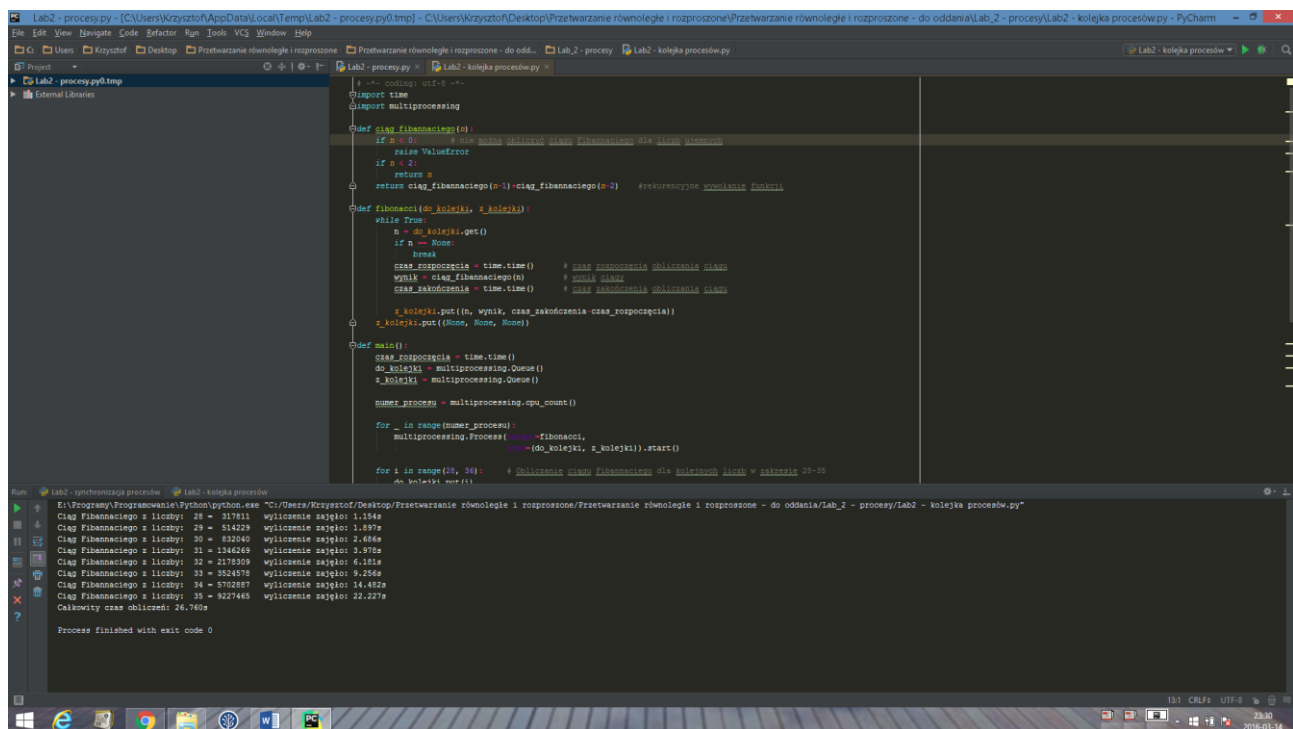


# OOR: Programowanie równoległe i rozproszone

## Laboratorium 2:

Za pomocą wybranego przez siebie języka programowania zademonstruj tworzenie i współpracę procesów.



```
Lab2 - procesy.py [C:\Users\Krzysztof\AppData\Local\Temp\Lab2 - procesy.py0.tmp] - C:\Users\Krzysztof\Desktop\Przetwarzanie równoległe i rozproszone\Przetwarzanie równoległe i rozproszone - do oddania\Lab2 - procesy\Lab2 - kolejka procesów.py - PyCharm
```

```
import time
import multiprocessing

def ciag_fibonnaciego(n):
    if n < 0:
        raise ValueError
    if n < 2:
        return n
    return ciag_fibonnaciego(n-1)+ciag_fibonnaciego(n-2) #rekurencyjne wywołanie funkcji

def fibonnaci(do_kolejki, z_kolejki):
    while True:
        n = do_kolejki.get()
        if n == None:
            break
        czas_rozpoczecia = time.time() # czas rozpoczęcia obliczenia ciągu
        wynik = ciag_fibonnaciego(n) # wynik ciągu
        czas_zakończona = time.time() # czas zakończenia obliczenia ciągu
        z_kolejki.put((n, wynik, czas_zakończona - czas_rozpoczecia))
        z_kolejki.put((None, None, None))

def main():
    czas_rozpoczecia = time.time()
    do_kolejki = multiprocessing.Queue()
    z_kolejki = multiprocessing.Queue()

    numer_procesow = multiprocessing.cpu_count()

    for _ in range(numer_procesow):
        multiprocessing.Process(target=fibonnaci, args=(do_kolejki, z_kolejki)).start()

    for i in range(28, 34): # obliczenie ciągu fibonnaciego dla kolejnych liczb = zakres 28-34
        do_kolejki.put(i)

    do_kolejki.put(None)
```

