Отчёт по лабораторной работе №11

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Тагиев Павел Фаикович

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов [1].

# 2 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 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 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.

# 3 Выполнение лабораторной работы

## 3.1 Задание 1

Листинг 1: Упрощенный механизм семафоров

#! /usr/bin/bash  
  
semaphore\_wait() {  
 # атомарная проверка того что ресурс не занят  
 while ! mkdir /tmp/$0.lock 2> /dev/null; do  
 sleep 0.5 # ждем освобождения ресурса  
 done  
}  
  
semaphore\_signal() {  
 rmdir /tmp/$0.lock  
}  
  
echo -e "\nпроцесс номер $$ пытается захватить ресурс"  
  
semaphore\_wait # попытка войти в критическую секцию  
  
echo -e "\nпроцесс номер $$ начал выполнять критическую секцию"  
  
sleep 2 # выполнение какой-то работы требующей сихронизации  
  
semaphore\_signal # выход из критической секции  
  
echo -e "\nпроцесс номер $$ освободил ресурс"

На лист. 1 можно увидеть простейшую реализацию бинарного семафора или мьютекса [2]. В качестве атомарной проверки внутри функции semaphore\_wait я использую команду mkdir [3]. Эта команда удобна тем что при попытке создания директории с существующим именем она возвращает ошибку.

Проверка существования директории, ее создание при отсутствии и возврат кода в рамках одной команды делает mkdir прекрасной альтернативой атомарной инструкции *CAS* [4] из других языков.

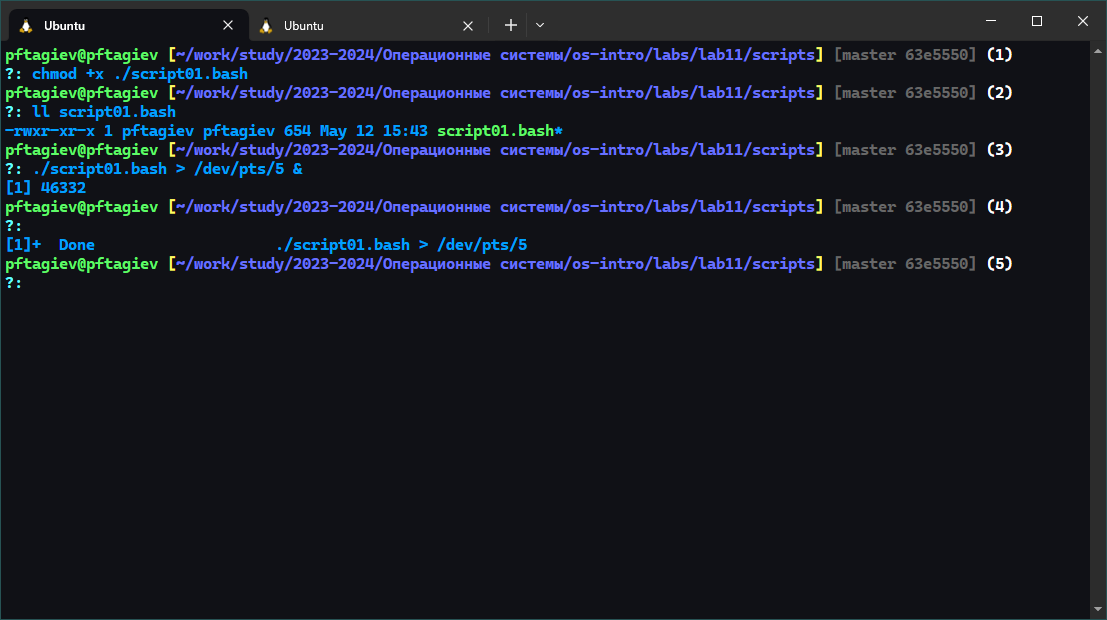


Рис. 1: Запуск скрипта в первом терминале

Начнем выполнение первой части задания. Для удобства сделаем скрипт исполняемым как показано на рис. 1 в промпте (1), затем запустим его в фоне перенаправив вывод в терминал dev/pts/5. В терминале dev/pts/5 запустим тот же командный файл в обычном режиме, как показано на рис. 2.

Как видно все на том же рис. 2 семафор работает корректно и критическую секцию исполняет только один запущенный экземпляр скрипта, пока другой ждет.

