17. Конкурентност

15 декември 2022

Въпрос

Кои са трите основни неща в един изпълним файл/DLL?

Кода на приложението

import таблица

export таблица

Въпрос+

Как се нарича преобръщането на данните подаваме стойности от C към Python (and vice-versa)?

marshalling

Въпрос++

Какво са built-in функциите в Питон?

Неща написани на C

... и вградени в интерпретатора (също е написан на C).

... или съществуващи във външни (native) библиотеки

Въпрос+++

Какви проблеми решава Python/C API-то?

Скорост

Достъп до неща, недостъпни попринцип в Питон интерпретатора

Преизползване на съществуващ C код в Python

Въпрос++++

Колко от вас са предали “домашното”, свързано с проектите?

3

Срокът е до следващият вторник.

Няма да можем да ви напомним отново.

Що е то конкурентност?

~~Когато две изчисления се случват едновременно.~~

Когато две изчисления нямат ясно дефинирана последователност на изпълнение.

Тоест:

Предварително знаем, че:

- След момент t0 ще започне изпълнението на две задачи T1 и T2
- В момент t1 и двете задачи ще са приключили

Няма как предварително да знаем:

- Реда, в който ще се изпълнят
- Дали ще си предават управлението
- Коя ще приключи първа

Паралелизъм?

Това, за което повечето хора си мислят, когато им говориш за конкурентност.

Две изчисления, които реално се изпълняват едновременно.

Предполага наличието на много ядра/процесори, така че няколко задачи наистина да вървят паралелно.

Пример

Конкурентност (MFF)

Паралелизъм (MFM)

Как се постига конкурентност в Python?

Отделни процеси

Нишки

- Зелени нишки - нашия код се грижи за предаването на контрол
- Системни нишки - операционната система се грижи за предаването на контрол

GIL

Един python процес винаги работи върху едно ядро.

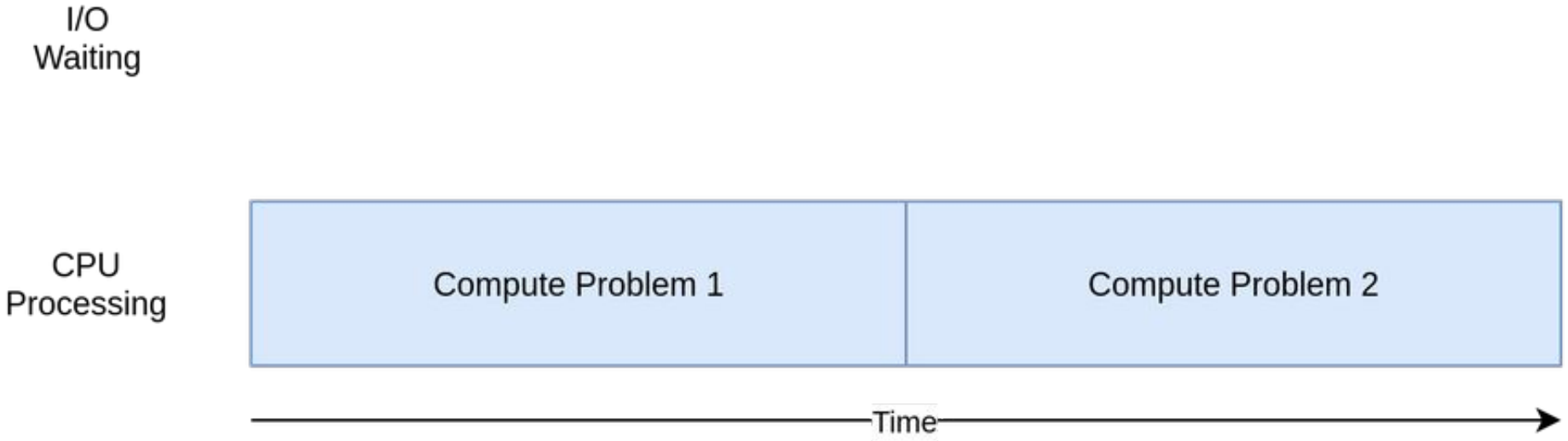
Досадно е, но уви спестява един ред други проблеми.

Възможно е дори реферирането на променлива в няколко нишки едновременно да създаде проблем (защо?).

