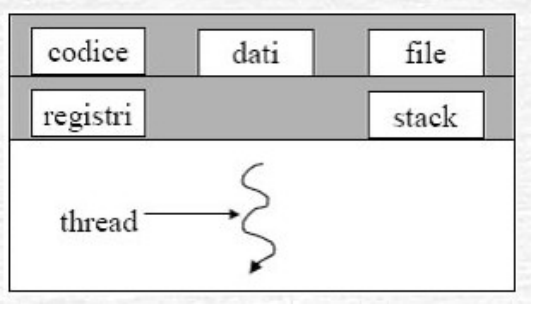
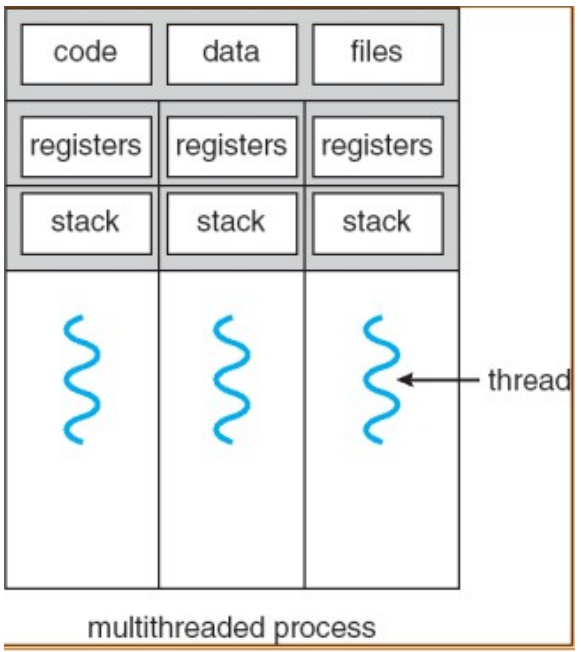
Differenza tra programma e processo

In informatica, un programma è costituito dal codice oggetto generato dalla compilazione del codice sorgente, ed è normalmente salvato sotto forma di uno o più file. Esso è un'entità statica, che rimane immutata durante l'esecuzione (entità passiva).
Il processo invece è l'entità utilizzata dal sistema operativo per rappresentare una specifica esecuzione di un programma (entità attiva). Esso è quindi un'entità dinamica, che dipende dai dati che vengono elaborati, e dalle operazioni eseguite su di essi. **Il processo è quindi caratterizzato, oltre che dal codice eseguibile, dall'insieme di tutte le informazioni che ne definiscono lo stato, come il contenuto della memoria indirizzata, i thread, i descrittori dei file e delle periferiche in uso. Infine, l'uso dell'astrazione dall'hardware è necessario al sistema operativo per realizzare la multiprogrammazione.**



Thread

Un Thread è un flusso separato di esecuzione. Nei nostri programmi che faranno uso del threading, i thread racchiuderanno una parte del nostro codice che vogliamo eseguire in modo indipendente dal resto del programma. Spesso quando parliamo di thread, pensiamo immediatamente alla possibilità di avere un programma che esegue contemporaneamente due diverse parti di codice. Ma in realtà i vari thread presenti in un programma, e stessa cosa, a livello più basso, i thread di un sistema operativo, non vengono eseguiti contemporaneamente, anche se sembra che lo facciano.



I thread creati condividono tra di loro sia la memoria che lo stato del processo. In altre parole condividono sia il codice, le istruzioni che i valori delle variabili. Quindi se un thread cambierà una variabile globale, questa cambierà per tutti gli altri thread. Comunque un thread ha anche a disposizione variabili locali che saranno accessibili esclusivamente al singolo thread e scompariranno al termine della sua esecuzione.

Il modulo threading

.

Python mette a disposizione il modulo threading per la programmazione di applicazioni Multithreaded: **import threading**

Un thread è rappresentato dalla classe Thread, il cui costruttore riceve in ingresso il nome della funzione eseguente il thread:
t = threading.Thread(target = nome_funzione)

Il metodo start() esegue la funzione in un nuovo thread
t.start()

Primo esempio

Implementeremo un singolo thread che attende un determinato periodo di tempo (per es. 2 secondi) prima di visualizzare un messaggio sulla console.
Nel corso dell'esecuzione faremo partire 2 thread contemporaneamente.

```python
import threading
import time


def funzione(num):
    print(f"Partenza  del Thread {num}")
    print("Elaboro.......\n")
    time.sleep(2)
    print(f"Finito lavoro {num}")
def main():
    t1 = threading.Thread(target=funzione,args=("Primo",))
    t2 = threading.Thread(target=funzione,args=("Secondo",))

    t1.start()
    t2.start()

    print("Fine chiamata main\n")


if __name__=="__main__":
    main()
```

È possibile passare un numero arbitrario di argomenti alla funzione da eseguire
Si usa il parametro con nome **args** della classe Thread

