

Шаблон отчёта по лабораторной работе № 13

1022204143

Надиа Эззакат

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	13
5	Контрольные вопросы	14

List of Tables

List of Figures

3.1	mcredit	8
3.2	ВЫВОД	9
3.3	mcredit	10
3.4	mcredit	10
3.5	ВЫВОД 1	11
3.6	ВЫВОД 2	11
3.7	mcredit	12
3.8	ВЫВОД	12

1 Цель работы

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

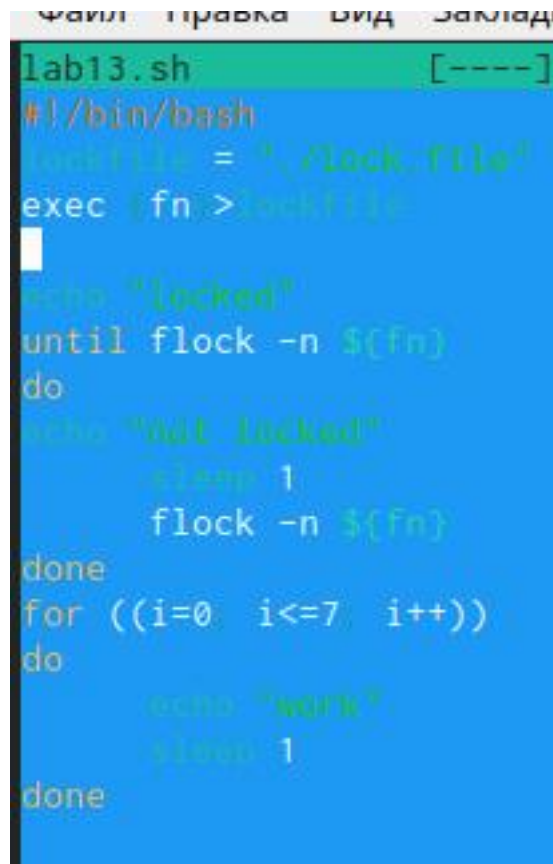
2 Задание

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 Выполнение лабораторной работы

1. Написала командный файл, реализующий упрощённый механизм семафоров.



```
lab13.sh [-----]
#!/bin/bash
lockfile="/lockfile"
exec {fn}>{lockfile}

echo "not locked"
until flock -n ${fn}
do
    echo "not locked"
    sleep 1
    flock -n ${fn}
done
for ((i=0; i<=7; i++))
do
    echo "work"
    sleep 1
done
```

Figure 3.1: mcedit


```
nehzzakat@dk4n64 ~ $ touch lab13.sh
nehzzakat@dk4n64 ~ $ chmod + lab13.sh
nehzzakat@dk4n64 ~ $ chmod +x lab13.sh
nehzzakat@dk4n64 ~ $ mcedit lab13.sh

nehzzakat@dk4n64 ~ $ ./lab13.sh
./lab13.sh: строка 2: lockfile: команда не найдена
locked
work
work
work
work
work
work
work
work
work
nehzzakat@dk4n64 ~ $
```

Figure 3.2: вывод

2. Реализовала команду man с помощью командного файла. Изучила содержимое каталога /usr/share/man/man1.

```

nvme-dsm.1.bz2
nvme-effects-log.1.bz2
nvme-endurance-log.1.bz2
nvme-error-log.1.bz2
nvme-flush.1.bz2
nvme-format.1.bz2
nvme-fw-commit.1.bz2
nvme-fw-download.1.bz2
nvme-fw-log.1.bz2
nvme-gen-hostqn.1.bz2
nvme-get-feature.1.bz2
nvme-get-log.1.bz2
nvme-get-ns-id.1.bz2
nvme-get-property.1.bz2
nvme-help.1.bz2
nvme-huawei-id-ctrl.1.bz2
nvme-huawei-list.1.bz2
nvme-id-ctrl.1.bz2
nvme-id-ns.1.bz2
nvme-id-nvmset.1.bz2
nvme-intel-id-ctrl.1.bz2
nvme-intel-internal-log.1.bz2
nvme-intel-lat-stats.1.bz2
nvme-intel-market-name.1.bz2
nvme-intel-smart-log-add.1.bz2
nvme-intel-temp-stats.1.bz2
nvme-io-passthru.1.bz2
nvme-list.1.bz2
nvme-list-ctrl.1.bz2
nvme-list-ns.1.bz2
nvme-list-subsys.1.bz2
nvme-lnvm-create.1.bz2
zlib-flate.1.bz2
zmore.1.bz2
znew.1.bz2
zonetab2pot.py.1.bz2
zresample.1.bz2
zretune.1.bz2
zrun.1.bz2
zsh.1.bz2
zshall.1.bz2
zshbuiltins.1.bz2
zshcalsys.1.bz2
zshcompctl.1.bz2
zshcompsys.1.bz2
zshcompwid.1.bz2
zshcontrib.1.bz2
zshexpn.1.bz2
zshmisc.1.bz2
zshmodules.1.bz2
zshoptions.1.bz2
zshparam.1.bz2
zshroadmap.1.bz2
zshtcpsys.1.bz2
zshzftpsys.1.bz2
zshzle.1.bz2
zsoelim.1.bz2
zstd.1.bz2
zstdcat.1.bz2
zstdgrep.1.bz2
zstdless.1.bz2
zvbi-chains.1.bz2
zvbid.1.bz2
zvbi-ntsc-cc.1.bz2

