Paradigma Secventiala versus Concurenta

Cursul nr. 11

Mihai Zaharia

Cum se face un logging mai serios

```
def alg_complicat(items):# ex1
    for i, item in enumerate(items):
        # corpul alg
        logger.debug('%s iteration, item=%s', i, item)
def handle_request(request):#ex2
    logger.info('Gestionez cererea %s', request)
    # tratare cerere
    result = 'result'
    logger.info('Rezultatul este: %s', result)
def start_service():
    logger.info('Pornesc serviciul pe portul %s ...', port)
    service.start()
    logger.info('Serviciul a pornit')
def authenticate(user_name, password, ip_address):# ex 3
    if user_name != USER_NAME and password != PASSWORD:
        logger.warn('Incercare esuata de intrare in sistem utilizator %s de la IP %s', user_name, ip_address)
        return False
    # executarea autentificarii
def get_user_by_id(user_id):
    user = db.read_user(user_id)
    if user is None:
        logger.error('Nu hasesc utilizatorul cu user_id=%s', user_id)
        return user
    return user
```

Cum se face un logging mai serios

```
try: #ex 1
   open('/path/to/does/not/exist', 'rb')
except (SystemExit, KeyboardInterrupt):
   raise
except Exception, e:
   logger.error('Nu am putut deschide fisierul', exc_info=True)
import logging

def foo():#ex 2
   logger = logging.getLogger(__name__)
   logger.info('Hi, foo')
class Bar(object):
   def __init__(self, logger=None):
      self.logger = logger or logging.getLogger(__name__)
   def bar(self):
      self.logger.info('Hi, bar')
```

Cum se face un logging mai serios

logging.json

```
{
   "version": 1,
   "disable_existing_loggers": false,
   "formatters": {
      "simple": {
         "format": "%(asctime)s - %(name)s - %(levelname)s -
%(message)s"
      }
   },
   "handlers": {
      "console": {
         "class": "logging.StreamHandler",
         "level": "DEBUG",
         "formatter": "simple",
         "stream": "ext://sys.stdout"      },
      "info_file_handler": {
         "class": "logging.handlers.RotatingFileHandler",
         "level": "INFO",
         "formatter": "simple",
         "filename": "info.log",
         "maxBytes": 10485760,
         "backupCount": 20,
         "encoding": "utf8"       },
      "error_file_handler": {
         "class": "logging.handlers.RotatingFileHandler",
         "level": "ERROR",
         "formatter": "simple",
         "filename": "errors.log",
         "maxBytes": 10485760,
         "backupCount": 20,
         "encoding": "utf8"
      }
   },
   "loggers": {
      "my_module": {
         "level": "ERROR",
         "handlers": ["console"],
         "propagate": false
      }
   },
   "root": {
      "level": "INFO",
      "handlers": ["console", "info_file_handler", "error_file_handler"]
   }
}
```

pentru a incarca acest fisier dintr-o cale prestabilita

```
LOG_CFG=my_logging.json python my_server.py
```

Python threading module

- pentru thread
- pentru Lock
- pentru RLock
- pentru semafoare
- pentru condiții
- pentru evenimente

Analiza comparativă - diverse biblioteci pentru paralelism

```
import threading
import multiprocessing
from concurrent.futures import ThreadPoolExecutor
import time
def countdown():
    x = 100000000
    while x > 0:
        x -= 1
def ver_1():#pseudoparalelism
    thread_1 = threading.Thread(target=countdown)
    thread_2 = threading.Thread(target=countdown)
    thread_1.start()
    thread_2.start()
    thread_1.join()
    thread_2.join()
def ver_2():#secvential
    countdown()
    countdown()
def ver_3():#paralelism cu multiprocessing
    process_1 = multiprocessing.Process(target=countdown)
    process_2 = multiprocessing.Process(target=countdown)
    process_1.start()
    process_2.start()
    process_1.join()
    process_2.join()
def ver_4():#paralelism cu concurrent.futurez
    with ThreadPoolExecutor(max_workers=2) as executor:
        future = executor.submit(countdown())
        future = executor.submit(countdown())
if __name__ == '__main__':
    start = time.time()
    ver_1()
    end = time.time()
    print("\n Timp executie pseudoparalelism cu GIL")
    print(end - start)
    start = time.time()
    ver_2()
    end = time.time()
    print("\n Timp executie secvential")
    print(end - start)
    start = time.time()
    ver_3()
    end = time.time()
    print("\n Timp executie paralela cu multiprocessing")
    print(end - start)
    start = time.time()
    ver_4()
    end = time.time()
    print("\n Timp executie paralela cu concurrent.futures")
    print(end - start)
```

si rezultatul executiei

Timp executie pseudoparalelism cu GIL - 13.273755550384521

Timp executie secvential - 10.081993579864502

Timp executie paralela cu multiprocessing - 5.0672242641448975

Timp executie paralela cu concurrent.futures - 13.14623498916626

Un fir de execuție

```
class threading.Thread(group=None,
        target=None,
        name=None,
        args=(),
        kwargs={})
```

Exemplu utlizare parametri funcție în fir

```
import threading

def function(i):
    print('Functia este apelata de firul %i\n' % i)

threads = []
for i in range(5):
    t = threading.Thread(target=function, args=(i,))
    threads.append(t)
    t.start()
    t.join()
```

