

---

# Protokoll

## A08 Watchdog + Synchronisation

---

Softwareentwicklung  
4CHIT 2016/17

Filip Scopulovic

Note:  
Betreuer: W. Rafeiner-Magor

Version 1.0  
Begonnen am 27. November 2016  
Beendet am 27. November 2016

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabe</b>	<b>1</b>
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:

1	Made number 24		queue element: 1
	Random number 24		queue element: 0
3	Queue is empty		queue element: 0

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

```

1  """
2  @author: Filip Scopulovic
3  @date: 21-11-2016
4  @use: watches when the thread needs to be stopped
5  """
6  import threading, time
7  from abc import ABCMeta, abstractmethod
8
9  class Stopable(metaclass=ABCMeta):
10     """
11     Abstract class for stopping
12     """
13     @abstractmethod
14     def stopping(self):
15         """
16         Abstract method is necessary to stop a thread in a more secure way
17
18         :return None:
19         """
20         pass
21
22 class WatchDog(threading.Thread):
23     """
24     Class for stopping a "Stopable" thread
25
26     :inheritance threading.Thread:
27     """
28     def __init__(self, stoptime, *threads):
29         """
30         Initializes the WatchDog-Thread
31
32         :param stoptime: takes a time that says when to stop
33         :param *threads: a list with threads
34         """
35         threading.Thread.__init__(self)
36         self.stoptime = stoptime
37         self.threads = threads
38
39     def run(self):
40         """
41         Waits the time that is in self.stoptime and stops the threads
42
43         :return None:
44         """
45         start = time.time()
46         end = start + self.stoptime
47         # waits until the end of time is reached
48         while time.time() < end:
49             # sleep should not be more than a second
50             time.sleep(0.9)
51         # stops all the threads
52         for t in self.threads:
53             t.stopping()

```

Listing 2: Watchdog





```
1  """
2  @author: Filip Scopulovic
3  @date: 21-11-2016
4  @use: starting script for the project A08_Thread_Synch
5  """
6  import watch_dog, consumer, producer, queue
7
8  #Empty list of threads
9  threads = []
10 #Initializes the queue
11 qu = queue.Queue()
12 #Sets the maximum size for the queue at 20
13 qu.maxsize = 20
14 #Initializes the Producer
15 p1 = producer.Producer(qu)
16 p2 = producer.Producer(qu)
17 #Initializes the Consumers
18 c1 = consumer.Consumer(qu)
19 c2 = consumer.Consumer(qu)
20 #Adds all the threads together
21 threads.append(p1)
22 threads.append(p2)
23 threads.append(c1)
24 threads.append(c2)
25
26 running_time = 2
27 #Initializes the watch dog
28 w = watch_dog.WatchDog(running_time, *threads)
29 w.start()
30 for thread in threads:
31     thread.start()
32 for thread in threads:
33     thread.join()
34 w.join()
```

Listing 5: Start

## Listings

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