Identificazione

Non è strettamente necessario passare un parametro quale identificatore di un thread
Ciascun thread ha già un proprio identificatore interno, che può essere ottenuto come segue
Si accede all'oggetto rappresentante il thread corrente tramite il metodo current_thread()
Si invoca il metodo name su tale oggetto threading.current_thread().name
È possibile impostare un nome arbitrario tramite l'argomento name del costruttore
In caso contrario, il nome assegnato di default è del tipo Thread-<num>

```python
import threading
import time


def funzione():
    print(f"Partenza  del ",threading.current_thread().name)
    print("Elaboro.......\n")
    time.sleep(2)
    print(f"Finito lavoro ",threading.current_thread().name)
def main():
    t1 = threading.Thread(target=funzione)
    t2 = threading.Thread(target=funzione)

    t1.start()
    t2.start()

    print("Fine chiamata main\n")


if __name__=="__main__":
    main()
```

```python
import threading
import time


def funzione():
    print(f"Partenza  del ",threading.current_thread().name)
    print("Elaboro.......\n")
    time.sleep(2)
    print(f"Finito lavoro ",threading.current_thread().name)
def main():
    t1 = threading.Thread(target=funzione,name="Primo")
    t2 = threading.Thread(target=funzione,name="Secondo")

    t1.start()
    t2.start()

    print("Fine chiamata main\n")


if __name__=="__main__":
    main()
```

Sincronizzazione

```python
import threading
import time


def funzione():
    print(f"Partenza  del ",threading.current_thread().name)
    print("Elaboro.......\n")
    time.sleep(2)
    print(f"Finito lavoro ",threading.current_thread().name)
def main():
    t1 = threading.Thread(target=funzione,name="Primo")
    t2 = threading.Thread(target=funzione,name="Secondo")

    t1.start()
    t2.start()
    t2.join()

    print("Fine chiamata main\n")


if __name__=="__main__":
    main()
```

Invocando il metodo join() su un oggetto Thread, il programma principale aspetta che il thread associato esca

```python
import threading
import time

luc=threading.Lock()
def funzione():
    luc.acquire()
    print(f"Partenza  del ",threading.current_thread().name)
    print("Elaboro.......\n")
    time.sleep(2)
    print(f"Finito lavoro ",threading.current_thread().name)
    luc.release()
def main():
    t1 = threading.Thread(target=funzione,name="Primo")
    t2 = threading.Thread(target=funzione,name="Secondo")

    t1.start()
    t2.start()
    t2.join()

    print("Fine chiamata main\n")


if __name__=="__main__":
    main()
```

Inserisco le funzioni **acquire()** e **release()** rispettivamente all'inizio e alla fine del blocco di istruzioni.
Con il metodo acquire() faccio assumere al thread il controllo dell'esecuzione e sospendo gli altri sottoprocessi in corso.

Al termine dell'esecuzione del thread, rilascio il controllo con la funzione release().

Server Multithread

SERVER

```python
import socket
import time
import threading

def operazione(conn):
    while True:
        ricevi = conn.recv(4096).decode()
        print(ricevi+" da ",threading.current_thread().name)
        risp = input("Inserisci una risposta a")
        conn.sendall((risp).encode())
        if ricevi == "exit":
            conn.close()
            break


def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("127.0.0.1", 8000))
    s.listen()
    while True:


        conn, adress = s.accept()
        print(f"Connesso con {adress}")


        t=threading.Thread(target=operazione,args=(conn,))
        t.start()


if __name__=="__main__":
    main()
```

CLIENT

```python
import threading
import socket

def client():
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect(("127.0.0.1",8000))
    while True:
        msg = input("inserisci un messaggio")
        s.sendall((msg).encode())
        rice = s.recv(4096).decode()
        print(rice)
        if msg == "exit":
            s.close()
            break
def main():
    client()

if __name__=="__main__":
    main()
```

Class Thread_server

```python
import socket
import time
import threading
class operazione(threading.Thread):
    def __init__(self,conn):
        threading.Thread.__init__(self)
        self.conn=conn
    def run(self):
        while True:
            self.ricevi = self.conn.recv(4096).decode()
            print(self.ricevi + " da ", threading.current_thread().name)
            self.risp = input("Inserisci una risposta a")
            self.conn.sendall((self.risp).encode())
            if self.ricevi == "exit":
                self.conn.close()
                break



def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("127.0.0.1", 8000))
    s.listen()
    while True:


        conn, adress = s.accept()
        print(f"Connesso con {adress}")


        t=operazione(conn)
        t.start()

```

Esercizio

Scrivere una applicazione client/server in cui

- Il server puo' gestire concorrentemente piu' client.
- Il client
  - Legge da standard input una espressione aritmetica <operando1> <operatore> <operando2> e la invia al server
  - Il server esegue l'operazione ed invia il risultato al client
  - Il client visualizza il risultato.
- termina quando l'utente digita la stringa exit.

Client e server comunicano attraverso socket TCP