Determinarea firului curent

```
import threading
import logging
logging.basicConfig(level=logging.INFO)
def first_function():
    logging.info(threading.currentThread().getName() + str(' porneste...'))
    logging.info(threading.currentThread().getName() + str(' se opreste...'))
    return
def second_function():
    logging.info(threading.currentThread().getName() + str(' porneste...'))
    logging.info(threading.currentThread().getName() + str(' se opreste...'))
    return
def third_function():
    logging.info(threading.currentThread().getName() + str(' porneste...'))
    logging.info(threading.currentThread().getName() + str(' se opreste...'))
    return
if __name__ == '__main__':
    t1 = threading.Thread(name='prima_functie', target=first_function)
    t2 = threading.Thread(name='a doua functie', target= second_function)
    t3 = threading.Thread(name='a treia functie', target=third_function)
    t1.start()
    t2.start()
    t3.start()
    logging.debug('Pauza')
    t1.join()
    t2.join()
    t3.join()
logging.info(threading.currentThread().getName() + str(' - main thread...'))
```

si rezultatul executiei

```
INFO:root:prima_functie porneste...
INFO:root:prima_functie se opreste...
INFO:root:a doua functie porneste...
INFO:root:a treia functie porneste...
INFO:root:a doua functie se opreste...
INFO:root:a treia functie se opreste...
INFO:root:MainThread - main thread...

Process finished with exit code 0
```

Utilizarea unui fir într-o subclasă

```
import threading
import time
EXIT_FLAG = 0
class Firisor(threading.Thread):
    def __init__(self, thread_id, name, counter):
        threading.Thread.__init__(self)
        self.thread_id = thread_id
        self.name = name
        self.counter = counter
    def run(self):
        print('Sunt %s si am pornit' % self.name)
        print_time(self.name, self.counter, 5)
        print('Sunt %s si am terminat\n' % self.name)
def print_time(thread_name, delay, counter):
    while counter:
        if EXIT_FLAG:
            thread.exit()
        time.sleep(delay)
        print('%s: %s' % (thread_name, time.ctime(time.time())))
        counter -= 1
thread1 = Firisor(1, 'Firul 1', 1)
thread2 = Firisor(2, 'Firul 2', 2)
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print('S-a terminat firul principal\n')
```

si rezultatul executiei

```
"/home/bugs/PycharmProjects/fir in subclasa/venv/bin/python"
"/home/bugs/PycharmProjects/fir in subclasa/fir in subclasa.py"
Sunt Firul 1 si am pornit
Sunt Firul 2 si am pornit
Firul 1: Sun Apr  7 13:30:44 2019
Firul 1: Sun Apr  7 13:30:45 2019
Firul 2: Sun Apr  7 13:30:45 2019
Firul 1: Sun Apr  7 13:30:46 2019
Firul 2: Sun Apr  7 13:30:47 2019
Firul 1: Sun Apr  7 13:30:47 2019
Firul 1: Sun Apr  7 13:30:48 2019
Sunt Firul 1 si am terminat

Firul 2: Sun Apr  7 13:30:49 2019
Firul 2: Sun Apr  7 13:30:51 2019
Firul 2: Sun Apr  7 13:30:53 2019
Sunt Firul 2 si am terminat

S-a terminat firul principal

Process finished with exit code 0
```

Exemplu de utilizare lock()

```
import threading
contor_cu_lock = 0
contor_fara_lock = 0
COUNT = 1000000
lock_contor = threading.Lock()
def safe_inc():
    global contor_cu_lock
    for _ in range(COUNT):
        lock_contor.acquire()
        contor_cu_lock += 1
        lock_contor.release()
def safe_dec():
    global contor_cu_lock
    for _ in range(COUNT):
        lock_contor.acquire()
        contor_cu_lock -= 1
        lock_contor.release()
def unsafe_inc():
    global contor_fara_lock
    for _ in range(COUNT):
        contor_fara_lock += 1
def unsafe_dec():
    global contor_fara_lock
    for _ in range(COUNT):
        contor_fara_lock += 1
if __name__ == '__main__':
    t1 = threading.Thread(target=safe_inc)
    t2 = threading.Thread(target=safe_dec)
    t3 = threading.Thread(target=unsafe_dec)
    t4 = threading.Thread(target=unsafe_inc)
    t1.start()
    t2.start()
    t3.start()
    t4.start()
    t1.join()
    t2.join()
    t3.join()
    t4.join()
    print('variabila comuna gestionata cu lock', contor_cu_lock)
    print('variabila comuna gestionata fara lock', contor_fara_lock)
```

si rezultatul executiei

variabila comuna gestionata cu lock 0

variabila comuna gestionata fara lock 1322023

Exemplu utilizare Rlock()

```
import threading
import time
class Cutiechibrituri(object):
    lock = threading.RLock()
    def __init__(self):
        self.total_chibrituri = 0
    def execute(self, n):
        Cutiechibrituri.lock.acquire()
        self.total_chibrituri += n
        Cutiechibrituri.lock.release()
    def pun(self):
        Cutiechibrituri.lock.acquire()
        self.execute(1)
        Cutiechibrituri.lock.release()
    def scot(self):
        Cutiechibrituri.lock.acquire()
        self.execute(-1)
        Cutiechibrituri.lock.release()
def pune(Cutiechibrituri, chibrituri):
    while chibrituri > 0:
        print('Pun un chibrit in Cutiechibrituri')
        Cutiechibrituri.pun()
        time.sleep(1)
        chibrituri -= 1
def scot(Cutiechibrituri, chibrituri):
    while chibrituri > 0:
        print('Scot un chibrit din Cutiechibrituri')
        Cutiechibrituri.scot()
        time.sleep(1)
        chibrituri -= 1
if __name__ == '__main__':
    chibrituri = 5
    print('Pun', chibrituri, 'chibrituri in Cutiechibrituri')
    Cutiechibrituri = Cutiechibrituri()
    t1 = threading.Thread(target=pune, args=(Cutiechibrituri,
chibrituri))
    t2 = threading.Thread(target=scot, args=(Cutiechibrituri,
chibrituri))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
    print('mai sunt', Cutiechibrituri.total_chibrituri, 'chibrituri in
Cutiechibrituri')
```

