

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

дисциплина: Операционные системы

Студент:

Афтаева Ксения Васильевна

Преподаватель:

Велиева Т.В.

Группа: НПИбд-01-20

МОСКВА 2021 г.

Цель работы:

Изучить основы программирования в оболочке **OS UNIX**. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

Задачи:

1. Изучить теоретический материал
2. 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, в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
3. Реализовать команду `man` с помощью командного файла. Изучить 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**.

4. Используя встроенную переменную "\$RANDOM", написать командный файл, генерирующий случайную последовательность букв латинского алфавита.
5. Ответить на контрольные вопросы.

Объект и предмет исследования:

Программирование в оболочке **ОС UNIX/Linux**

Техническое оснащение:

Ноутбук, на котором установлена виртуальная машина с Linux

Теоретические вводные данные [1] :

bash (сокр. от «*Bourne-Again shell*») — это командная оболочка (или «интерпретатор командной строки»), используемая по умолчанию в операционных системах на базе Unix и Linux, созданная в 1989 году Брайаном Фоксом с целью усовершенствования командной оболочки sh.

bash позволяет автоматизировать различные задачи, устанавливать программное обеспечение, настраивать конфигурации для своего рабочего окружения и многое другое.

Циклы BASH:

Циклы позволяют выполнять один и тот же участок кода необходимое количество раз. В большинстве языков программирования существует несколько типов циклов. Большинство из них поддерживаются оболочкой Bash.

- **for** - позволяет перебрать все элементы из массива или использует переменную-счетчик для определения количества повторений;
- **while** - цикл выполняется пока условие истинно;
- **until** - цикл выполняется пока условие ложно.

Bash позволяет использовать циклы как в скриптах, так и непосредственно в командной оболочке.

Другие команды:

flock — утилита, которая позволяет использовать лок-файл для предотвращения запуска копии процесса (вашего скрипта, крона или чего-то еще).

sleep – одна из самых простых команд. Как видно из названия, его единственная функция – спать. Другими словами, он вводит задержку на указанное время.

\$RANDOM - возвращает псевдослучайные целые числа в диапазоне 0 - 32767.

Условные обозначения и символы:

- `$*` — отображается вся командная строка или параметры оболочки;
 - `$?` — код завершения последней выполненной команды;
 - `$$` — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
 - `!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
 - `$-` — значение флагов командного процессора;
 - `${#*}` — возвращает целое число — количество слов, которые были результатом `$*`;
 - `${#name}` — возвращает целое значение длины строки в переменной `name`;
 - `${name[n]}` — обращение к `n`-му элементу массива;
 - `${name[*]}` — перечисляет все элементы массива, разделённые пробелом;
 - `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных;
 - `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
 - `${name:value}` — проверяется факт существования переменной;
 - `${name=value}` — если `name` не определено, то ему присваивается значение `value`;
 - `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
 - `name + value` — это выражение работает противоположно `{name-value}`. Если переменная определена, то подставляется `value`;
 - `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
 - `${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.
-

Выполнение работы:

Задание 1

1. Создала командный файл и открыла его на редактирование одной командой `emacs lab13z1.sh`, где **emacs** - редактор, в котором я буду работать, а **lab13z1.sh**

- название файла. В данном файле написала программу, выполняющую требуемые функции (Рис.1)

```
[kvaftaeva@kvaftaeva ~]$ emacs lab13z1.sh
lab13z1.sh - emacs@kvaftaeva.localdomain

File Edit Options Buffers Tools Sh-Script Help

[Icons: File, Folder, Print, Close, Save, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
exec {fn}>./lock.file
while test -f ./lock.file
do
if flock -n ${fn}
then
echo "File was locked"
sleep 5
flock -u ${fn}
echo "File was unlocked"
sleep 1
else
echo "The file could not be locked"
sleep 5
fi
done
```

Рис.1 Программа к заданию 1

Структура программы: в первой строке вызываем интерпритатор (в нашем случае **bash**). Далее мы присваиваем файлу номер, так как **flock** работает только с дескрипторами. Затем идем циклом, условием которого является **test -f ./lock.file**. Данное условие будет истинно, если файл существует и является именно файлом. Затем условием **if flock -n \${fn}** мы проверяем, заблокирован ли файл (уже запущен). Если нет, то с ним можно работать. Тогда мы выводим информацию о том, что файл заблокирован, ждем 5 секунд (строка **sleep 5**), выводим информацию о том, что он разблокирован, ждем секунду. Иначе выводим сообщение, что не можем с ним работать.

