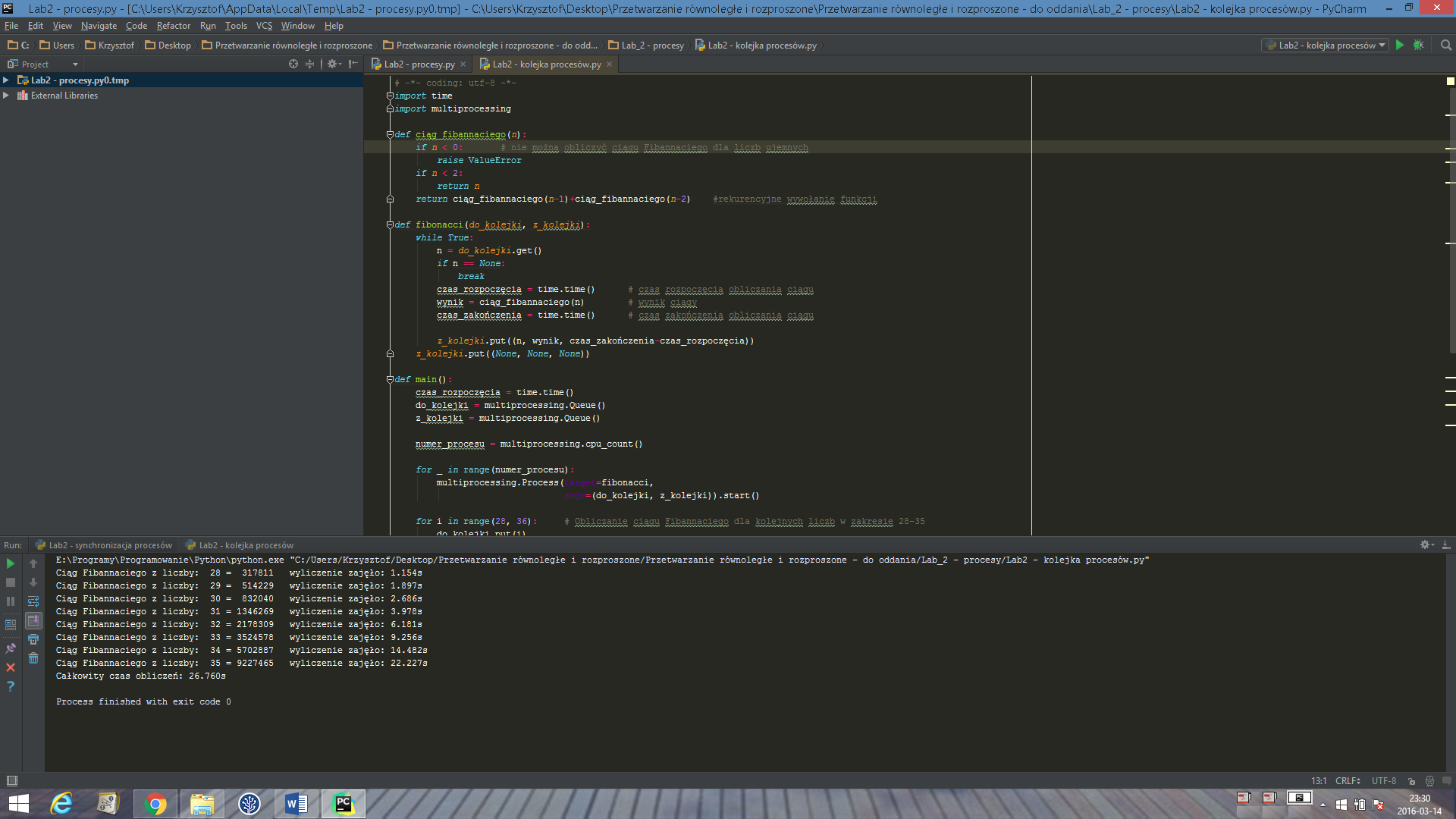
# **OOR: Programowanie równoległe i rozproszone**