Exemplu semafoare

```
import threading
import time
import random
semafor = threading.Semaphore(0)
def consumator():
    print('Consumatorul in asteptare')
    semafor.acquire()
    print('Consumatorur a fost anuntat si a folosit ', element, ' elemente')
def producator():
    global element
    time.sleep(1)#simulare complexiate operatiuni in caz real
    element = random.randint(0, 1000)
    print('Producatorul a fost anuntat si aprodus ', element,' elemente')
    semafor.release()
if __name__ == '__main__':
    for i in range(5):
        t1 = threading.Thread(target=producator)
        t2 = threading.Thread(target=consumator)
        t1.start()
        t2.start()
        t1.join()
        t2.join()
```

si un exemplu de executie

```
Consumatorul in asteptare
Producatorul a fost anuntat si aprodus  398  elemente
Consumatorur a fost anuntat si a folosit  398  elemente
....
Consumatorul in asteptare
Producatorul a fost anuntat si aprodus  701  elemente
Consumatorur a fost anuntat si a folosit  701  elemente
```

Fir cu Condiție

```
from threading import Thread, Condition
import time
elemente = []
conditie = Condition()
class Consumator(Thread):
    def __init__(self):
        Thread.__init__(self)
    def consumator(self):
        global conditie#utlizarea variabilelor globale
NERECOMANDATA in caz real
        global elemente
        conditie.acquire()
        if len(elemente) == 0:
            conditie.wait()
            print('mesaj de la consumator: nu am nimic disponibil')
        elemente.pop()
        print('mesaj de la consumator : am utlizat un element')
        print('mesaj de la consumator : mai am disponibil',
len(elemente), 'elemente')
        conditie.notify()
        conditie.release()
    def run(self):
        for i in range(5):
            self.consumator()
class Producator(Thread):
    def __init__(self):
        Thread.__init__(self)
    def producator(self):
        global conditie
        global elemente
        conditie.acquire()
        if len(elemente) == 10:
            conditie.wait()
            print('mesaj de la producator : am disponibile',
len(elemente), 'elemente')
            print('mesaj de la producator : am oprit productia')
        elemente.append(1)
        print('mesaj de la producator : am produs',
len(elemente), 'elemente')
        conditie.notify()
        conditie.release()
    def run(self):
        for i in range(5):
            self.producator()
if __name__ == '__main__':
    producator = Producator()
    consumator = Consumator()
    producator.start()
    consumator.start()
    producator.join()
    consumator.join()
```

Fir cu eveniment

```
import time
from threading import Thread, Event
import random
elemente = []
eveniment = Event()
class Consumator(Thread):
    def __init__(self, elemente, eveniment):
        Thread.__init__(self)
        self.elemente = elemente
        self.eveniment = eveniment
    def run(self):
        for i in range(5):
            self.eveniment.wait()
            try:
                item = self.elemente.pop()
            except IndexError:
                print('Nu pot scoate dintr-o coada goala!')
            print('\nMesaj de la consumator : %d a fot generat de
%s' % (item, self.name))
```

```
class Producator(Thread):
    def __init__(self, elemente, eveniment):
        Thread.__init__(self)
        self.elemente = elemente
        self.eveniment = eveniment
    def run(self):
        for i in range(5):
            item = random.randint(0, 256)
            self.elemente.append(item)
            print('\nMesaj de la producator : elementul # %d a
fost adaugat la lista de %s' % (
                item, self.name))
            print('Mesaj de la producator  : eveniment generat de
%s' % self.name)
            self.eveniment.set()
            print('Mesaj de la producator : eveniment anulat de
%s' % self.name)
            self.eveniment.clear()
if __name__ == '__main__':
    t1 = Producator(elemente, eveniment)
    t2 = Consumator(elemente, eveniment)
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

Utilizarea 'with'

```
import threading
import logging
logging.basicConfig(
    level=logging.DEBUG,
    format='(%(threadName)-8s) %(message)s',
)
def thread_cu_with(statement):
    with statement:
        logging.debug('%s achizitionat cu with' % statement)
def thread_fara_with(statement):
    statement.acquire()
    try:
        logging.debug('%s achizitionatt direct' % statement)
    finally:
        statement.release()
if __name__ == '__main__':
    lock = threading.Lock()
    rlock = threading.RLock()
    conditie = threading.Condition()
    mutex = threading.Semaphore(1)
    threading_synchronisation_list = [lock, rlock, conditie, mutex]
    for statement in threading_synchronisation_list:
        t1 = threading.Thread(target=thread_cu_with, args=(statement,))
        t2 = threading.Thread(target=thread_fara_with, args=(statement,))
        t1.start()
        t2.start()
        t1.join()
        t2.join()
```

si rezultatul executiei

```
(Thread-1) <locked _thread.lock object at 0x7fc8cce9dcb0> achizitionat cu with
(Thread-2) <locked _thread.lock object at 0x7fc8cce9dcb0> achizitionatt direct
(Thread-3) <locked _thread.RLock object owner=140500325627648 count=1 at
0x7fc8ccdbb390> achizitionat cu with
(Thread-4) <locked _thread.RLock object owner=140500325627648 count=1 at
0x7fc8ccdbb390> achizitionatt direct
(Thread-5) <Condition(<locked _thread.RLock object owner=140500325627648 count=1
at 0x7fc8ccdbb420>, 0)> achizitionat cu with
(Thread-6) <Condition(<locked _thread.RLock object owner=140500406716160 count=1
at 0x7fc8ccdbb420>, 0)> achizitionatt direct
(Thread-7) <threading.Semaphore object at 0x7fc8ccd56320> achizitionat cu with
(Thread-8) <threading.Semaphore object at 0x7fc8ccd56320> achizitionatt direct
```

Comunicare inter-thread utlizând cozi

