

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

**Лабораторная работа No 12. Программирование в командном
процессоре ОС UNIX. Расширенное программирование**

Абдуллина Ляйсан Раисовна НПИбд-01-21

Содержание

1	Цель работы	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
3.1	2	8
4	Контрольные вопросы	12
5	Выводы	15
6	Список литературы	16

Список иллюстраций

3.1	Написанная программа	7
3.2	Написанная программа	7
3.3	Проверка - работает успешно	8
3.4	Написанная программа	9
3.5	Проверка - работает успешно	9
3.6	Проверка - работает успешно	10
3.7	Проверка - работает успешно	10
3.8	Написанная программа	11
3.9	Проверка - работает успешно	11

1 Цель работы

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

2 Теоретическое введение

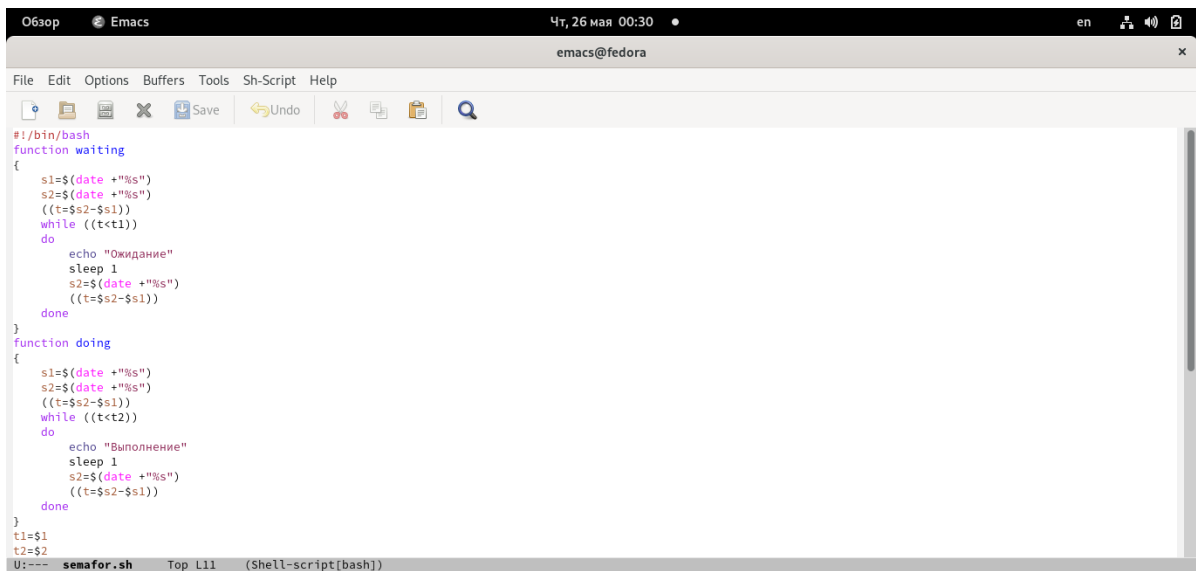
Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: - оболочка Борна (Bourne shell 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.