```

nehzzakat@dk4n64 /usr/share/man/man1 \$

Figure 3.3: mcedit

```

lab13_2.sh [----] 18
#!/bin/bash
cd /usr/share/man/man1
if (test -f $1.1.bz2)
then mv $1.1.bz2
else echo "Not working"
fi

```

Figure 3.4: mcedit

```

zvbid(1)                                VBI proxy daemon                                zvbid(1)

NAME
    zvbid - VBI proxy daemon

SYNOPSIS
    zvbid [ options ]

DESCRIPTION
    zvbid is a proxy for VBI devices, i.e. it forwards one or more VBI data streams to one or more con-
    nected clients and manages channel change requests.

OPTIONS
    -dev path
        Path of a device from which to read data. This argument can be given several times with dif-
        ferent devices.

    -buffers count
        Number of buffers to allocate for capturing VBI raw data from devices which support streaming
        (currently only video4linux, rev. 2) A higher number of buffers can prevent data loss in case
        of high latency. The downside is higher memory consumption (typically 65kB per buffer.) De-
        fault count is 8, maximum is 32.

    -nodetach
        Daemon process remains connected to the terminal from which it was started (e.g. so that you
        can stop it by pressing Control-C keys). Intended for trouble shooting only.

    -kill
        Terminates a proxy daemon running for the given device.

    -debug level
        Enables debug output: 0= off(default); 1= general messages; In addition 2, 4, 8, ... can be
        added to enable debug output for various categories.
zvbid 1 h72 lines 1-32

```

Figure 3.5: вывод 1

```

nehzzakat@dk4n64 ~ $ ./lab13_2.sh cdd
Not working
nehzzakat@dk4n64 ~ $ ./lab13_2.sh zvbid
nehzzakat@dk4n64 ~ $ ./lab13_2.sh zvbid.1
Not working
nehzzakat@dk4n64 ~ $

```

Figure 3.6: вывод 2

- 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.

```
lab13_3.sh [----] 10 L:[ 1+ 7 8/ 12] *(103 /
#!/bin/bash

for symbol in {A..Z} {a..z};
do SYMBOLS=$SYMBOLS$symbol;
done
STR_LEN=20
STR=""
for i in `seq 1 $STR_LEN`
do
STR=$STR${SYMBOLS:$(expr $RANDOM % ${#SYMBOLS}):1}
done
echo $STR
```

Figure 3.7: mcedit

```
nehzzakat@dk4n64 ~ $ mcedit lab13_3.sh

nehzzakat@dk4n64 ~ $ ./lab13_3.sh
lKIjsOdDqOMIxqyFTVDR
nehzzakat@dk4n64 ~ $ ./lab13_3.sh
pKVzbKNlTebigdeYaQiy
nehzzakat@dk4n64 ~ $ ./lab13_3.sh
XkpNlqgIBSXRKcgzphjI
nehzzakat@dk4n64 ~ $
nehzzakat@dk4n64 ~ $ ./lab13_3.sh
ryXZPRkWCQFbTpEgiylP
nehzzakat@dk4n64 ~ $ ./lab13_3.sh
vJpXBFewungiudqlzRBE
nehzzakat@dk4n64 ~ $
```

Figure 3.8: вывод

4 Выводы

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

5 Контрольные вопросы

1. \$1 следует внести в кавычки.
2. С помощью знака >| можно объединить несколько строк в одну.
3. Эта утилита выводит последовательность целых чисел с заданным шагом.
Также можно реализовать с помощью утилиты jot.
4. Результатом вычисления выражения $\$((10/3))$ будет число 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 рутины 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, исключаяющей написание скриптов, используйте zsh. Если вам часто приходится писать свои скрипты, только bash! Впрочем, можно комбинировать.
6. Синтаксис конструкции for ((a=1; a <= LIMIT; a++)) верен.
7. Язык bash и другие языки программирования:
 - а. Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на си/си++, скомпилированных с максимальной оптимизацией;

- b. 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;
- c. 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;
- d. Скорость кодов, генерируемых компилятором языка си фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;
- e. Скорость ассемблерных кодов x86-64 может меньше, чем аналогичных кодов x86, примерно на 10%;
- f. Оптимизация кодов лучше работает на процессоре Intel;
- g. 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;
- h. 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;

- i. В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, осо-

бенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета $\text{ask}(5,2,3)$