Thread

Th.S Trần Đức Lợi
Pythonvietnam.info

Ôn tập bài cũ

- Ôn lại nội dung đã học về **Cơ Sở Dữ Liệu**
- Chữa bài **Làm việc với redis**

Mục đích bài học

- Tìm hiểu về đa luồng và xử lý đa luồng trong python

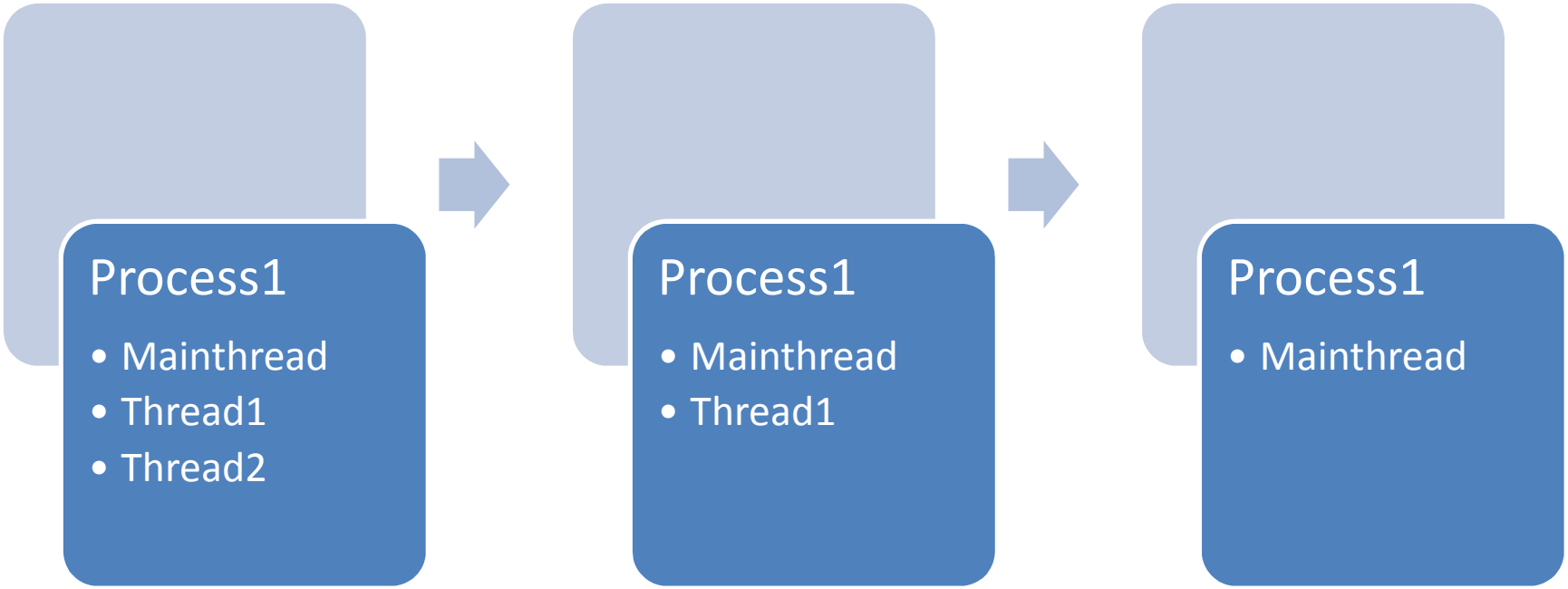
Thread

- Process và thread đối với hệ điều hành
- Tại sao cần phải xử lý đa luồng
  - GUI
  - Browser Tabs
  - Server

Process

- Process là gì?
- Một process có thể có nhiều threads
- Các process thông thường không sử dụng chung tài nguyên
- Thread được coi là một lightweight process

Thread


Process1
• Mainthread
• Thread1
• Thread2
Process1
• Mainthread
• Thread1
Process1
• Mainthread

Bổ trợ: Logging

- Giúp debug và làm tường minh thông tin trong chương trình.
- Cú pháp:
  - import logging
  - logging.basicConfig(***format***='[%(levelname)s]:[%(threadName)-10s]:%(message)s', ***level***=logging.DEBUG)
  - logging.info("Init completed.")

