

Отчет по лабораторной работе №12

Дисциплина: Операционные системы

Кашкин Ивнн Евгеньевич

Содержание

Цель работы	5
Задание	6
Теоретическое введение	7
Выполнение лабораторной работы	8
Выводы	16
Контрольные вопросы	17
Список литературы	20

Список иллюстраций

0.1	Создание первой программы	8
0.2	Программа №1	9
0.3	Ввод программы	10
0.4	Ввод программы	11
0.5	Ввод программы	12
0.6	Смотрим каталог	12
0.7	Создание программы	13
0.8	Программа №2	13
0.9	Ввод программы	13
0.10	Ввод программы	14
0.11	Создание программы	14
0.12	Программа №3	15

Список таблиц

Цель работы

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

Задание

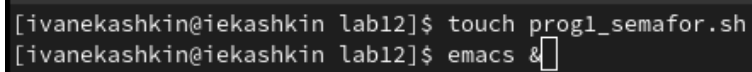
1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию 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 имела 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.

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;
- С-оболочка (или csh) — надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Выполнение лабораторной работы

1. Зашёл в каталог лабораторной работы, создал файл 1 программы prog1_semafor.sh и открыл emacs (рис. [-@fig:001])



```
[ivanekashkin@iekashkin lab12]$ touch prog1_semafor.sh  
[ivanekashkin@iekashkin lab12]$ emacs &
```

Рис. 0.1: Создание первой программы

- Написал командный файл, реализующий упрощённый механизм семафоров. (рис. [-@fig:002])


```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t1))
do
    echo "Ожидание"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
s1=$(date +%s)
s2=$(date +%s)
((t=s2-s1))
while ((t<t2))
do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s)
    ((t=s2-s1))
done
□
```

Рис. 0.2: Программа №1

- Далее сохранил файл и запустил его из командной строки. Прописываем команду “./prog1_semafor.sh 5 4” и наблюдаем выполнение программы (рис. [-@fig:003])

```
[ivanekashkin@iekashkin lab12]$ chmod +x *.sh
[1]+  Завершён      emacs
[ivanekashkin@iekashkin lab12]$ ./prog1_semafor.sh
[ivanekashkin@iekashkin lab12]$ ./prog1_semafor.sh 5 4
Ожидание
Ожидание
Ожидание
Ожидание
Ожидание
Выполнение
Выполнение
Выполнение
Выполнение
[ivanekashkin@iekashkin lab12]$
```

Рис. 0.3: Ввод программы

- После модернизируем командный файл по заданию, чтобы имелась возможность взаимодействия трёх и более процессов.(рис. [-@fig:004])

```
#!/bin/bash
function ozhid{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t1))
    do
        echo "Ожидание"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
function vipoln{
    s1=$(date +%s)
    s2=$(date +%s)
    ((t=s2-s1))
    while ((t<t2))
    do
        echo "Выполнение"
        sleep 1
        s2=$(date +%s)
        ((t=s2-s1))
    done
}
t1=$1
t2=$2
com=$3
while true
do
    if [ "$com"=="Exit" ]
    then
        echo "Exit"
        exit 0
    fi
    if [ "$com"=="Ожидание" ]
    then ozhid
    fi
    if [ "$com"=="Выполнение" ]
    then vipoln
    fi
    echo "Следующее действие: "
    read com
done
```

Рис. 0.4: Ввод программы

- Далее сохранил файл и запустил его из командной строки. Прописываем команду “./prog1_semafor.sh 5 4” и наблюдаем выполнение программы (рис. [-@fig:005])

```
[ivanekashkin@iekashkin lab12]$ sudo ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/1 &
[1] 13519
bash: /dev/pts/1: Отказано в доступе
[1]+ Выход 1          sudo ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/1
[ivanekashkin@iekashkin lab12]$ sudo ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/2 &
[1] 13554
bash: /dev/pts/2: Отказано в доступе
[1]+ Выход 1          sudo ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/2
[ivanekashkin@iekashkin lab12]$ ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/2 &
[1] 13571
bash: /dev/pts/2: Отказано в доступе
[1]+ Выход 1          ./prog1_semafor.sh 5 4 Ожидание > /dev/pts/2
[ivanekashkin@iekashkin lab12]$
```

