Threading

Program vs Process

We can listen to songs and surf the Internet while playing games.

At this time, each application program is a process.

A program is simply a set of code or compiled instructions.

When you run (double click) these programs in the storage device (HDD), they are loaded into memory and become processes.

So we can say that a process is a running/executing program.

Multiprocessor

Multiprocessor means that your computer (CPU and OS) can run multiple processes concurrently or parallelly using multi-cores or multi-cpus.

Concurrently means that it looks like you run multiple process simultaneously but actually it runs one at a time.

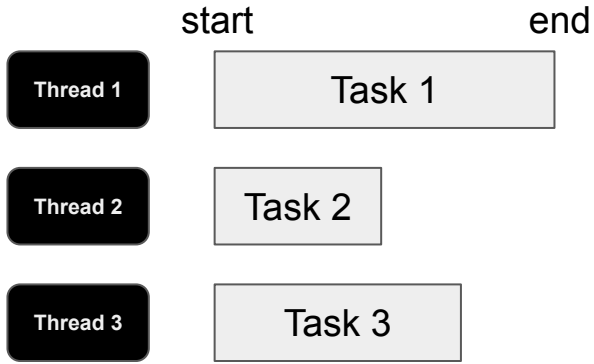
Parallely means that the multiple processes are executed at the same time.

Thread

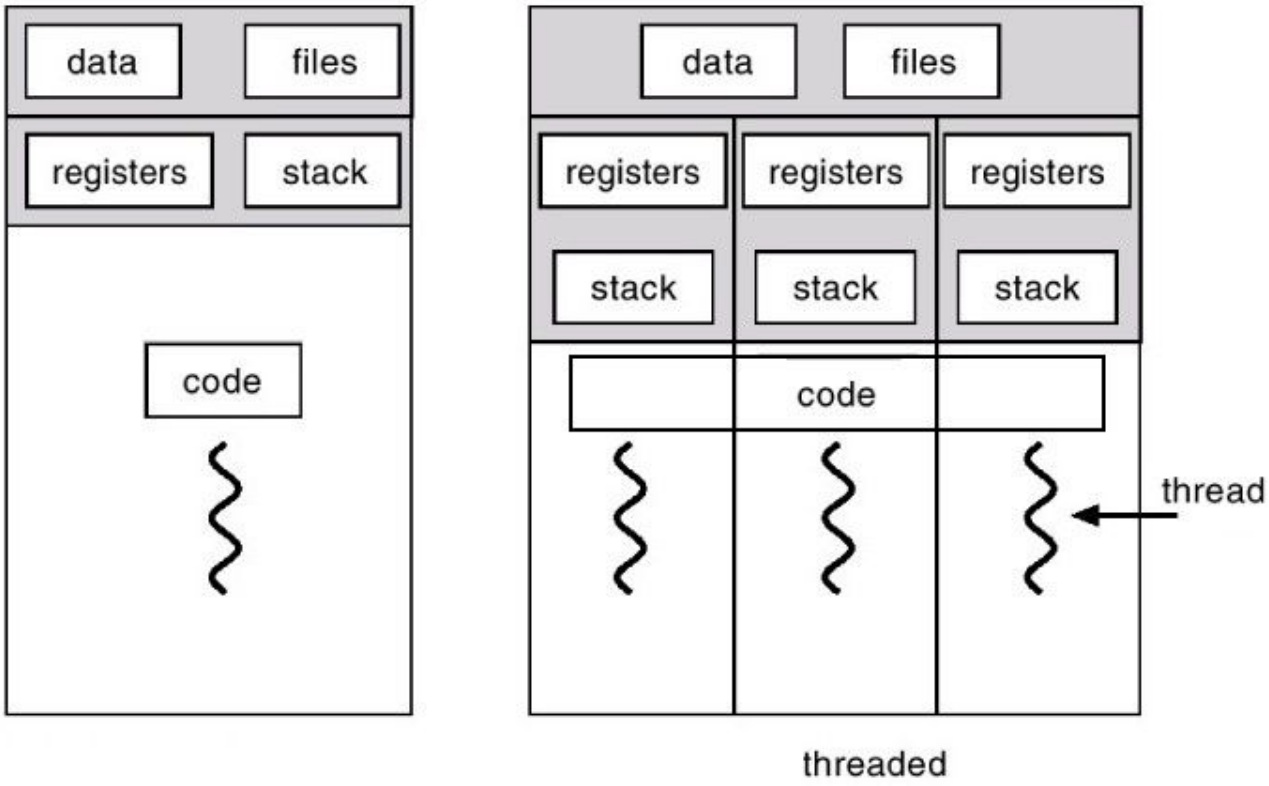
It can be said that a thread divides a process into several and executes it. In other words, different threads execute various parts of the running program at the same time. Using threads, you can do many things at the same time, rather than one program at a time.

