**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**ОТЧЕТ**

**по лабораторной работе № 13**

*дисциплина: Операционные системы*

*тема: Программирование в командном процессоре ОС UNIX. Расширенное программирование*

Подготовила: Голощапова И.Б.

Группа: НФИбд\_01-20

# Цель работы

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

# Библиография

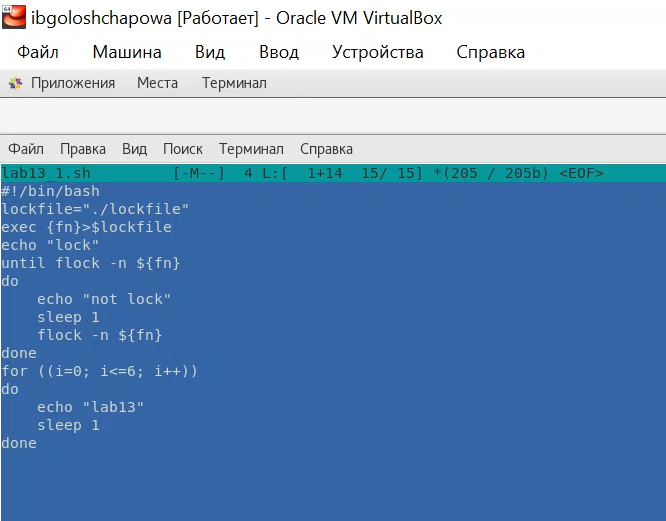
[Командные файлы в Linux](https://sgww.livejournal.com/8836.html)

[Википедия - Командная оболочка](https://ru.wikipedia.org/wiki/Командная_оболочка_Unix)

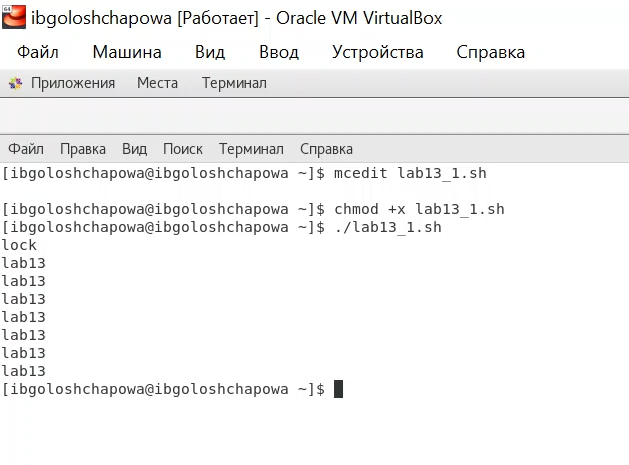
[Программные процессоры](https://infopedia.su/24x10498.html)

# Выполнение лабораторной работы

1. Написала командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустила 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. Доработала программу так, чтобы  
   имелась возможность взаимодействия трёх и более процессов.

