ОСРВ

1) Работа с командной строкой (составить протокол по выполению всех пунктов)

1) Определить тип файлов /dev/hd0, /dev/console, /dev/ttyp0, /dev/shmem, /dev/mem.

с помощью команды: ls -la

block special, character special, character special, sticky directory,
character special

2) Определить, какой каталог делается рабочим при входе в систему. Почему?

Для определения каталога, который делается рабочим при входе в систему, нужно проверить значение переменной окружения "HOME". Эта переменная содержит путь к домашнему каталогу пользователя, который является его рабочим каталогом по умолчанию при входе в систему.

3) Создать каталог LAB1 и сделать его рабочим.

```
mkdir LAB1; cd LAB1
```

4) Определить (с помощью программы ls), в каком каталоге содержится файл services. Посмотреть его содержимое.

ls /etc/ | grep services; cat /etc/services

5) Сколько скрытых файлов в вашем домашнем каталоге?

```
ls -a | grep "^\\." | wc -l
```

6) Определить полное дерево подкаталогов в /boot . Сколько там файлов, размер которых меньше 1К байт? Сколько там исполняемых файлов?

```
ls -R /boot - рекурсивный вывод содержимого в boot
find /boot -type f -size -1k | wc -l - подсчет количества файлов, размером 1Кб
sudo find /boot -type f -executable | wc -l - поиск и подсчет всех исполняемых файлов
```

7) Сколько жестких связей у каталога /boot и почему?

У каталога /boot нет жестких связей, потому что он является корневым каталогом файловой системы и не может быть связан с другими каталогами. Жесткие связи могут быть созданы только для обычных файлов и каталогов внутри файловой системы.

8) Создать текстовый файл с помощью редактора vi. Какие флаги доступа устанавливаются у вновь создаваемого файла? Почему? Как это исправить?

Файл создан с помощью vi. Флаги доступа при создании файла 664. Изменить права доступа можно с помощью команды chmod

9) Сделать каталог и создать в нем 10 копий некоторого файла. Перенести три из них в вышестоящий каталог. Удалить (с подтверждением) некоторые из оставшихся файлов. Проверить влияние флага w на команду удаления файла.

```
mkdir mydir
cp myfile.txt mydir/ - копирование файла в папку mydir
cd mydir - переход в папку
for i in {1..10}; do cp myfile.txt "myfile$i.txt"; done
mv myfile{1..3}.txt ..
rm -i myfile{4..10}.txt
```

Флаг w (write) означает, что файл можно записывать. Если у файла нет прав на запись, то удалить его не получится.

10) Определить значения переменных среды PATH, LOGNAME, HOME, HOSTNAME, PWD, RANDOM. Меняются ли они со временем?

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:
LOGANAME=vladimir
HOME=/home/vladimir
HOSTNAME=vladimir-IdeaPad-3-15ADA05
PWD=/home/vladimir (Изменяется)
RANDOM=31245 (Изменяется)
```

11) Определить коды завершения команд ls /bin и ls /pin

```
0 - успех, 2 - ошибка доступа
```

12) 1. Вывести содержимое каталога /bin в файл в несколько колонок. Затем добавить к нему распечатку каталога /usr/bin

```
ls -C /bin > file.txt && ls -a /usr/bin >> file.txt
```

13) Сколько файлов удалили бы команды rm /usr/bin/g* и rm /usr/bin/t?? ? (просьба файлы не удалять)

```
ls -la /usr/bin/g* | wc -l,  ls -la /usr/bin/t??| wc -l
```

14) Сколько всего пользователей зарегистрировано в системе?

```
cat /etc/passwd | wc -l
```

15) Сколько различных групп пользователей в системе?

```
cat /etc/group | wc -l
```

16) Определить имена пользователей, у которых нет пароля.

```
sudo getent shadow | grep '^[^:]*:.\?:' | cut -d: -f1
```

17) Защитить файл для чтения со стороны владельца, проверить.

```
chmod o-r file.txt
```

18) Защитить файл для чтения со стороны других пользователей, проверить.

```
chmod o-r file.txt
```

19) Защитить файл для записи со стороны владельца, проверить.

```
chmod u-w file.txt
```

20) Защитить файл для записи со стороны других пользователей, проверить.

```
chmod o-w file.txt
```

21) Открыть / закрыть свой основной каталог для доступа со стороны других пользователей, проверить.

```
chmod o-r ~ && chmod o+r ~
```

22) Разрешить доступ к своему основному каталогу, но запретить его изменение, проверить.

```
chmod -w ~
```

23) Разрешить доступ к файлам только с известными именами, проверить.

```
chmod 400 file1.txt
chmod 400 file2.txt
chmod 400 file3.txt

chmod 000 *
```

2) Создание простых скриптов

1) Написать скрипт, который просто выводит значения переданных ему параметров.

```
#!/bin/bash

# Цикл для обхода всех переданных параметров
for param in "$@"; do
    echo "$param"
done
```

2) Написать скрипт, который с помощью утилит pidin и grep выводит на экран информацию об указанном по имени процессе.

```
#!/bin/bash

# Проверяем наличие аргумента с именем процесса
if [ -z "$1" ]; then
  echo "Использование: $0 <имя_процесса>"
```

```
  exit 1
fi

# Получаем идентификаторы процессов по имени
pids=$(pidof $1)

# Проверяем, найдены ли процессы
if [ -z "$pids" ]; then
  echo "Процесс с именем '$1' не найден."
  exit 1
fi

# Выводим информацию о каждом процессе
for pid in $pids; do
  echo "Информация о процессе с PID $pid:"
  pidin $pid | grep -E "PID|Name|State|PPID|Threads"
  echo "------------------------------"
done
```

3) Написать скрипт, который компилирует указанную программу и при отсутствии ошибок запускает её. Если же есть ошибки, то автоматически вызывает редактор для их исправления.

```
#!/bin/bash

# Проверяем наличие аргумента с именем исходного файла
if [ -z "$1" ]; then
  echo "Использование: $0 <файл.c>"
  exit 1
fi

# Имя исходного файла
source_file="$1"

# Имя исполняемого файла (без расширения)
executable_file="${source_file%.*}"

# Компилируем программу
gcc -o "$executable_file" "$source_file"

# Проверяем наличие ошибок компиляции
if [ $? -ne 0 ]; then
  echo "Компиляция завершилась с ошибками. Открываем редактор..."
  # Открываем редактор для исправления ошибок
  nano "$source_file"
  exit 1
fi

# Запускаем исполняемый файл
./"$executable_file"
```

3) Разработка программ

1) Написать программу, выводящую сообщение "HELLO" в центре чистого экрана.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>

void clear_screen() { system("clear"); }

void print_center(const char *message) {
  int terminal_width;
  int terminal_height;
  int message_length;
  int padding;
```

```
  struct winsize w;
  ioctl(0, TIOCGWINSZ, &w);
  terminal_width = w.ws_col;
  terminal_height = w.ws_row;

  message_length = strlen(message);

  padding = terminal_height / 2;
  for (int i = 0; i < padding; i++) {
    printf("\n");
  }
  padding = (terminal_width - message_length) / 2;

  for (int i = 0; i < padding; i++) {
    printf(" ");
  }

  printf("%s\n", message);
  padding = terminal_height / 2;

  for (int i = 0; i < padding; i++) {
    printf("\n");
  }
}

int main() {
  clear_screen();
  print_center("HELLO");
  return 0;
}
```

2) Написать программу, позволяющую определять коды нажимаемых клавиш и восстанавливающую исходный вид терминала (цвет, курсор) при выходе.

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

struct termios original_termios;

void restore_terminal() {
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &original_termios);
}

void enable_raw_mode() {
  struct termios raw_termios;

  // Получаем исходные настройки терминала
  tcgetattr(STDIN_FILENO, &original_termios);
  atexit(restore_terminal);

  raw_termios = original_termios;

  // Отключаем канонический (строчный) режим
  raw_termios.c_lflag &= ~(ICANON | ECHO);

  // Отключаем обработку сигналов и символьных кодов
  raw_termios.c_lflag &= ~ISIG;
  raw_termios.c_iflag &= ~(IXON | ICRNL);

  // Отключаем вывод символов в зависимости от их значения
  raw_termios.c_iflag &= ~ISTRIP;
  raw_termios.c_oflag &= ~OPOST;
  raw_termios.c_cflag |= CS8;

