LẬP TRÌNH KHÔNG ĐỒNG BỘ TRONG PYTHON

Nội dung

- Cập nhật một thanh tiến trình bằng cách sử dụng một thread (luồng)
- Cập nhật hai thanh tiến trình bằng hai luồng
- Cập nhật các thanh tiến trình bằng cách sử dụng các luồng bị ràng buộc với một cơ chế khóa
- Cập nhật các thanh tiến trình đồng thời bằng các hoạt động không đồng bộ
- Quản lý tài nguyên bằng trình quản lý bối cảnh

Luồng (Thread) trong Python & PyQt

- **Luồng (Thread)** cho phép nhiều tác vụ chạy **đồng thời trong cùng một chương trình**, giúp ứng dụng **không bị treo** khi xử lý nặng (rất quan trọng với PyQt).
- Mỗi luồng hoạt động **độc lập**, có thể **tạm dừng (sleep)** và tiếp tục chạy.
- Python sử dụng mô-đun **threading** để quản lý luồng.

Các hàm thường dùng trong threading

- activeCount() – số luồng đang chạy
- currentThread() – luồng hiện tại
- enumerate() – danh sách các luồng đang hoạt động

Lớp Thread và các phương thức chính

- start() – khởi động luồng
- run() – nội dung công việc của luồng
- join() – chờ luồng kết thúc
- isAlive() – kiểm tra luồng còn chạy hay không
- getName(), setName() – lấy / đặt tên luồng

Kết hợp đa luồng/async giúp giao diện không bị treo khi xử lý tác vụ nặng hoặc I/O.

- **Đa luồng (Multithreading):** Nhiều luồng chạy song song giúp chương trình nhanh hơn, đặc biệt khi chờ I/O.
- **Đồng bộ hóa & Khóa (Lock):** Tránh xung đột khi nhiều luồng cùng truy cập tài nguyên chung; chỉ một luồng giữ khóa và thực thi tại một thời điểm.
- **Lập trình không đồng bộ (Async):** Cho phép nhiều tác vụ chạy song song với luồng chính, tăng hiệu năng.
- **asyncio:** Dùng **vòng lặp sự kiện** để lập lịch các tác vụ không đồng bộ; hàm khai báo bằng **async** (coroutine) và tạm dừng bằng **await** (ví dụ await asyncio.sleep()).

Giới Thiệu

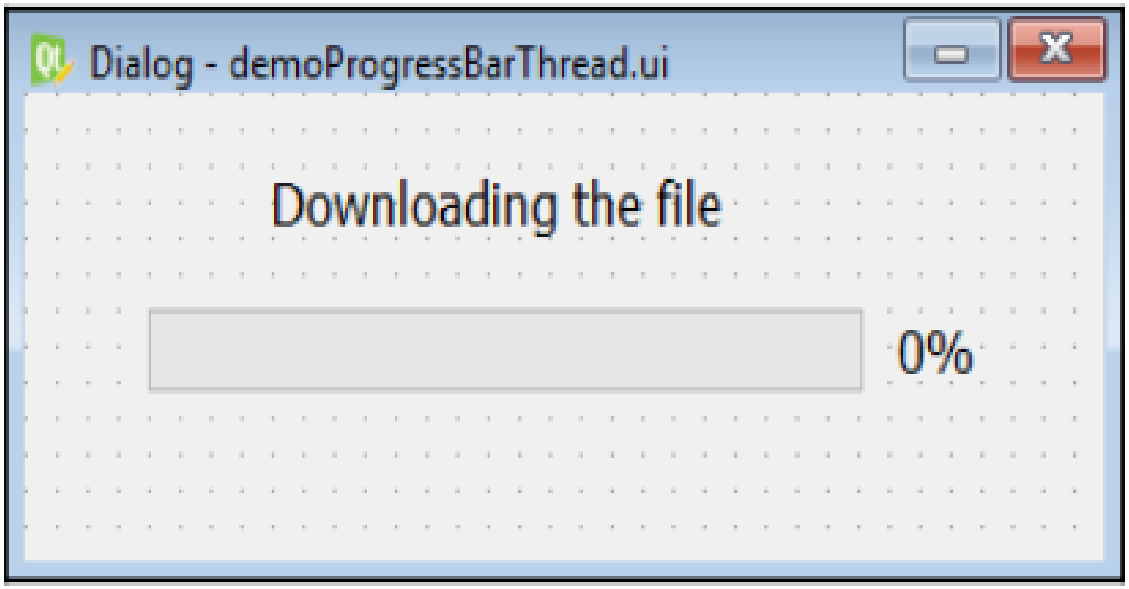
- Sau đây là các phương thức được yêu cầu trong lập trình không đồng bộ:
  - **asyncio.get_event_loop()**: Phương thức này được sử dụng để lấy sự kiện mặc định
  - **loop().** Vòng lặp sự kiện lên lịch và chạy các tác vụ không đồng bộ.
  - **loop.run_until_complete():** Phương thức này sẽ không trả về cho đến khi tất cả các tác vụ không đồng bộ được thực hiện.