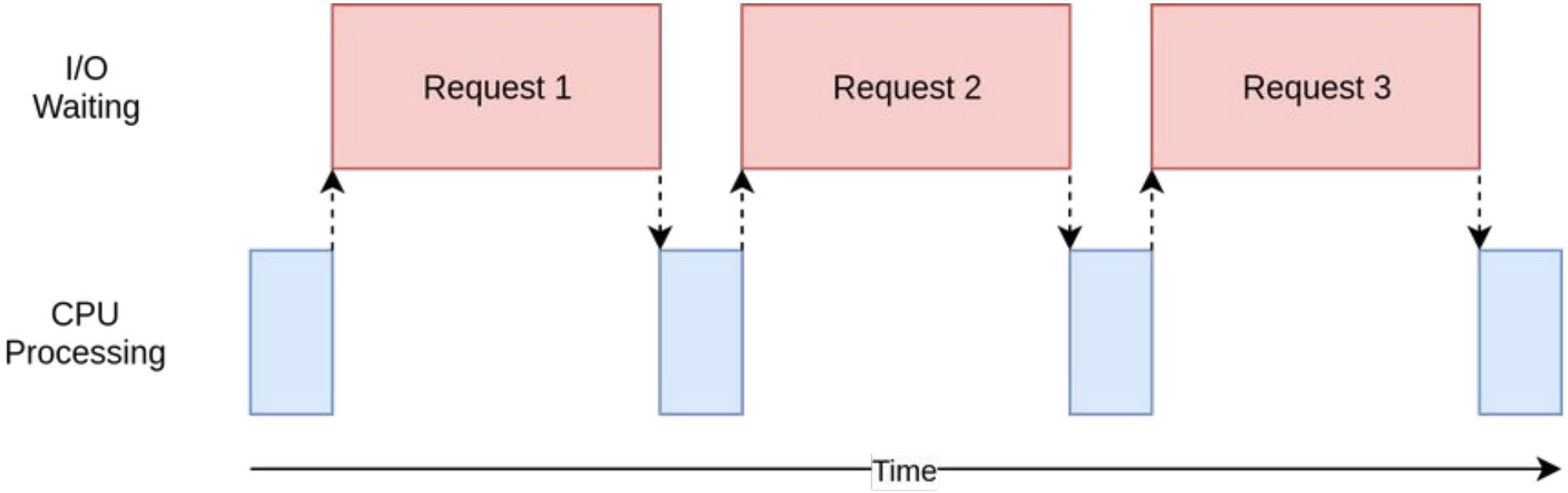
Чакай малко, това звучи супер неефективно!

- Има нещо вярно
- Но!
- Нишките си имат приложение и в Python
- IO-bound vs CPU-bound процеси

CPU-bound


I/O
Waiting
CPU
Processing
Compute Problem 1
Compute Problem 2
Time

IO-bound


I/O
Waiting
CPU
Processing
Request 1
Request 2
Request 3
Time

fork

Как (unixy) операционна система реализира паралелизъм

- fork създава ново копие на програмата, която изпълняваме
- Всички ресурси и променливи запазват стойността си в процеса-син
- След създаването на новия процес, всички промени са локални
- Все едно клонираме хора, за да вършим повече работа едновременно

Пример с fork (на C)

```c
#include <stdio.h>

int main()
{
    printf("before\n");
    if (fork())
        printf("father\n");
    else
        printf("son\n");
    printf("both\n");
}
```