## **Laboratorium 2:**

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

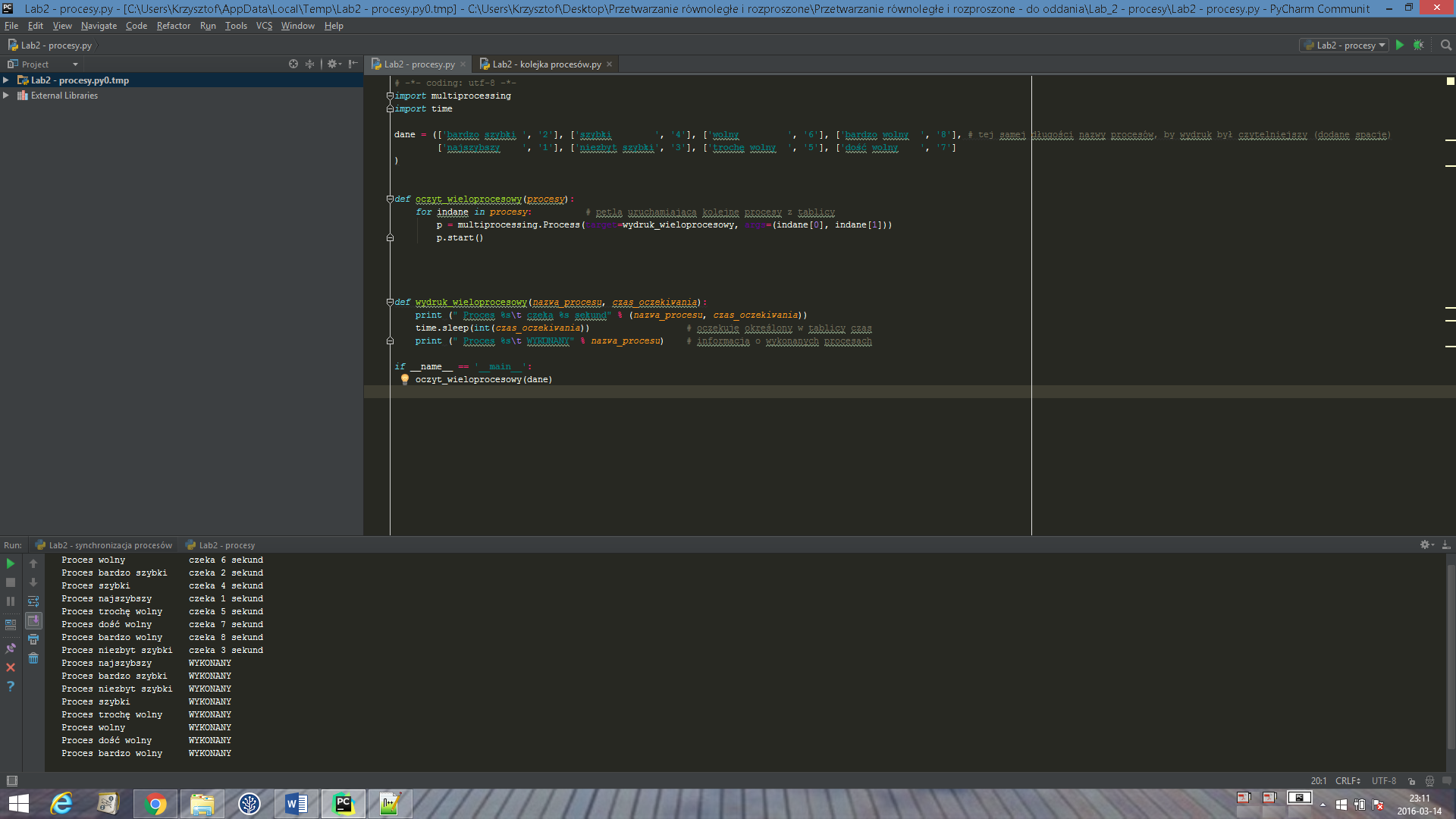


Przykład ilustrujący uruchamianie wielu wątków i równoległe ich wykonywanie.

