МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Управление процессами в Python»

Отчет по лабораторной работе № 2.25

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Кучеренко С. Ю. _____« » 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

Цель работы: приобретение навыков написания многозадачных приложений на языке программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process


def func():
    print("Hello from child Process")


if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
```

```
Hello from main Process
Hello from child Process

Process finished with exit code 0
```

Рисунок 1 – Результат работы программы

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process


def func():
    print("Hello from child Process")


if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
    print("Goodbye")
```

```
Hello from main Process
Goodbye
Hello from child Process

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process


def func():
    print("Hello from child Process")


if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
    proc.join()
    print("Goodbye")
```

```
Hello from main Process
Hello from child Process
Goodbye

Process finished with exit code 0
```

Рисунок 3 – Результат работы программы

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process


def func():
    print("Hello from child Process")


if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
    print(f"Proc is_alive status: {proc.is_alive()}")
    proc.join()
    print("Goodbye")
    print(f"Proc is_alive status: {proc.is_alive()}")
```

```
Hello from main Process
Proc is_alive status: True
Hello from child Process
Goodbye
Proc is_alive status: False

Process finished with exit code 0
```

Рисунок 4 – Результат работы программы

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process
from time import sleep


class CustomProcess(Process):
    def __init__(self, limit):
        Process.__init__(self)
        self._limit = limit
    def run(self):
        for i in range(self._limit):
            print(f"From CustomProcess: {i}")
            sleep(0.5)

if __name__ == "__main__":
    cpr = CustomProcess(3)
    cpr.start()
```

```
From CustomProcess: 0
From CustomProcess: 1
From CustomProcess: 2

Process finished with exit code 0
```

Рисунок 5 – Результат работы программы

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process
from time import sleep


def func():
    counter = 0
    while True:
        print(f"counter = {counter}")
        counter += 1
        sleep(0.1)
```

```
if __name__ == "__main__":
    proc = Process(target=func)
    proc.start()
    sleep(0.7)
    proc.terminate()
```

```
counter = 0
counter = 1
counter = 2
counter = 3
counter = 4
counter = 5
counter = 6

Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from multiprocessing import Process
from time import sleep


def func(name):
    counter = 0
    while True:
        print(f"proc {name}, counter = {counter}")
        counter += 1
        sleep(0.1)


if __name__ == "__main__":
    proc1 = Process(target=func, args=("proc1",), daemon=True)
    proc2 = Process(target=func, args=("proc2",))
    proc2.daemon = True
    proc1.start()
    proc2.start()
    sleep(0.3)
```

```
proc proc2, counter = 0
proc proc1, counter = 0
proc proc2, counter = 1
proc proc1, counter = 1
proc proc1, counter = 2
proc proc2, counter = 2

Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

4. Выполните клонирование созданного репозитория.

5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

6. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

7. Создайте проект PyCharm в папке репозитория.

8. Для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from multiprocessing import Process, Queue
import math

EPS = .0000001


def inf_sum(x, out):
    summa = 1.0
    temp = 0
    n = 1
    while abs(summa - temp) > EPS:
        temp = summa
        summa += math.sin(n * x) / n
        n += 1

    out.put(summa)
```

```
def check(x, out):
    res = - math.log(2 * math.sin(0.5 * x))

    out.put(res)


if __name__ == '__main__':
    x = math.pi

    out1 = Queue()
    out2 = Queue()
    process_1 = Process(target=inf_sum, args=(x, out1))
    process_2 = Process(target=check, args=(x, out2))
    process_1.start()
    process_2.start()
    result1 = out1.get()
    result2 = out2.get()
    process_1.join()
    process_2.join()

    print(f"The sum is: {result1}")
    print(f"The check sum is: {result2}")
```

```
The sum is: 1.0000000000000002
The check sum is: -0.6931471805599453

Process finished with exit code 0
```

Рисунок 8 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main (master).
11. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы:

1. Как создаются и завершаются процессы в Python?

Процессы в *Python* позволяют запустить выполнение нескольких задач в параллельном режиме. По сути, при старте процесса запускает еще одна копия интерпретатора *Python*, в котором выполняется указанная функция. Таким образом, если мы запустим пять процессов, то будет запущено пять отдельных интерпретаторов, в этом случае уже не будет проблем с *GIL*. Такой способ позволяет параллельно запускать задачи активно использующие *CPU*. Они будут распределяться между несколькими процессами (ядрами), что значительно увеличит производительность вычислений.

2. В чем особенность создания классов-наследников от Process?

В классе наследнике от *Process* необходимо переопределить метод *run()* для того, чтобы он (класс) соответствовал протоколу работы с процессами. Ниже представлен пример с реализацией этого подхода.

3. Как выполнить принудительное завершение процесса?

В отличии от потоков, работу процессов можно принудительно завершить, для этого класс *Process* предоставляет набор методов:

- `terminate()` - принудительно завершает работу процесса. В *Unix* отправляется команда *SIGTERM*, в *Windows* используется функция *TerminateProcess()*.
- `kill()` - метод аналогичный *terminate()* по функционалу, только вместо *SIGTERM* в *Unix* будет отправлена команда *SIGKILL*.

4. Что такое процессы-демоны? Как запустить процесс-демон?

Процессы демоны по своим свойствам похожи на потоки-демоны, их суть заключается в том, что они завершают свою работу, если завершился родительский процесс.

Указание на то, что процесс является демоном должно быть сделано до его запуска (до вызова метода *start()*). Для демонического процесса запрещено самостоятельно создавать дочерние процессы. Эти процессы не являются демонами (сервисами) в понимании *Unix*, единственное их свойство – это завершение работы вместе с родительским процессом.

Указать на то, что процесс является демоном можно при создании экземпляра класса через аргумент *daemon*, либо после создания через свойство *daemon*.