```
from threading import Thread
from queue import Queue
import time
import random
class Producator(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue
    def run(self):
        for i in range(10):
            element = random.randint(0, 256)
            self.queue.put(element)
            print('Mesaj de la producator : element N%d adaugat
la coada de %s\n' % (
                element, self.name))
            time.sleep(1)
class Consumator(Thread):
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue
    def run(self):
        while True:
            element = self.queue.get()
            print('Mesaj de la consumator : %d scos din coada de
%s' % (
                element, self.name))
            self.queue.task_done()
if __name__ == '__main__':
    queue = Queue()
    t1 = Producator(queue)
    t2 = Consumator(queue)
    t3 = Consumator(queue)
    t4 = Consumator(queue)
    t1.start()
    t2.start()
    t3.start()
    t4.start()
    t1.join()
    t2.join()
    t3.join()
    t4.join()
```

Paralelism real - multiprocessing

```
import multiprocessing
import time
def proces_gol():
    nume = multiprocessing.current_process().name
    print('\nPornesc un proces numit: %s' % nume)
    time.sleep(3)#simulez o executie
    print('Am terminat procesul numit: %s' % nume)
if __name__ == '__main__':
    proces_demon = multiprocessing.Process(
        name='proces demon', target=proces_gol)
    proces_demon.daemon = True
    proces_normal = multiprocessing.Process(
        name='proces normal', target=proces_gol)
    proces_normal.daemon = False
    proces_demon.start()
    proces_normal.start()
    print('am terminat procesul normal')
```

si rezultatul executiei

```
am terminat procesul normal

Pornesc un proces numit: proces demon

Pornesc un proces numit: proces normal
Am terminat procesul numit: proces normal

Process finished with exit code 0
```

gestiunea stării curente a unui proces

```
import multiprocessing
import time
import signal
def proces_gol():
    print('Pornesc executia procesului')
    time.sleep(0.1)
    print('S-a terminat executia procesului')
if __name__ == '__main__':
    proces_test = multiprocessing.Process(target=proces_gol)
    print('Starea procesului inainte de lansarea in executie:', proces_test, proces_test.is_alive())
    proces_test.start()
    print('Procesul se executa:', proces_test, proces_test.is_alive())
    proces_test.terminate()
    try:
        print('Procesul s-a terminat:', proces_test, proces_gol().is_alive())
    except AttributeError:
        print('Nu exista informatii dupa comanda terminare')
    proces_test.join()
    try:
        print('Procesul dupa join:', proces_test, proces_gol().is_alive())
    except AttributeError:
        print('Nu am informatii dupa join')
    if signal.SIG_DFL == proces_test.exitcode:
        print('Procesul dupa un exit code')
```

si rezultatul executiei

```
Starea procesului inainte de lansarea in executie:
<Process(Process-1, initial)> False
Procesul se executa: <Process(Process-1, started)> True
Pornesc executia procesului
S-a terminat executia procesului
Nu exista informatii dupa comanda terminare
Pornesc executia procesului
S-a terminat executia procesului
Nu am informatii dupa join

Process finished with exit code 0
```

utilizarea unui proces in subclasă

```
import multiprocessing
class ProcesTest(multiprocessing.Process):
    def run(self):
        print ('am apelat metoda run() in procesul: %s' %self.name)
        return

if __name__ == '__main__':
    jobs = []

    for i in range(5):
        p = ProcesTest()
        jobs.append(p)
        p.start()
        p.join()
```

si rezultatul executiei

```
am apelat metoda run() in procesul: ProcesTest-1
am apelat metoda run() in procesul: ProcesTest-2
am apelat metoda run() in procesul: ProcesTest-3
am apelat metoda run() in procesul: ProcesTest-4
am apelat metoda run() in procesul: ProcesTest-5

Process finished with exit code 0
```

Cozi pentru comunicare interproces

```
import multiprocessing
import random
class Producator(multiprocessing.Process):
    def __init__(self, queue):
        multiprocessing.Process.__init__(self)
        self.queue = queue
    def run(self):
        for _ in range(10):
            element = random.randint(0, 256)
            self.queue.put(element)
            print('Proces Producator : elementul %d s-a
addaugat in coada % s' % (element, self.name ))
            print('Dimensiunea cozii este %s' %
self.queue.qsize())
class Consumator(multiprocessing.Process):
    def __init__(self, queue):
        multiprocessing.Process.__init__(self)
        self.queue = queue
    def run(self):
        while True:
            if self.queue.empty():
                print('Coada este goala')
                break
            else:
                element = self.queue.get()
                print('Proces Consumator : elementul %d a
fost scos din %s\n' % (element, self.name))

if __name__ == '__main__':
    queue = multiprocessing.Queue()
    proces_producator = Producator(queue)
    proces_consumator = Consumator(queue)
    proces_producator.start()
    proces_consumator.start()
    proces_producator.join()
    proces_consumator.join()
```

Comunicare utilizând pipe

```
import multiprocessing
def creare_elemente(pipe):
    pipe_iesire, _ = pipe
    for element in range(4):
        pipe_iesire.send(element)
    pipe_iesire.close()
def multiply_elements(pipe1, pipe2):
    close, pipe_intrare = pipe1
    close.close()
    pipe_iesire, _ = pipe2
    try:
        while True:
            element = pipe_intrare.recv()
            print('am primit in pipe1:',element)
            x = element * element
            pipe_iesire.send(x)
            print('am trimis in pipe2:',x)
    except EOFError:
        pipe_iesire.close()
```

```
if __name__ == '__main__':
    # primul pipe cu elemente de la 0 la 9
    pipe1 = multiprocessing.Pipe(True)
    process_pipe1 = multiprocessing.Process(
        target=creare_elemente, args=(pipe1,))
    process_pipe1.start()
    # al doilea pipe
    pipe2 = multiprocessing.Pipe(True)
    process_pipe2 = multiprocessing.Process(
        target=multiply_elements, args=(pipe1, pipe2) )
    process_pipe2.start()
    pipe1[0].close()
    pipe2[0].close()
    try:
        while True:
            print('Am scos elementul:',pipe2[1].recv())
    except EOFError:
        print('End')
```

Sincronizarea proceselor

- Lock
- Event
- Condition
- Semaphore
- RLock
- Barrier

Exemplu simplu de apel barieră