После написания программы сохранила ее комбинацией клавиш **С-х С-с** и закрыла текстовый редактор.

2. Таким образом, при запуске на нескольких терминалах данного файла (команда **./lab13z1.sh**), терминалы будут "конфликтовать". Т.е если в одном терминале выводится "File was locked", в других будет выводиться строка "The file could not be locked". Если же выводится строка о разблокировке, то другие терминалы могут работать с файлом. Я открыла 3 терминала и запустила в них файл (Рис.2)

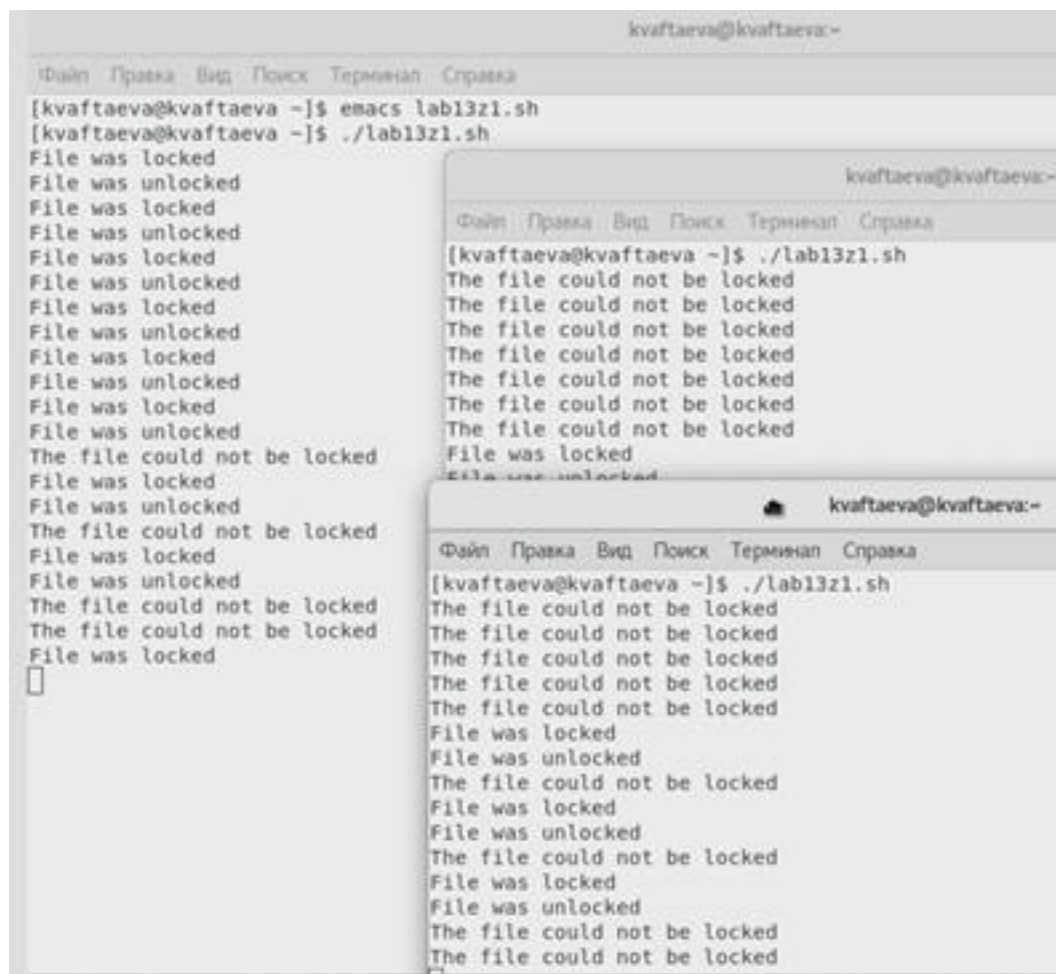
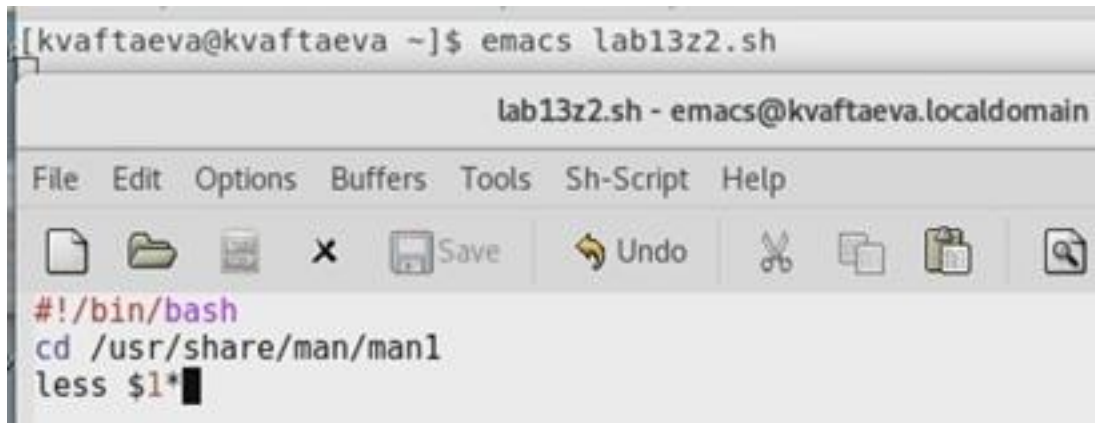


