

Отчет по лабораторной работе №12

Дисциплина Операционные системы

Колобова Елизавета, гр. НММбд-01-22

Содержание

1	Цель работы	5
2	Контрольные вопросы	9
	Список литературы	12

Список иллюстраций

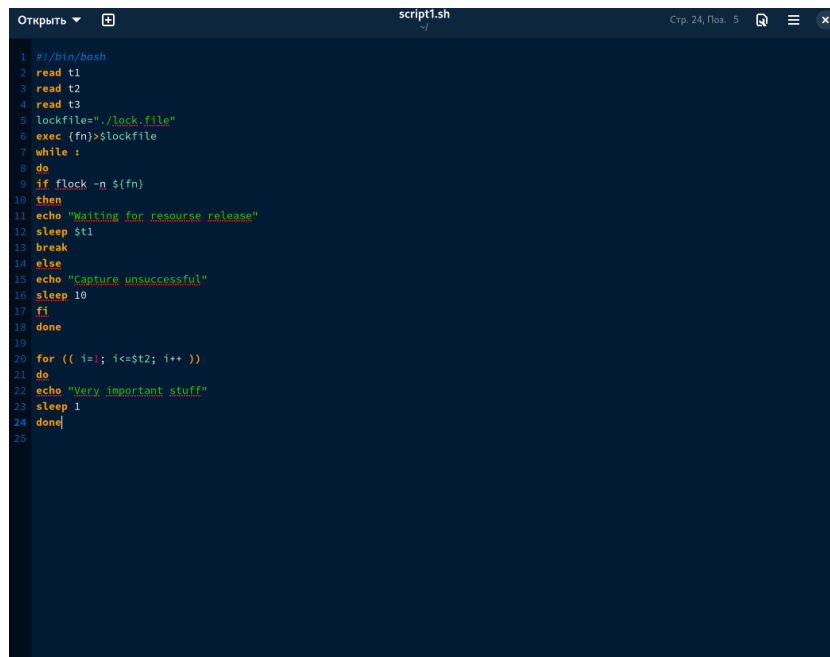
1.1	Рис. 1. Скрипт упрощенного механизма семафоров	6
1.2	Рис. 2. Реализация команды map	7
1.3	Рис. 3. Скрипт создания буквенной последовательности	8

Список таблиц

1 Цель работы

Цель работы - изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать сложные командные файлы с использованием логических управляющих конструкций и циклов. # Выполнение лабораторной работы

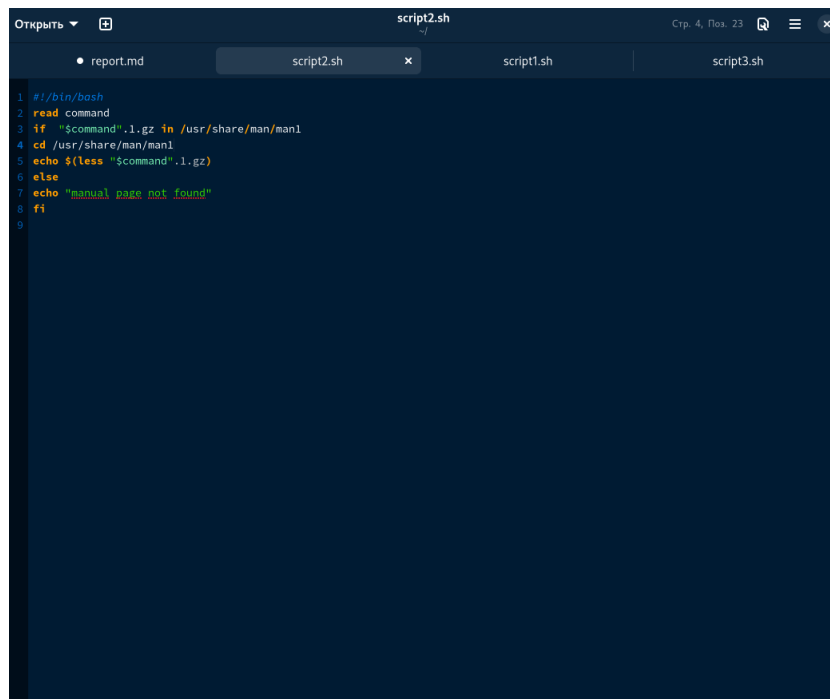
1. Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл в течение некоторого времени t_1 дожидается освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использует его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустим командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой ($> /dev/tty\#$ или $> /dev/psa/\#$, $\#$ - номер терминала). (рис. [1.1])



```
1 #!/bin/bash
2 read t1
3 read t2
4 read t3
5 lockfile="./lock.file"
6 exec {fn}>$lockfile
7 while :
8 do
9 if flock -n ${fn}
10 then
11 echo "Waiting for resource release"
12 sleep $t1
13 break
14 else
15 echo "capture unsuccessful"
16 sleep 10
17 fi
18 done
19
20 for (( i=1; i<=$t2; i++ ))
21 do
22 echo "Very important stuff"
23 sleep $t3
24 done
25
```