```
import multiprocessing
from multiprocessing import Barrier, Lock,
Process
from time import time
import datetime as dt
def test_bariera(barrier, lock):
    name =
multiprocessing.current_process().name
    barrier.wait()
    now = time()
    with lock:
        print('Procesul %s -----> %s' % (name,
dt.datetime.fromtimestamp(now)))
def test_fara_bariera():
    name =
multiprocessing.current_process().name
    now = time()
    print('Procesul %s -----> %s' % (name,
dt.datetime.fromtimestamp(now)))
if __name__ == '__main__':
    barrier = Barrier(2)
    lock = Lock()
    Process(name='p1 - test_cu_bariera',
        target=test_bariera,
        args=(barrier, lock)).start()
    Process(name='p2 - test_cu_bariera',
        target=test_bariera,
        args=(barrier, lock)).start()
    Process(name='p3 - test_fara_bariera',
        target=test_fara_bariera).start()
    Process(name='p4 - test_fara_bariera',
        target=test_fara_bariera).start()
```

si rezultatul executiei

```
Procesul p2 - test_cu_bariera ----> 2019-04-10 07:38:19.407982
Procesul p1 - test_cu_bariera ----> 2019-04-10 07:38:19.408019
Procesul p3 - test_fara_bariera -----> 2019-04-10 07:38:19.408314
Procesul p4 - test_fara_bariera -----> 2019-04-10 07:38:19.408933
Process finished with exit code 0
```

Gestiunea stărilor între procese

```
import multiprocessing as mp
def worker(dictionary, cheie, element, contor):
    lock=mp.Lock()#trebuie?
    with lock:
        contor[0]=contor[0]+1
    dictionary[cheie] = element
    print('Cheie:', cheie, 'Ialoare:', element, 'sunt la al', contor[0], '-lea apel')
if __name__ == '__main__':
    manager = mp.Manager()  # handler de variabila comuna
    dictionary = manager.dict()
    contor = manager.list([0])
    contor[0] = 0
    sarcini = [mp.Process(target=worker, args=(dictionary, i, i * 2, contor))
            for i in range(5)]
    for treaba in sarcini:
        treaba.start()
    for treaba in sarcini:
        treaba.join()
    print('Rezultate:', dictionary)
```

si rezultatul executiei

```
Cheie: 0 Ialoare: 0 sunt la al 1 -lea apel
Cheie: 1 Ialoare: 2 sunt la al 2 -lea apel
Cheie: 2 Ialoare: 4 sunt la al 2 -lea apel
Cheie: 3 Ialoare: 6 sunt la al 3 -lea apel
Cheie: 4 Ialoare: 8 sunt la al 4 -lea apel
Rezultate: {0: 0, 1: 2, 2: 4, 3: 6, 4: 8}
```

Utlizarea unui pool de procese

```
import multiprocessing as mp
import  time as tm
import signal
import sys
def la_patrat(data):
    #tm.sleep(10000)
    val = data* data
    return val
def signal_handler(signal, frame):
    print('a aparut o operatie externa', signal)
    pool.terminate()
    pool.join()
    print('am terminat fortat procesul')
    sys.exit(0)
if __name__ == '__main__':
    intrari = list(range(10))
    pool = mp.Pool(processes=4)
    signal.signal(signal.SIGINT, signal_handler)
    calcul_pool = pool.map(la_patrat, intrari)
    pool.close()
    pool.join()
    print('Pool:', calcul_pool)
```

si rezultatul executiei

```
Pool: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Process finished with exit code 0
```

Cozi în multiprocessing

```
import multiprocessing
import os
from multiprocessing import Queue
q = Queue()
def proc_pid(n):
    q.put(os.getpid())#linux pid
    print("\n[{0}] Salut!".format(n))
procese = [ ]
for i in range(5):
    t = multiprocessing.Process(target=proc_pid, args=(i,))
    procese.append(t)#creez un pool de procese
    t.start()
for un_proces in procese:
    print(un_proces.name) #utlizez referinta la fiecare proces
    un_proces.join()
olista = [ ]
while not q.empty():
    olista.append(q.get())#scot din coada
print(olista,'de lungimea',len(olista))
```

si rezultatul executiei

```
[0] Salut!
[1] Salut!
[2] Salut!
Process-1
Process-2
Process-3
[3] Salut!
Process-4
[4] Salut!
Process-5
[28746, 28747, 28748, 28750, 28753]
de lungimea 5
Process finished with exit code 0
```

Concurent.future

- concurrent.futures.Executor:
- submit (function ,argument):
- map (function,argument):
- shutdown (Wait = True):
- concurrent.futures.Future:

Executors - Gestionari ai execuției

- concurrent.futures.ThreadPoolExecutor(max_workers)
- concurrent.futures.ProcessPoolExecutor(max_workers)

Reanalizăm performanțele

```
import concurrent.futures as cf
import time
lista_numere = list(range(1, 5))
def numara(numar):
    i = 0
    for i in range(10**7):
        i += 1
    return i*numar
def evaluare(element):
    element_rezultat = numara(element)
    print('element %s, rezultat este %s' % (element,
element_rezultat))
if __name__ == '__main__':
    # secvential
    start = time.time()
    tpornire = time.time()
    for element in lista_numere:
        evaluare(element)
    print('Executia secventiala a durat %s secunde' %
(time.time() - tpornire))
    # cu pool fire
    tpornire = time.time()
    with cf.ThreadPoolExecutor(max_workers=5) as
executor:
        for element in lista_numere:
            executor.submit(evaluare, element)
    print('Executia pool-ului de fire a durat %s secunde' %
(time.time() - tpornire))
    # cu pool procese
    tpornire = time.time()
    with cf.ProcessPoolExecutor(max_workers=5) as
executor:
        for element in lista_numere:
            executor.submit(evaluare, element)
    print('Executia pool-ului de procese a durat %s
secunde' % (time.time() - tpornire))
```

extras din rezultat de executie

