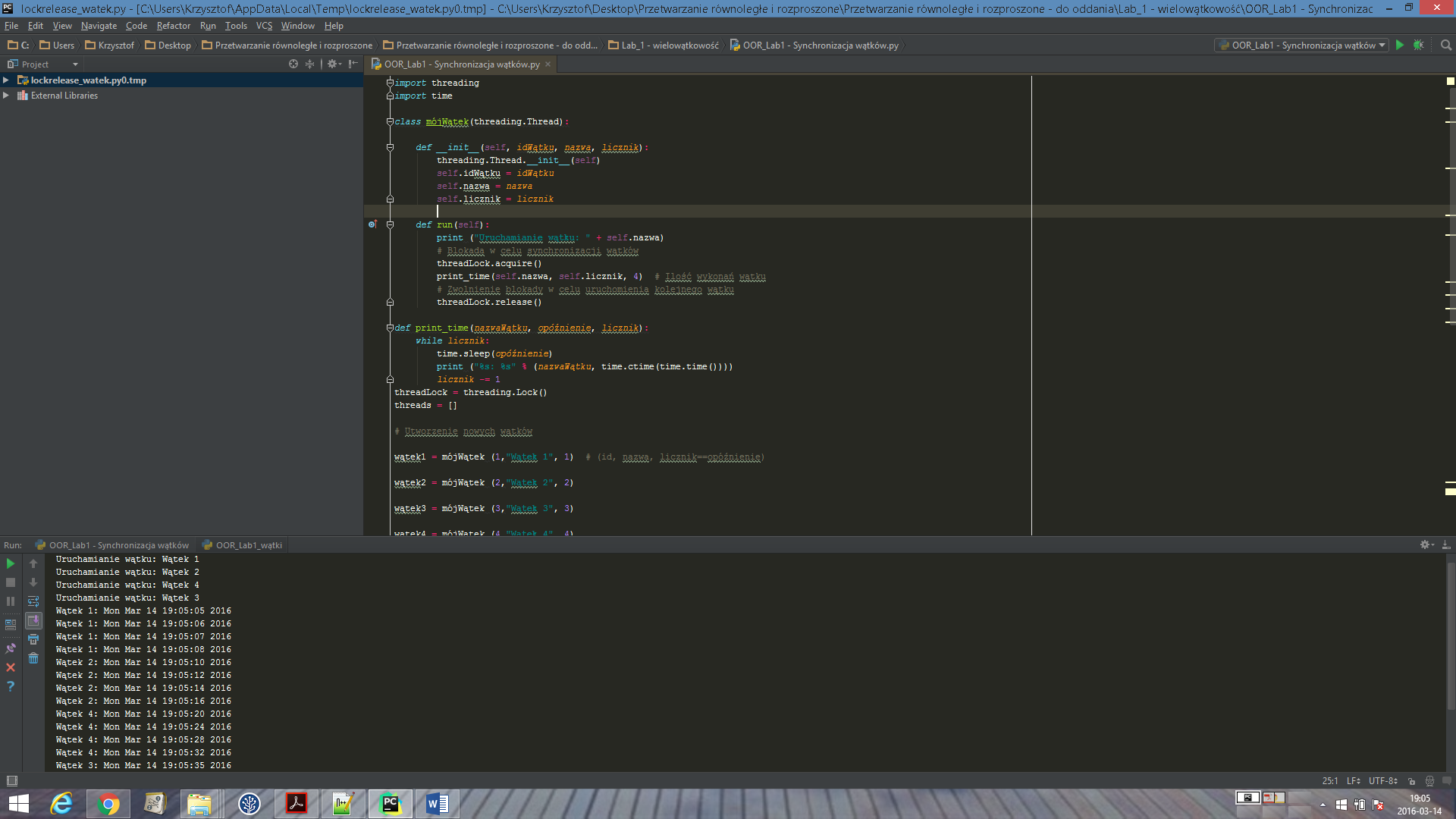
# **OOR: Programowanie równoległe i rozproszone**

## **Laboratorium 1:**

Za pomocą wybranego przez siebie języka programowania zademonstruj   
tworzenie i synchronizowanie wątków.

Zaprezentuj użycie metod run() i join() związanych z wątkami.

