Цель работы

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

Задание

Написать командные файлы, выполняющие действия, описанные в задании

Ход работы

Необходимо выполнить следующие задания:

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, 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.

    - Написал командный файл, которая принимает на вход два параметра - время ожидания и время выполнения. Если командный файл за время ожидания не может получить доступ к "ресурсу", он выведет сообщение об этом и начнет ожидание заново. После того как "ресурс" стал доступен, командный файл его занимает и начинает в течение времени **t2** выводит сообщение о своей работе. После завершения "ресурс" освобождается. Код командного файла: (Рис.1)

```
task1.sh           [----] 66 L:[  1+21  22/ 28] *(946 /1162b) 1088 0x440
lock=/tmp/resource<----><------>#Путь к "ресурсу"
t1=$1<-><------><------><------>#Время ожидания (первый аргумент)
t2=$2<-><------><------><------>#Время выполнения (второй аргумент)
while true<----><------><------>#Бесконечный цикл
    do
    if [ -f $lock ]<---><------>#Проверка занятости "ресурса"
    then
<------>echo "Please, free resource"<-->#Сообщение о занятости "ресурса"
<------>for (( i=1; i<=t1; i++))<------>#Цикл ожидания
<------>do
<------>    echo "Resource occupied by another process" #Сообщение о занятости "ресурса"
<------>    sleep 1<---><------><------><------><------>#Секундное ожидание
<------>    if [ ! -f $lock ]<-><------><------><------>#Проверка занятости "ресурса"
<------>    then
<------><------>break<-><------><------><------><------>#Выход из цикла, если "ресурса" нет
<------>    fi
<------>done
    else
<------>break<-><------><------><------><------><------>#Выход из цикла, если "ресурса" нет
    fi
done
touch $lock<---><------><------><------><------><------>#Занятие "ресурса"
for (( i=1; i<=t2; i++))<------><------><------><------>#Цикл выполнения
do
    echo "Using resource"
    sleep 1
done
rm $lock<------><------><------><------><------><------>#Освобождение "ресурса" по завершении работы
```

Рис.1

- Результат работы командного файла в нескольких терминалах (предварительно дал файлу разрешение на выполнение с помощью команды **chmod +x имяфайла**) (Рис.2):

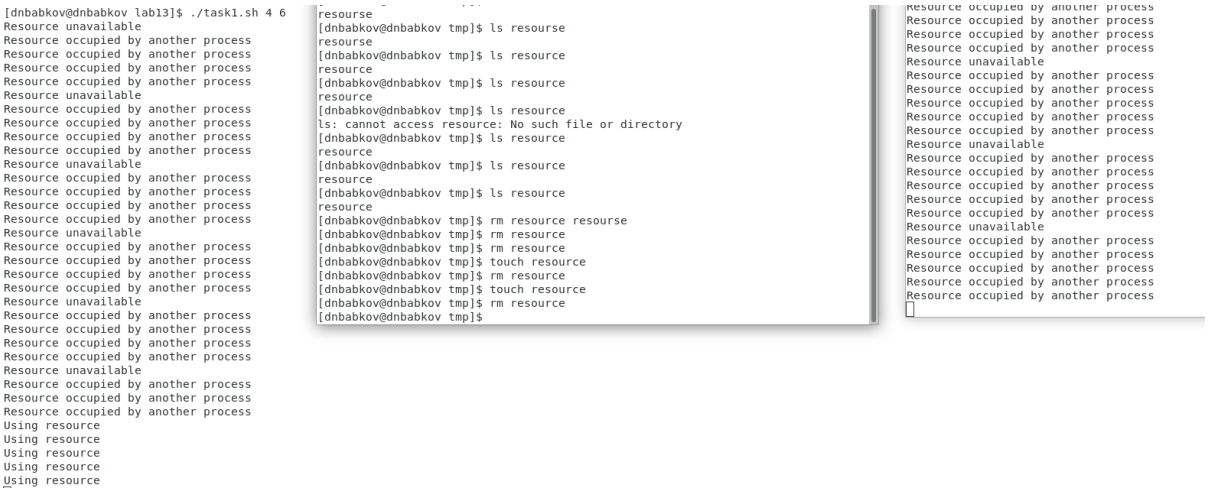

Рис.2

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.