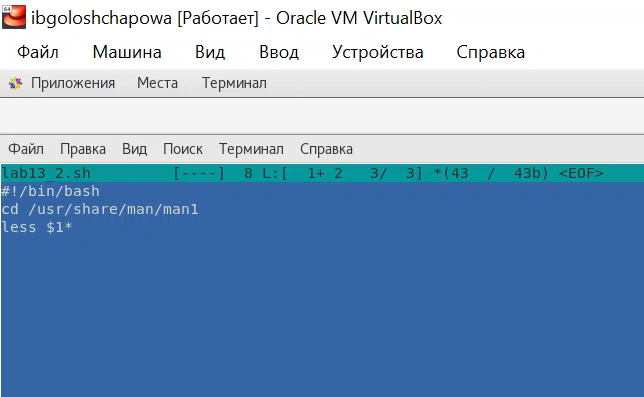


*Рис.1 "Код программы (1)"*

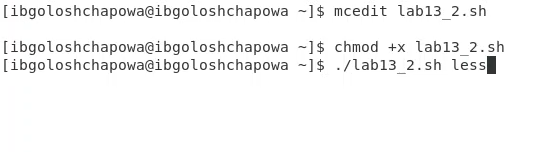


*Рис.2 "Запуск программы (1)"*

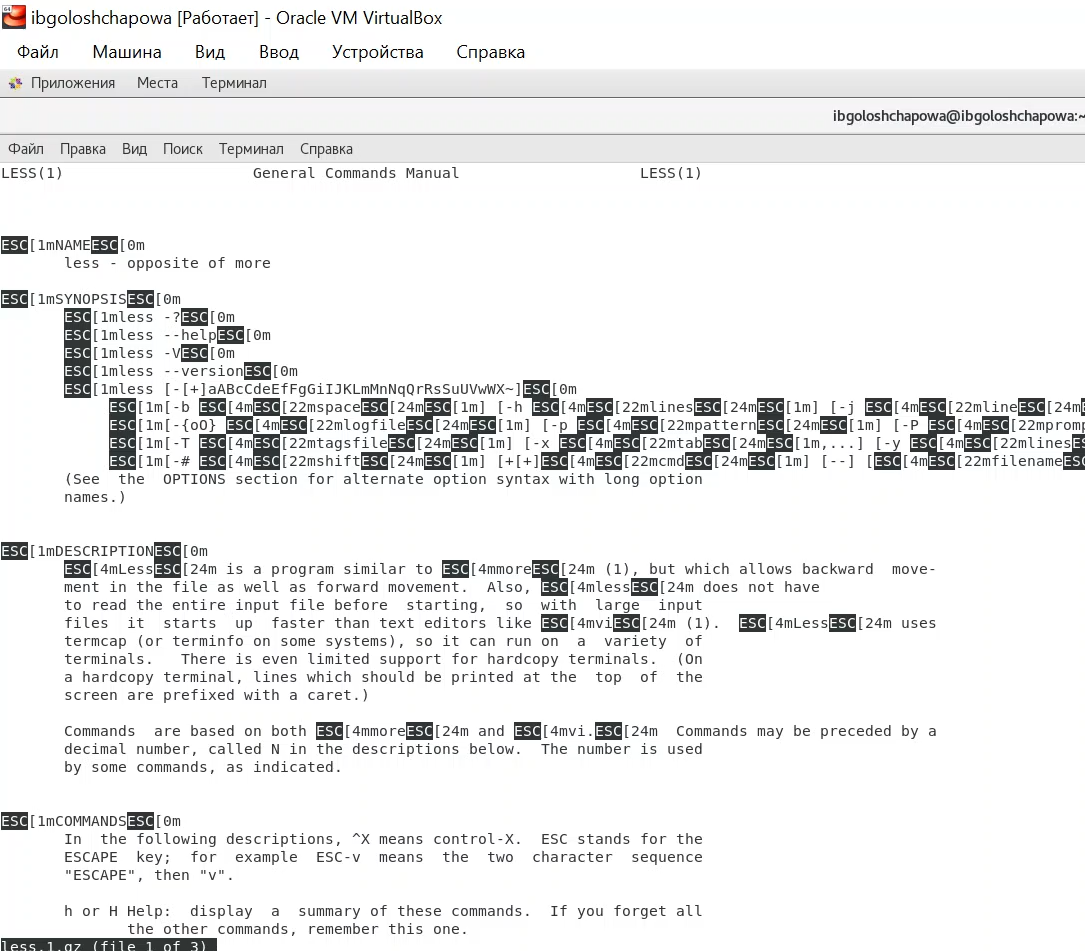
1. Реализовала команду man с помощью командного файла. Изучила содержимое  
   каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов,  
   содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое 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.3 "Код программы (2)"*