  // Устанавливаем минимальное количество символов и время ожидания
  raw_termios.c_cc[VMIN] = 0;
  raw_termios.c_cc[VTIME] = 1;
```

```
  // Применяем новые настройки терминала
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw_termios);
}

int main() {
  enable_raw_mode();

  char c;
  while (1) {
    // Чтение символа с клавиатуры
    if (read(STDIN_FILENO, &c, 1) == -1) {
      perror("read");
      exit(1);
    }

    // Если нажата клавиша 'q', выходим из программы
    if (c == 'q') {
      break;
    }

    // Выводим код нажатой клавиши
    printf("Код клавиши: %d\n", c);
  }

  // Восстанавливаем исходные настройки терминала
  restore_terminal();

  return 0;
}
```

3) Написать программу, рисующую движущийся символ (при выключенном курсоре, без использования функции стирания экрана)

```
#include <stdio.h>
#include <unistd.h>

void clear_line() {
  printf("\r\033[K");
  fflush(stdout);
}

int main() {
  // Выключаем отображение курсора
  printf("\033[?25l");
  fflush(stdout);

  int position = 0;
  int direction = 1;

  while (1) {
    clear_line();

    // Выводим пробелы перед символом
    for (int i = 0; i < position; i++) {
      printf(" ");
    }

    // Выводим символ
    printf("X");
    fflush(stdout);

    // Задержка для создания эффекта движения
    usleep(10000);

    // Изменяем позицию символа
    position += direction;

    // Изменяем направление при достижении границ экрана
    if (position == 0 || position == 50) {
      direction *= -1;
    }
```

```
  }

  // Включаем отображение курсора перед выходом
  printf("\033[?25h");
  fflush(stdout);

  return 0;
}
```

4) Написать программу, рисующую бесконечно движущийся символ. Характер движения (скорость, направление, цвет и т.д.) задавать с помощью параметров командной строки. Предусмотреть восстановление параметров дисплея (цвет, курсор) при принудительном завершении программы. Осуществить запуск нескольких экземпляров программы с разными параметрами движения (запуск с одного терминала, вывод на другой)

```
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>

volatile sig_atomic_t stop;

void handle_signal(int signal) { stop = 1; }

void clear_screen() {
  printf("\033[2J");
  printf("\033[1;1H");
  fflush(stdout);
}

void move_cursor(int x, int y) {
  printf("\033[%d;%dH", y, x);
  fflush(stdout);
}

void set_color(int color) {
  printf("\033[38;5;%dm", color);
  fflush(stdout);
}

int main(int argc, char *argv[]) {
  if (argc < 5) {
    printf(
        "Usage: ./moving_symbol <symbol> <color> <x_velocity> <y_velocity>\n");
    return 1;
  }

  char symbol = argv[1][0];
  int color = atoi(argv[2]);
  int x_velocity = atoi(argv[3]);
  int y_velocity = atoi(argv[4]);

  struct termios original_termios;
  tcgetattr(STDIN_FILENO, &original_termios);

  struct termios raw_termios = original_termios;
  raw_termios.c_lflag &= ~(ECHO | ICANON);
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw_termios);

  int original_flags = fcntl(STDIN_FILENO, F_GETFL);
  fcntl(STDIN_FILENO, F_SETFL, original_flags | O_NONBLOCK);

  signal(SIGINT, handle_signal);

  int x = 1;
  int y = 1;
  int x_max, y_max;
```

```
  while (!stop) {
    clear_screen();
    move_cursor(x, y);
    set_color(color);
    printf("%c", symbol);
    fflush(stdout);

    usleep(100000);

    // Расчет новой позиции
    x += x_velocity;
    y += y_velocity;

    // Получение размера терминала
    printf("\033[999B\033[999C");
    fflush(stdout);
    printf("\033[6n");
    fflush(stdout);
    scanf("\033[%d;%dR", &y_max, &x_max);

    if (x > x_max) {
      x = 1;
    } else if (x < 1) {
      x = x_max;
    }

    if (y > y_max) {
      y = 1;
    } else if (y < 1) {
      y = y_max;
    }
  }

  tcsetattr(STDIN_FILENO, TCSAFLUSH, &original_termios);
  fcntl(STDIN_FILENO, F_SETFL, original_flags);

  return 0;
}
```