



${\bf Protokoll}$ ${\bf A08~Watchdog+Synchronisation}$

Note:

Betreuer: W. Rafeiner-Magor

 $\begin{array}{c} {\rm Software entwicklung} \\ {\rm 4CHIT} \ \ 2016/17 \end{array}$

Filip Scopulovic

Version 1.0
Begonnen am 27. November 2016

Beendet am 27. November 2016

Inhaltsverzeichnis

1	Auf	gabe	1
	1.1	Vorgangsweise	1
	1.2	Aufwand	1
	1.3	Resultat	1
	1.4	Beobachtungen	1
	1.5	Schwierigkeiten	1
	1.6	Code	2

1 Aufgabe

1.1 Vorgangsweise

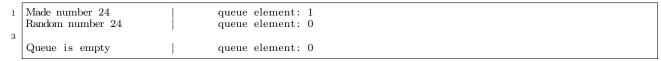
Mit einem von dem Herr Professor Rafeiner-Magor bereitgestelltem pdf-File. Mit diesem haben wir uns über Watchdogs informiert. Diese waren die einzige Neuheit bei der Aufgabe, ansonsten war alles wie gewohnt.

1.2 Aufwand

Der Aufwand war gering. Einerseits hatten wir die Watchdogs-Klasse vorgegebenen und andererseits haben wir schon ein "schwereres" Consumer-Producer-Schema implementiert.

1.3 Resultat

Ich habe ein Ausgabeschema überlegt, welches schön aussieht. So circa sieht es aus:



Listing 1: Ausgabeschema

1.4 Beobachtungen

Meine Beobachtung war die, das man den Consumer mit time.sleep() warten lassen musste, damit die Queue nicht immer leer ist.

1.5 Schwierigkeiten

Schwierigkeiten hat mir der letzte Punkt der erweiterten Aufgabenstellung eingebracht, da ich nicht wusste, wie man das implementiert. Deshalb habe ich es auch gelassen.

1.6 Code

```
@author: Filip Scopulovic
    @date: 21-11-2016
    Quse: watches when the thread needs to be stopped
   import threading, time
 6
    from abc import ABCMeta, abstractmethod
 8
    class Stopable(metaclass=ABCMeta):
10
         Abstract class for stopping
12
         @abstractmethod
         def stopping(self):
14
             Abstract method is necessary to stop a thread in a more secure way
16
18
              :return None:
20
             pass
    class WatchDog(threading.Thread):
22
         Class for stopping a "Stoppable" thread
24
         :inheritance threading.Thread:
26
        \label{eq:def_def} \begin{array}{l} \operatorname{def} \  \, \inf_{\Pi \in \Pi} (\operatorname{self} \, , \ \operatorname{stoptime} \, , \ *\operatorname{threads}) \, : \end{array}
28
             Initializes the WatchDog-Thread
30
              :param stoptime: takes a time that says when to stop
32
              :param *threads: a list with threads
34
             {\it threading.Thread.}\_{\it init}\_{\it (self)}
36
              self.stoptime = stoptime
              self.threads = threads
38
         def run(self):
40
             Waits the time that is in self.stoptime and stops the threads
42
              :return None:
44
             start = time.time()
             \mathrm{end} = \mathrm{start} + \mathrm{self.stoptime}
46
             # waits until the end of time is reached
             while time.time() < end:
48
                  # sleep should not be more than a second
                  time.sleep(0.9)
             # stops all the threads
52
             for t in self.threads:
                  t.stopping()
```

Listing 2: Watchdog

```
1
   @author: Filip Scopulovic
   @date: 21-11-2016
3
   Quse: consumer prints out the numbers from the queue
5
   import threading, queue, time
  from watch dog import Stopable
7
   class Consumer(threading.Thread, Stopable):
9
      Takes numbers from the queue and prints them out
11
      :inheritance threading.Thread:
13
      :inheritance Stopable:
15
          _{\parallel \parallel \parallel \parallel} init__(self, queue):
      \mathbf{def}
17
          Calls the Base constructor from threading. Thread
19
          Initializes both parameters
          :param queue: takes the queue as a parameter
21
          :param running: takes a boolean that says if the thread should run
23
          threading.Thread.\__init\__(self)
25
          self.queue = queue
          self.running = True
27
      def stopping(self):
29
          Class is taken from the abstract class Stopable and it sets the running method False
31
          :return None:
33
          self.running = False
35
      def run(self):
37
          Prints out the number that waits in the queue while self.running is True
39
          :return None:
41
          while self.running:
             time.sleep(0.01)
43
              if self.queue.empty():
                 45
                 #self.queue.task_done()
              else:
47
                 #Koennte wie beim Producer ein Einzeiler sein, wollte aber einen schoeneren Output haben
                      und deshalb so "kompliziert"
                 got number = self.queue.get()
49
                 if got_number < 10:
                     51
                         \verb|self.queue.qsize())|, \verb|end=""|
                     53
                 self.queue.task done()
```

Listing 3: Verbraucher

```
@author: Filip Scopulovic
2
   @date: 21-11-2016
   @use: producer makes a random number and prints it out before putting it in the queue
6
  import threading, random, queue
   from watch dog import Stopable
8
   class Producer(threading.Thread, Stopable):
10
      Takes numbers from the queue and prints them out
12
      :inheritance threading.Thread:
14
      :inheritance Stopable:
      def __init__(self , queue):
16
          Calls the Base constructor from threading. Thread
18
          Initializes both parameters
20
          :param queue: takes the queue as a parameter
          :param running: takes a boolean that says if the thread should run
22
          threading.Thread.\__init\__(self)
24
          self.queue = queue
          self.running = True
26
      def stopping(self):
          Class is taken from the abstract class Stopable and it sets the running method False
30
          :return None:
32
          self.running = False
34
36
      def run(self):
          Makes a random number, prints it out and puts it in the queue
38
          :return None:
40
          while self.running:
42
             rand num = random.randint(0, 254)
              if self.queue.full():
                 46
                 self.queue.put(rand\_num)
                 48
              \operatorname{self.queue.join}()
```

Listing 4: Erzeuger

```
1
   @author: Filip Scopulovic
   @date: 21-11-2016
   @use: starting script for the project A08_Thread_Synch
   import watch_dog, consumer, producer, queue
   #Empty list of threads
   threads = []
   #Initalizes the queue
  qu = queue.Queue()
   #Sets the maximum size for the queue at 20
13 \mid qu.maxsize = 20
   #Initalizes the Producer
15 | p1 = producer.Producer(qu)
   p2 = producer.Producer(qu)
  #Initalizes the Consumers
   c1 = consumer.Consumer(qu)
  c2 = consumer.Consumer(qu)
   #Adds all the threads together
   threads.append(p1)
   threads.append(p2)
   threads.append(c1)
   threads.append(c2)
  running_time = 2
#Initializes the watch dog
27
   w = watch_dog.WatchDog(running_time, *threads)
  w.start()
29
   for thread in threads:
       thread.start()
   for thread in threads:
       thread.join()
   w.join()
```

Listing 5: Start

Listings

1	Ausgabeschema	1
2	Watchdog	2
3	Verbraucher	3
4	Erzeuger	4
5	Start	5