3 Выполнение лабораторной работы

##1

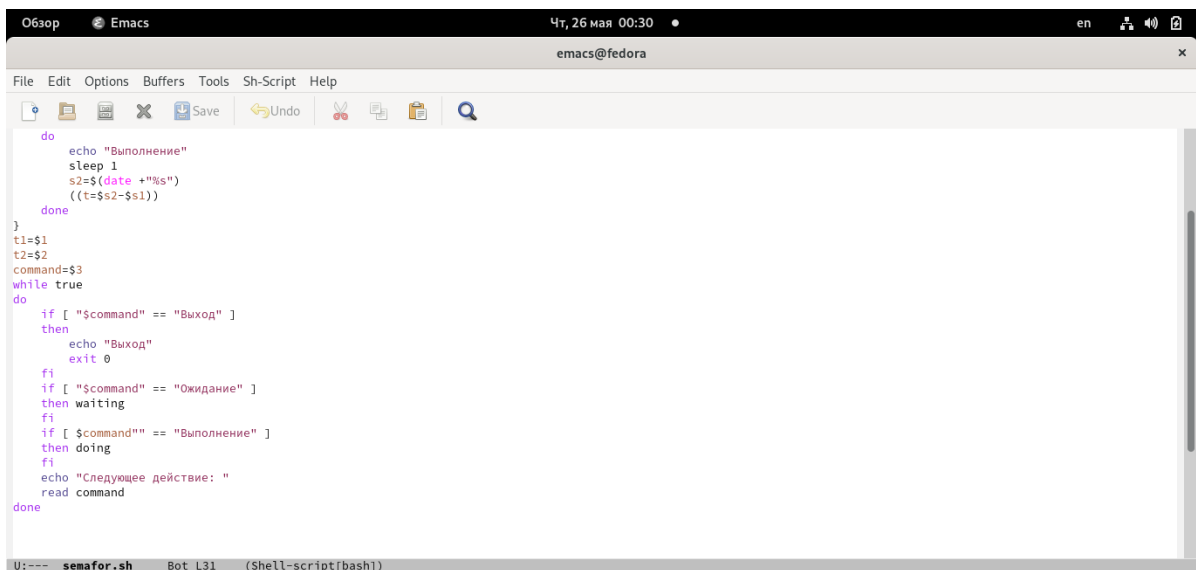
Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени t_1 будет дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, 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.

Для этого мы создаем файл `semafor.sh` и пишем программу (скриншоты 3.1, 3.2)



```
#!/bin/bash
function waiting
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=$s2-$s1))
    while ((t<t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s")
        ((t=$s2-$s1))
    done
}
function doing
{
    s1=$(date +%s")
    s2=$(date +%s")
    ((t=$s2-$s1))
    while ((t<t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s")
        ((t=$s2-$s1))
    done
}
t1=$1
t2=$2
U:--- semafor.sh Top L11 (Shell-script[bash])
```

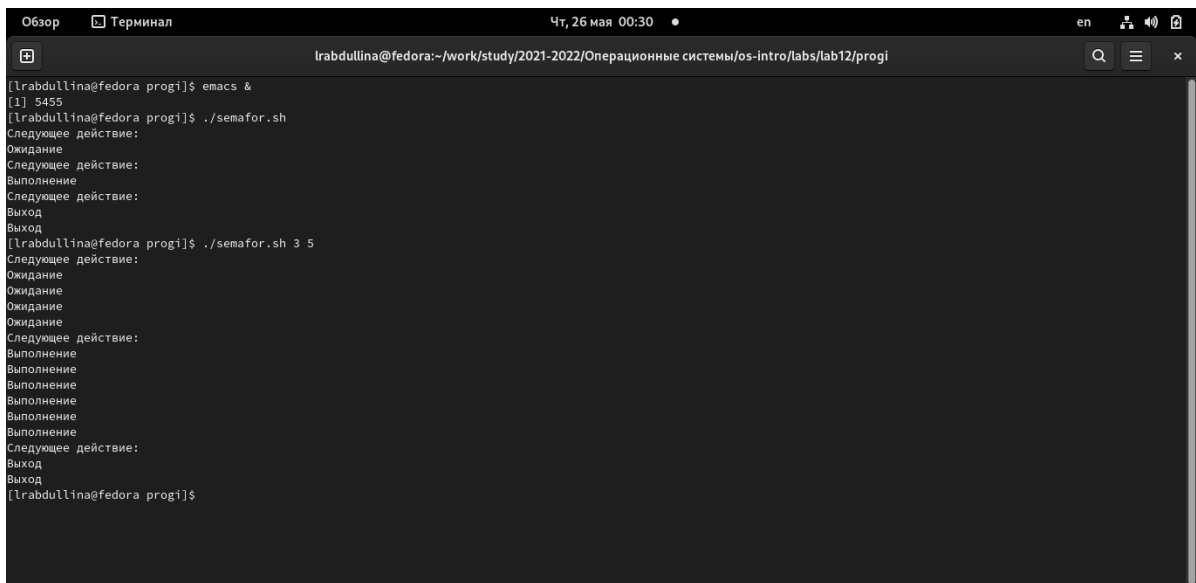
Скриншот 3.1: Написанная программа



```
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then
        waiting
    fi
    if [ "$command" == "Выполнение" ]
    then
        doing
    fi
    echo "Следующее действие: "
    read command
done
U:--- semafor.sh Bot L31 (Shell-script[bash])
```

Скриншот 3.2: Написанная программа

Не забудем сделать наш файл с программой исполняемым через команду `chmod +x semafor.sh`. После этого вводим команды в консоли `./semafor` и `./semafor 3 5` и смотрим на результат. (скриншот 3.3)



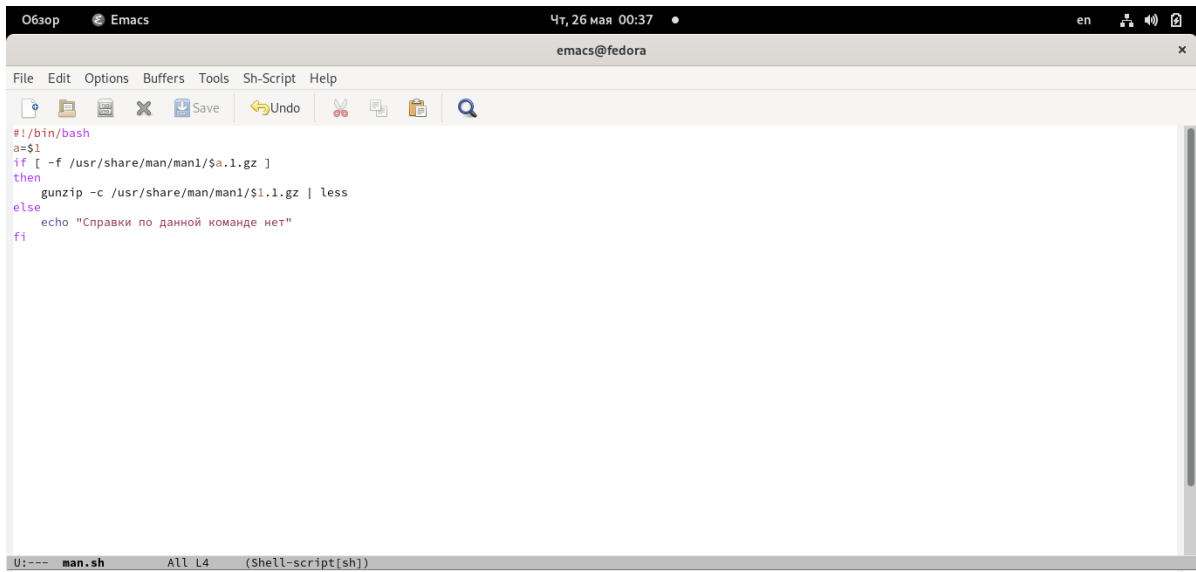
```
[lrabdullina@fedora prog1]$ emacs &
[1] 5455
[lrabdullina@fedora prog1]$ ./semafor.sh
Следующее действие:
Ожидание
Следующее действие:
Выполнение
Следующее действие:
Выход
Выход
[lrabdullina@fedora prog1]$ ./semafor.sh 3 5
Следующее действие:
Ожидание
Ожидание
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
Выход
[lrabdullina@fedora prog1]$
```

Скриншот 3.3: Проверка - работает успешно

3.1 2

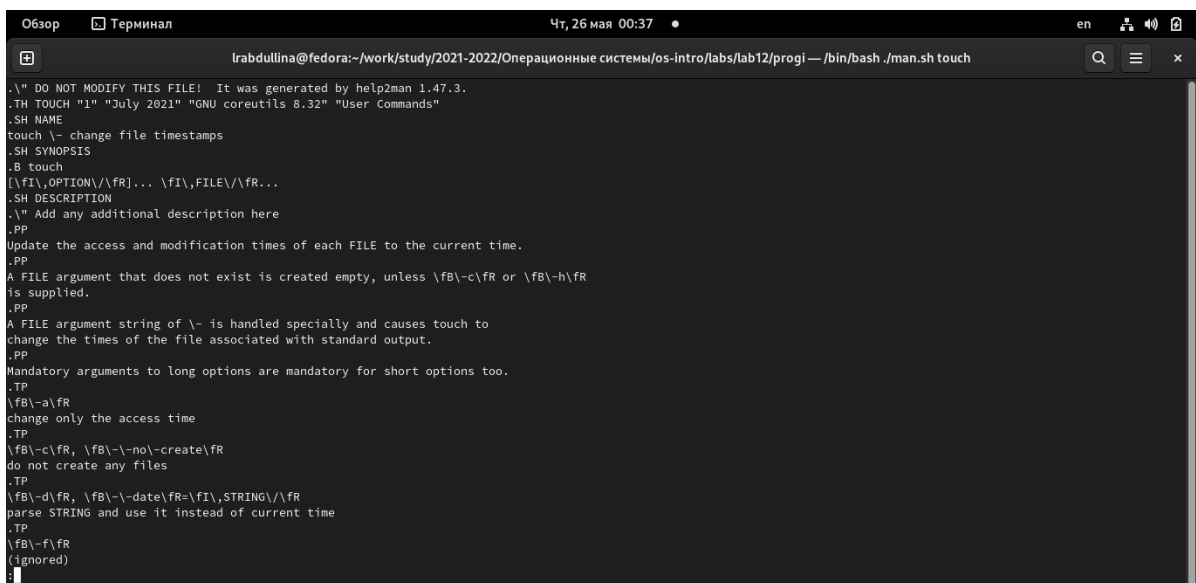
Реализуем команду `man` с помощью командного файла. Изучим содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. 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`

Для этого мы создаем файл `man.sh` и пишем код (скриншот 3.4)

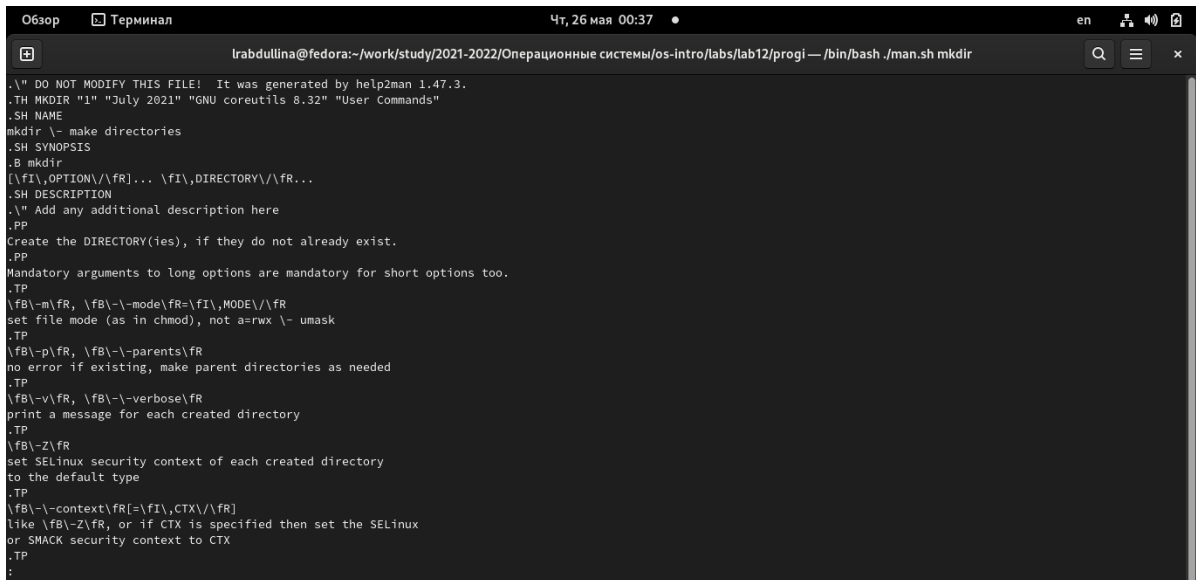


Скриншот 3.4: Написанная программа

Сделаем файл исполняемым через команду `chmod +x man.sh`. И проверим его работу. Пишем команды `./man.sh touch ./man.sh mkdir ./man.sh mksir` (скриншоты 3.5, 3.6, 3.7)

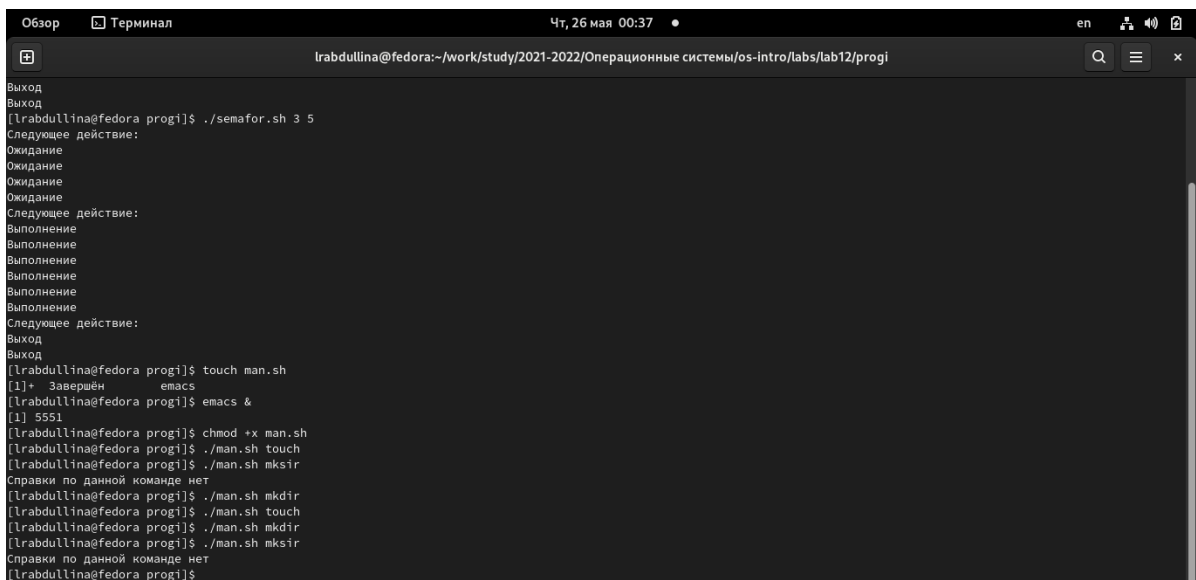


Скриншот 3.5: Проверка - работает успешно



```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab12/progi — /bin/bash ./man.sh mkdir
.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "July 2021" "GNU coreutils 8.32" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[ \fI\,OPTION\ \fR ]... \fI\,DIRECTORY\ \fR...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\-\m\fR, \fB\-\mode\fR=\fI\,MODE\ \fR
set file mode (as in chmod), not a=rwx \-\ umask
.TP
\fB\-\p\fR, \fB\-\parents\fR
no error if existing, make parent directories as needed
.TP
\fB\-\v\fR, \fB\-\verbose\fR
print a message for each created directory
.TP
\fB\-\Z\fR
set SELinux security context of each created directory
to the default type
.TP
\fB\-\context\fR=\fI\,CTX\ \fR
like \fB\-\Z\fR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
.
```

Скриншот 3.6: Проверка - работает успешно



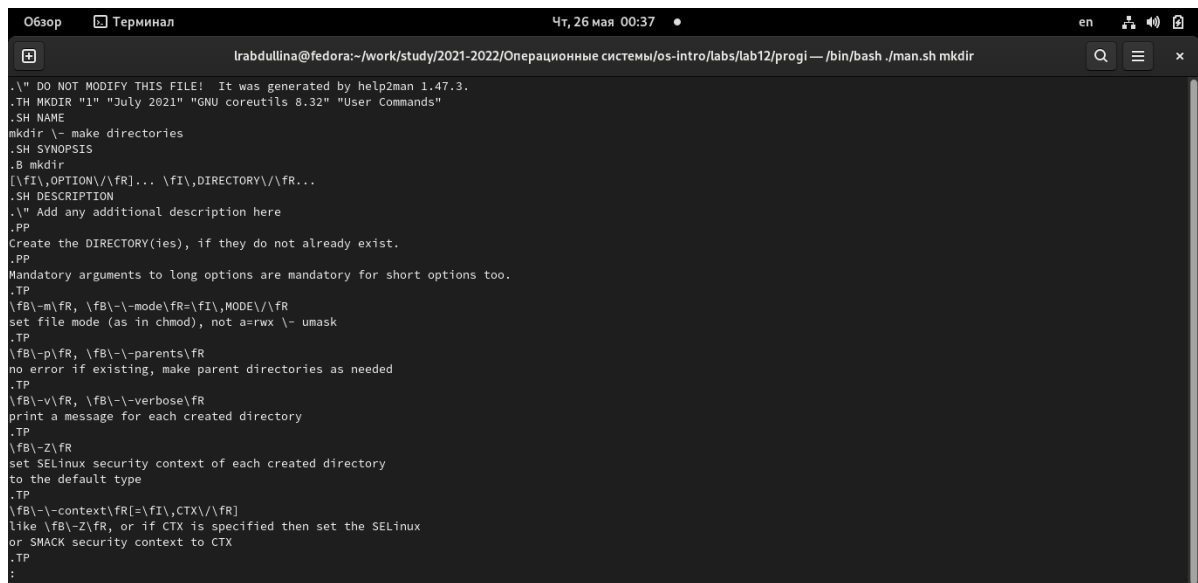
```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab12/progi
Выход
[lrabduullina@fedora progi]$ ./semaphor.sh 3 5
Следующее действие:
Ожидание
Ожидание
Ожидание
Ожидание
Следующее действие:
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
[lrabduullina@fedora progi]$ touch man.sh
[1]+  Завершен      emacs
[lrabduullina@fedora progi]$ emacs &
[1] 5551
[lrabduullina@fedora progi]$ chmod +x man.sh
[lrabduullina@fedora progi]$ ./man.sh touch
[lrabduullina@fedora progi]$ ./man.sh mksir
Справки по данной команде нет
[lrabduullina@fedora progi]$ ./man.sh mkdir
[lrabduullina@fedora progi]$ ./man.sh touch
[lrabduullina@fedora progi]$ ./man.sh mkdir
[lrabduullina@fedora progi]$ ./man.sh mksir
Справки по данной команде нет
[lrabduullina@fedora progi]$
```