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 był czytelniejszy (dodane spacje)  
 [**'najszybszy '**, **'1'**], [**'niezbyt szybki'**, **'3'**], [**'trochę wolny '**, **'5'**], [**'dość wolny '**, **'7'**]  
)  
  
  
*def* oczyt\_wieloprocesowy(*procesy*):  
 *for* indane *in procesy*: # pętla uruchamiająca kolejne procesy z tablicy  
 p = multiprocessing.Process(target=wydruk\_wieloprocesowy, args=(indane[0], indane[1]))  
 p.start()  
  
  
  
  
*def* wydruk\_wieloprocesowy(*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\_\_'**:  
 oczyt\_wieloprocesowy(dane)

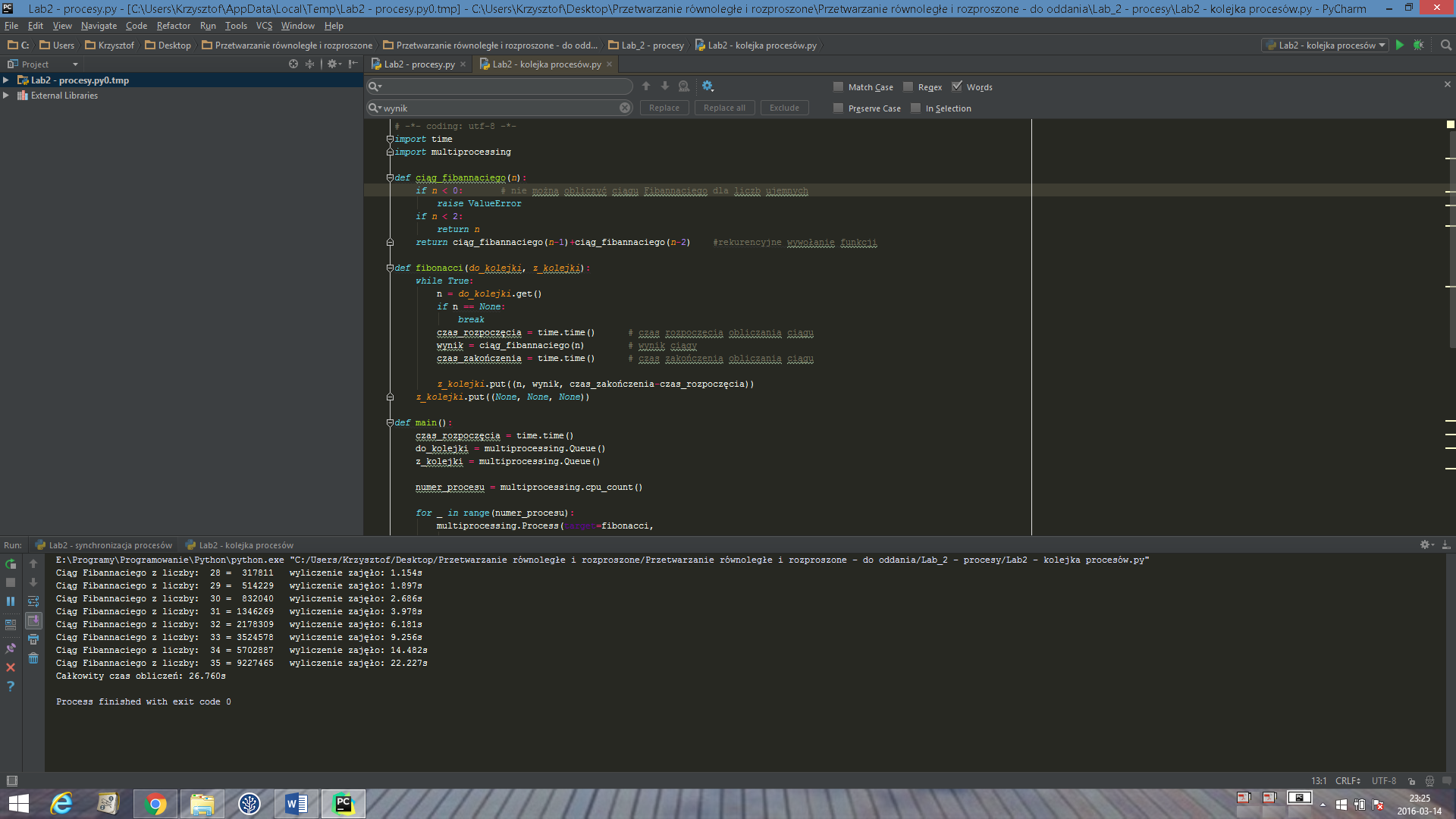
## **Kod wynikowy:**



**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 ciągy  
 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:**



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).