Рис.2 Работа файла задание 1

Видим, что действительно есть "конфликт".

Задание 2

1. Создала командный файл и открыла его на редактирование одной командой `emacs lab13z2.sh`, где **emacs** - редактор, в котором я буду работать, а **lab13z2.sh** - название файла. В данном файле написала программу, выполняющую требуемые функции (Рис.3)



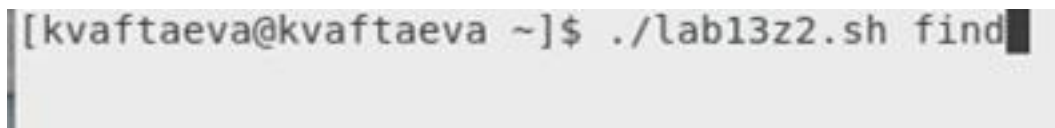
```
[kvaftaeva@kvaftaeva ~]$ emacs lab13z2.sh  
lab13z2.sh - emacs@kvaftaeva.localdomain  
File Edit Options Buffers Tools Sh-Script Help  
Save Undo  
#!/bin/bash  
cd /usr/share/man/man1  
less $1
```

Рис.3 Программа к заданию 2

Структура программы: она состоит всего из 3 строк. 1 строка - вызов интерпретатора (в нашем случае **bash**). 2 строка - переход в каталог **/usr/share/man/man1**, где находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд, которые мы будем просматривать. Переход был осуществлен командой **cd**. Командой **less** просмотрим содержимое справки, используя указатель на файл.

После написания программы сохранила ее комбинацией клавиш **C-x C-s** и закрыла текстовый редактор.

2. Даю право на выполнение этого файла, командой **chmod +x lab13z2.sh**. Запускаю его, введя **./lab13z2.sh**, указав после него команду, справку о которой хочу увидеть - **find** (Рис.4)



```
[kvaftaeva@kvaftaeva ~]$ ./lab13z2.sh find
```

Рис.4 Запуск программы 2

Видим, что у нас действительно вывелась справка о данной команде (Рис.5)

```
kvaftaeva@kvaftaeva:~  
Файл Правка Вид Поиск Терминал Справка  
FIND(1) General Commands Manual FIND(1)  
  
NAME  
find - search for files in a directory hierarchy  
  
SYNOPSIS  
find [-H] [-L] [-P] [-D debugopts] [-O level] [path...] [expression]  
  
DESCRIPTION  
This manual page documents the GNU version of find. GNU  
find searches the directory tree rooted at each given file name by evaluating the  
given expression from left to right, according to the rules of precedence  
(see section OPERATORS), until the outcome is known (the left hand side is  
false for & and operations, true for &or operations), at which point  
find moves on to the next file name.  
  
If you are using find in an environment where security is important  
find.1.gz (file 1 of 4)
```

Рис.5 Справка о команде `find`

Задание 3

1. Создала командный файл и открыла его на редактирование одной командой `emacs lab13z3.sh`, где **emacs** - редактор, в котором я буду работать, а **lab13z3.sh** - название файла. В данном файле написала программу, выполняющую требуемые функции (Рис.6)