For example, suppose you have 10 tasks that take the same amount of time. If it takes 1 minute for one operation, it will take 10 minutes in total. However, if you use 2 threads to process the work at the same time, you will be able to complete the work in about 5 minutes.

Single thread vs Multi-thread



Process vs Thread



How to use thread in python

We are going to use `threading` module.

To implement a new thread using the threading module, (1) you have to define a new subclass of the Thread class. (2) Override the __init__(self [,args]) method to add additional arguments. (3) Then, override the run(self [,args]) method to implement what the thread should do when started.

(4) Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run() method.

The methods of Thread class

The methods provided by the Thread class are as follows −

`run()` − The run() method is the entry point for a thread.

`start()` − The start() method starts a thread by calling the run method.

`join()` − The join() waits for threads to terminate.

`isAlive()` − The isAlive() method checks whether a thread is still executing.

`getName()` − The getName() method returns the name of a thread.

`setName()` − The setName() method sets the name of a thread.

```python
import threading
import time
import random


class myThread (threading.Thread):
    def __init__(self, threadID):
        threading.Thread.__init__(self)
        self.threadID = threadID

    def run(self):
        for i in range(50):
            print(self.threadID, i)
            time.sleep(random.random())


# Create new threads
thread1 = myThread(1)
thread2 = myThread(2)

# Start new Threads
thread1.start()
thread2.start()
```

```
2 38
2 39
2 40
2 41
1 44
1 45
2 42
2 43
1 46
2 44
1 47
2 45
1 48
1 49
2 46
2 47
2 48
2 49

Process finished with exit code 0
```

Example

Let's think about an ATM machine. Two persons share a bank account and they want to deposit some money ($100 or $100000) at the same time.

To test the threads, we deposit only $1 per each transaction.

What is the output of this program?

```python
import threading


class account:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, m):
        self.balance += m


class myThread (threading.Thread):
    def __init__(self, name, acc):
        threading.Thread.__init__(self)
        self.name = name
        self.acc = acc

    def run(self):
        for i in range(100000):
            self.acc.deposit(1)
```

```python
a = account(0)
# Create new threads
thread1 = myThread("Kim", a)
thread2 = myThread("Park", a)

# Start new Threads
thread1.start()
thread2.start()

threads = []
threads.append(thread1)
threads.append(thread2)
# Wait for all threads to complete
for t in threads:
    t.join()

print(a.balance)
```

Synchronizing Threads

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the Lock() method, which returns the new lock.

```
def run(self):
    for i in range(100000):
        threadLock.acquire()
        self.acc.deposit(1)
        threadLock.release()
```

The acquire(blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.

If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The release() method of the new lock object is used to release the lock when it is no longer required.

```python
import threading

class account:
    def __init__(self, balance):
        self.balance = balance
    def deposit(self, m):
        self.balance += m

class myThread (threading.Thread):
    def __init__(self, name, acc):
        threading.Thread.__init__(self)
        self.name = name
        self.acc = acc

    def run(self):
        for i in range(100000):
            threadLock.acquire()
            self.acc.deposit(1)
            threadLock.release()

a = account(0)
threadLock = threading.Lock()
thread1 = myThread("kim", a)
thread2 = myThread("park", a)

thread1.start()
thread2.start()

threads = []
threads.append(thread1)
threads.append(thread2)
for t in threads:
    t.join()

print(a.balance)
```