

Отчёт по лабораторной работе №12

Программирование в командном процессоре ОС UNIX. Расширенное
программирование.

Самигуллин Эмиль Артурович

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.

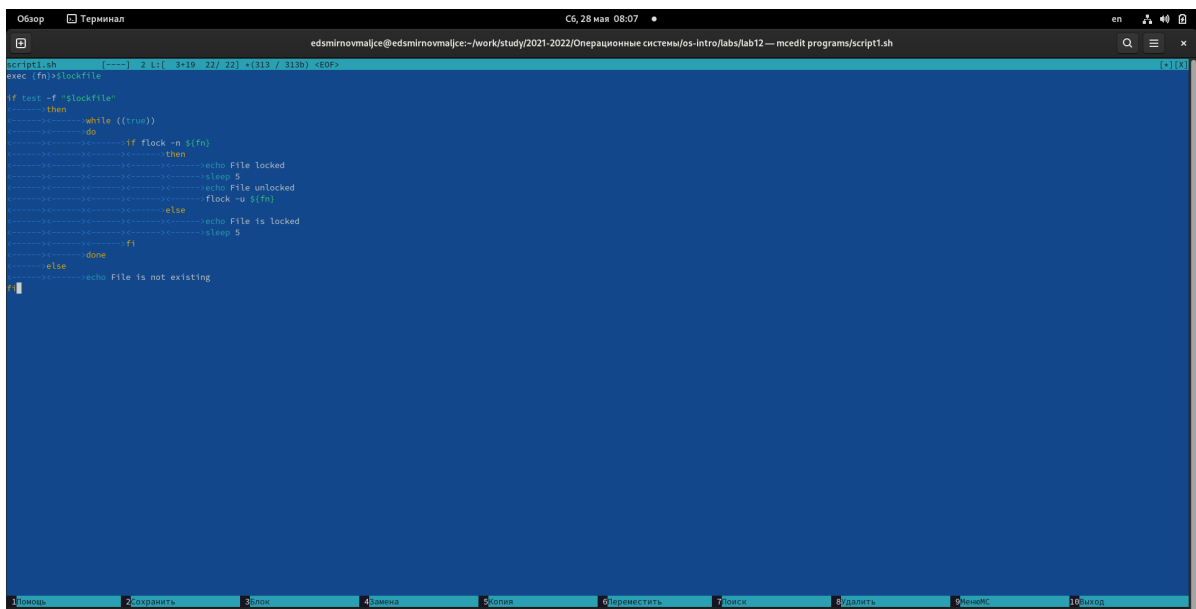
3 Теоретическое введение

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 оболочкой 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.

4 Выполнение лабораторной работы

1. Написал 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). Запустил 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. Доработал программу так, чтобы имелась возможность взаимодействия трёх и более процессов.



```
script1.sh [-----] 0 L1 3x19 22/ 23 1313 / 3136 <EOF>
exec (fn)>lockfile

if test -f "lockfile"
then
while ((true))
do
if flock -n $(fn)
then
echo File locked
sleep 5
echo File unlocked
flock -u $(fn)
else
echo File is locked
sleep 5
done
else
echo File is not existing
fi
```

рис. 1: текст первого скрипта.

2. Реализовал команду `man` с помощью командного файла. Изучил 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`.



рис. 2: выполнение второго скрипта.

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

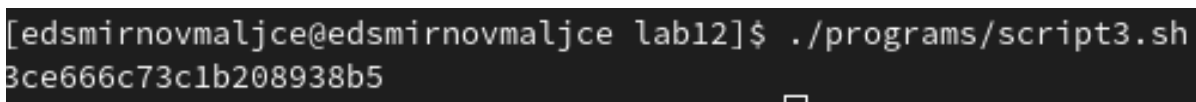


рис. 3: выполнение третьего скрипта.

5 Ответы на контрольные вопросы

1. В строке

```
while [ $1 != "exit" ]
```

пропущены пробелы рядом с квадратными скобками.

2. Две строки можно объединить записав их в третью переменную:

```
var3="$var1$var2"
```

3. Утилита `seq` выводит последовательность целых чисел с шагом, заданным пользователем. Ее функционал можно реализовать простейшим циклом, который прибавляет число, введенное пользователем и выводит его.

4. Выражение $\$(10/3)$ выдает число 3.

5. Основные отличия `zsh` от `bash`:

- `Zsh` более интерактивный и настраиваемый, чем `Bash`.
- У `Zsh` есть поддержка с плавающей точкой, которой нет у `Bash`.
- В `Zsh` поддерживаются структуры хеш-данных, которых нет в `Bash`.
- Функции вызова в `Bash` лучше по сравнению с `Zsh`.
- Внешний вид подсказки можно контролировать в `Bash`, тогда как `Zsh` настраивается.
- Конфигурационными файлами являются `.bashrc` в интерактивных оболочках без регистрации и `.profile` или `.bash_profile` в оболочках входа в `Bash`. В `Zsh` оболочками, не входящими в систему, являются `.zshrc`, а оболочками для входа - `.zprofile`.

- Массивы Zsh индексируются от 1 до длины, тогда как Bash индексируется от -1 до длины.
- В Zsh, если шаблоны не совпадают ни с одним файлом, выдается ошибка. Находясь в Баше, он остается без изменений.
- Правая часть конвейера запускается как родительская оболочка в Zsh, в то время как в Bash она запускается как подоболочка.
- В Zsh функция `zmv` используется для массового переименования, тогда как в Bash мы должны использовать функцию расширения параметров.
- Bash имеет хорошие возможности написания сценариев в одной строке, в то время как в Zsh мы не смогли найти то же самое.
- По умолчанию выходные данные хранятся во временном файле в Zsh, а в Bash - нет.
- Многие встроенные функции в Bash упрощают сложные программы, тогда как в Zsh встроенных функций для сложных программ меньше.
- Zsh эффективно управляет своими файлами, в то время как Bash плохо умеет работать с файлами.

6. Синтаксис следующей строчки верен.

```
for ((a=1; a <= LIMIT; a++))
```

7. bash позволяет напрямую обращаться к командам, одно такие языки, как Python значительно проще.

6 Выводы

- Я научился писать командные файлы с использованием управляющих конструкций и циклов.