```
Executia secventiala a durat 2.10807728767395 secunde
Executia pool-ului de fire a durat 3.307506799697876 secunde
Executia pool-ului de procese a durat 0.5447263717651367 secunde
Process finished with exit code 0
```

Gestiunea evenimentelor cu Asyncio

- Event loop:
- Coroutines:
- Futures:
- Tasks:

Metode specifice gestiunii buclei de evenimente

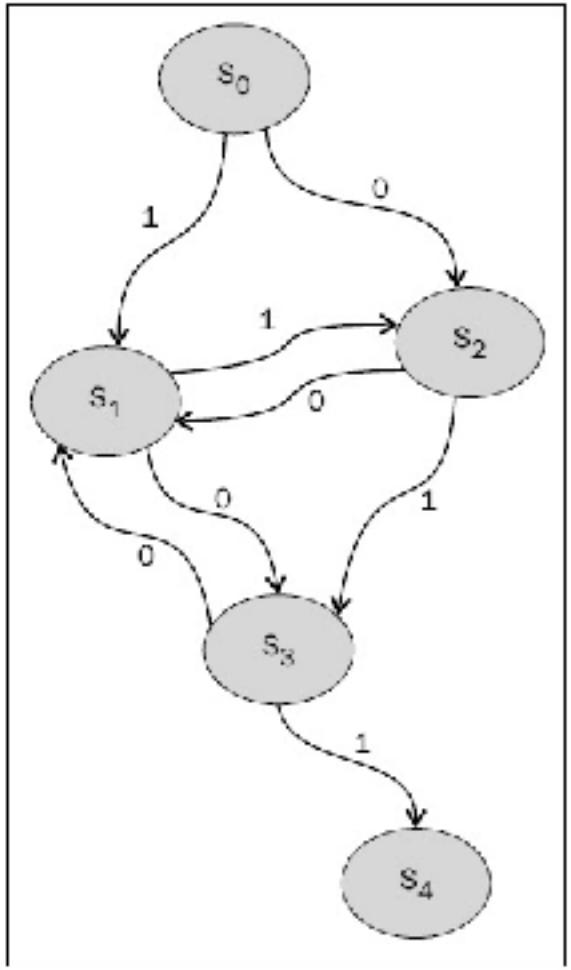
- loop = get_event_loop():
- loop.call_later(time_delay,callback,argument):
- loop.call_soon(callback,argument):
- loop.time():
- asyncio.set_event_loop():
- asyncio.new_event_loop():
- loop.run_forever():

Un prim exemplu de utlizare asyncio

```
import asyncio as asy
def functia_1(tstop, bucla):
    print('functia 1 apelata')
    if (bucla.time() + 1.0) < tstop:
        bucla.call_later(1, functia_2, tstop, bucla)
    else:
        bucla.stop()
def functia_2(tstop, bucla):
    print('functia 2 apelata')
    if (bucla.time() + 1.0) < tstop:
        bucla.call_later(1, functia_3, tstop, bucla)
    else:
        bucla.stop()
def functia_3(tstop, bucla):
    print('functia 3 apelata')
    if (bucla.time() + 1.0) < tstop:
        bucla.call_later(1, functia_1, tstop, bucla)
    else:
        bucla.stop()
if __name__ == '__main__':
    bucla_asincrona = asy.get_event_loop()
    sfarsit_bucla = bucla_asincrona.time() + 6.0
    bucla_asincrona.call_soon(functia_1,
sfarsit_bucla, bucla_asincrona)
    bucla_asincrona.run_forever()
    bucla_asincrona.close()
```

```
functia 1 apelata
functia 2 apelata
functia 3 apelata
functia 1 apelata
functia 2 apelata
functia 3 apelata
Process finished with exit code 0
```

Corutine



Să testăm următorul program cu sevența de intrări prezentate mai jos

Start → S_0 → I_0=0

↓

S_2 → I_2=0

↓

S_1 → I1=0

↓

S_3 → I3=1

↓

S_4 → Stop

Și programul ...

```
import asyncio as asy
@asy.coroutine
def stare0_start():
   print('Start din S0 \n')
   valoare_intrare = int(input('Valoare Intrare in S0='))
   if valoare_intrare == 0:
      rezultat = yield from stare2(valoare_intrare)
   else: rezultat = yield from stare1(valoare_intrare)
@asy.coroutine
def stare1(valoare_tranzitie):
   valoare_iesire = 'stare S1 cu valoare de intrare = %s\n' %
valoare_tranzitie
   valoare_intrare = int(input('Valoare Intrare in S1='))
   print('...S1 - calculez...')
   if valoare_intrare == 0:
      rezultat = yield from stare3(valoare_intrare)
   else: rezultat = yield from stare2(valoare_intrare)
   return valoare_iesire + 'apel stare S1 cu %s' % rezultat
@asy.coroutine
def stare2(valoare_tranzitie):
   valoare_iesire = 'stare S2 cu valoare de tranzitie = %s\n' %
valoare_tranzitie
   valoare_intrare = int(input('Valoare Intrare in S2='))
   print('...S2 - calculez...')
   if valoare_intrare == 0:
      rezultat = yield from stare1(valoare_intrare)
   else: rezultat = yield from stare3(valoare_intrare)
   return valoare_iesire + 'apel stare 2 cu %s' % rezultat
@ asy.coroutine
def stare3(valoare_tranzitie):
   valoare_iesire = 'stare S3 cu valoare de tranzitie = %s\n' %
valoare_tranzitie
   valoare_intrare = int(input('Valoare Intrare in S3='))
   print('...S3 - calculez...')
   if valoare_intrare == 0:
      rezultat = yield from stare1(valoare_intrare)
   else: rezultat = yield from stare4_stop(valoare_intrare)
   return valoare_iesire + 'apel stare 3 cu %s' % rezultat
@ asy.coroutine
def stare4_stop(valoare_tranzitie):
   print('...S4 - calculez...')
   valoare_iesire = 'sfarsit stare with tranzitie value = %s\n' %
valoare_tranzitie
   print('...Oprire...')
   return valoare_iesire
if __name__ == '__main__':
   print('Executie FSM utilizand asyncIO si Corutine [I
reprezinta intrarea in stare]')
   bucla = asy.get_event_loop()
   bucla.run_until_complete(stare0_start())
```

Gestiune task-uri cu asyncio