Скриншот 3.7: Проверка - работает успешно

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

Для начала созданием новый файл random.sh и напишем программу. (скриншот

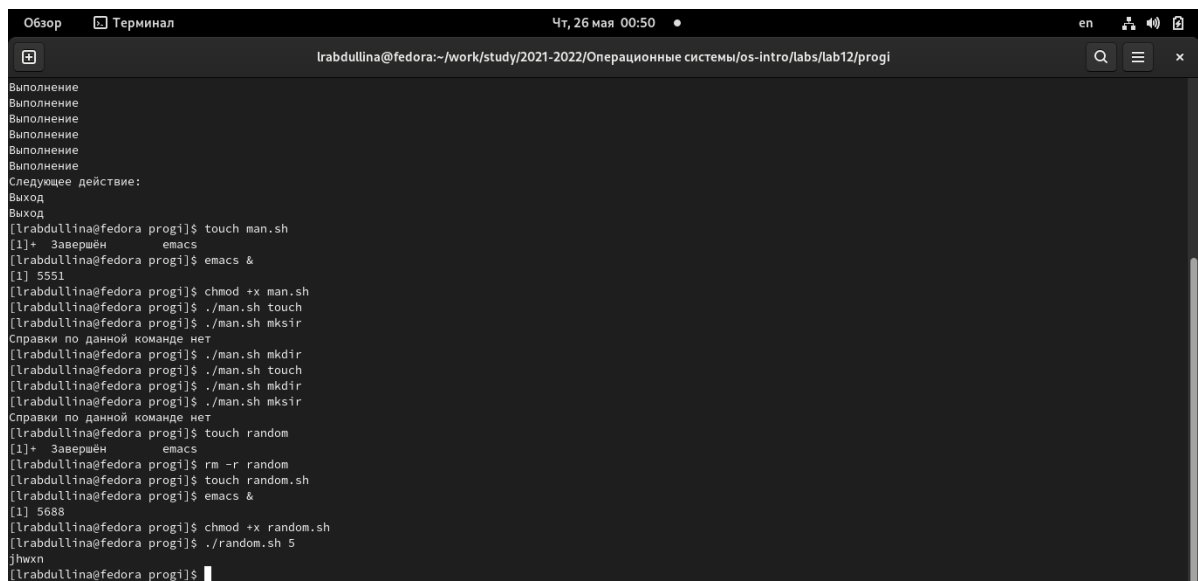
3.8)



```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab12/progi — /bin/bash ./man.sh mkdir
.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH MKDIR "1" "July 2021" "GNU coreutils 8.32" "User Commands"
.SH NAME
mkdir \- make directories
.SH SYNOPSIS
.B mkdir
[\fI\,OPTION\[\fR]... \fI\,DIRECTORY\[\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
Create the DIRECTORY(ies), if they do not already exist.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\-\m\[\fR, \fB\-\mode\[\fR=\fI\,MODE\[\fR]
set file mode (as in chmod), not a=rwx \-\ umask
.TP
\fB\-\p\[\fR, \fB\-\parents\[\fR
no error if existing, make parent directories as needed
.TP
\fB\-\v\[\fR, \fB\-\verbose\[\fR
print a message for each created directory
.TP
\fB\-\Z\[\fR
set SELinux security context of each created directory
to the default type
.TP
\fB\-\context\[\fR[=\fI\,CTX\[\fR]
like \fB\-\Z\[\fR, or if CTX is specified then set the SELinux
or SMACK security context to CTX
.TP
:
```

Скриншот 3.8: Написанная программа

Сделаем файл исполняемым через команду `chmod +x random.sh`. И проверим его работу. Напишем случайное число и посмотрим на результат. (скриншот 3.9)



```
lrabduullina@fedora:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab12/progi
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие:
Выход
Выход
[lrabduullina@fedora progi]$ touch man.sh
[1]+  Завершён      emacs
[lrabduullina@fedora progi]$ emacs &
[1] 5551
[lrabduullina@fedora progi]$ chmod +x man.sh
[lrabduullina@fedora progi]$ ./man.sh touch
[lrabduullina@fedora progi]$ ./man.sh mkdir
Справки по данной команде нет
[lrabduullina@fedora progi]$ ./man.sh mkdir
[lrabduullina@fedora progi]$ ./man.sh touch
[lrabduullina@fedora progi]$ ./man.sh mkdir
[lrabduullina@fedora progi]$ ./man.sh mkdir
Справки по данной команде нет
[lrabduullina@fedora progi]$ touch random
[1]+  Завершён      emacs
[lrabduullina@fedora progi]$ rm -r random
[lrabduullina@fedora progi]$ touch random.sh
[lrabduullina@fedora progi]$ emacs &
[1] 5688
[lrabduullina@fedora progi]$ chmod +x random.sh
[lrabduullina@fedora progi]$ ./random.sh 5
jhwxn
[lrabduullina@fedora progi]$
```

Скриншот 3.9: Проверка - работает успешно

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:

- `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="$VAR2" echo "$VAR3"` Результат: Hello, World
- Второй: `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. Скажите 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

- `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 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

5 Выводы

В ходе лабораторной работы мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 Список литературы

<https://esystem.rudn.ru/course/view.php?id=5790> :::