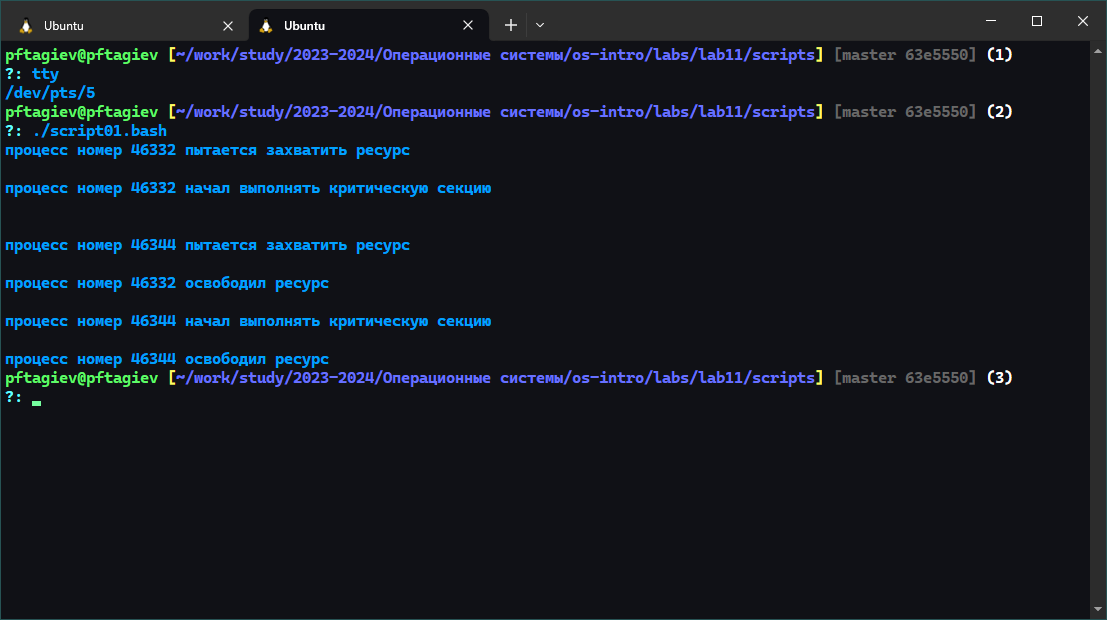


Рис. 2: Запуск скрипта во втором терминале

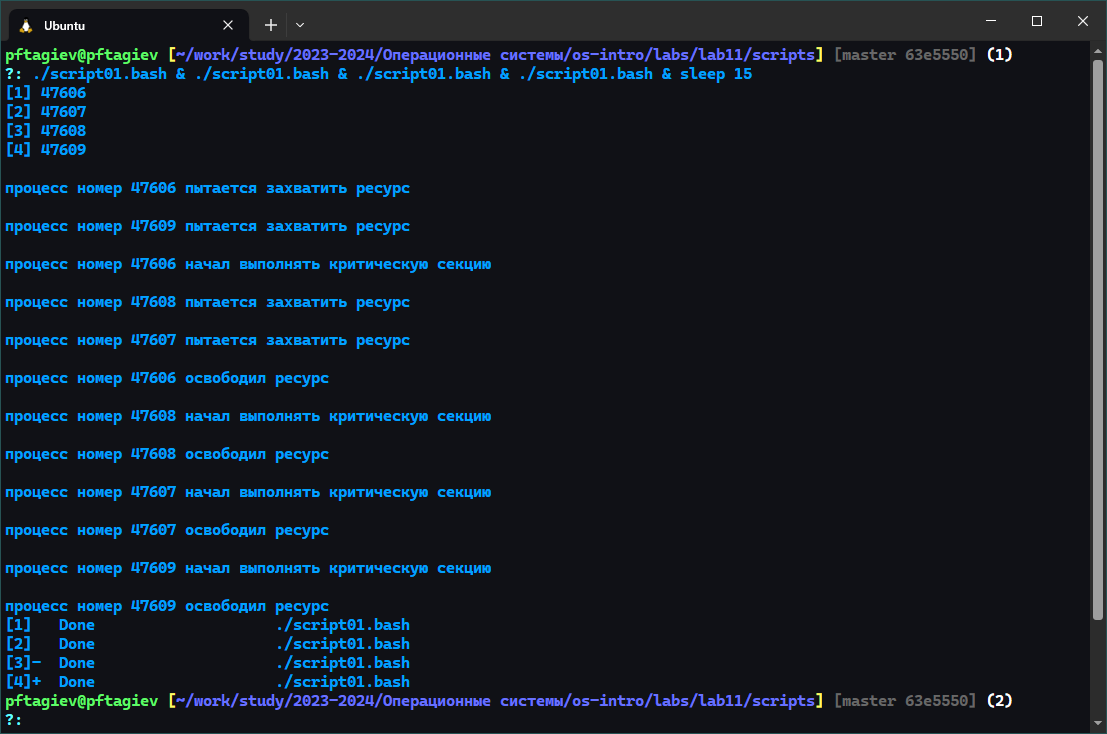


Рис. 3: Запуск трех и более процессов

Пойдем дальше и запустим сразу четыре экземпляра скрипта, как показано на рис. 3 в промте (1). Тут sleep 15 нужен чтобы терминал не создавал новый промпт 15 секунд и не мешал выводу запущенных на фоне процессов. Как видно семафор отлично справляется с тремя и более процессами.

## 3.2 Задание 2

Листинг 2: Реализация команды man

#! /usr/bin/bash  
  
file=/usr/share/man/man1/$1.1.gz # путь к мануалам  
  
if [ -f $file ]; then  
 # если архив существует  
 # читаем его и передаем  
 # интерпретатору языка разметки roff  
 # затем читаем его постранично  
 zcat $file | groff -man -T utf8 -i | less  
else  
 echo Для $1 не существует справки  
fi

В этом задании требуется написать простую реализацию команды man, для этого изучим содержимое директории /usr/share/man/man1, в которой хранятся архивы мануалов команд. Как можно увидеть на рис. 4 имя архива имеет следующий шаблон: имя\_команды.1.gz.

Однако открывая архив просто передав его команде less, как предлагается в задании, я получил совсем не тот результат что ожидал. Как видно на рис. 5 файлы мануалов написаны с использованием специального языка разметки roff. Поэтому, чтобы добиться сопоставимого с командой man вывода, прежде чем печатать файлы мануалов на экран их нужно передать специальному интерпретатору groff, что я и сделал в своей реализации команды man на лист. 2.