Рис. 0.5: Ввод программы

2. По заданию мы просматриваем каталог /usr/share/man/man1 (рис. [-@fig:005])

```
[ivanekashkin@iekashkin lab12]$ cd /usr/share/man/man1
[ivanekashkin@iekashkin man1]$ ls
.: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-core.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
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
```

Рис. 0.6: Смотрим каталог

- Создал файлы для второй программы prog2_man.sh (рис. [-@fig:007])

```
[ivanekashkin@iekashkin lab12]$ touch prog2_man.sh
[ivanekashkin@iekashkin lab12]$ emacs &
```

Рис. 0.7: Создание программы

- После написал программу для файла .sh (рис. [-@fig:008])

```
#!/bin/bash
a=$1
if [ -f /usr/share/man/man1/$a.1.gz ]
then
    guzip -c /usr/share/man/man1/$a.1.gz | less
else
    echo "Справки нет"
fi
```

Рис. 0.8: Программа №2

- Делае я проверил работы программы, запустив ее из терминала
“./prog2_man.sh ls” (рис. [-@fig:009]) (рис. [-@fig:0010])

```
.\\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.47.3.
.TH LS "1" "March 2022" "GNU coreutils 8.32" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[\fI\,OPTION\|\fR]... [\fI\,FILE\|\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of \fB\-cftuvSUX\|fR nor \fB\-\-sort\|fR is specified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\-a\|fR, \fB\-\-all\|fR
do not ignore entries starting with .
.TP
\fB\-A\|fR, \fB\-\-almost-\|fR
do not list implied . and ..
.TP
\fB\-\-author\|fR
with \fB\-l\|fR, print the author of each file
.TP
\fB\-b\|fR, \fB\-\-escape\|fR
print C\|fR-style escapes for nongraphic characters
```

Рис. 0.9: Ввод программы

```
[ivanekashkin@iekashkin lab12]$ chmod +x *.sh
[ivanekashkin@iekashkin lab12]$ ./prog2_man.sh ls
[ivanekashkin@iekashkin lab12]$ ./prog2_man.sh apple
Справки нет
```

Рис. 0.10: Ввод программы

3. Создал файлы для третьей программы prog3_random.sh (рис. [-@fig:0011])

```
[ivanekashkin@iekashkin lab12]$ touch prog3_random.sh
[ivanekashkin@iekashkin lab12]$ emacs &
```

Рис. 0.11: Создание программы

- Используя встроенную переменную \$RANDOM, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. (рис. [-@fig:0012])

```
#!/bin/bash

a=$1
for ((i=0; i<$a; i++))
do
    ((char=$RANDOM%26+1))
    case $char in
        1) echo -n A;; 14) echo -n N;;
        2) echo -n B;; 15) echo -n O;;
        3) echo -n C;; 16) echo -n P;;
        4) echo -n D;; 17) echo -n Q;;
        5) echo -n E;; 18) echo -n R;;
        6) echo -n F;; 19) echo -n S;;
        7) echo -n G;; 20) echo -n T;;
        8) echo -n H;; 21) echo -n U;;
        9) echo -n I;; 22) echo -n V;;
        10) echo -n J;; 23) echo -n W;;
        11) echo -n K;; 24) echo -n X;;
        12) echo -n L;; 25) echo -n Y;;
        13) echo -n M;; 26) echo -n Z;;
    esac
done
echo

```

Рис. 0.12: Программа №3

- Я сохранил командный файл и использовал его командой (рис. [-@fig:0011])

Ввод программы

Выводы

- Изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов

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 ["$1" != "exit"]`
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:
 - Первый: `VAR1="Hello," VAR2=" World" VAR3="$VAR1VAR2" 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 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. Преимущества 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

Список литературы

::: {#Лабораторная работа No 10. Программирование в командном процессоре ОС UNIX. Командные файлы} :::