Lab2 - synchronizacja procesów    Lab2 - kolejka procesów

```
E:\Programy\Programowanie\Python\python.exe "C:\Users\Krzysztof\Desktop\Przetwarzanie równoległe i rozproszone\Przetwarzanie równoległe i rozproszone - do oddania\Lab2 - procesy\Lab2 - kolejka procesów.py"
Ciąg Fibonnaciego z liczb: 28 = 317811    wyliczenie zajęło: 1.154s
Ciąg Fibonnaciego z liczb: 29 = 514229    wyliczenie zajęło: 1.497s
Ciąg Fibonnaciego z liczb: 30 = 832040    wyliczenie zajęło: 2.486s
Ciąg Fibonnaciego z liczb: 31 = 1346269    wyliczenie zajęło: 3.978s
Ciąg Fibonnaciego z liczb: 32 = 2178109    wyliczenie zajęło: 6.355s
Ciąg Fibonnaciego z liczb: 33 = 3524578    wyliczenie zajęło: 9.256s
Ciąg Fibonnaciego z liczb: 34 = 5702887    wyliczenie zajęło: 14.492s
Ciąg Fibonnaciego z liczb: 35 = 9227465    wyliczenie zajęło: 22.227s
Całkowity czas obliczeń: 34.760s

Process finished with exit code 0
```

Przykład ilustrujący uruchamianie wielu wątków i równoległe ich wykonywanie.

Czas wykonywania kolejnych procesów jest różny, dlatego o kolejności zakończenia nie decyduje kolejność uruchomienia, lecz czas wykonywania danego wątku (dlatego, że wątki są wykonywane niezależnie i szybciej skończą swoje działanie wątki o krótszym czasie wykonywania).

### Kod programu obsługującego wiele procesów w języku Python:

```
# -*- coding: utf-8 -*-
import multiprocessing
import time

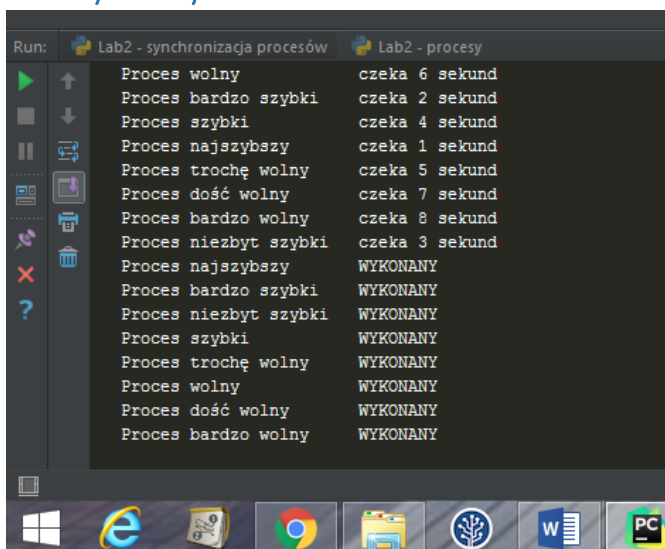
dane = ([ 'bardzo szybki ', '2'], [ 'szybki ', '4'], [ 'wolny ', '6'], [ 'bardzo wolny ', '8'], # tej samej długości nazwy procesów, by wydruk
      [ 'najszybszy ', '1'], [ 'niezbyt szybki', '3'], [ 'trochę wolny ', '5'], [ 'dość wolny ', '7']
    )

def odczyt_wielopprocesowy(procesy):
    for indane in procesy: # pętla uruchamiająca kolejne procesy z tablicy
        p = multiprocessing.Process(target=wydruk_wielopprocesowy,
        args=(indane[0], indane[1]))
        p.start()

def wydruk_wielopprocesowy(nazwa_procesu, czas_oczekiwania):
    print (" Proces %s\t czeka %s sekund" % (nazwa_procesu, czas_oczekiwania))
    time.sleep(int(czas_oczekiwania)) # oczekuje określony w tablicy czas
    print (" Proces %s\t WYKONANY" % nazwa_procesu) # informacja o wykonanych procesach

if __name__ == '__main__':
    odczyt_wielopprocesowy(dane)
```