Tạo một thread mới

- Cú pháp:
- import threading
- t1 = threading.Thread(name="Thread 1", target=Hello, ***args***=("thread 1",))
- t1.start()

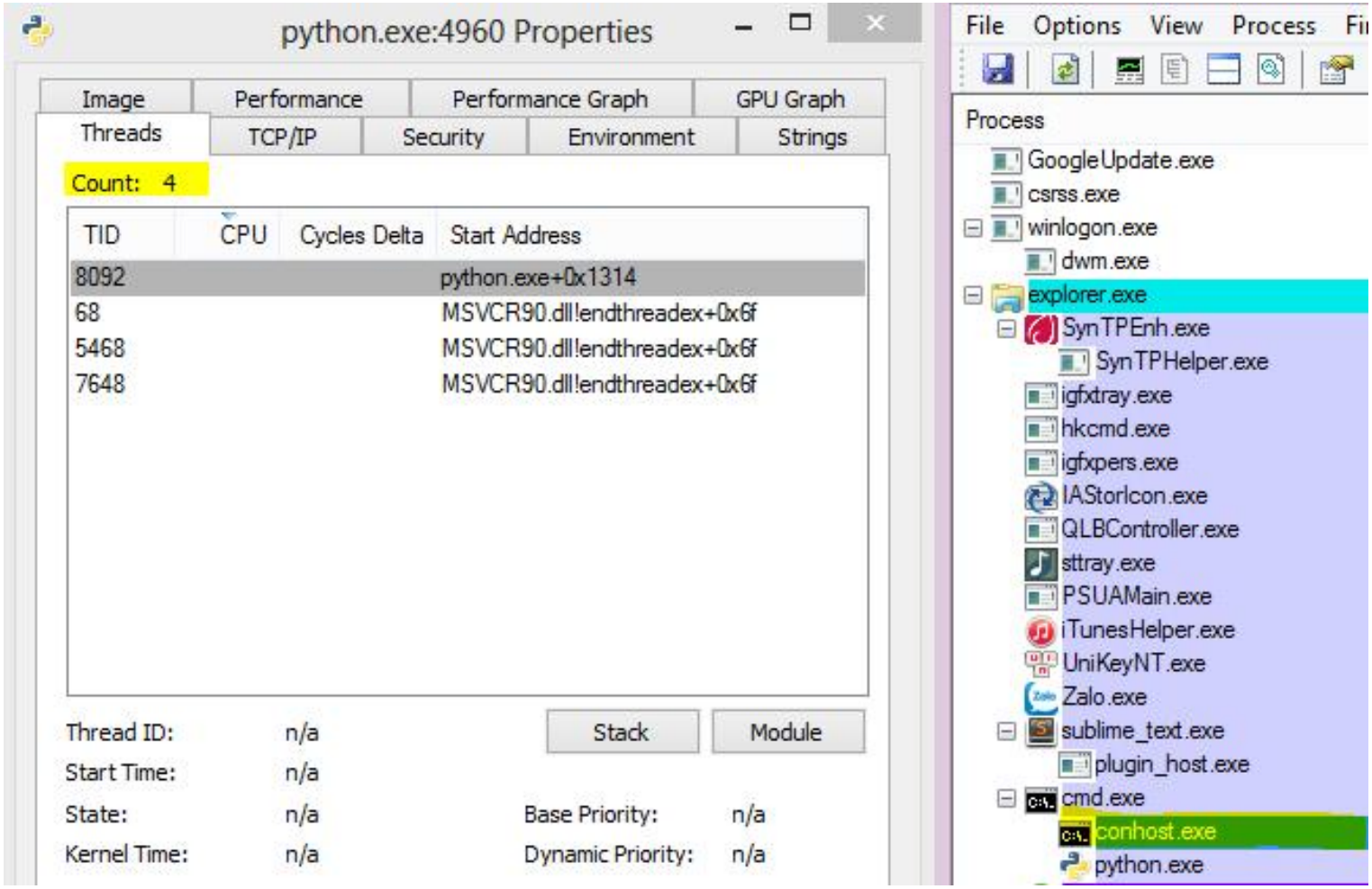
Tạo một thread mới

- Truyền tham số từ main thread vào hàm thực thi của thread bằng ***args***
- Khởi động thread bằng lệnh start()

Thread: Current Thread

- Dùng lệnh:
  - Threading.currentThread().getName()
- Thực thi và kiểm tra số thread được tạo trên hệ điều hành
- Windows: Process Explorer
- https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