```
import asyncio as asy
@asy.coroutine
def factorial(number):
    fact = 1
    for i in range(2, number + 1):
        print('Calculez factorial(%s)' % i)
        yield from asy.sleep(1)
        fact *= i
    print(' factorial(%s) = %s' % (number, fact))
@asy.coroutine
def fibonacci(number):
    a, b = 0, 1
    for i in range(number):
        print('Calculez fibonacci(%s)' % i)
        yield from asy.sleep(1)
        a, b = b, a + b
    print(' fibonacci(%s) = %s' % (number, a))
@asy.coroutine
def coeficient_binomial(n, k):
    rezultat = 1
    for i in range(1, k + 1):
        rezultat =  rezultat*(n - i + 1)/i
        print('Calculez coeficientul binomial(%s)' % i)
        yield from asy.sleep(1)
    print(' coeficientul binomial(%s, %s) = %s' % (n,
k,  rezultat))
if __name__ == '__main__':
    tasks = [asy.Task(factorial(7)),
             asy.Task(fibonacci(7)),
             asy.Task(coeficient_binomial(14, 7))]
    bucla = asy.get_event_loop()
    bucla.run_until_complete(asy.wait(tasks))
    bucla.close()
```

Asyncio și Futures

- instanțiere obiect future

import asyncio

future = asyncio.Future()

- metodele acestei clase sunt următoarele:
  - cancel():
  - result():
  - exception():
  - add_done_callback(fn):
  - remove_done_callback(fn):
  - set_result(result):
  - set_exception(exception):

Și un exemplu de utilizare

```
import asyncio as asy
@asy.coroutine
def prima_corutina(future, numar):
    contor = 0
    for i in range(1, numar + 1):
        contor += 1
    yield from asy.sleep(1)
    future.set_result('In prima corutina calculez
suma a %s numere = %s' % (numar, contor))
@asy.coroutine
def a_doua_corutina(future, numar):
    contor = 1
    for i in range(2, numar + 1):
        contor *= i
    future.set_result('In a doua corutina calculez
factorial(%s) = %s' % (numar, contor))
def preiau_rezultatul(future):
    print(future.result())
if __name__ == '__main__':
    numar1 = int(input('Numarul 1 = '))
    numar2 = int(input('Numarul 2 = '))
    bucla = asy.get_event_loop()
    future1 = asy.Future()
    future2 = asy.Future()
    tasks = [prima_corutina(future1, numar1),
            a_doua_corutina(future2, numar2)]
    future1.add_done_callback(preiau_rezultatul)
    future2.add_done_callback(preiau_rezultatul)
    bucla.run_until_complete(asy.wait(tasks))
    bucla.close()
```

și exemplu de utilizare

```
Numarul 1 = 10
Numarul 2 = 10
In a doua corutina calculez factorial(10) = 3628800
In prima corutina calculez suma a 10 numere = 10
Process finished with exit code 0
```

Tratarea canalelor de comunicare cu socket

- **Deschiderea unui socket la server pentru recepționare date**

```
canal_comunicare = socket(AF_INET, SOCK_STREAM)
canal_comunicare.bind((serverHost, serverPort))
canal_comunicare.listen(3)
conn, addr = canal_comunicare.accept()
date_primite = conn.recv(1024)
```

- Pentru început se crează canalul de comuincare pe server cu

```
socket(family, type [, proto]),
```

- care ne oferă un canal de comunicare din categoria *family*.
- Categorii de canale de comunicare specifice bibliotecii socket din python:

Categorie	Descriere
AF_INET	Protocol Ipv4 (TCP, UDP)
AF_INET6	Protocol Ipv6 (TCP, UDP)
AF_UNIX	Protocoale de domeniu

Tratarea canalelor de comunicare

- Tipuri de canale de comunicare din biblioteca socket

Tip	Descriere
SOCK_STREAM	Deschide un fisier existent pentru citire.
SOCK_DGRAM	Deschide un fisier pentru scriere. Cu suprascriere.
SOCK_RAW	Deschide un fisier existent pentru adăugări sau modificări
SOCK_RDM	Deschide un fisier atât pentru scriere cât și pentru citire, nu face suprascriere.
SOCK_SEQPACKET	Deschide un fisier atât pentru scriere cât și pentru citire, face suprascriere.

Gestiune simplă a rețelei

```
import socket
def afla_informatii_despre_masina_locala():
    nume_masina = socket.gethostname()
    adresa_ip = socket.gethostbyname(nume_masina)
    print("Numele sistemului local: %s" % nume_masina)
    print("Adresa de IP a sistemului local: %s" % adresa_ip)
def afla_informatii_despre_masina_la_distanta(nume):
    try:
        print("Adresa IP a masinii cu numele %s: %s" % (nume, socket.gethostbyname(nume)))
    except socket.error as err_msg:
        print("%s: %s" % (nume, err_msg))
if __name__ == '__main__':
    afla_informatii_despre_masina_locala()
    afla_informatii_despre_masina_la_distanta('www.tuiasi.ro')
```

si rezultatul executiei

```
Numele sistemului local: home
Adresa de IP a sistemului local: 127.0.1.1
Adresa IP a masinii cu numele www.tuiasi.ro:
81.180.223.65