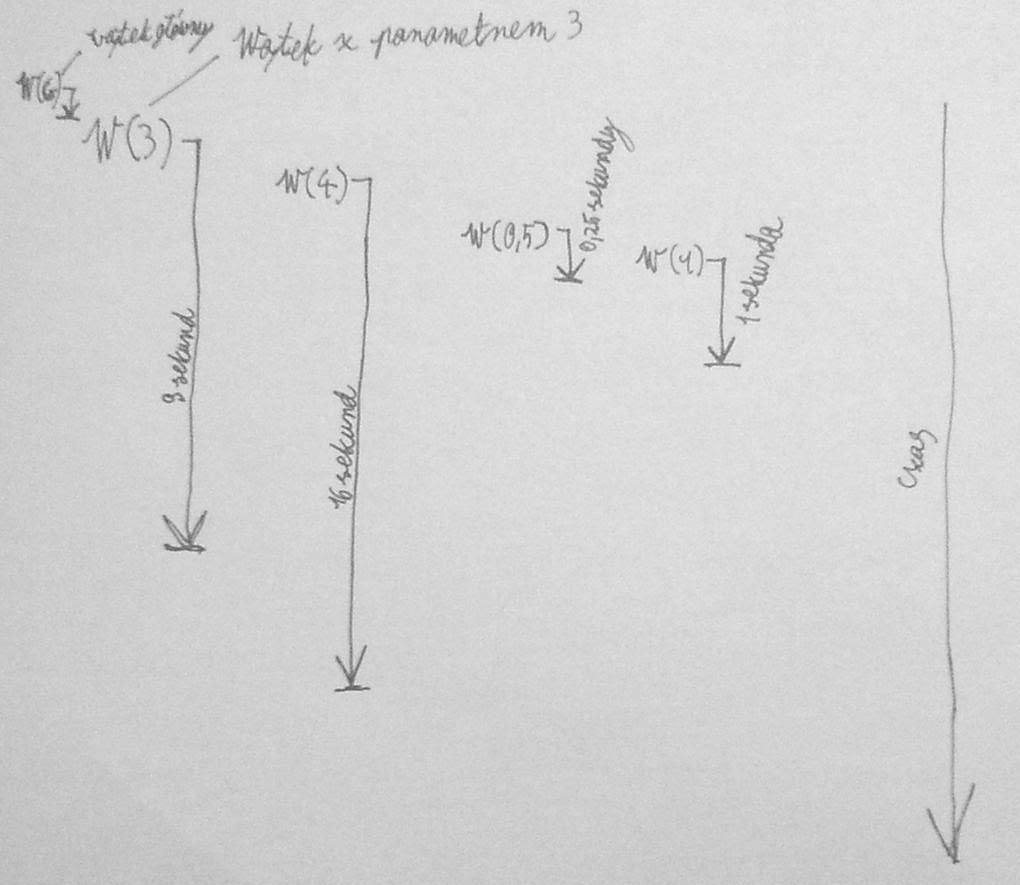
Zaprezentuj użycie metod acquire() i release() związanych z blokadami (Lock).



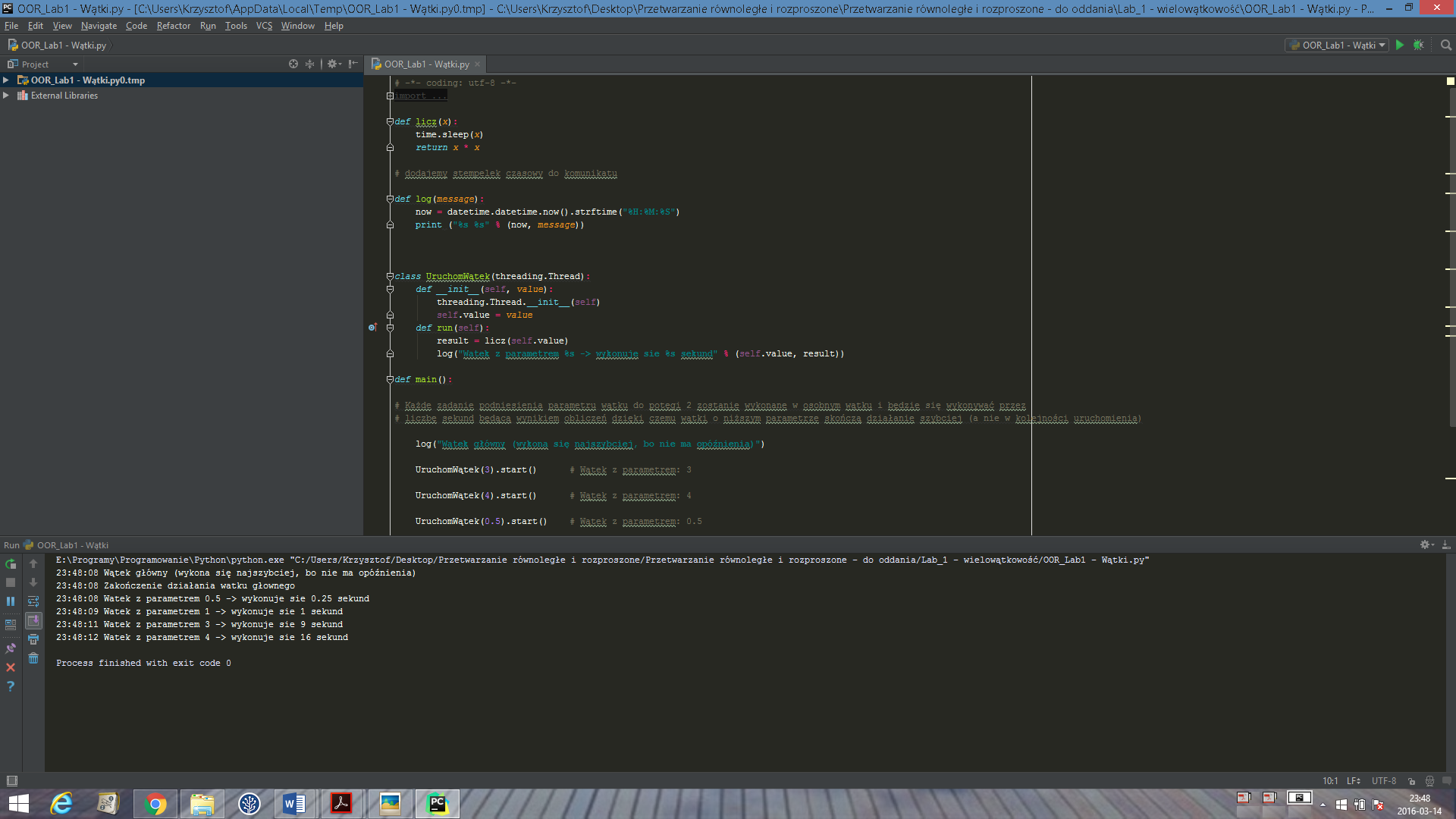
Przykład ilustrujący uruchamianie wielu wątków i równoległe ich wykonywanie.