Cập Nhật Thanh Tiến Trình Bằng Cách Sử Dụng Luồng

- **Thanh tiến trình (Progress Bar)** dùng để cho người dùng biết tác vụ nền đang chạy.
- **Luồng (Thread)** giúp thực hiện tác vụ nền mà không làm treo giao diện.
- Trong PyQt, **giá trị thanh tiến trình được cập nhật từ luồng đang chạy**.
- Kết hợp **Progress Bar + Thread** giúp giao diện mượt và phản hồi tốt khi xử lý tác vụ dài.

Cập Nhật Thanh Tiến Trình Bằng Cách Dùng Luồng

- Tạo ứng dụng **PyQt** theo mẫu *Dialog without Buttons*.
- Dùng **Qt Designer** thêm QLabel và QProgressBar bằng kéo–thả.
- Đặt QLabel.text = "Downloading the file" và giữ QProgressBar.objectName mặc định.
- Lưu giao diện thành **demoProTHERBarThread.ui** (file .ui là XML).
- Chuyển file .ui sang **code Python**.
- Xem **de** **nhu** file giao diện (header) và **impo**
- Tạo file n.



```
import sys
import threading
import time
from PyQt5.QtWidgets import QDialog, QApplication
from demoProgressBarThread import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.show()
class myThread (threading.Thread):
        counter=0
    def __init__(self, w):
        threading.Thread.__init__(self)
```

Cập Nhật Thanh Tiến Trình Bằng Cách Sử Dụng Luồng

```
            self.w=w
            self.counter=0
        def run(self):
            print ("Starting " + self.name)
            while self.counter <=100:
                time.sleep(1)
                w.ui.progressBar.setValue(self.counter)
                self.counter+=10
                print ("Exiting " + self.name)
    if __name__=="__main__":
        app = QApplication(sys.argv)
        w = MyForm()
        thread1 = myThread(w)
        thread1.start()
        w.exec()
        sys.exit(app.exec_())
```

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

- Để đa nhiệm, bạn cần nhiều hơn một luồng chạy cùng lúc. Trọng tâm của bài này là tìm hiểu làm thế nào hai tác vụ có thể được thực hiện không đồng bộ thông qua hai luồng, nghĩa là thời gian CPU được phân bổ cho hai luồng này và cách chuyển đổi được thực hiện giữa chúng.
- Bài này sẽ giúp bạn hiểu làm thế nào hai luồng chạy độc lập mà không can thiệp lẫn nhau. Chúng ta sẽ sử dụng hai thanh tiến trình trong bài này. Một thanh tiến trình sẽ biểu thị tiến trình tải xuống file và thanh tiến trình khác sẽ biểu thị tiến trình quét vi-rút trên ổ đĩa hiện tại. Cả hai thanh tiến trình sẽ tiến triển độc lập với nhau thông qua hai luồng khác nhau.

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

- Chúng ta hãy tìm hiểu làm thế nào hai thanh tiến trình được quản lý bởi hai luồng. Để hiểu cách thời gian CPU được phân bổ cho mỗi luồng đang chạy để thực hiện đồng thời hai tác vụ, hãy thực hiện các bước sau:

1. Hãy tạo một ứng dụng dựa trên mẫu Dialog without Buttons. Chúng ta cần hai cặp widget **QLabel** và **QProgressBar** trong ứng dụng này.
2. Thêm một **QLabel** và tiện ích **QProgressBar** vào biểu mẫu bằng cách kéo và thả tiện ích **Label** vào form và bên dưới tiện ích **Label**, kéo và thả thanh tiến trình trên form.

