

Лабораторная работа №12

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

Заболотная Кристина Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	13
4.1	Ответы на контрольные вопросы	13
5	Выводы	16
	Список литературы	17

Список иллюстраций

3.1	создаем lab12.sh	8
3.2	первый скрипт	9
3.3	создаем lab12-1.sh	9
3.4	второй скрипт	10
3.5	less	10
3.6	создаем lab12-2.sh	11
3.7	второй скрипт	11
3.8	вывод 10 слов	12

Список таблиц

1 Цель работы

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

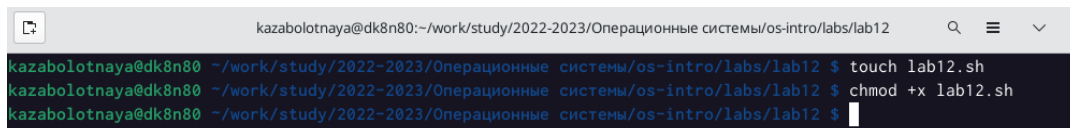
2 Задание

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. Написан 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). Запустили 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. Имеется возможность взаимодействия трёх и более процессов.



```
kazaboltnaya@dk8n80:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12.sh
kazaboltnaya@dk8n80:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12.sh
kazaboltnaya@dk8n80:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $
```

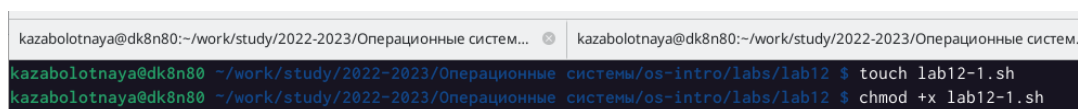
Рис. 3.1: создаем lab12.sh



```
1 #!/bin/bash
2 lockfile="./lockfile"
3 exec {fn}>$lockfile
4 echo "lock"
5 until flock -n ${fn}
6 do
7     echo "not lock"
8     sleep 1
9     flock -n ${fn}
10 done
11 for ((i=0; i<=5; i++))
12 do
13     echo "work"
14     sleep 1
15 done
16
17
```

Рис. 3.2: певый скрипт

2. Реализована команду man с помощью командного файла. Изучино 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.



```
kazabolotnaya@dk8n80:~/work/study/2022-2023/Операционные систем...  kazabolotnaya@dk8n80:~/work/study/2022-2023/Операционные систем.
kazabolotnaya@dk8n80 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12-1.sh
kazabolotnaya@dk8n80 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12-1.sh
```

Рис. 3.3: создаем lab12-1.sh

```
lab12-1.sh - GNU Emacs at dk8n8l

File Edit Options Buffers Tools Sh-Script Outline Hic

#!/bin/bash
cd /usr/share/man/man1
less $1*
```