Рис. 1.1: Рис. 1. Скрипт упрощенного механизма семафоров

2. Реализуем команду `man` с помощью командного файла. Изучим 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. [1.2])



```
1 #!/bin/bash
2 read command
3 if "$command".1.gz in /usr/share/man/man1
4 cd /usr/share/man/man1
5 echo $(less "$command".1.gz)
6 else
7 echo "manual page not found"
8 fi
9
```

Рис. 1.2: Рис. 2. Реализация команды man

- Используя встроенную переменную \$RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. [1.3])

```
Открыть  script3.sh  Стр. 12, Пос. 28
1  #!/bin/bash
2  arr1=([a-z])
3  arr2=(0)
4  count=0
5  echo "-----"
6  while true
7  do
8      number=$RANDOM
9      ((count++))
10     if ((number < 27))
11     then
12         arr2[count]=$number
13         echo ${arr1[$number]}
14     fi
15     if ${#arr2[@]}=27
16     then
17         break
18     fi
19 done
```

Рис. 1.3: Рис. 3. Скрипт создания буквенной последовательности

2 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]` Этот арифметический оператор сравнивает константы, а не переменные, либо условия цикла в скобки не заключаются
2. Как объединить (конкатенация) несколько строк в одну? Самый простой способ объединить две или более строковые переменные — записать их одну за другой, также можно объединить одну или несколько переменных с литеральными строками. Другой способ объединения строк в `bash` — добавление переменных или литеральных строк к переменной с помощью оператора `+=`
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`? Эта утилита выводит последовательность целых чисел с шагом, заданным пользователем. По-умолчанию, выводимые числа отделяются друг от друга символом перевода строки, однако, с помощью ключа `-s` может быть задан другой разделитель. Также можно реализовать через вывод на консоль считанной последовательности путем обращения ко всем аргументам через `“$# / $@”`
4. Какой результат даст вычисление выражения `$((10/3))`? 3
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`. `zsh` представляет собой оболочку в стиле Борна, которая содержит функции, которые вы найдете в `bash`, и даже больше. Например, у `zsh` есть проверка орфографии, возможность отслеживать входы / выходы из системы, некоторые встроенные функции программирования, такие как байт-код, поддержка

научной нотации в синтаксисе, арифметика с плавающей точкой и другие функции. Эта новая оболочка совместима с `bash`, но включает в себя больше возможностей. Оболочка `zsh` предлагает встроенную коррекцию орфографии, улучшенное завершение командной строки, загружаемые модули, которые выступают в качестве плагинов для вашей оболочки, глобальные псевдонимы, которые позволяют использовать псевдонимы имен файлов или чего-либо еще в командной строке вместо просто команд, и больше поддержки тем. Это похоже на `bash`, но с множеством дополнительных возможностей, дополнительных функций и настраиваемых параметров, которые вы могли бы оценить, если бы вы проводили много времени в командной строке.

6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))` -да, верен
7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки? `Bash` (Bourne Again Shell) включает в себя простой язык программирования, который позволяет при помощи условных операторов и операторов цикла использовать утилиты и программы операционной системы для написания как простых, так и сложных скриптов. В этом плане `Bash`, несомненно, обладает некоторыми преимуществами, в частности, универсальностью и доступностью. Для того, чтобы написать скрипт на `Bash`, установка дополнительных пакетов не требуется. Достаточно создать файл вида `script_name.sh` с последовательно исполняемыми операциями и запустить его, либо добавить в качестве задачи планировщика `cron`. Стоит отметить, что возможности командного интерпретатора зачастую используются не полностью. Многие администраторы выбирают `Bash` для написания простых или средних по сложности скриптов. В крупных проектах, где есть специфические задачи и требуется работа с разнообразными входными данными, многомерными массивами и сокетами больше доверяют `Perl`, `Python` или `Ruby`. Отчасти это

связано с проблемами переносимости bash-скриптов на другие платформы, (например, Windows), отчасти с тем, что Bash воспринимается скорее как средство автоматизации работы с файлами и утилитами, чем полноценный скриптовый язык, даже несмотря на наличие в арсенале sed и awk. Ещё одним минусом Bash является то, что при выполнении скрипта каждая запущенная с его помощью утилита создаёт свой процесс, что отражается на скорости выполнения и уровне использования ресурсов системы. # Выводы

Результатом проделанной работы является изучение основ программирования в оболочке ОС UNIX/Linux.

Список литературы