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

3. Lặp lại quy trình cho một cặp tiện ích Label và Progress Bar khác.

4. Đặt thuộc tính text của tiện ích Label đầu tiên thành *Downloading the file*.

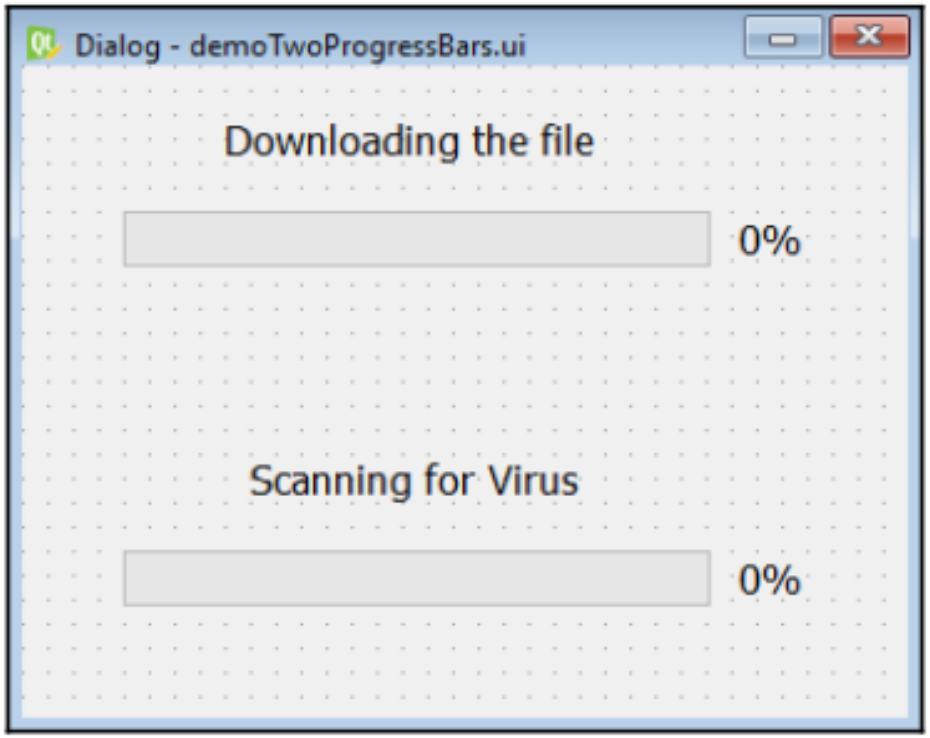
5. Đặt thuộc tính text của tiện ích Label thứ hai thành Scanning for Virus.

6. Đặt thuộc tính objectName của thanh tiến trình đầu tiên thành progressBarFileDownload.

7. Đặt thuộc tính objectName của thanh tiến trình thứ hai thành progressBarVirusScan.

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

8. Lưu ứng dụng dưới dạng demoTwoProTHERBars.ui. Sau khi thực hiện các bước trước, biểu mẫu sẽ xuất hiện như được hiển thị trong ảnh chụp màn hình sau:



Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

Giao diện người dùng được tạo bằng Qt Designer được lưu trữ trong file .ui là file XML. Bằng cách áp dụng tiện ích pyuic5, file XML có thể được chuyển đổi thành mã Python.

9. Hãy coi tập lệnh demoTwoProTHERBars.py như một file tiêu đề và nhập nó vào file mà bạn sẽ gọi thiết kế giao diện người dùng của nó.

10. Tạo một file Python khác với tên callProTHERBarTwoThreads.pyw và nhập code demoTwoProTHERBars.py vào đó:

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

```
import sys
import threading
import time
from PyQt5.QtWidgets import QDialog, QApplication
from demoTwoProgressBars import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.show()
class myThread (threading.Thread):
        counter=0
    def __init__(self, w, ProgressBar):
        threading.Thread.__init__(self)
        self.w=w
        self.counter=0
        self.progreassBar=ProgressBar
    def run(self):
        print ("Starting " + self.name+"n")
        while self.counter <=100:
            time.sleep(1)
            self.progreassBar.setValue(self.counter)
            self.counter+=10
            print ("Exiting " + self.name+"n")
```

Cập Nhật Hai Thanh Tiến Trình Bằng Hai Luồng

```
if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    thread1 = myThread(w, w.ui.progressBarFileDownload)
    thread2 = myThread(w, w.ui.progressBarVirusScan)
    thread1.start()
    thread2.start()
    w.exec()
    thread1.join()
    thread2.join()
    sys.exit(app.exec_())
```

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

- Phần này sẽ giúp bạn hiểu làm thế nào hai luồng có thể tránh sự mơ hồ bằng cách sử dụng các khóa. Đó là cách các tài nguyên được chia sẻ có thể được truy cập và thao tác bởi hai luồng đồng thời, mà không đưa ra kết quả mơ hồ.
- Chúng ta sẽ sử dụng hai thanh tiến trình trong phần này. Một thanh tiến trình sẽ biểu thị tiến trình tải file xuống và thanh tiến trình khác sẽ biểu thị tiến trình quét vi-rút trên ổ đĩa hiện tại. Chỉ một thanh tiến trình sẽ tiến triển tại một thời điểm.

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

Các bước sau đây sẽ giúp bạn hiểu làm thế nào hai luồng có thể chạy đồng thời, cập nhật tài nguyên có thể chia sẻ chung mà không đưa ra kết quả mơ hồ:

1. Hãy tạo một ứng dụng dựa trên mẫu **Dialog without Buttons**. Chúng ta cần hai cặp widget QLabel và QProgressBar trong ứng dụng này.
2. Thêm tiện ích QLabel và tiện ích QProgressBar vào form bằng cách kéo và thả tiện ích Label trên form và bên dưới tiện ích Label , kéo và thả tiện ích Progress Bar (Thanh tiến trình) trên form.
3. Lặp lại quy trình cho một cặp tiện ích Label và Progress Bar khác.

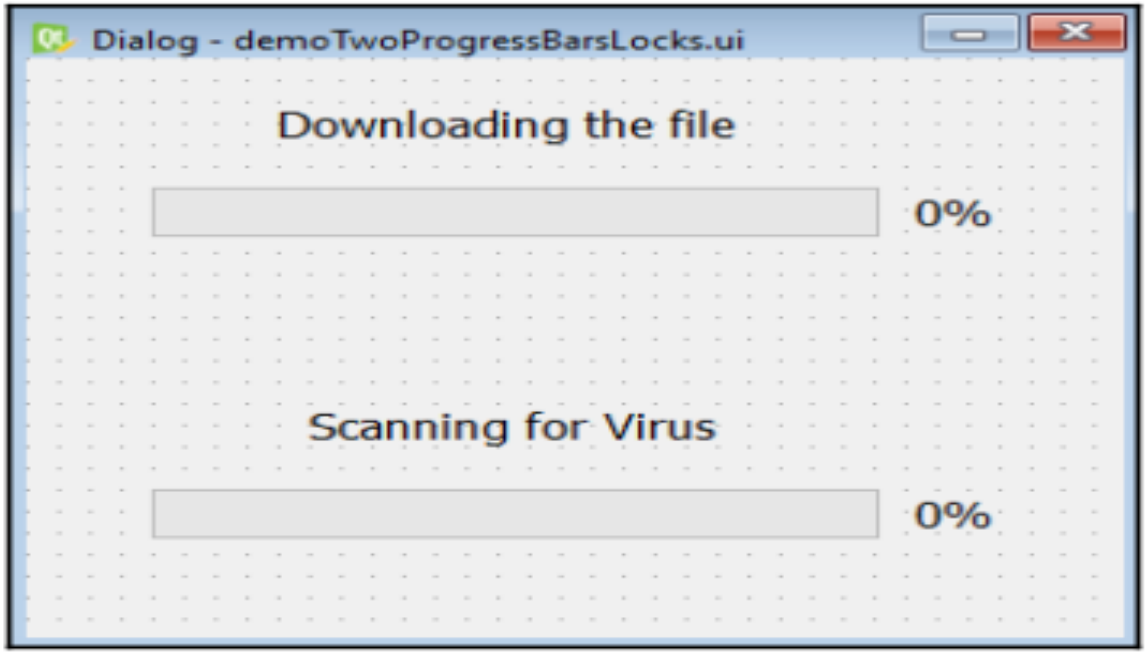
Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

4. Đặt thuộc tính text của tiện ích Label đầu tiên thành *Downloading the file* và tiện ích Label thứ hai thành *Scanning for Virus.*

5. Đặt thuộc tính objectName của tiện ích Thanh tiến trình đầu tiên thành **progressBarFileDownload**.
6. Đặt thuộc tính objectName của tiện ích Thanh tiến trình thứ hai thành **ProgressBarVirusScan**.

7. Lưu ứng dụng dưới dạng demoTwoProTHERBarsLocks.ui. Form sẽ xuất hiện như được hiển thị trong ảnh chụp sau:

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa



Giao diện người dùng được tạo bằng Qt Designer được lưu trữ trong file .ui, là file XML và cần chuyển đổi thành mã Python. Lệnh pyuic5 được sử dụng để chuyển đổi file XML thành tập lệnh Python.

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

8. Hãy coi file demoTwoProTHERBarsLocks.py là một file tiêu đề và nhập nó vào file mà bạn sẽ gọi thiết kế giao diện người dùng của nó.

9. Tạo một file Python khác có tên callProTHERBarTwoThreadsLocks.pyw và nhập mã demoTwoProTHERBarsLocks.py vào nó:

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

```
import sys
import threading
import time
from PyQt5.QtWidgets import QDialog, QApplication
from demoTwoProgressBarsLocks import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.show()
class myThread (threading.Thread):
    counter=0
```

Cập Nhật Các Thanh Tiến Trình Bằng Cách Sử Dụng Các Luồng Bị Ràng Buộc Với Một Cơ Chế Khóa

```
    def __init__(self, w, ProgressBar):
        threading.Thread.__init__(self)
        self.w=w
        self.counter=0
        self.progreassBar=ProgressBar
    def run(self):
        print ("Starting " + self.name+"n")
        threadLock.acquire()
        while self.counter <=100:
            time.sleep(1)
            self.progreassBar.setValue(self.counter)
            self.counter+=10
            threadLock.release()
            print ("Exiting " + self.name+"n")
if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    thread1 = myThread(w, w.ui.progressBarFileDownload)
    thread2 = myThread(w, w.ui.progressBarVirusScan)
    threadLock = threading.Lock()
    threads = []
    thread1.start()
    thread2.start()
    w.exec()
    threads.append(thread1)
    threads.append(thread2)
    for t in threads:
        t.join()
    sys.exit(app.exec_())
```

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

- Phần này sẽ giúp bạn hiểu cách các hoạt động không đồng bộ được thực hiện trong Python. asyncio là một thư viện trong Python hỗ trợ lập trình không đồng bộ. Phương tiện không đồng bộ, ngoài luồng chính, một hoặc nhiều tác vụ cũng sẽ thực thi song song. Trong khi sử dụng asyncio, bạn nên nhớ rằng chỉ có code được viết trong các phương thức được gắn cờ là async mới có thể gọi bất kỳ code nào theo cách không đồng bộ. Bên cạnh đó, code async chỉ có thể chạy bên trong một vòng lặp sự kiện. Vòng lặp sự kiện là mã thực hiện đa nhiệm.

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

- Điều đó cũng có nghĩa là để thực hiện lập trình không đồng bộ trong Python, chúng ta cần tạo một vòng lặp sự kiện hoặc lấy đối tượng vòng lặp sự kiện mặc định của luồng hiện tại.
- Chúng ta sẽ sử dụng hai thanh tiến trình và cả hai sẽ được cập nhật đồng thời thông qua các hoạt động không đồng bộ.

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

- Điều đó cũng có nghĩa là để thực hiện lập trình không đồng bộ trong Python, chúng ta cần tạo một vòng lặp sự kiện hoặc lấy đối tượng vòng lặp sự kiện mặc định của luồng hiện tại.
- Chúng ta sẽ sử dụng hai thanh tiến trình và cả hai sẽ được cập nhật đồng thời thông qua các hoạt động không đồng bộ.
- Thực hiện các bước sau để hiểu cách thực hiện các thao tác không đồng bộ:

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

1. Hãy tạo một ứng dụng dựa trên **mẫu Dialog without Buttons** . Chúng tôi sẽ yêu cầu hai cặp widget QLabel và QProgressBar trong ứng dụng này.
2. Thêm tiện ích QLabel và tiện ích QProgressBar vào form bằng cách kéo và thả tiện ích Label trên form và bên dưới tiện ích Label , kéo và thả tiện ích Progress Bar (Thanh tiến trình) trên form.
3. Lặp lại quy trình cho một cặp tiện ích Label và Progress Bar khác.
4. Trên cặp Label và Thanh tiến trình, kéo và thả nút ấn trên form.

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

5. Đặt thuộc tính text của nút ấn thành **Start**.

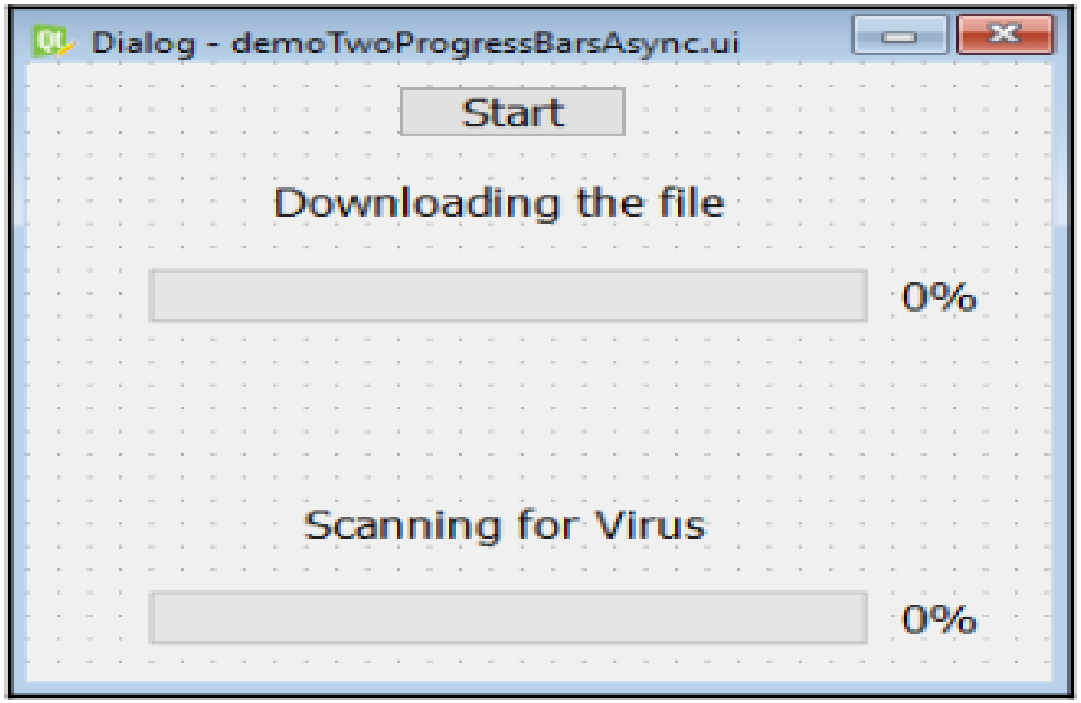
6. Đặt thuộc tính text của tiện ích Label đầu tiên thành *Downloading the file* và tiện ích Label thứ hai thành *Scanning for Virus*

7. Đặt thuộc tính objectName của nút ấn thành *pushButtonStart*.

8. Đặt thuộc tính objectName của tiện ích Thanh tiến trình đầu tiên thành *progressBarFileDownload* và thuộc tính của thanh tiến trình thứ hai thành *ProgressBarVirusScan*.

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

9. Lưu ứng dụng dưới dạng demoTwoProTHERBarsAsync.ui. Form sẽ xuất hiện như hình sau:



Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

Giao diện người dùng được tạo bằng Qt Designer được lưu trữ trong file .ui, đây là file XML và cần chuyển đổi thành code Python. Lệnh pyuic5 được sử dụng để chuyển đổi file XML thành code Python.

10. Hãy coi tập lệnh demoTwoProTHERBarsAsync.py như một file tiêu đề và nhập nó vào file mà bạn sẽ gọi thiết kế giao diện người dùng của nó.

11. Tạo một file Python khác với tên callProTHERBarAsync1.pyw và nhập code demoTwoProTHERBarsAsync.py vào nó

Cập Nhật Các Thanh Tiến Trình Đồng Thời Bằng Các Hoạt Động Không Đồng Bộ

```
import sys, time
import asyncio
from PyQt5.QtWidgets import QDialog, QApplication
from quamash import QEventLoop
from demoTwoProgressBarsAsync import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.pushButtonStart.clicked.connect(self.invokeAsync)
        self.show()
    def invokeAsync(self):
        asyncio.ensure_future(self.updt(0.5, self.ui.
        progressBarFileDownload))
        asyncio.ensure_future(self.updt(1, self.ui.
        progressBarVirusScan))
@staticmethod
async def updt(delay, ProgressBar):
    for i in range(101):
        await asyncio.sleep(delay)
        ProgressBar.setValue(i)
    def stopper(loop):
        loop.stop()
if __name__=="__main__":
    app = QApplication(sys.argv)
    loop = QEventLoop(app)
    asyncio.set_event_loop(loop)
    w = MyForm()
    w.exec()
    with loop:
        loop.run_forever()
        loop.close()
    sys.exit(app.exec_())
```

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

Trong phần này, bạn sẽ học cách cập nhật đồng thời hai thanh tiến trình bằng cách sử dụng hai luồng. Việc đồng bộ hóa giữa hai luồng và khóa chúng sẽ được xử lý thông qua trình quản lý bối cảnh. Quản lý bối cảnh là gì?

Trình quản lý bối cảnh cho phép chúng ta phân bổ và giải phóng tài nguyên bất cứ khi nào muốn. Để tối ưu hóa việc sử dụng tài nguyên, điều cần thiết là, khi bất kỳ tài nguyên nào được phân bổ bởi bất kỳ ứng dụng hoặc luồng nào, chúng sẽ được giải phóng hoặc dọn sạch để chúng có thể được sử dụng bởi một số ứng

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

hoặc luồng khác. Nhưng đôi khi chương trình gặp sự cố trong khi thực thi hoặc vì một số lý do khác khiến chương trình không kết thúc đúng cách, do đó các tài nguyên được phân bổ không được giải phóng đúng cách. Các nhà quản lý bối cảnh giúp đỡ trong các tình huống như vậy bằng cách đảm bảo việc dọn sạch các tài nguyên được phân bổ diễn ra. Đây là một ví dụ nhỏ về việc sử dụng trình quản lý bối cảnh:

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

with method_call() as variable_name:

statements that use variable_name

.....

.....

variable_name is automatically cleaned

Từ khóa *with* đóng vai trò chính trong trình quản lý bối cảnh. Sử dụng từ khóa *with*, chúng ta có thể gọi bất kỳ phương thức nào trả về trình quản lý bối cảnh. Chúng ta gán trình quản lý bối cảnh được trả về cho bất kỳ biến nào bằng cách sử dụng, dưới dạng *variable_name*.

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

variable_name sẽ chỉ tồn tại trong khối thụt lề của câu lệnh *with* và sẽ được tự động dọn sạch khi khối kết thúc. Trình quản lý bối cảnh rất hữu ích trong khi sử dụng nhiều luồng. Trong khi sử dụng nhiều luồng, bạn cần có được các khóa khi một luồng truy cập vào một tài nguyên chung. Ngoài ra, khi một tác vụ trên tài nguyên chung được thực hiện, bạn cần giải phóng khóa. Nếu các khóa không được phát hành vì một số ngoại lệ, nó có thể dẫn đến bế tắc. Trình quản lý bối cảnh sẽ tự động giải phóng khóa bằng cách sử dụng khóa với từ khóa *with*. Đây là một ví dụ nhỏ cho thấy việc mua và giải phóng các khóa:

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

threadLock.acquire()

statements that use resource

.....

.....

threadLock.release()

Bạn có thể thấy rằng một khi khóa được lấy, tài nguyên được sử dụng và cuối cùng khóa được giải phóng. Nhưng code này có thể gây ra thảm họa nếu lệnh threadLock.release () không thực thi do một số ngoại lệ trong các câu lệnh trước. Trong tình huống như vậy, tốt hơn là sử dụng trình quản lý bối cảnh.

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

Đây là cú pháp để tự động phát hành khóa bằng trình quản lý bối cảnh:

with threadLock:

statements that use resource

.....

.....

lock is released automatically

Bạn có thể thấy trong cú pháp trước đó rằng thời điểm khối kết thúc, khóa được tự động giải phóng mà không cần thực hiện phương thức release ().

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

Hãy bắt đầu với việc tạo một ứng dụng trong đó hai thanh tiến trình được cập nhật bằng hai luồng và các khóa trong luồng được xử lý bằng trình quản lý ngữ cảnh.

Hãy cùng nhau tạo một ứng dụng dựa trên mẫu Dialog without Buttons với các bước sau:

1. Chúng tôi cần hai cặp widget QLabel và QProgressBar trong ứng dụng này. Thêm tiện ích QLabel vào form bằng cách kéo và thả tiện ích Label trên form.
2. Bên dưới tiện ích Label, kéo và thả tiện ích Thanh tiến trình trên form.

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

3. Lặp lại quy trình cho một cặp tiện ích Label và Thanh tiến trình khác.

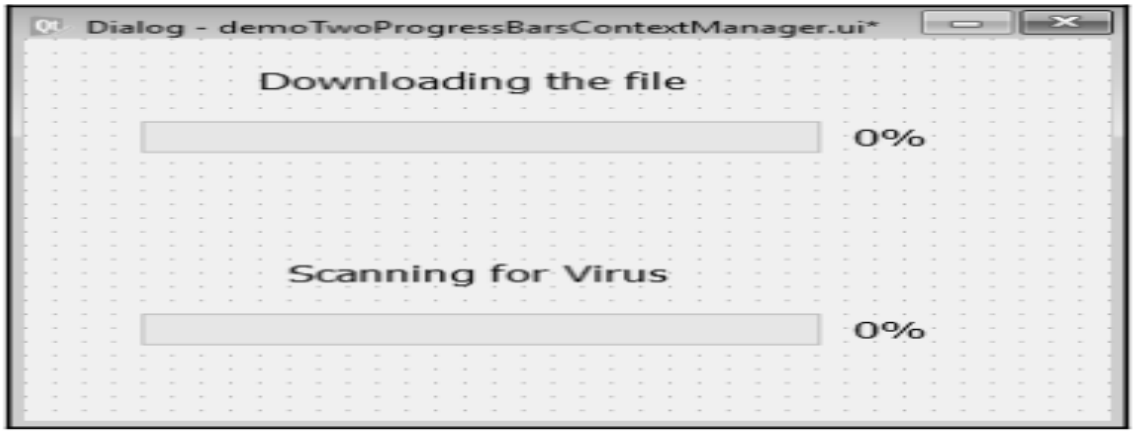
4. Đặt thuộc tính text của tiện ích Label đầu tiên thành *Downloading the file* và tiện ích Label thứ hai thành Scanning for Virus.

5. Đặt thuộc tính objectName của tiện ích Thanh tiến trình đầu tiên thành *progressBarFileDownload*.

6. Đặt thuộc tính objectName của tiện ích Thanh tiến trình thứ hai thành *ProgressBarVirusScan*.

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

7. Lưu ứng dụng dưới dạng demoTwoProTHERBarsContextManager.ui. Form sẽ xuất hiện như sau:



Giao diện người dùng được tạo bằng Qt Designer được lưu trữ trong file .ui, đây là file XML và cần chuyển đổi thành code Python. Tiện ích pyuic5 được sử dụng để chuyển đổi file XML thành code Python

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

8. Hãy coi file demoTwoProTHERBarsContextManager.py làm file tiêu đề và nhập nó vào file mà bạn sẽ gọi thiết kế giao diện người dùng của nó.

9. Tạo một file Python khác có tên callProTHERBarContextManager.pyw và nhập mã demoTwoProTHERBarsContextManager.py vào nó:

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

```
import sys
import threading
import time
from PyQt5.QtWidgets import QDialog, QApplication
from demoTwoProgressBarsContextManager import *
class MyForm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.show()
```

Quản Lý Tài Nguyên Bằng Trình Quản Lý Bối Cảnh

```
class myThread (threading.Thread):
counter=0
    def __init__(self, w, ProgressBar):
        threading.Thread.__init__(self)
        self.w=w
        self.counter=0
        self.progreassBar=ProgressBar

    def run(self):
        print ("Starting " + self.name+"\n")
        with threadLock:
            while self.counter <=100:
                time.sleep(1)
                self.progreassBar.setValue(self.counter)
                self.counter+=10
                print ("Exiting " + self.name+"\n")
if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    thread1 = myThread(w, w.ui.progressBarFileDownload)
    thread2 = myThread(w, w.ui.progressBarVirusScan)
    threadLock = threading.Lock()
    threads = []
    thread1.start()
    thread2.start()
    w.exec()
    threads.append(thread1)
    threads.append(thread2)
    for t in threads:
        t.join()
    sys.exit(app.exec_())
```