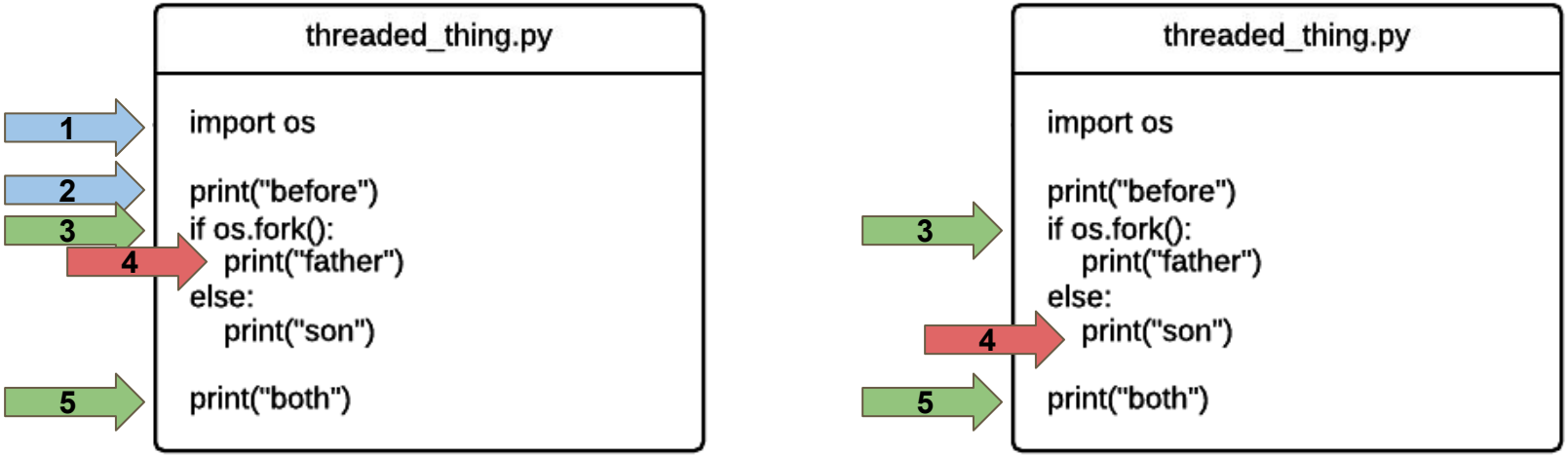
```
before
son
both
father
both
```

Пример с fork (на Python)

```
import os

print("before")
if os.fork():
    print("father")
else:
    print("son")
print("both")
```

В картинки


threaded_thing.py
import os
print("before")
if os.fork():
print("father")
else:
print("son")
print("both")
1
2
3
4
5
threaded_thing.py
import os
print("before")
if os.fork():
print("father")
else:
print("son")
print("both")
3
4
5

По-сложен пример с fork

```
import os
import time

def log(msg): print("\n* " + msg)

orders = 0
while True:
    order = input('Enter order: ')
    if not order: continue
    if order in ('q', 'x', 'quit', 'exit'): break
    pid = os.fork()
    if pid == 0:
        time.sleep(3)
        log("Order '{0}' is ready!".format(order))
        break
    else:
        log("Roger that '{0}'({1}). Please, wait.".format(order, orders))
        orders += 1
```

Друг пример с fork

```
import os

pid = os.fork()
if pid == 0:
    os.execlp('date', 'date')
else:
    status = os.wait()
    print("Father: son has finished {0}".format(status))
```

Можем да ползваме и сигнали - `import signal`

Предимства и недостатъци на fork

Против:

- Само за UNIX
- Създаването на нов процес е бавно и паметоемко
- Комуникацията между процеси е трудна - нямат обща памет

За:

- Стабилност
- Синът е независим - ако омаже нещо, бащата няма да пострада

Създаване на нова нишка

Даваме функция с параметри към нея

или

- Нишка — наследник на threading.Thread
- Код — в метода run
- Създаване и изпълнение на кода паралелно — метод start

Пример с нишки (функция)

```
import threading

def f(name):
    print("Hello from {0}".format(name))

thread = threading.Thread(target=f, args=('Bob',))
thread.start()
thread.join()
```

Пример с нишки 2 (наследник на Thread)

```
import threading
import time

orders = 0

class Chef(threading.Thread):

    def __init__(self, order):
        self.order = order
        threading.Thread.__init__(self)

    def run(self):
        time.sleep(3)
        log("Order '{0}' is ready!".format(self.order))

while True:
    order = input('Enter order: ')
    if not order: continue
    if order in ('q', 'x', 'quit', 'exit'): break
    chef = Chef(order)
    chef.start()
    log("Roger that '{0}'. Please, wait in quiet desperation.".format(order))
    orders += 1
```

Критични секции

- Части от кода, които могат да бъдат изпълнени само от една нишка/процес в даден момент се наричат критични секции
- Те са критична част от многозадачното програмиране
- Има много похвати за реализирането на критични секции
- Всичките са равномощни на нещо, наречено **семафор**

threading.Lock

- threading.Lock() ни връща Lock обект
- Викането на метода му acquire() ни гарантира, че само ние притежаваме този Lock
- Викането на release() освобождава Lock-а и разрешава някой друг да го заключи с acquire()
- Ако викнем acquire() докато Lock-а е зает — методът чака, докато не се освободи

Забележка: Кодът от този и всички следващи примери, можете да намерите [ето тук](#).

with и обекти с acquire и release

- Всички Lock-оподбни обекти от threading са и context manager-и
- Ако ги ползваме с with ни се гарантира викането на acquire() преди и на release() след блока

```
with bathroom:
    self.log("--> (entered the bathroom)")
    time.sleep(random.random())
    self.log("<-- (left the bathroom)")
```