Рис. 4: Изучение директории /usr/share/man/man1

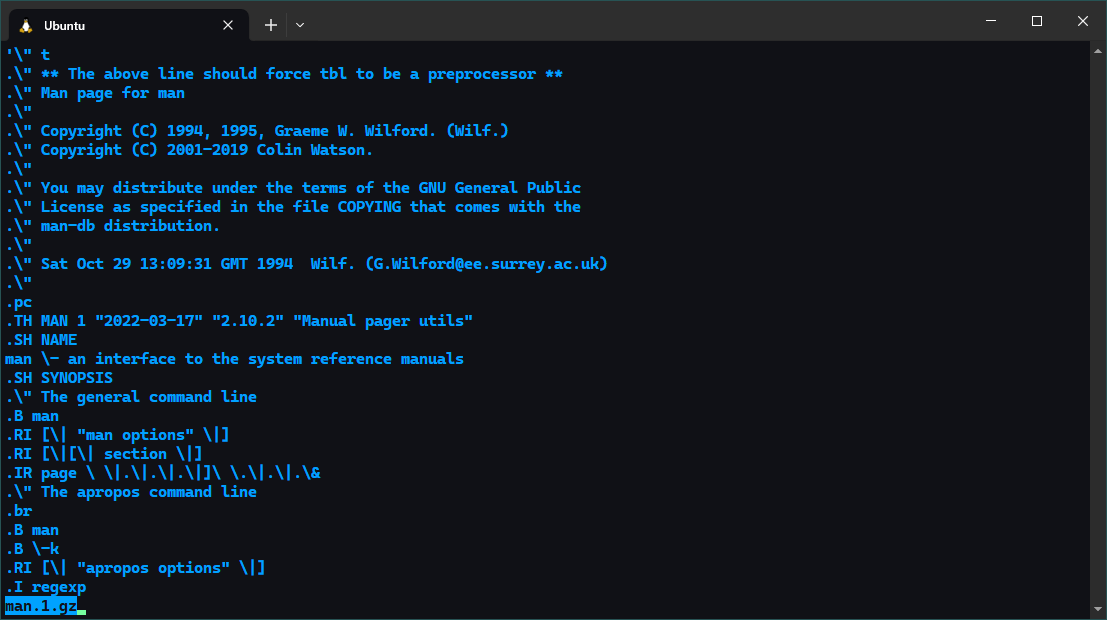


Рис. 5: Вывод напрямую через less

Запустив данную реализацию команды man, как показано на рис. 6, мы получим красивую, обработанную справку к реальной команде man (рис. 7).