Thread


python.exe:4960 Properties
Image
Performance
Performance Graph
GPU Graph
Threads
TCP/IP
Security
Environment
Strings
Count: 4
TID
CPU
Cycles Delta
Start Address
8092
python.exe+0x1314
68
MSVCR90.dll!endthreadex+0x6f
5468
MSVCR90.dll!endthreadex+0x6f
7648
MSVCR90.dll!endthreadex+0x6f
Thread ID: n/a
Start Time: n/a
State: n/a
Kernel Time: n/a
Stack
Module
Base Priority: n/a
Dynamic Priority: n/a
File
Options
View
Process
Process
GoogleUpdate.exe
csrss.exe
winlogon.exe
dwm.exe
explorer.exe
SynTPEnh.exe
SynTPHelper.exe
igfxtray.exe
hkcmd.exe
igfxpers.exe
IAStorIcon.exe
QLBController.exe
sttray.exe
PSUAMain.exe
iTunesHelper.exe
UniKeyNT.exe
Zalo.exe
sublime_text.exe
plugin_host.exe
cmd.exe
conhost.exe
python.exe

Non-daemon thread

- Mặc định khi tạo thread mới
- Khi mainthread sẽ đợi cho đến khi tất cả các thread con hoàn thành xong mới thoát
- Chương trình thoát hoàn toàn
- Cú pháp:
  - t1.setDaemon(True)

Daemon Thread

- Phải set tường minh
- Người dùng phải tự kiểm soát thread dạng này
- Chương trình chính có thể tự thoát khi hoàn thành mà không phụ thuộc vào daemon thread
- Cú pháp:
  - t2.setDaemon(False)

Ví dụ

- Ví dụ về cách sử dụng Daemon và Non-Daemon Thread

Bài tập

- Viết chương trình push-pop redis dưới dạng thread

Thread: Join()

- Để giúp chỉ tường minh main thread phải đợi daemon thread hoàn thành xong mới được thoát
- Cú pháp:
  - Daemonthr.start()
  - Daemonthr.join()

Thread: join()

- Mặc định join() sẽ block main thread mãi mãi đến khi daemon thread hoàn thành
- Set thêm tham số timeout để chỉ thời gian join() hết hiệu lực
- Cú pháp: d.join(seconds)

Thread: Đồng bộ threads

- Giao tiếp thông qua đối tượng Event theo cơ chế: **một thread** sẽ đưa ra tín hiệu và các thread khác đợi tín hiệu
- Số lượng thread đợi 1 tín hiệu là không giới hạn
- Khai báo event:
  - event = threading.Event()

Thread: Đồng bộ threads

- Set() hoặc clear() một event
- Các thread sẽ đợi cho đến khi event được set để thực hiện hoặc khi hết timeout của lệnh wait(timeoutseconds)
- Kiểm tra xem Event đã được set chưa:
  - isSet() hoặc is_set()

Thread: Đồng bộ threads

- Giới thiệu chương trình ví dụ về đồng bộ giữa các thread với Events

Thread: Đồng bộ threads

- So sánh thực thi các thread khi wait có timeout và không có timeout
- Event không phải là phương thức đồng bộ duy nhất giữa các threads (Condition)