Process finished with exit code 0
```

Caut servicii

```
import socket as sk

def caut_servcii(nume_protocol, port_list):
    for port in port_list:
        try:
            s=sk.getservbyport(port, nume_protocol)
            print("Pe portul: %s am serviciul cu numele %s care utilizeaza protocolul %s" % (port,
s,nume_protocol))
        except: continue

if __name__ == '__main__':
    port_list =[1,80,8080]
    nume_protocol = 'udp'
    caut_servicii(nume_protocol,port_list)
    nume_protocol = 'tcp'
    caut_servicii(nume_protocol,port_list)
```

Mărunțișuri

```
import socket as sk
DIM_BUFF_SEND = 4096
DIM_BUFF_RECV = 4096
def test_socket_timeout():
    s = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    print("Timp maxim de astepare: %s" % s.gettimeout())
    s.settimeout(100)
    print("Timpul de asteptare curent: %s" % s.gettimeout())
def modific_dim_buffer(trimitere,receptie):
    sock = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
    bufsize = sock.getsockopt(sk.SOL_SOCKET, sk.SO_SNDBUF)
    print("Dimensiune tampon inainte de modificare:%d" % bufsize)
    sock.setsockopt(sk.SOL_TCP, sk.TCP_NODELAY, 1)
    sock.setsockopt( sk.SOL_SOCKET, sk.SO_SNDBUF, trimitere)
    sock.setsockopt( sk.SOL_SOCKET, sk.SO_RCVBUF, receptie)
    bufsize = sock.getsockopt(sk.SOL_SOCKET, sk.SO_SNDBUF)
    print("Dimensiune tampon dupa de modificare:%d" % bufsize)
if __name__ == '__main__':
    test_socket_timeout()
    modific_dim_buffer(DIM_BUFF_SEND,DIM_BUFF_RECV)
```

si rezultatul executiei

```
Timp maxim de astepare: None
Timpul de asteptare curent: 100.0
Dimensiune tampon inainte de modificare:16384
Dimensiune tampon dupa de modificare:8192

Process finished with exit code 0
```

Aplicații client server – serverul

```
from socket import *
serverHost = ''  # asculta pe toate  interfetele de retea
serverPort = 8888
canal_comunicare_server = socket(AF_INET, SOCK_STREAM)
canal_comunicare_server.bind((serverHost, serverPort))
canal_comunicare_server.listen(3)
while True:
    conexiune,addr = canal_comunicare_server.accept()
    print("Conexiune cu un client:", addr)
    while True:
        data = conexiune.recv(1024)
        if not data:
            break
        print('Serverul a primit:', repr(data))
        conexiune.sendall(data)
    conexiune.close()
```

Aplicatii client server – clientul

```
import sys
from socket import *
serverHost = ''
serverPort = 8888
mesaj = ['Mesajul unu de la client', 'Mesajul doi']
if len(sys.argv) > 1:
   serverHost = sys.argv[1]
canal_comunicare_client = socket(AF_INET, SOCK_STREAM)
canal_comunicare_client.connect((serverHost, serverPort))
for element in mesaj:
    canal_comunicare_client.sendall(element.encode())
   date_receptionate = socketul_client.recv(1024)
   print("Clientul a primit:", date_receptionate)
canal_comunicare_client.close()
```

Aplicatii client server – server cu socketserver

```
import socketserver
HOST, PORT = "", 8889

class MyTCPRequestHandler(socketserver.StreamRequestHandler):
    def handle (self):
        self.date_primite = self.request.recv(1024).strip()
        print("{} a trimis:".format(self.client_address[0]))
        print(self.date_primite)
        self.request.sendall(self.date_primite.upper())
server = socketserver.TCPServer((HOST,PORT),MyTCPRequestHandler)

server.serve_forever()
```

Și un client pentru serverul anterior

```
import socket
HOST, PORT = "", 8889
date_test = "Test de emisie receptie\nsi a doua linie"
canal_comunicare = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    canal_comunicare.connect((HOST, PORT))
    canal_comunicare.sendall(bytes(date_test + "\n", "utf-8"))
    date_receptionate = str(canal_comunicare.recv(1024), "utf-8")
finally:
    canal_comunicare.close()
print("Am trimis la sever:     {}".format(date_test))
print("Am primit de la server: {}".format(date_receptionate))
```

Server Ecou utilizând TCP-IP bazat pe asyncio

```
import asyncio as asy
class ServerEcou(asy.Protocol):
    def connection_made(self, transport):
        nume_client =
transport.get_extra_info('nume client')
        print("Conexiune acceptata cu
{}".format(nume_client))
        self.transport = transport
    def data_received(self, date):
        mesaj = date.decode()
        print("Date Receptionate:
{}".format(mesaj))
        print("Trimt: {!r}".format(mesaj))
        self.transport.write(date)
        print("Inchid socket-ul clientului")
        self.transport.close()
bucla = asy.get_event_loop()
c1 = bucla.create_server(ServerEcou, '127.0.0.1',
8888)
server = bucla.run_until_complete(c1)
try:
    bucla.run_forever()
except KeyboardInterrupt:
    pass
server.close()
bucla.run_until_complete(server.wait_closed())
bucla.close()
```

Client Ecou utilizând TCP-IP bazat pe asyncio

```
import asyncio as asy
class ClientEcou(asy.Protocol):
  def __init__(self, mesaj, bucla):
    self.mesaj = mesaj
    self.bucla = bucla
  def connection_made(self, transport):
    transport.write(self.mesaj.encode())
    print('Date trimise:
{!r}'.format(self.mesaj))
  def data_received(self, data):
    print("Date primite:
{!r}".format(data.decode()))
  def connection_lost(self, exc):
    print("serverul a terminat conexiunea")
    print("Oprire bucla de evenimente")
    self.bucla.stop()
bucla = asy.get_event_loop()
mesaj = 'Am trimis ceva'
c = bucla.create_connection(lambda:
ClientEcou(mesaj, bucla), '127.0.0.1',
8888)
bucla.run_until_complete(c)
bucla.run_forever()
bucla.close()
```