   - Перед написанием командного файла я изучил содержимое каталога **/usr/share/man/man1** (Рис.3)

```
[dnbabkov@dnbabkov man1]$ cd /usr/share/man/man1
[dnbabkov@dnbabkov man1]$ ls -l
total 12188
-rw-r--r--. 1 root root     40 Apr  1  2020 :.1.gz
-rw-r--r--. 1 root root     40 Apr  1  2020 [.1.gz
-rw-r--r--. 1 root root   4644 Feb  2 19:32 a2p.1.gz
-rw-r--r--. 1 root root   3523 Jan  9  2013 ab.1.gz
-rw-r--r--. 1 root root   1109 Oct  1  2020 abrt-action-analyze-backtrace.1.gz
-rw-r--r--. 1 root root    950 Oct  1  2020 abrt-action-analyze-c.1.gz
-rw-r--r--. 1 root root    932 Oct  1  2020 abrt-action-analyze-ccpp-local.1.gz
-rw-r--r--. 1 root root    810 Oct  1  2020 abrt-action-analyze-core.1.gz
-rw-r--r--. 1 root root   1003 Oct  1  2020 abrt-action-analyze-oops.1.gz
-rw-r--r--. 1 root root   1004 Oct  1  2020 abrt-action-analyze-python.1.gz
-rw-r--r--. 1 root root   1049 Oct  1  2020 abrt-action-analyze-vmcore.1.gz
-rw-r--r--. 1 root root    835 Oct  1  2020 abrt-action-analyze-vulnerability.1.gz
-rw-r--r--. 1 root root   1138 Oct  1  2020 abrt-action-analyze-xorg.1.gz
-rw-r--r--. 1 root root    955 Oct  1  2020 abrt-action-check-oops-for-hw-error.1.gz
-rw-r--r--. 1 root root   1045 Oct  1  2020 abrt-action-generate-backtrace.1.gz
-rw-r--r--. 1 root root   1185 Oct  1  2020 abrt-action-generate-core-backtrace.1.gz
-rw-r--r--. 1 root root   1012 Oct  1  2020 abrt-action-install-debuginfo.1.gz
-rw-r--r--. 1 root root    916 Oct  1  2020 abrt-action-list-dsos.1.gz
-rw-r--r--. 1 root root   1131 Oct  1  2020 abrt-action-notify.1.gz
-rw-r--r--. 1 root root    991 Oct  1  2020 abrt-action-perform-ccpp-analysis.1.gz
-rw-r--r--. 1 root root    958 Oct  1  2020 abrt-action-save-kernel-data.1.gz
-rw-r--r--. 1 root root   1074 Oct  1  2020 abrt-action-save-package-data.1.gz
-rw-r--r--. 1 root root    839 Oct  1  2020 abrt-action-trim-files.1.gz
-rw-r--r--. 1 root root    873 Oct  1  2020 abrt-applet.1.gz
-rw-r--r--. 1 root root   1475 Oct  1  2020 abrt-auto-reporting.1.gz
-rw-r--r--. 1 root root   1055 Oct  1  2020 abrt-cli.1.gz
-rw-r--r--. 1 root root   1071 Oct  1  2020 abrt-dump-oops.1.gz
-rw-r--r--. 1 root root    948 Oct  1  2020 abrt-dump-xorg.1.gz
-rw-r--r--. 1 root root    810 Oct  1  2020 abrt-handle-upload.1.gz
-rw-r--r--. 1 root root    793 Oct  1  2020 abrt-harvest-pstoreoops.1.gz
-rw-r--r--. 1 root root    875 Oct  1  2020 abrt-harvest-vmcore.1.gz
-rw-r--r--. 1 root root    729 Oct  1  2020 abrt-install-ccpp-hook.1.gz
-rw-r--r--. 1 root root    798 Oct  1  2020 abrt-merge-pstoreoops.1.gz
-rw-r--r--. 1 root root   1369 Oct  1  2020 abrt-retrace-client.1.gz
-rw-r--r--. 1 root root   1139 Oct  1  2020 abrt-server.1.gz
-rw-r--r--. 1 root root    787 Oct  1  2020 abrt-watch-log.1.gz
```