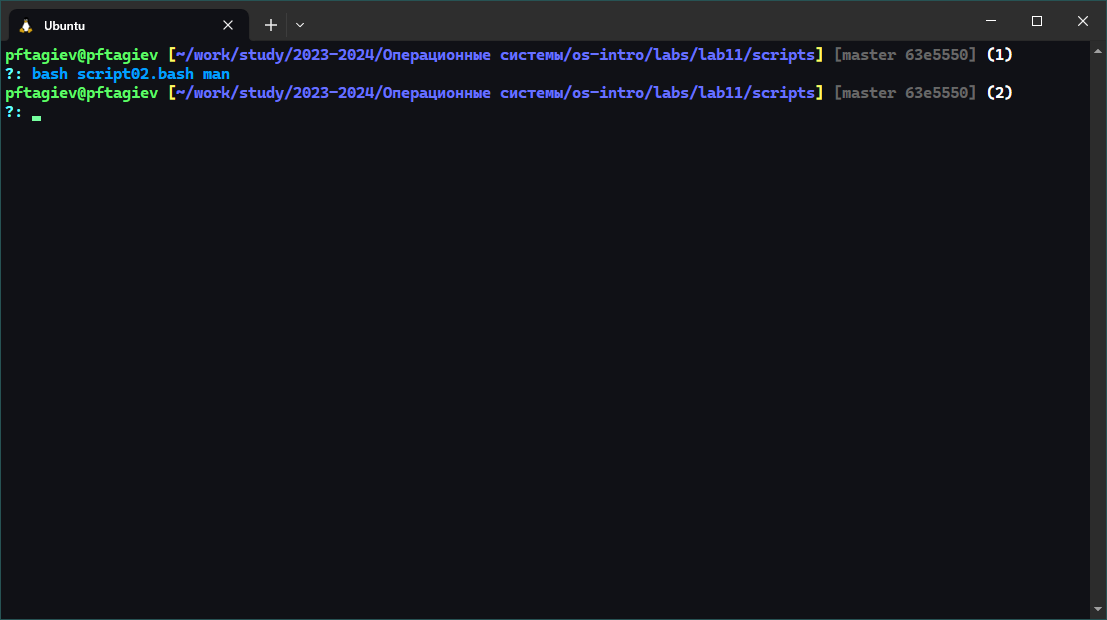


Рис. 6: Запуск реализации команды man

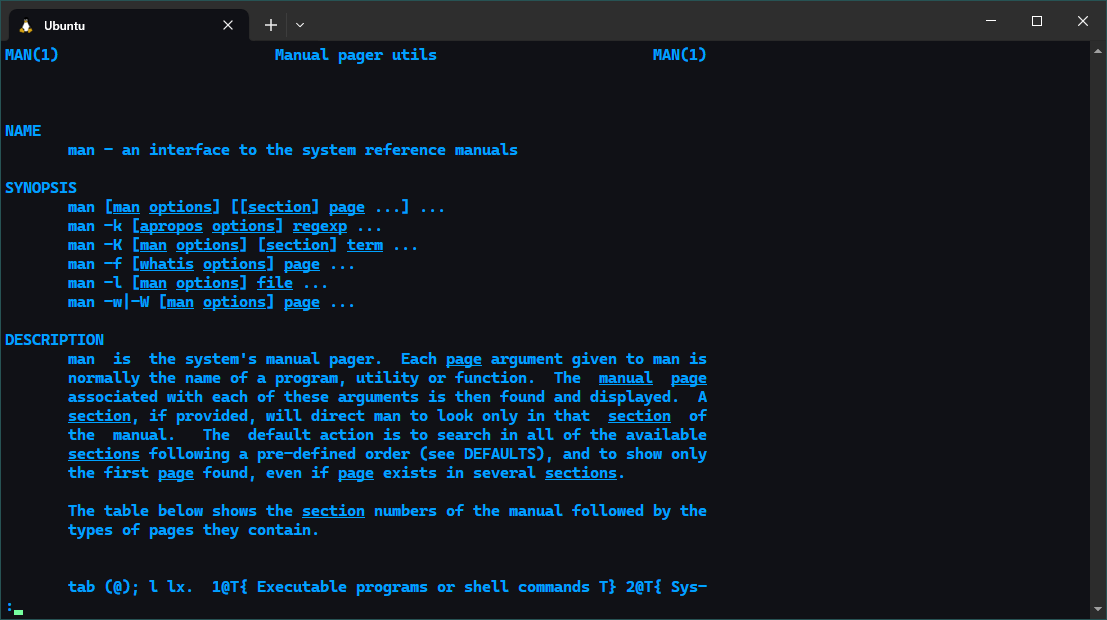


Рис. 7: Обработанный вывод мануала

## 3.3 Задание 3

Листинг 3: Генерация случайной последовательности

#! /usr/bin/bash  
  
lowercase=(a b c d e f g h i j k l m n o p q r s t u v w x y z)  
uppercase=(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)  
  
# все буквы английского алфавита  
letters=("${lowercase[@]}" "${uppercase[@]}")  
  
