Программирование в командном процессоре ОС UNIX.Расширенное программирование.

Отчет по лабораторной работе №13

Башкирова Я.Д.

Содержание

Список таблиц

Список иллюстраций

1 Теоретическое введение

UNIX - это многозадачная, многопользовательская операционная система (ОС) общего назначения. Многозадачность означает, что в среде одновременно могут выполняться несколько программ. Термин многопользовательская “говорит” о том, что в системе одновременно могут работать несколько пользователей.

Операционная система UNIX была создана в первой половине 70-х годов в подразделении фирмы AT&T, называемом Bell Telephone Labs. Первая версия системы была реализована на мини-ЭВМ PDP/7 и предназначалась, в основном, для обработки текстовой информации. Пройдя успешную проверку, система получила дальнейшее развитие. Большая часть кодов была переписана с ассемблера PDP на язык высокого уровня Си, разрабатываемый в то же время в упомянутой фирме. UNIX удачно отражал особенности существовавших в то время операционных систем и был применен в ряде университетов Северной Америки для обучения студентов принципам построения ОС. В начале 80-х годов UNIX был установлен на ряде вычислительных машин, отличных по своей архитектуре от PDP. В середине 80-х система привлекла внимание фирм-производителей компьютерного оборудования и программного обеспечения. Появились первые коммерческие версии UNIX. Они получили широкое распространение во всем мире. Сегодня трудно назвать ЭВМ, для которой не была бы разработана версия этой ОС (в дальнейшем версии системы мы также буден называть ее диалектами).

Широкое распространение системы UNIX объясняется простотой ее устройства и легкостью переноса на компьютеры разной архитектуры. Последнее есть следствие того, что ОС написана на языке высокого уровня, и машинно-зависимые

ее части невелики по объему и четко выделены. Но слаба машинная зависимость имеет и свои недостатки. Основным из них является некоторое снижение производительности UNIX по сравнению с системами, разработанными для конкретной ЭВМ и в большей степени учитывающими ее аппаратные особенности.

2 Цель работы

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

3 Задание

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 в фоновом, а 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.

4 Ход работы

1. Я написала командный файл, реализующий упрощённый механизм семафоров. Я запустила 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 в фоновом, а привилегированном режиме. Я доработала программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

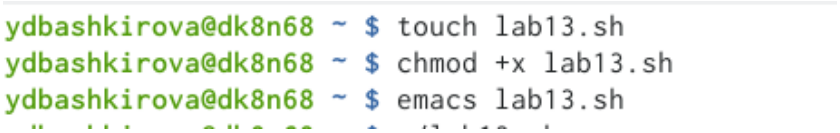

Рис. 4.1: Создание файла и дача ему прав

Рис. 4.2: Командный файл

Рис. 4.3: Упрощенный механизм семафоров

2. Я реализовала команду man с помощью командного файла. Я изучила содержимое каталога /usr/share/man/man1. 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.

```
ydbashkirova@dk8n68 ~ $ touch lab131.sh
ydbashkirova@dk8n68 ~ $ chmod +x lab131.sh
ydbashkirova@dk8n68 ~ $ emacs lab131.sh
```

Рис. 4.4: Создание файла и дача ему прав

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

Рис. 4.5: Командный файл

```
ydbashkirova@dk8n68 ~ $ ./lab131.sh
ydbashkirova@dk8n68 ~ $
```

Рис. 4.6: Открытие архива

```
411toppm User Manual(0)

NAME
       411toppm - convert Sony Mavica .411 image to PPM

SYNOPSIS
       411toppm [-width width] [-height height] [411file]

DESCRIPTION
       This program is part of Netpbm(1)

        411toppm reads a .411 file, such as from a Sony Mavic camera, and converts it to a PPM image as output.

       Output is to Standard Output.

       The  originator of this program and decipherer of the .411 format, Steve Allen <sla@alumni.caltech.edu>, has this to say about
       nail (especially when you have the full size JPG file) that the only point in doing this was to answer the implicit challenge

OPTIONS
       All options may be abbreviated to the shortest unique prefix.

       -width The width (number of columns) of the input image.  Default is 64.

       -height
              The height (number of rows) of the input image.  Default is 48.

SEE ALSO
       ppm(5)

DOCUMENT SOURCE
       This manual page was generated by the Netpbm tool 'makeman' from HTML source.  The master documentation is at

              http://netpbm.sourceforge.net/doc/411toppm.html
```

Рис. 4.7: Содержимое справки

3. Я, используя встроенную переменную $RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита.

```
ydbashkirova@dk8n68 ~ $ touch lab130.sh
ydbashkirova@dk8n68 ~ $ chmod +x lab130.sh
ydbashkirova@dk8n68 ~ $ touch lab130.sh less
ydbashkirova@dk8n68 ~ $ emacs lab130.sh
```

Рис. 4.8: Создание файла и дача ему прав

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

Рис. 4.9: Командный файл

```
ydbashkirova@dk8n68 ~ $ ./lab130.sh

10 random words:
hbbbijcecb
1
chdbcdbhbb
2
dbcdhcgbhd
3
cbbbccbcce
4
cbccfcbbcc
5
bbccdjbccd
6
bbccbdfcbb
7
hcdibcdgbd
8
hccbbbcbcb
9
jcdbccbgbc
10
ydbashkirova@dk8n68 ~ $
```

Рис. 4.10: Случайная последовательность букв

5 Выводы

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

6 Библиография

https://www.opennet.ru/docs/RUS/xtoolkit/x-1.html

7 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=“$VAR1$VAR2”

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:

Встроенная 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 exec, 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 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;

-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

ack(5,2,3)