Рис. 3.4: второй скрипт

```
LESS(1)                                General Commands Manual                                LESS(1)

ESC[1mNAMEESC[0m
    less - opposite of more

ESC[1mSYNOPSISESC[0m
    ESC[1mless -?ESC[0m
    ESC[1mless --helpESC[0m
    ESC[1mless -VESC[0m
    ESC[1mless --versionESC[0m
    ESC[1mless [-[+ ]aAbcCdEfFgGiIjKlMmNnOoPpQqRrSsTtUuVvWwXx~]ESC[0m
    ESC[1m[-b ESC[4mESC[22mspaceESC[24mESC[1m] [-h ESC[4mESC[22mlinesESC[24mESC[1m] [-j ESC[4mESC[22mlineESC[24mESC[1m] [-k
    ESC[1m[-o0] ESC[4mESC[22mlogfileESC[24mESC[1m] [-p ESC[4mESC[22mpatternESC[24mESC[1m] [-P ESC[4mESC[22mpromptESC[24mESC[1m]
    ESC[1m[-T ESC[4mESC[22mtagsfileESC[24mESC[1m] [-x ESC[4mESC[22mtabESC[24mESC[1m,...] [-y ESC[4mESC[22mlinesESC[24mESC[1m]
    ESC[1m[-# ESC[4mESC[22mshiftESC[24mESC[1m] [+][+ ]ESC[4mESC[22mcmdESC[24mESC[1m] [--] [ESC[4mESC[22mfilenameESC[24mESC[1m]
    (See the OPTIONS section for alternate option syntax with long option names.)

ESC[1mDESCRIPTIONESC[0m
    ESC[4mLessESC[24m is a program similar to ESC[4mmoreESC[24m(1), but which allows backward movement in the file as well as
    forward movement. Also, ESC[4mlessESC[24m does not have to read the entire input file before starting, so
    with large input files it starts up faster than text editors like ESC[4mviESC[24m(1). ESC[4mLessESC[24m uses termcap (or
    terminfo on some systems), so it can run on a variety of terminals. There is even limited sup-
    port for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top
    of the screen are prefixed with a caret.)

    Commands are based on both ESC[4mmoreESC[24m and ESC[4mviESC[24m. Commands may be preceded by a decimal number, called N
    in the descriptions below. The number is used by some commands, as indicated.

ESC[1mCOMMANDSESC[0m
    In the following descriptions, ^X means control-X. ESC stands for the ESCAPE key; for example
    ESC-v means the two character sequence "ESCAPE", then "v".

    h or H Help: display a summary of these commands. If you forget all the other commands, remem-
    ber this one.

    SPACE or ^V or f or ^F
    Scroll forward N lines (if N is zero, scroll forward from the bottom line to the top line). If N is non-zero, the
```

Рис. 3.5: less

3. Используя встроенную переменную \$RANDOM, написан командный файл, генерирующий случайную последовательность букв латинского алфавита. \$RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

```
kazabolotnaya@dk8n80 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ touch lab12-2.sh
kazabolotnaya@dk8n80 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $ chmod +x lab12-2.sh
kazabolotnaya@dk8n80 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab12 $
```

Рис. 3.6: создаем lab12-2.sh



The screenshot shows a text editor window titled '*lab12-2.sh' with the following script content:

```
1 #!/bin/bash
2 M=10
3 c=1
4 d=1
5 echo
6 echo "10 random words:"
7 while (($c!=($M+1)))
8 do
9     echo $(for((i=1;i<=10;i++)); do printf '%s' "${RANDOM:0:1}"; done | tr '[:0-9]' '[:a-z]')
10    echo $d
11    ((c+=1))
12    ((d+=1))
13 done
```

Рис. 3.7: второй скрипт

10 random words:

ccccgcbbcd

1

cddgcbeccc

2

debbbbbibcc

3

jbccddcfhb

4

jcbdccdbcf

5

cbjebfdccc

6

cbgbcibbdb

7

bdgcdcbdcb

8

ccbceibbbc

9

Рис. 3.8: вывод 10 слов

4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]`
2. Как объединить (конкатенация) несколько строк в одну?
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?
4. Какой результат даст вычисление выражения `$((10/3))`?
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.
6. Проверьте, верен ли синтаксис данной конструкции: `for ((a=1; a <= LIMIT; a++))`
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?

4.1 Ответы на контрольные вопросы

1. В строке `while [$1 != "exit"]` квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, `VAR1="Hello,"` `VAR2="World"` `VAR3="␣␣␣1VAR2"` `echo "$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 $i" done`
4. Результатом вычисления выражения `$((10/3))` будет число 3.
5. Список того, что можно получить, используя `Z Shell` вместо `Bash`: Встро-

енная 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 `exes`, чтобы заменить оболочку другой командой.

6. Синтаксис конструкции `for ((a=1; a <= LIMIT; a++))` верен.
7. 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 вир-

туальной 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 ask(5,2,3)

5 Выводы

В ходе выполнения данной лабораторной работы мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Список литературы