Рис.3

- Написал командный файл, который принимает на вход название команды, находит соответствующий архив в указанном выше каталоге, распаковывает его и выводит содержимое справки с помощью конвейера и команды **less** и, если такой справки нет, выводит соответствующее сообщение (Рис.4)

```
task2.sh           [-M--] 56 L:[  1+ 6   7/  8] *(378 / 381b) 0010 0x00A
c=$1<-->#Название искомой команды
path="/usr/share/man/man1/"<--->#Путь к справкам по командам
if [ -f $path$c.1.gz ]<>#Проверка существования справки по команде
then
    gunzip -c $path$c.1.gz | less<----->#Вывод справки
else
    echo "No reference abailable"<----->#Вывод сообщения
fi
```

Рис.4

- Результат работы командного файла для команды **ls** (Рис.5)

  ```
  .\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.43.3.
  .TH LS "1" "November 2020" "GNU coreutils 8.22" "User Commands"
  .SH NAME
  ls \- list directory contents
  .SH SYNOPSIS
  .B ls
  [\fIOPTION\fR]... [\fIFILE\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\-all\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\-style escapes for nongraphic characters
  .TP
  \fB\-\-block\-size\fR=\fISIZE\fR
  scale sizes by SIZE before printing them; e.g.,
  \&'\-\-block\-size=M' prints sizes in units of
  1,048,576 bytes; see SIZE format below
  .TP
  \fB\-B\fR, \fB\-\-ignore\-backups\fR
  do not list implied entries ending with ~
  .TP
  \fB\-c\fR
  with \fB\-lt\fR: sort by, and show, ctime (time of last
  modification of file status information);
  with \fB\-l\fR: show ctime and sort by name;
  otherwise: sort by ctime, newest first
  .TP
  ```

  Рис.5

3. Используя встроенную переменную **$RANDOM**, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что эта переменная выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

   - Написал командный файл, который принимает на вход одно число - кол-во выводимых символов. Затем с помощью цикла **for** и оператора **case** выводятся символы в зависимости от значений переменной **$RANDOM**, которая принимает значения от 1 до 26 (Рис.6)

     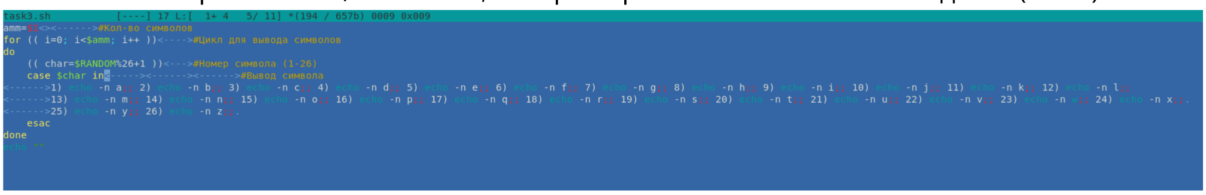

     Рис.6

- Результат работы командного файла: (Рис.7)

```
[dnbabkov@dnbabkov lab13]$ ./task3.sh 5
tyrkj
[dnbabkov@dnbabkov lab13]$ ./task3.sh 5
kzzse
[dnbabkov@dnbabkov lab13]$ ./task3.sh 5
xmpgo
[dnbabkov@dnbabkov lab13]$ ./task3.sh 16
hdrozwnicqfnfqbo
[dnbabkov@dnbabkov lab13]$ ./task3.sh 16
pdupugtslnmjtlyu
```

Рис.7

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 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 языков общего назначения
    - Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на быстроте выполнения этого скрипта
    - Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий

Вывод

В ходе выполнения данной лабораторной работы я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.