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')
```