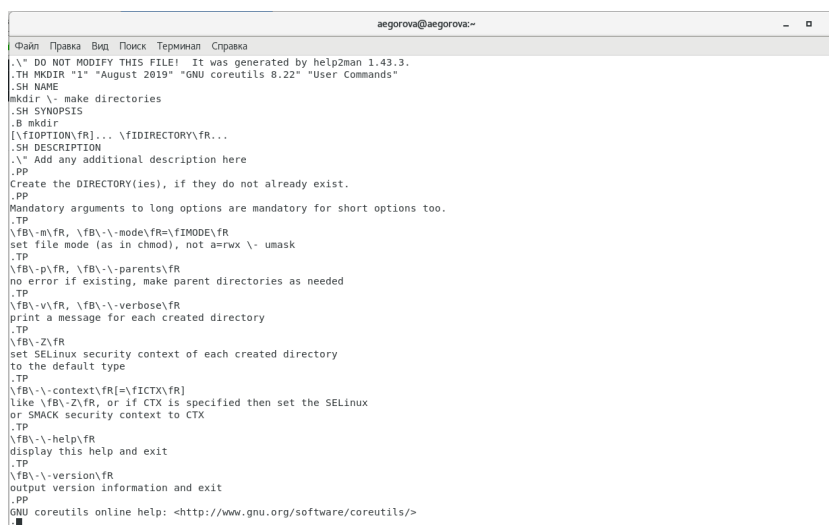
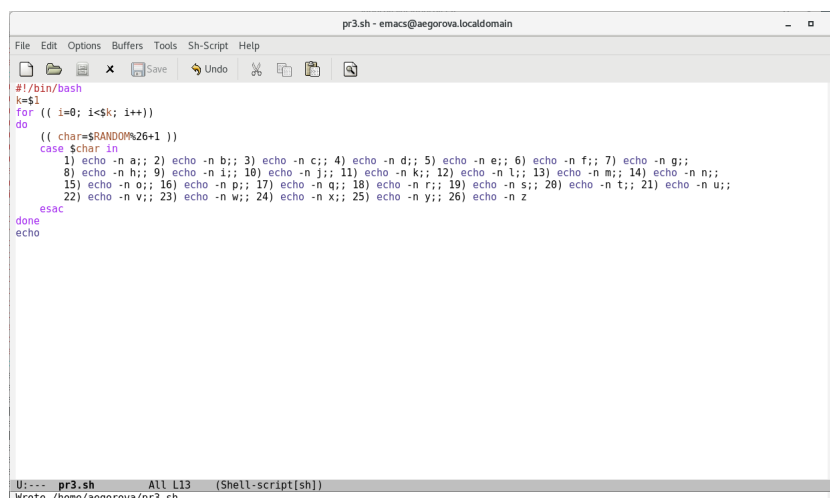


Figure 3.11: Проверка скрипта

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

Для данной задачи я создала файл: pr3.sh и написала соответствующий скрипт. (рис. -fig. 3.12)



```
pr3.sh - emacs@aegorova.localdomain
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
k=$1
for (( i=0; i<=$k; i++))
do
  (( char=$((RANDOM%26+1)) ))
  case $char in
    1) echo -n a;; 2) echo -n b;; 3) echo -n c;; 4) echo -n d;; 5) echo -n e;; 6) echo -n f;; 7) echo -n g;;
    8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;; 12) echo -n l;; 13) echo -n m;; 14) echo -n n;;
    15) echo -n o;; 16) echo -n p;; 17) echo -n q;; 18) echo -n r;; 19) echo -n s;; 20) echo -n t;; 21) echo -n u;;
    22) echo -n v;; 23) echo -n w;; 24) echo -n x;; 25) echo -n y;; 26) echo -n z
  esac
done
echo
```

U:--- pr3.sh All L13 (Shell-script[sh])
Wrote /home/aegorova/pr3.sh

Figure 3.12: Третий скрипт

Далее я проверила работу написанного скрипта («./pr3.sh 5» и «./pr3.sh 13»), предварительно добавив право на исполнение файла («chmod +x pr3.sh»). Скрипт работает корректно. (рис. -fig. 3.13)

```
[aegorova@aegorova ~]$ chmod +x pr3.sh
[2]- Done emacs
[aegorova@aegorova ~]$ ./pr3.sh 5
aqitv
[aegorova@aegorova ~]$ ./pr3.sh 13
igavlmetdxttx
```

Figure 3.13: Проверка скрипта

4 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, а также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

- 1) `while [$1 != "exit"]`. В данной строчке допущены следующие ошибки: 1) не хватает пробелов после первой скобки `[` и перед второй скобкой `]`; 2) выражение `$1` необходимо взять в `"`, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`.
- 2) 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) Команда `seq` в Linux используется для генерации от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. чисел. 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) Результатом данного выражения $\$(10/3)$ будет 3, потому что это целочисленное деление без остатка.
- 5) 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) for ((a=1; a <= LIMIT; a++)) синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать \$ перед переменными ().
- 7) Преимущества 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.