

Отчет по лабораторной работе №11

Операционные системы

Шихалиева Зурият Арсеновна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	14
5	Ответы на контрольные вопросы	15

Список иллюстраций

3.1	Создание и исполнение файла	8
3.2	Доработанный код программы	9
3.3	Изучение содержимого папки	10
3.4	Код программы	11
3.5	Исполнение программы	11
3.6	Результат работы программы	12
3.7	Создание и исполнение файла	12
3.8	Код программы	12

Список таблиц

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

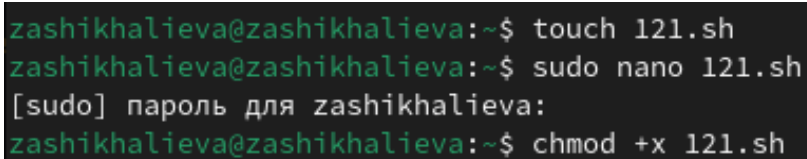
2 Задание

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

Создаю командный файл для первой программы, пишу ее, проверяю ее работу (рис. fig. 3.1).



```
zashikhalieva@zashikhalieva:~$ touch 121.sh
zashikhalieva@zashikhalieva:~$ sudo nano 121.sh
[sudo] пароль для zashikhalieva:
zashikhalieva@zashikhalieva:~$ chmod +x 121.sh
```

Рис. 3.1: Создание и исполнение файла

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 (рис. fig. 3.2).


```
#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
if flock -n ${fn}
then
    echo "File is blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done
```

Рис. 3.2: Доработанный код программы

```
#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
```

```

if flock -n ${fn}
then
    echo "File is blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done

```

Чтобы реализовать команду `man` с помощью командного файла, изучаю 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 (рис. fig. 3.3).

```

zashikhalieva@zashikhalieva:~$ ls /usr/share/man/man1
.:1.gz
'[:1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
nbdkit-sparse-random-plugin.1.gz
nbdkit-split-plugin.1.gz
nbdkit-ssh-plugin.1.gz
nbdkit-swab-filter.1.gz
nbdkit-tls.1.gz
nbdkit-tls-fallback-filter.1.gz
nbdkit-truncate-filter.1.gz
nbdkit-zero-plugin.1.gz
nc.1.gz
ncat.1.gz
ncursesw6-config.1.gz
ndctl.1.gz
ndctl-activate-firmware.1.gz
ndctl-check-labels.1.gz
ndctl-check-namespace.1.gz

```

Рис. 3.3: Изучение содержимого папки

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. 3.4).

```
#!/bin/bash

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "There is no such command"
fi
```

Рис. 3.4: Код программы

```
#!/bin/bash

a=$1
if test -f "/usr/share/man/man1/$a.1.gz"
then less /usr/share/man/man1/$a.1.gz
else
echo "There is no such command"
fi
```

Проверяю работу командного файла (рис. fig. 3.5).

```
zashikhalieva@zashikhalieva:~$ ./12.sh rmdir
```

Рис. 3.5: Исполнение программы

Командный файл работает так же, как и команда `man`, открывает справку по указанной утилите (рис. fig. 3.6).

```
ESC[4mRMDIRESC[24m(1)
ESC[4mRMDIRESC[24m(1)
ESC[1mNAMEESC[0m
rmmdir - remove empty directories
ESC[1mSYNOPSISESC[0m
ESC[1mrmmdir ESC[22mESC[4mOPTIONESC[24m... ESC[4mDIRECTORYESC[24m...
ESC[1mDESCRIPTIONESC[0m
Remove the DIRECTORY(ies), if they are empty.
ESC[1m--ignore-fail-on-non-emptyESC[0m
ignore each failure to remove a non-empty directory
ESC[1m-pESC[22m, ESC[1m--parentsESC[0m
remove DIRECTORY and its ancestors; e.g., 'rmmdir ESC[1m-p ESC[22ma/b' is similar to 'rmmdir a/b a'
ESC[1m-vESC[22m, ESC[1m--verboseESC[0m
output a diagnostic for every directory processed
ESC[1m--help ESC[22mdisplay this help and exit
ESC[1m--versionESC[0m
output version information and exit
```

Рис. 3.6: Результат работы программы

Создаю файл для кода третьей программы, пишу программу и проверяю ее работу (рис. fig. 3.7).

```
zashikhalieva@zashikhalieva:~$ nano 111.sh
```

Рис. 3.7: Создание и исполнение файла

Используя встроенную переменную \$RANDOM, пишу командный файл, генерирующий случайную последовательность букв латинского алфавита. Т.к. \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767, ввожу ограничения так, чтобы была генерация чисел от 1 до 26 (рис. fig. 3.8).

```
#!/bin/bash
a=$1
if [[ -z "$a" ]]; then
    echo "Usage: $0 <number>"
    exit 1
fi
for ((i=0; i<a; i++))
do
    char=$((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
```

Рис. 3.8: Код программы

GNU nano 7.2

111.sh

```
#!/bin/bash
```

```
a=$1
```

```
if [[ -z "$a" ]]; then
```

```
    echo "Usage: $0 <number>"
```

```
    exit 1
```

```
fi
```

```
for ((i=0; i<a; 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;;
```

```
        7) echo -n g;; 8) echo -n h;; 9) echo -n i;; 10) echo -n j;; 11) echo -n k;
```

```
        13) echo -n m;; 14) echo -n n;; 15) echo -n o;; 16) echo -n p;; 17) echo -n q;
```

```
        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
```

4 Выводы

При выполнении данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на контрольные вопросы

1. `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 ["$1" != "exit"]`
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый: `VAR1="Hello," VAR2=" World"`
`VAR3="$VAR1$VAR2" echo "$VAR3"` Результат: Hello, World
Второй: `VAR1="Hello," VAR1+=" World" echo "$VAR1"` Результат: Hello, World
3. 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. 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. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?

Преимущества и недостатки 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