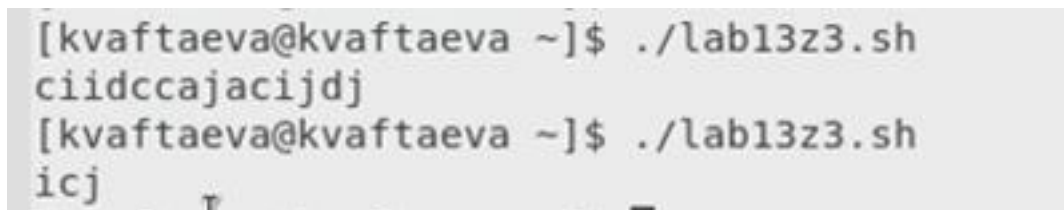
```
[kvaftaeva@kvaftaeva ~]$ emacs lab13z3.sh  
lab13z3.sh - emacs@kvaftaeva.localdomain  
File Edit Options Buffers Tools Sh-Script Help  
Save Undo  
#!/bin/bash  
n=$((1+$RANDOM%3))  
for ((i=1; i<n;i++))  
do  
echo -n[$RANDOM |tr '[0-9]' '[a-z]'  
done  
echo $RANDOM |tr '[0-9]' '[a-z]'
```


Рис.6 Программа к заданию 2

Структура программы: в первой строке вызов интерпритатора (в нашем случае **bash**). Далее можно было обойтись последней строкой. Однако в таком случае, почти всегда были бы 5 значные строки (так как диапазон рандома до 32767 и пятизначных чисел значительно больше). Поэтому во второй строке я ввожу переменную **n**, куда помещаю случайное число от 1 до 3. Далее я иду циклом от 1 до **n**. В каждой итерации я создаю случайное число (**\$RANDOM**), заменяю все цифры в нем на буквы (****tr '[0-9]' '[a-z]'**) и вывожу на экран с помощью команды **echo** и опции **-n**, которая позволяет убрать переход на другую строку (склеивает их). Однако количество итераций не совпадает с количеством раз, которые изначально были запланированы. Это связано с тем, что в последнем выводе символов нужно убрать опцию, склеивающую строки. Поэтому в последней строке я делаю последний вывод без опции.

После написания программы сохранила ее комбинацией клавиш **C-x C-s** и закрыла текстовый редактор.

2. Запускаю программу несколько раз, вводя **./lab13z3.sh** (Рис.7)



```
[kvaftaeva@kvaftaeva ~]$ ./lab13z3.sh
ciidccajacijdj
[kvaftaeva@kvaftaeva ~]$ ./lab13z3.sh
icj
```

Рис.7 Результат программы 3

Видим, что выводятся строки с случайным набором букв и разной длиной.

Контрольные вопросы:

1. Нужно взять в кавычки «\$1» [2]
2. Написать переменные одну за другой. Например: **A = "BC"** . Либо с помощью оператора **+=**. Например: **B += C** [3]
3. Эта утилита выводит последовательность целых чисел с заданным шагом. Также можно реализовать с помощью утилиты **jot**. [4]
4. 3
5. В **zsh** можно настроить отдельные сочетания клавиш так, как вам нравится. Использование истории команд в **zsh** ничем особенным не отличается от **bash**. **Zsh** очень удобен для повседневной работы и делает добрую половину рутинных за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в **zsh** после **for** обязательно вставлять пробел,

нумерация массивов в zsh начинается с 1, чего совершенно невозможно понять. Так, если вы используете shell для повседневной работы, исключающей написание скриптов, используйте zsh. Если вам часто приходится писать свои скрипты, только bash! Впрочем, можно комбинировать. Как установить zsh в качестве оболочки по умолчанию для отдельного пользователя: [о](#)

6. Синтаксис верен. [\[2\]](#)

7. Преимущества:

- По сравнению с cmd у bash больше возможностей.
- По сравнению с нескриптовыми языками программирования у него более низкий порог вхождения.
- Его не нужно отдельно устанавливать, он встроен в операционную систему.

Недостатки:

- В интернете меньше дополнительной информации про него, чем про языки программирования.
- Сложнее отлаживать программу.

Заключение:

Теоретический материал изучен и пригодится в дальнейшей работе. Все цели и задачи выполнены.

Вывод:

Я изучила основы программирования в оболочке **ОС UNIX**. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

Библиографический список:

[1]: [Программирование в bash](#)

[2]: [Описание лабораторной работы №11](#)

[3]: [Команды bash](#)

[4]: [Команды jot, seq](#)