Разработайте bash-скрипт **os_0007.sh**, который принимает два параметра: Pid процесса, строку fd (необязательный параметр). В результате работы bash-скрипт выводит в консоль: наименование исполняемого файла, Pid-процесса, Pid-родительского процесса (PPid), перечень дескрипторов (номеров) дескрипторов (fd) открытых потоков. Приведенные ниже скриншоты демонстрируют работу скрипта os_0007.sh. Лабораторная работа 03б

ПСКП

**Задание 01**

1. Разработайте функцию ***firstJob***, которая будет возвращать Promise.
2. Promise должен разрешаться успешно со значением «Hello World» через 2 секунды после вызова функции ***firstJob***.
3. Вызовите функцию ***firstJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await с try/catch.

**Задание 02**

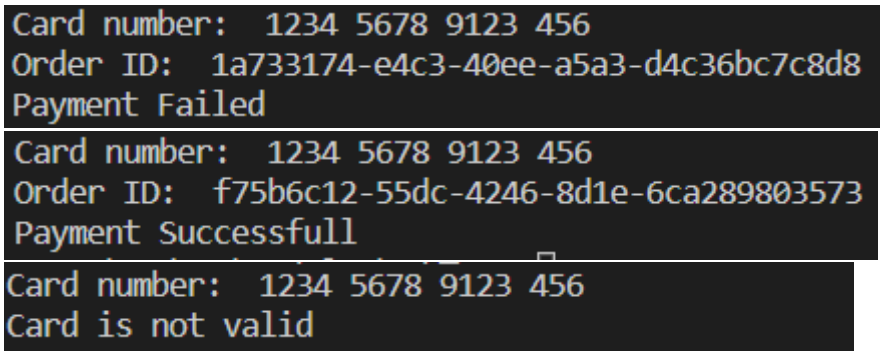
4. Разработайте функцию ***secondJob***, которая будет возвращать Promise.
5. Promise должен отклоняется с сообщением об ошибке через 3 секунды.
6. Вызовите функцию ***secondJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await с try/catch.

**Задание 03**

7. Разработайте еще одну функцию ***thirdJob***, принимающую параметр data и возвращающую Promise.
8. Если data не является числом, ***немедленно*** вернуть отклоненный Promise со значением «error».
9. Если data является нечетным числом, вернуть через 1 секунду успешно разрешенный Promise со значением «odd».
10. Если data является четным числом, вернуть через 2 секунды отклоненный Promise со значением «even».
11. Вызовите функцию ***thirdJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await с try/catch.

## Задание 04

12. Разработайте функцию ***createOrder***, в которой будет создаваться Promise. Эта функция должна принимать в качестве параметра номер карты покупателя, проверять ее, а также назначать идентификатор заказу.
    А) Если карта покупателя невалидна, то отклонять Promise с ошибкой «Card is not valid».
    Б) Если же карта прошла проверку, то генерировать номер заказа (например, с помощью uuid) и успешно разрешать Promise с этим номером спустя 5 сек.
13. Для проверки карты необходимо создать функцию ***validateCard***. Она должна принимать номер карты, выводить его на консоль и рандомно возвращать true или false.
14. Также разработайте функцию ***proceedToPayment***, которая должна вызываться после ***createOrder***, если проверка карты прошла успешно. В ней необходимо принимать номер заказа, выводить его на консоль и возвращать Promise, который рандомно разрешается либо с успешным значением «Payment successfull», либо с ошибкой «Payment failed».
15. Вызовите функции ***createOrder*** и ***proceedToPayment*** в правильном порядке и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await с try/catch. Должны получаться следующие результаты:




## Задание 05

16. Разработайте три отдельные функции для вычисления квадрата, куба и четвертой степени заданного числа. Каждая из функций должна возвращать Promise, который либо разрешается с вычисленным значением, либо отклоняется с сообщением об ошибке, если ввод недействителен.

17. Далее используйте ***Promise.all()*** для вычисления этих функций.
18. Обработайте результат с помощью обработчиков Promise. Протестируйте работу с правильным и неправильным вводом.

**Задание 06**

19. Используйте задание 5. Добавьте к каждой функции разрешение Promise спустя некоторый промежуток времени.
20. Используйте ***Promise.race()*** для вычисления математических функций.
21. Замените использование на ***Promise.any()*** для получения первого Promise, который разрешается.

**Задание 07**

22. Перепишите код со скриншота ниже и выполните его. Объясните полученный результат.
23. На примере этого кода объясните, как работает Event Loop.

```
function f1() {
    console.log('f1');
}

function f2() {
    console.log('f2');
}

function f3() {
    console.log('f3');
}

function main() {
    console.log('main');

    setTimeout(f1, 50);
    setTimeout(f3, 30);

    new Promise((resolve, reject) =>
        resolve('I am a Promise, right after f1 and f3! Really?')
    ).then(resolve => console.log(resolve));

    new Promise((resolve, reject) =>
        resolve('I am a Promise after Promise!')
    ).then(resolve => console.log(resolve));

    f2();
}

main();
```

**Задание 08.** Ответьте следующие на вопросы.

24. Что такое **callback**?
25. В чем минусы использования коллбэков? Какие есть способы их решения?
26. Что такое **Promise** и как он работает?
27. В каких **состояниях** может находиться Promise?
28. Как изменить состояние Promise?
29. Как изменить значение Promise?
30. Что такое **цепочки промисов** и как они работает?
31. Назовите два способа обработки ошибок в Promise.
32. Для чего нужен метод **Promise.all()**?
33. В чем отличия методов **Promise.race()** и **Promise.any()**?
34. Что такое **async/await**?