

# Лабораторная работа №12

Отчет

Анастасия дмитриевна Форис

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	8
Выводы	10
Контрольные вопросы	11

## Список иллюстраций

## Список таблиц

## Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится 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 в 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 нём 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.

# Выполнение лабораторной работы

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 в фоновом, а в привилегированном режиме. Доработать программу так, чтобы имела возможность взаимодействия трёх и более процессов. Рис.1 Выполнение задания
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`. Рис.2 Командный файл и исполнение
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. Рис.3 Командный файл и исполнение

# Выводы

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке `while [$1 != "exit"] )$1` следует внести в кавычки («»)
2. Как объединить (конкатенация) несколько строк в одну? С помощью знака `>|`
3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать ее функционал при программировании на `bash`? Эта утилита выводит последовательность целых чисел с заданным шагом. Также можно реализовать с помощью утилиты `jot`.
4. Какой результат даст вычисление выражения `$((10/3))`? Результат: 3.
5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`. В `zsh` можно настроить отдельные сочетания клавиш так, как вам нравится. Использование истории команд в `zsh` ничем особенным не отличается от `bash`. `Zsh` очень удобен для повседневной работы и делает добрую половину рутины за вас. Но стоит обратить внимание на различия между этими двумя оболочками. Например, в `zsh` после `for` обязательно вставлять пробел, нумерация массивов в `zsh` начинается с 1, чего совершенно невозможно понять. Так, если вы используете `shell` для повседневной работы, исключаяющей написание скриптов, используйте `zsh`. Если вам часто приходится писать свои скрипты, только `bash`! Впрочем, можно комбинировать. Как установить `zsh` в качестве оболочки по-умолчанию для отдельного пользователя:о
6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))` Синтаксис верен.
7. Сравните язык `bash` с языками программирования, которые вы знаете. Какие

преимущества у `bash` по сравнению с ними? Какие недостатки?

1. Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на `си/си++`, скомпилированных с максимальной оптимизацией;
2. 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;
3. 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;
4. Скорость кодов, генерируемых компилятором языка `си` фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM;
5. Скорость ассемблерных кодов `x86-64` может меньше, чем аналогичных кодов `x86`, примерно на 10%;
6. Оптимизация кодов лучше работает на процессоре Intel;
7. 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;
8. 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;
9. 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)`