Модерно строителство, ресторанти и семафори

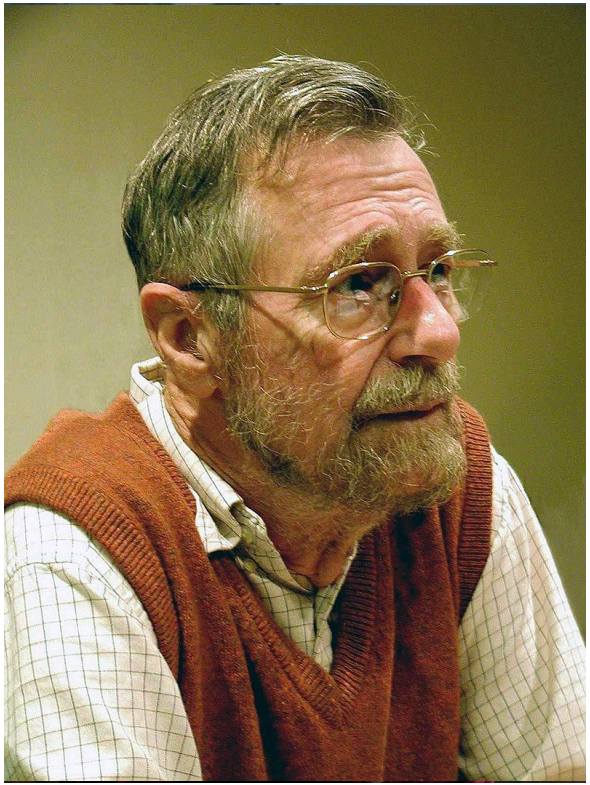
А ако има повече от една тоалетна в сградата?

Или още по-добре:

- Ресторант с 10 човека
- Всеки е едновременно готвач и сервитьор
- 5 фурни

Enter семафори

- Числова променлива v
- Операция P(v) — чакай докато v > 0, след което v -= 1
- Операция V(v) — v += 1
- Предложени от Дийкстра
- Не този
- Този (Едсхер)



threading.Semaphore

- threading.Semaphore(k) ни връща семафор с интерфейс като на Lock и стойност k
- При всяко изпълнение на acquire() стойността се намалява с 1
- При всяко изпълнение на release() стойността се увеличава с 1
- Ако стойността е 0, acquire() спи, докато някой не я увеличи с release()
- Lock() е еквивалентен на Semaphore(1)

threading.Event

- По-проста комуникация, в която няколко нишки изчакват събитие от друга
- `wait()` блокира докато събитието не се случи
- `set()` „случва“ събитието

Коледа на село

- 2 баби, майсторки баничарки, пекат баници за 3-мата си внука
- Искаме след като бабите опекат нещо, някой от свободните внуци да започва да яде
- Бабите се казват producer-и, внуците — consumer-и

threading.Condition

- `wait()` — чака, докато някоя баба не произведе нова баница
- `notify()` — това трябва да каже една баба, която е опекла баница. Ще събуди някой от чакащите. Ако няма чакащи няма да направи нищо...
- `notify_all()` — ще събуди всички чакащи
- `release()` и `acquire()` работят върху вътрешен за Condition Lock, който може да се подаде при конструиране
- `wait()` и `notify()` работят само ако владеем вътрешния Lock

threading.local

- Инстанцира се чрез `threading.local()`
- Позволява свободен достъп до атрибути
- Всяка нишка вижда различни стойности зад атрибутите на този обект

multiprocessing

multiprocessing модулът

- Дава ни подобни възможности като threading, но за процеси
- Crossplatform - Unix/Windows
- Благинки за синхронизация - Semaphore, Lock, RLock, Condition, Event
- Благинки за обмен на данни - Queue, Pipe
- Възможности за обща памен (Value, Array) от елементарни данни (int/float/byte/...) и ctypes структури
- Възможност за общи обекти - Manager
- Басейн (Pool)

multiprocessing - примери

- “Нормален” пример
- Пример с обща памет
- Пример с lock
- Manager
- Нещо много по-удобно - map (ама multiprocessing map)

Паралелизиран map

```
import urllib
from multiprocessing import Pool

urls = ['http://www.python.org', 'http://www.python.org/about/',
        'http://github.com/', 'http://www.bg-mamma.com/']

def fetch(url):
    try:
        return urllib.request.urlopen(url).status
    except urllib.error.URLError as e:
        return e.code

pool = Pool(4)
statuses = pool.map(fetch, urls)
pool.close()
pool.join()
print(statuses)
```

Неща, за които трябва да се внимава под Windows

- Аргументите на Process.__init__() трябва да са „pickable“
- Не е хубаво да достъпвате глобални променливи
- Задължително if __name__ == '__main__'
- Трябва да се внимава с действията при import

Въпроси?