Czas wykonywania kolejnych wątków 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 wątków w języku Python:**

# -\*- coding: utf-8 -\*-  
*import* time  
*import* datetime  
*import* threading  
  
*def* licz(*x*):  
 time.sleep(*x*)  
 *return x* \* *x*# dodajemy stempelek czasowy do komunikatu  
  
*def* log(*message*):  
 now = datetime.datetime.now().strftime(**"%H:%M:%S"**)  
 print (**"%s %s"** % (now, *message*))  
  
  
  
*class* UruchomWątek(threading.Thread):  
 *def \_\_init\_\_*(self, *value*):  
 threading.Thread.\_\_init\_\_(self)  
 self.value = *value  
 def* run(self):  
 result = licz(self.value)  
 log(**"Watek z parametrem %s -> wykonuje sie %s sekund"** % (self.value, result))  
  
*def* main():  
  
# Każde zadanie podniesienia parametru wątku do potęgi 2 zostanie wykonane w osobnym wątku i będzie się wykonywać przez  
# liczbę sekund będącą wynikiem obliczeń dzięki czemu wątki o niższym parametrze skończą działanie szybciej (a nie w kolejności uruchomienia)  
  
 log(**"Wątek główny (wykona się najszybciej, bo nie ma opóźnienia)"**)  
  
 UruchomWątek(3).start() # Wątek z parametrem: 3  
  
 UruchomWątek(4).start() # Wątek z parametrem: 4  
  
 UruchomWątek(0.5).start() # Wątek z parametrem: 0.5  
  
 UruchomWątek(1).start() # Wątek z parametrem: 1  
  
 log(**"Zakończenie działania watku głownego"**)  
  
*if* \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()