result=  
  
# по умолчанию длина последовательности 10  
for ((i=${1:-10}; i--;)); do  
 result+=${letters[$RANDOM%${#letters[@]}]}  
done  
  
echo $result

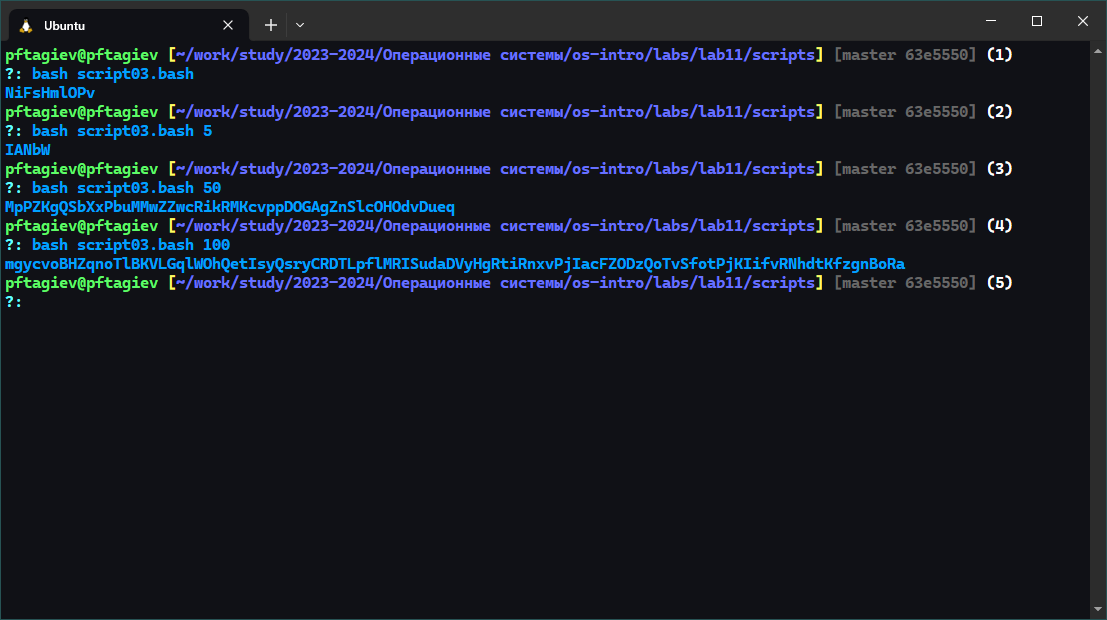


Рис. 8: Пример использования

Следуя заданию, напишем командный файл, который генерирует случайную последовательность букв английского алфавита используя встроенную переменную $RANDOM. Реализацию можно увидеть на лист. 3, а пример использования на рис. 8.