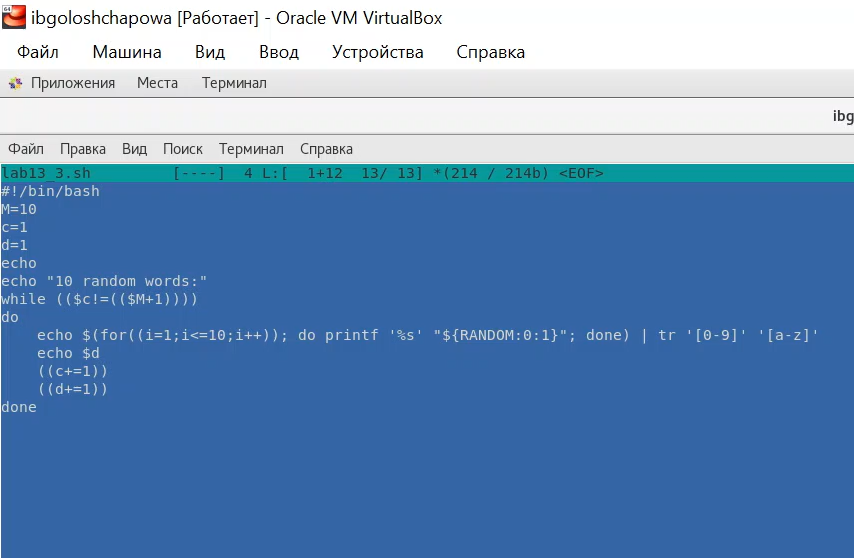


*Рис.4 "Запуск программы (2)"*

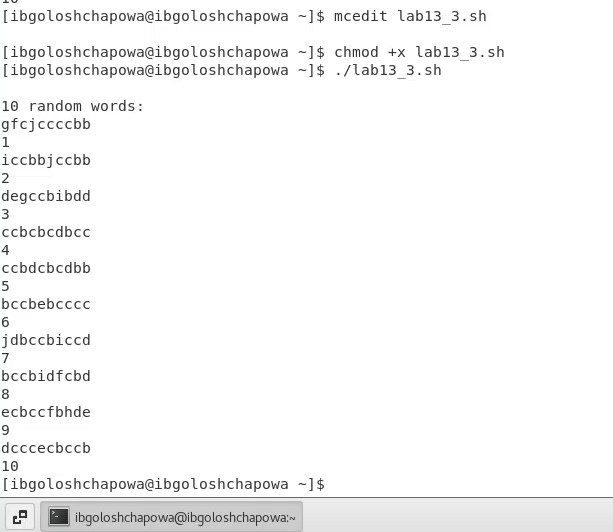


*Рис.5 "Работа программы (2)"*

1. Используя встроенную переменную $RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. Учла,  
   что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.



*Рис.6 "Код программы (3)"*



*Рис.5 "Работа программы (3)"*

# Выводы

В ходе лабораторной работы я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Контрольные вопросы

1.В строке while [$1 != "exit"] квадратные скобки нужно заменить на круглые.

2.Есть несколько видов конкатенации строк. Например,

VAR1="Hello,"

VAR2=" World"

VAR3="\(VAR1\)VAR2"

echo "$VAR3"

3.Команда seq выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. В bash можно использовать seq с циклом for, используя подстановку команд. Например,

$ for i in $(seq 1 0.5 4)

do

echo "The number is $i"

done

1. Результатом вычисления выражения $((10/3)) будет число 3.
2. Список того, что можно получить, используя Z Shell вместо Bash:

Встроенная команда zmv поможет массово переименовать файлы/директории, например, чтобы добавить ‘.txt’ к имени каждого файла, запустите zmv –C '(\*)(#q.)' '$1.txt'.

Утилита zcalc — это замечательный калькулятор командной строки, удобный способ считать быстро, не покидая терминал.

Команда zparseopts — это однострочник, который поможет разобрать сложные варианты, которые предоставляются скрипту.

Команда autopushd позволяет делать popd после того, как с помощью cd, чтобы вернуться в предыдущую директорию.

Поддержка чисел с плавающей точкой (коей Bash не содержит).

Поддержка для структур данных «хэш».

Есть также ряд особенностей, которые присутствуют только в Bash:

Опция командной строки –norc, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл .bashrc

Использование опции –rcfile с bash позволяет исполнять команды из определённого файла.

Отличные возможности вызова (набор опций для командной строки)

Может быть вызвана командой sh

Bash можно запустить в определённом режиме POSIX. Примените set –o posix, чтобы включить режим, или ––posix при запуске.

Можно управлять видом командной строки в Bash. Настройка переменной PROMPT\_COMMAND с одним или более специальными символами настроит её за вас.

Bash также можно включить в режиме ограниченной оболочки (с rbash или --restricted), это означает, что некоторые команды/действия больше не будут доступны:

Настройка и удаление значений служебных переменных SHELL, PATH, ENV, BASH\_ENV

Перенаправление вывода с использованием операторов ‘>’, ‘>|’, ‘<>’, ‘>&’, ‘&>’, ‘>>’

Разбор значений SHELLOPTS из окружения оболочки при запуске

Использование встроенного оператора exec, чтобы заменить оболочку другой командой

1. Синтаксис конструкции for ((a=1; a <= LIMIT; a++)) верен.
2. Язык bash и другие языки программирования:

-Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на си/си++, скомпилированных с максимальной оптимизацией;

-Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Ява-машина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам;

-Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости

исполнения программ;

-Скорость кодов, генерируемых компилятором языка си фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;

-Скорость ассемблерных кодов x86-64 может меньше, чем аналогичных кодов x86, примерно на 10%;

-Оптимизация кодов лучше работает на процессоре Intel;

-Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, аук (gawk, mawk) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32-разрядных кодах;

-Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (gcc, icc, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром;

-В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета ack(5,2,3)