### Kod wynikowy:



Kod programu obliczającego ciąg Fibannaciego dla liczb w zakresie 28-30, obsługującego wiele procesów, z synchronizacją procesów i kolejką, w języku Python:

```
# -*- coding: utf-8 -*-
import time
import multiprocessing

def ciąg_fibannaciego(n):
    if n < 0: # nie można obliczyć ciągu Fibannaciego dla liczb ujemnych
        raise ValueError
    if n < 2:
        return n
    return ciąg_fibannaciego(n-1)+ciąg_fibannaciego(n-2) #rekurencyjne
#wywołanie funkcji

def fibonacci(do_kolejki, z_kolejki):
    while True:
        n = do_kolejki.get()
        if n == None:
            break
        czas_rozpoczęcia = time.time() # czas rozpoczęcia obliczania ciągu
        wynik = ciąg_fibannaciego(n) # wynik ciągu
        czas_zakończenia = time.time() # czas zakończenia obliczania ciągu

        z_kolejki.put((n, wynik, czas_zakończenia-czas_rozpoczęcia))
        z_kolejki.put((None, None, None))

def main():
    czas_rozpoczęcia = time.time()
    do_kolejki = multiprocessing.Queue()
    z_kolejki = multiprocessing.Queue()

    numer_procesu = multiprocessing.cpu_count()

    for _ in range(numer_procesu):
        multiprocessing.Process(target=fibonacci,
                                args=(do_kolejki, z_kolejki)).start()

    for i in range(28, 36): # Obliczanie ciągu Fibannaciego dla kolejnych
        #liczb w zakresie 28-35
        do_kolejki.put(i)

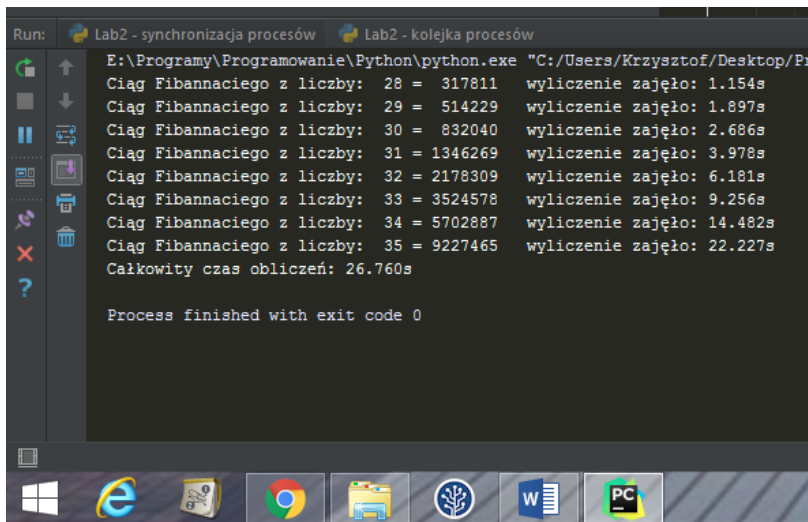
    for _ in range(numer_procesu):
        do_kolejki.put(None)

    while True:
        n, wynik, czas_wyliczenia = z_kolejki.get()
        if n == None:
            numer_procesu -= 1
            if numer_procesu == 0: break
        else:
            print ("Ciąg Fibannaciego z liczby: %3d = %7d   wyliczenie zajęło:
%0.3fs" % ( n, wynik, czas_wyliczenia))

            czas_zakończenia = time.time()
            print ("Całkowity czas obliczeń: %0.3fs" % (czas_zakończenia -
czas_rozpoczęcia))

if __name__ == "__main__":
    main()
```

## Kod wynikowy:



```
Run: Lab2 - synchronizacja procesów Lab2 - kolejka procesów
E:\Programy\Programowanie\Python\python.exe "C:/Users/Krzysztof/Desktop/P...
Ciąg Fibannaciego z liczby: 28 = 317811 wyliczenie zajęło: 1.154s
Ciąg Fibannaciego z liczby: 29 = 514229 wyliczenie zajęło: 1.897s
Ciąg Fibannaciego z liczby: 30 = 832040 wyliczenie zajęło: 2.686s
Ciąg Fibannaciego z liczby: 31 = 1346269 wyliczenie zajęło: 3.978s
Ciąg Fibannaciego z liczby: 32 = 2178309 wyliczenie zajęło: 6.181s
Ciąg Fibannaciego z liczby: 33 = 3524578 wyliczenie zajęło: 9.256s
Ciąg Fibannaciego z liczby: 34 = 5702887 wyliczenie zajęło: 14.482s
Ciąg Fibannaciego z liczby: 35 = 9227465 wyliczenie zajęło: 22.227s
Całkowity czas obliczeń: 26.760s

Process finished with exit code 0
```

Synchronizacja procesów jest niezbędna, gdy operacje są od siebie zależne i muszą być wykonywane jedna po drugiej (do wykonania akcji 2, wymagane jest zakończenie akcji 1).