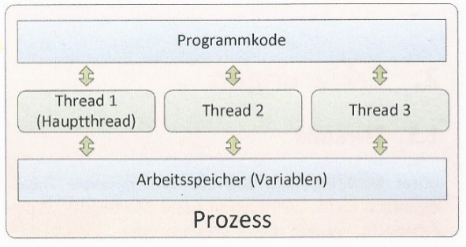
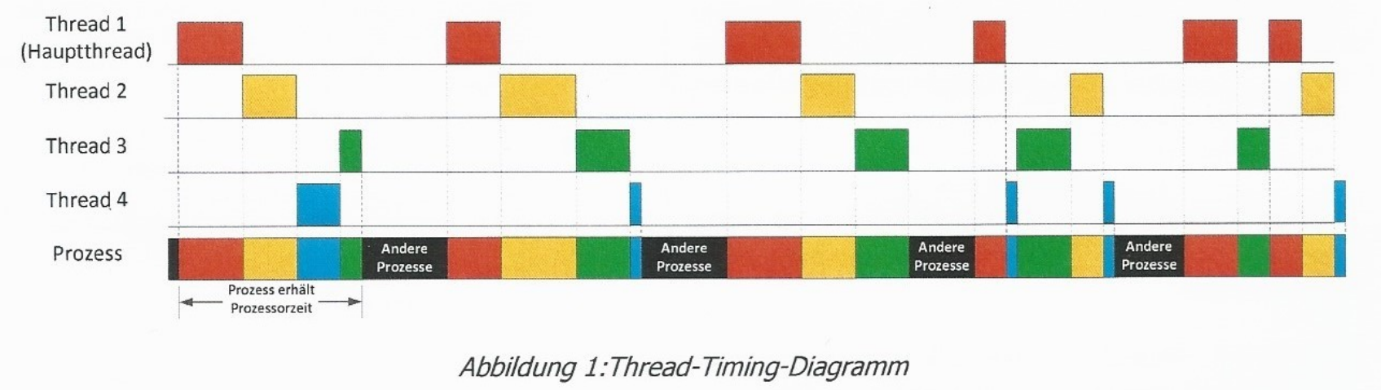
**Threads:**

**MT1:**

🡪 Jedes Programm ist ein Prozess. In einem Prozess können jedoch mehrere Threads laufen, die auf den gleichen Programmkode zugreifen und den gleichen Arbeitsspeicher haben.

Das Betriebssystem gibt jeden Thread abwechselnd Rechenleistung. Ein Thread kann nicht auf die Daten eines anderen Prozesses zu greifen. Man benötigt sonst Techniken wie Inter Prozess Communication (IPC).  
Jeder Thread bekommt nach dem anderen ein wenig Prozessorzeit zugewiesen. Somit bekommt man den Anschein das die Prozesse parallel laufen.

Wechseln zwischen Threads nennt man Kontextswitch

🡪 bei der sleep Funktion wird der Prozessor viel weniger belastet. Da der Thread dann für eine bestimmte Zeit keine Rechenleistung bekommt. Je länger geschlafen wird, umso weniger wird der Prozessor belastet

**MT2:**

🡪 from threading import Thread  
thread = Thread(target=funktion, args=(parameter,), name = „Thread1“)  
thread.start()

Mit global kann man lokale Variablen Global machen damit jeder Thread zugreifen kann

Logging Modul:

Import logging  
 🡪 Level: Wichtigkeit -> Flitern: alle Meldungen „über“ den Level angezeigt.  
 Konfiguration: logging.basicConfig(level=logging.LEVELDEBUG, format=“%(threadname)s:“(messages)s“  
zum Ausgeben: logging.warning(„Aussage“) oder logging.info(„hallo“),…

Timer:

from threading import Thread  
 t = threading.Timer(sekunden, Funktion)  
t.start()

**MT3:**

Ressource-Locking:

Das Problem ist, dass das Inkrementieren aus 3 Operationen besteht also 1. Register laden 2. Hochzählen 3. Arbeitsspeicher im Ram). Somit besteht die Gefahr, dass der eine Thread Änderungen an andere Threads überschreibt. Lösung = lock Objekt vom Modul threading damit die 3 Operationen nicht unterbrochen werden können.

Locking:

from threading import RLock.  
 lock = Rlock() = wird ganz oben angelegt  
 lock.acquire() Diese Funktion blockiert, wenn ein anderer Thread den Lock schon hält  
 try:  
 Bearbeitet geteilte Ressource  
 finally:  
 lock.release

Queue:

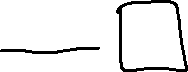
Ist eine Warteschlange diese hat 4 Funktionen (put(), get(), taskdone() und join()).



Druckerwarteschlange:



Implementierung:



import queue  
 q = queue.Queue()  
 q.put(String oder Integer)  
 elementausderQueue = q.get() bei der Liste wäre es pop |sonst wäre alles aufgehalten|  
 q.task\_done



task\_done(): Zeigt an, das eine zuvor in die Warteschlange gestellte Aufgabe abgeschlossen ist. Bei jeden get() das zum Abrufen des Tasks verwendet wird, teil ein nachfolgender Aufruf von task:done() der Warteschlange mit, dass die Verarbeitung des Tasks abgeschlossen ist.

Thread.join blockiert den Main Thread bis Nebenthread vorbei ist.

Die Funktion eval() kann einen String in eine Python Syntax umändern. Somit wird zum Beispiel aus dem String „21+5“ eine Rechnung gemacht 21+5 und Python gibt 26 aus

FIFO = First in First out  
LIFO = Last in First out.

**MT4:**

Problem: Warteschlange 🡪 womit kann man eine Synchronisation zweier Threads erreichen?

Antwort: Mit Event kann man ein Ereignis setzen, womit der Thread anfängt zu arbeiten.

Event:

- Im Haupthread wir das Objekt Event angelegt. 🡪 **event =Event()**  
- Der Arbeitsthread wird im Schlafzustand versetzt durch funktion **wait()** und wartet bis das Event gesetzt wird  
- Arbeitsthread wird aufgeweckt durch die Funktion **set()**  
- Beim Aufwachen des Threads muss das gesetzte Event gelöscht werden durch die Funktion **clear()** sonst wird jeder Aufruf der Funktion wait() unmittelbar zurückkehren, da das Event noch immer gesetzt ist.

🡪 mit is\_set() kann man überprüfen ob das event gesetzt ist oder nicht

MSC Diagramm = Message Sequence Chart:

