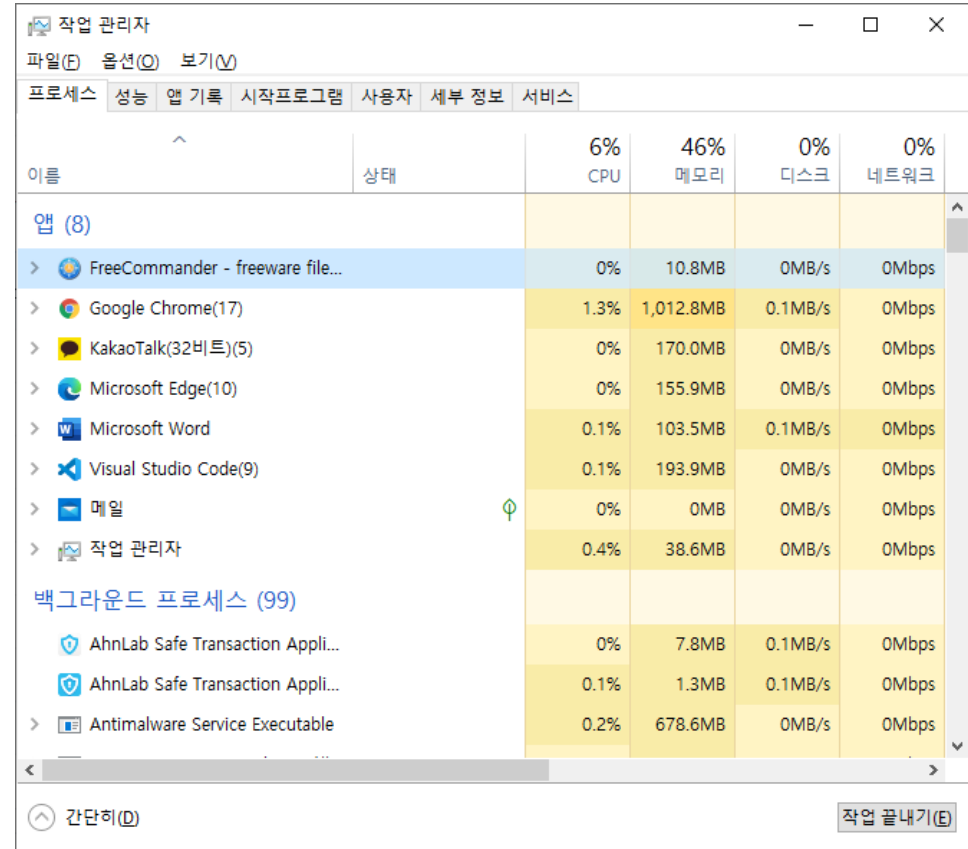


QThread

PyQT를 활용한 GUI 프로그래밍

프로세스 (Process)

- 프로세서(Processor)는 CPU를 의미함(Hardware)
- 프로그램(Program)은 '실행할 수 있는 파일'을 의미함
 - 윈도우라면 exe 파일
- 프로세스(Process)는 '실행중인 프로그램'을 의미함
 - 윈도우의 작업 관리자에서 실행되는 앱
 - 카카오톡
 - 워드
 - 크롬

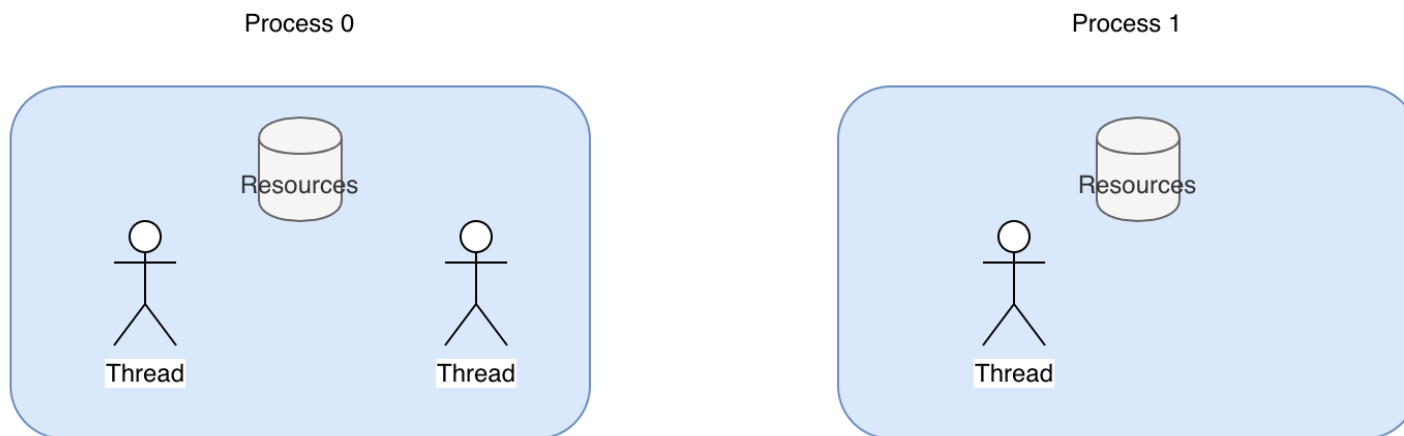


The screenshot shows the Windows Task Manager window titled '작업 관리자' (Task Manager). The '프로세스' (Processes) tab is selected, displaying a list of running applications and their resource usage. The columns are: 이름 (Name), 상태 (Status), CPU (6%), 메모리 (46%), 디스크 (0%), and 네트워크 (0%). The list is divided into '앱 (8)' (Apps) and '백그라운드 프로세스 (99)' (Background processes). The '앱' section includes FreeCommander, Google Chrome (17), KakaoTalk (32비트) (5), Microsoft Edge (10), Microsoft Word, Visual Studio Code (9), 메일 (Mail), and 작업 관리자 (Task Manager). The '백그라운드 프로세스' section includes AhnLab Safe Transaction Appli... (two instances) and Antimalware Service Executable.

이름	상태	CPU	메모리	디스크	네트워크
앱 (8)					
FreeCommander - freeware file...		0%	10.8MB	0MB/s	0Mbps
Google Chrome(17)		1.3%	1,012.8MB	0.1MB/s	0Mbps
KakaoTalk(32비트)(5)		0%	170.0MB	0MB/s	0Mbps
Microsoft Edge(10)		0%	155.9MB	0MB/s	0Mbps
Microsoft Word		0.1%	103.5MB	0.1MB/s	0Mbps
Visual Studio Code(9)		0.1%	193.9MB	0MB/s	0Mbps
메일		0%	0MB	0MB/s	0Mbps
작업 관리자		0.4%	38.6MB	0MB/s	0Mbps
백그라운드 프로세스 (99)					
AhnLab Safe Transaction Appli...		0%	7.8MB	0.1MB/s	0Mbps
AhnLab Safe Transaction Appli...		0.1%	1.3MB	0.1MB/s	0Mbps
Antimalware Service Executable		0.2%	678.6MB	0MB/s	0Mbps

프로세스와 스레드

- 프로세스가 운영체제에 의해 스케줄링되는 단위가 스레드임
 - 프로세스는 최소 하나의 스레드로 구성됨(단일 스레드)
 - 프로세스는 여러 스레드를 가질 수도 있음(멀티 스레드)
 - 프로세스는 실행될 때 운영체제로부터 독립된 자원을 할당 받음
 - **한 프로세스는 다른 프로세스의 자원(메모리)에 직접 접근할 수 없음**
 - 프로세스가 다른 프로세스의 자원을 접근하려면 프로세스간 통신을 해야함
 - 이와 달리 한 프로세스 내의 스레드는 서로 자원 공유가 가능함



PyQt와 QThread (1/2)

- PyQt에서 사용하는 스레드
 - QThread를 상속받아 기능을 추가

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
```

```
class Worker(QThread):
```

```
    def __init__(self):
```

```
        super().__init__()
```

← 부모 클래스의 생성자 호출

```
    def run(self):
```

```
        while True:
```

```
            print("worker thread")
```

```
            self.sleep(2)
```

PyQt와 QThread (2/2)

- 메인 쓰레드에서 Qthread 생성
 - QThread 객체의 start 메서드를 호출하면 run 메서드가 호출
 - 호출 된 이후에는 메인 thread와 자식 thread가 동시에 동작

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
```

```
class Worker(QThread):
    def __init__(self):
        super().__init__()

    def run(self):
        while True:
            print("Worker thread")
            self.sleep(2)
```

```
class MyWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.worker = Worker()
        self.worker.start()
```

QThread 주의사항 (1/2)

- QThread의 객체를 MainThread에서 생성할 때 변수 선언 주의하기
 - 생성자 내에서 선언된 변수는 생성자 호출이 끝나면 삭제 됨
 - 변수가 바인딩하는 객체 역시 소멸됨
 - 클래스의 메서드 내에서 self를 붙여서 변수를 선언하면 윈도우 객체가 소멸될 때 까지 유지

```
class MyWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        worker = Worker()  
        worker.start()
```

오류 코드

```
class MyWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.worker = Worker()  
        self.worker.start()
```

정상 코드

QThread 주의사항 (2/2)

- 메인 스레드 이외에서는 widget에 접근 불가
 - 예외로 인한 프로그램 강제 종료

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

class Worker(QThread):
    def __init__(self):
        super().__init__()

    def run(self):
        while True:
            self.label.setText('Hello')
            self.sleep(2)
```

QThread에서 값의 전달

- 사용자 정의 시그널을 이용
 - signal-slot 구조로 값을 전달

```
from PyQt5.QtCore import pyqtSignal
```

```
class Worker(QThread):
```

```
    mysignal = pyqtSignal(int)
```

```
    def __init__(self):  
        super().__init__()
```

```
    def run(self):  
        num = 0  
        while True:  
            self.mysignal.emit(num)  
            num += 1  
            self.sleep(1)
```

```
class MyWindow(QMainWindow):
```

```
    def __init__(self):  
        super().__init__()  
        self.worker = Worker()  
        self.worker.start()
```

```
        self.worker.mysignal.connect(self.receive_data)
```

```
    @pyqtSlot(int)  
    def receive_data(self, data):  
        print(data)
```


QThread로 값의 전달

- 생성자를 사용해서 쓰레드로 값을 전달

```
from PyQt5.QtCore import pyqtSignal
```

```
class Worker(QThread):
```

```
    mysignal = pyqtSignal(int)
```

```
    def __init__(self, name):
```

```
        super().__init__()
```

```
        self.name = name
```

```
    def run(self):
```

```
        num = 0
```

```
        while True:
```

```
            self.mysignal.emit(num)
```

```
            num += 1
```

```
            self.sleep(1)
```

```
class MyWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.worker = Worker("YJH")
```

```
        self.worker.start()
```

```
        self.worker.mysignal.connect(self.receive_data)
```

```
    @pyqtSlot(int)
```

```
    def receive_data(self, data):
```

```
        print(data)
```

연습 문제 - 1

- 다음 코드는 비트코인 현재 가격을 출력한다. 이를 이용해서 1초에 1회 가격을 조회하는 프로그램을 작성하라. 조회한 결과는 레이블에 출력한다.

```
import pyupbit
price = pyupbit.get_current_price("KRW-BTC")
print(price)
```

연습 문제 - 2

- get_current_price 함수는 입력하는 값에 따라 다른 자산의 가격을 조회한다. 다음 입력 값을 사용해서 가격을 조회하라. 각각의 자산은 1초에 한 번씩 조회한다.
 - KRW-BTC / KRW-ETH / KRW-XRP