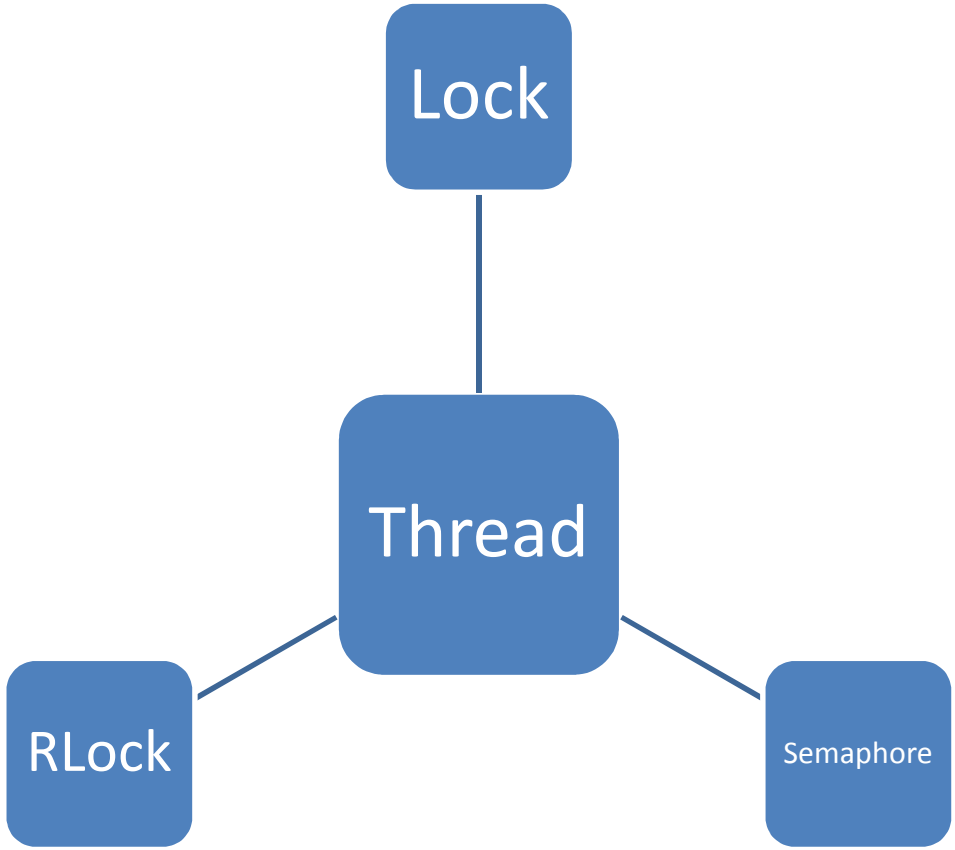
Thread: Đồng bộ threads

- Bài tập:
  - Thực hiện một chương trình đọc các giá trị từ Redis và DB MySQL. Chỉ thực hiện đọc từ DB MySQL nếu như đọc trên redis không có giá trị trả về. Thời gian chờ là 30s.

Thread: Quản lý tài nguyên

- Vấn đề khi làm việc đa luồng là ***quản lý truy nhập tài nguyên***
- Có nhiều hơn 1 thread cùng muốn truy cập, sửa đổi một biến, tài nguyên, … dùng chung

Thread: Quản lý tài nguyên



Thread: Quản lý tài nguyên

- Cơ chế Lock
  - Tại một thời điểm một lock chỉ có thể được sở hữu bởi nhiều nhất là 1 thread
  - Nếu một thread cố gằng sở hữu lock đang bị chiếm bởi 1 thread khác thì thread đó sẽ phải đợi cho đến khi lock được giải phóng

Thread: Quản lý tài nguyên

- ***.acquire()*** để chiếm lock
- ***.release()*** để giải phóng lock
- Hoặc sử dụng ***with lock:***
- ***.acquire(False)*** để cố gắng chiếm lock nhưng sẽ trả về fail nếu như lock đã bị chiếm và do đó dùng để kiểm tra
- ***.locked()*** để kiểm tra lock đã bị chiếm chưa

Thread: Quản lý tài nguyên

- Vấn đề phát sinh:
  - Không quan tâm thread nào đang chiếm lock ngay kể cả thread hiện thời mà lock
- Ý tưởng:
  - Rlock (re-entrant locks) : chỉ chặn việc lấy lock đối với các thread khác thread hiện thời đang cầm lock

Thread: Quản lý tài nguyên

- lock = threading.Lock()
- lock.acquire()
- lock.acquire() *# this will block*

- lock = threading.RLock()
- lock.acquire()
- lock.acquire() *# this won't block*

Thread: Semaphore

- Là cơ chế cao cấp hơn Lock
- Cho phép một số lượng thread nhất định có thể giữ ***semaphore***
- Nếu vượt quá sẽ block, không cho giữ semaphore
- Biến đếm sẽ giảm dần sau mỗi lần ***semaphore*** bị chiếm giữ và tăng dần nếu ngược lại
- Nếu biến đếm bằng 0 -> semaphore sẽ bị block

Thread: Semaphore

- Semaphore được dùng để hạn chế truy nhập đến tài nguyên
- Giá trị mặc định là 1 connection

Thread: Semaphore

- semaphore = threading.***BoundedSemaphore***()
- semaphore.acquire() *# decrements the counter*
- ... access the shared resource
- semaphore.release() *# increments the counter*

Tổng kết bài học

- Định nghĩa, cú pháp thread, process
- Daemon vs Non-Daemon thread
- Đồng bộ giữa các thread với Event
- Quản lý tài nguyên dùng chung

Bài tập

- Viết lại chương trình pubsub với thread