# 4 Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:  
   while [$1 != "exit"]  
   Ошибка в том что после [ и до ] должен идти пробел, еще имеет смысл обернуть $1 в *двойные кавычки*, чтобы эта переменная корректно обрабатывалась даже если она содержит пробелы.
2. Как объединить (конкатенировать) несколько строк в одну?  
   Есть множество способов конкатенации строк в *bash*, я разберу два основных. Предположим у нас есть переменные a и b, чтобы добавить к строке a строку b справа можно использовать следующий синтаксис: a+=$b. Записать результат конкатенации в новую переменную можно так: c="$a$b".
3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?  
   Утилита seq используется для генерации последовательностей, ее можно использовать в циклах. Например: for i in $(seq 0 2 10), переменная i будет принимать значения от 0 до 10 включительно с шагом 2. Очевидно, что чтобы добиться похожего результата можно использовать *Си-подобный* цикл for: for ((i=0;i<=10;i+=2))
4. Какой результат даст вычисление выражения $((10/3))?  
   Результат будет 3, так как будет выполнено целочисленное деление.
5. Укажите кратко основные отличия командной оболочки *zsh* от *bash*.
   * *Интерактивность*: *zsh* известен своими улучшенными интерактивными возможностями, такими как более продвинутый автодополнение команд и файлов, что делает его более удобным для интерактивного использования.
   * *Темы и плагины*: *zsh* поддерживает темы и плагины, что позволяет пользователям настраивать свой рабочий интерфейс и расширять функциональность оболочки.
   * *Синтаксис*: *zsh* обладает более гибким синтаксисом, включая улучшенные возможности для работы с массивами и ассоциативными массивами.
   * *Совместимость*: *zsh* во многом совместим с *bash*, но включает множество дополнительных возможностей, которые могут не работать в *bash* без изменений.
   * *Контекстное автодополнение*: *zsh* предлагает более продвинутое контекстное автодополнение, которое может учитывать не только имена файлов и команд, но и их параметры.
   * *Модульность*: *zsh* разработан с учётом модульности, что позволяет легко добавлять новые функции и интегрировать внешние скрипты.
6. Проверьте, верен ли синтаксис данной конструкции  
   for ((a=1; a <= LIMIT; a++))  
   Да, синтаксис верен. В двойных круглых скобках можно использовать переменные без знака доллара.
7. Сравните язык *bash* с какими-либо языками программирования. Какие преимущества у *bash* по сравнению с ними? Какие недостатки?
   * Преимущества *bash*:
     + *Скорость разработки*: *bash-скрипты* обычно проще и быстрее писать для простых задач, особенно для автоматизации командной строки и системных операций.
     + *Встроенная поддержка UNIX-команд*: *bash* нативно интегрируется с *UNIX-командами* и утилитами, что делает его мощным инструментом для системного администрирования.
     + *Портативность*: *bash-скрипты* легко переносить между различными *UNIX-подобными* системами без изменений.
   * Недостатки *bash*:
     + *Ограниченные возможности программирования*: *bash* не имеет такого богатого набора функций программирования, как *C++* или *Python*, например, объектно-ориентированное программирование или обширные стандартные библиотеки.
     + *Медленная производительность*: Для сложных задач или задач, требующих интенсивных вычислений, *bash* может работать медленнее, чем *C++* или *Python*.
     + *Сложность синтаксиса*: Некоторые аспекты синтаксиса *bash* могут быть непривычными или запутанными для новичков, особенно при работе с текстовыми строками и файлами.

# 5 Выводы

В этой лабораторной работе мы закрепили свои знания об основах программирования на языке *bash*. Написав простейшие реализации семафора и команды man, использовав переменную $RANDOM для генерации последовательности букв английского алфавита.

# Список литературы

1. Кулябов. Операционные системы. Москва: РУДН, 2016. 118 с.

2. Semaphore (programming) [Электронный ресурс]. 2024. URL: <https://en.wikipedia.org/wiki/Semaphore_(programming)>.

3. How can I ensure that only one instance of a script is running at a time (mutual exclusion, locking)? [Электронный ресурс]. 2024. URL: <https://mywiki.wooledge.org/BashFAQ/045>.

4. Compare-and-swap [Электронный ресурс]. 2024. URL: <https://en.wikipedia.org/wiki/Compare-and-swap>.