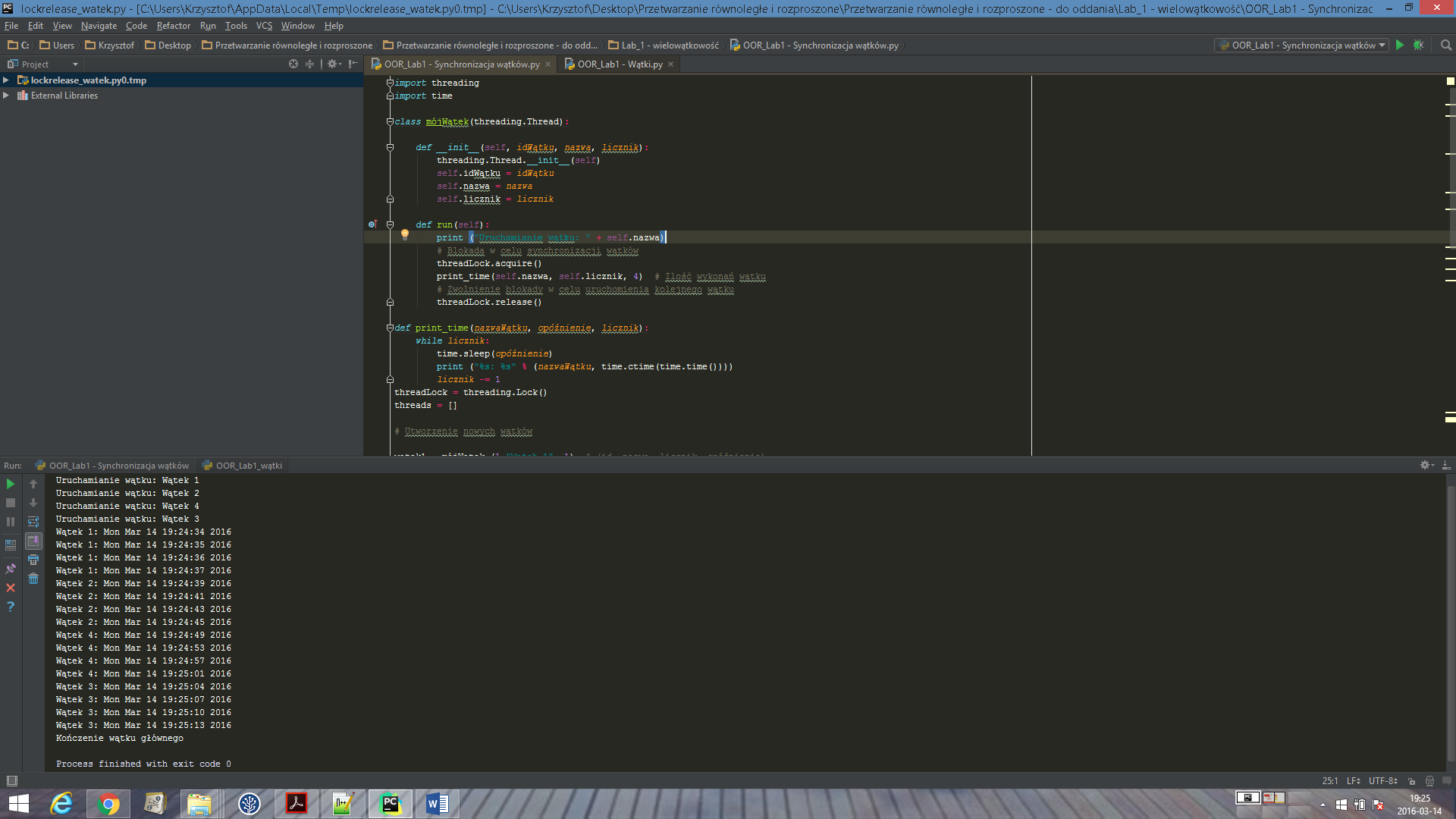
## **Kod wynikowy:**



**Kod programu obsługującego wiele wątków, z synchronizacją wątków w języku Python:**

*import* threading  
*import* time  
  
*class* mójWątek(threading.Thread):  
   
 *def \_\_init\_\_*(self, *idWątku*, *nazwa*, *licznik*):  
 threading.Thread.\_\_init\_\_(self)  
 self.idWątku = *idWątku* self.nazwa = *nazwa* self.licznik = *licznik  
   
 def* run(self):  
 print (**"Uruchamianie wątku: "** + self.nazwa)  
 # Blokada w celu synchronizacji wątków  
 threadLock.acquire()  
 print\_time(self.nazwa, self.licznik, 4) # Ilość wykonań wątku  
 # Zwolnienie blokady w celu uruchomienia kolejnego wątku  
 threadLock.release()  
  
*def* print\_time(*nazwaWątku*, *opóźnienie*, *licznik*):  
 *while licznik*:  
 time.sleep(*opóźnienie*)  
 print (**"%s: %s"** % (*nazwaWątku*, time.ctime(time.time())))  
 *licznik* -= 1  
threadLock = threading.Lock()  
threads = []  
  
# Utworzenie nowych wątków  
  
wątek1 = mójWątek (1,**"Wątek 1"**, 1) # (id, nazwa, licznik==opóźnienie)  
  
wątek2 = mójWątek (2,**"Wątek 2"**, 2)  
  
wątek3 = mójWątek (3,**"Wątek 3"**, 3)  
  
wątek4 = mójWątek (4,**"Wątek 4"**, 4)  
  
# Uruchomienie nowych wątków  
  
wątek1.start()  
  
wątek2.start()  
  
wątek4.start() # Wątek 4 zostanie uruchomiony przed wątkiem 3  
  
wątek3.start()  
  
# Dodawanie wątków do listy wątków  
  
threads.append(wątek1)  
threads.append(wątek2)  
threads.append(wątek4)  
threads.append(wątek3)  
  
# Oczekiwanie na zakończenie wszystkich wątków  
  
*for* t *in* threads:  
 t.join()  
  
print (**"Kończenie wątku głównego"**)

## **Kod wynikowy:**



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