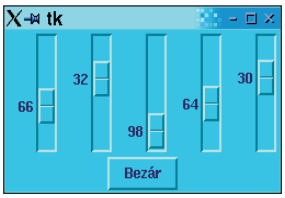
Tkinter felsőfokon – Python-szálak Tkinterből

Ebben a részben a Python szálkezelési lehetőségeivel ismerkedünk meg, egyúttal azt is megtudhatjuk, miként érhetjük el grafikus felületről ezeket a lehetőségekkel.

A Python-szálak használatával lehetőségünk nyílik arra, hogy egyetlen programon belül több végrehajtási szálat hozzunk létre. A szálakat alprogramokként képzelhetjük el, amelyek teljesen ugyanúgy néznek ki, mint a főprogram, ugyanazt a memóriaterületet használják, ugyanazokkal a fájlkezelőkkel bírnak, és még a végrehajtódó kód is teljesen egyező. Az egyetlen különbség közöttük az, hogy az egyes szálakban a végrehajtás a kód más-más részén helyezkedik el. Alapesetben a program futása egyetlen szálon hajtódik végre, vagyis a folyamatnak (process) saját memóriaterülete és egyéb erőforrásai vannak, melyeket az `os.fork()`-kal vagy `os.system()`-mel létrehozott új folyamatok nem érnek el.
Szálak létrehozására akkor van szükség, ha egy program által kizárólagosan használt erőforrást többféleképpen is el szeretnénk érni, vagy pedig ha programunknak két vagy több működő feladatot kell folyamatosan ellátnia. Ez az utóbbi főként a GUI-k használata során merül fel, amikor a felhasználói felület is folyamatos frissítést kíván, illetve a programnak egyéb, halasztást nem tűrő, olykor hosszadalmas teendői is lehetnek – ha ezek a főprogrammal egy szálon futnának, azt eredményeznék, hogy a felhasználói felület látszólag huzamosabb időszakokra teljesen lebénul.
A többszálú programok futása úgy zajlik, hogy a folyamat, vagyis a teljes program számára rendelkezésre álló idő megoszlik a program szálai között. Ez annyit tesz, hogyha például egy programban tíz szál fut, beleértve az elsődleges szálat is, akkor az egyes szálakra a teljes folyamatra összesen jutó idő 1/10 része esik.
A szálak (thread) használata során programunkat különleges gondossággal kell megtervezni, főleg az egyes szálak adatelérését és a közös adatok módosítását tekintve. Ha egy adatot a program több különböző szála nagyjából megegyező időben módosít, a programfutás kimenetele onnantól kezdve teljesen bizonytalanná válik. Ennek elkerülése érdekében a programban a kritikus részeknél kölcsönös kizárások (úgynevezett `mutex`-ek), vagy egyéb összehangoló eszközök használata szükséges, biztosítva, hogy egy adott részt egyszerre csak egy szál hajthasson végre.
A Python parancsértelmezőjében egy időben egyszerre csak egy szál futhat, melyet egy szigorúan szabályozott központi `lock`-oló eljárás biztosít. A parancsértelmező dolga, hogy bizonyos időközönként más szál kerüljön végrehajtásra. Alapértelmezésben ez a „bizonyos időközönként” 10 bájtkódnyi időt jelent. Ennyi utasítást hajt végre a Python, mielőtt a következő szálnak adná át a vezérlést. Ezen a `sys.setcheckinterval()` függvény segítségével változtathatunk.
Ha Python-programjainkból olyan külső C/C++ modulokat használunk, amelyeket nem kifejezetten többszálas végrehajtásra terveztek, akkor a külső program futása során az összes többi szál várakozni kénytelen. Szerencsére a Pythonban található modulok nagy részét felkészítették a többszálas végrehajtásra, így ezekkel nagy valószínűséggel nem lesz gondunk. Itt kell még megjegyeznünk, hogy többszálú programok futtatása esetén programunk a jelek és megszakítások fogadásakor furcsán viselkedhet. Például a billentyűzetmegszakítás kivételét bármely szál elkaphatja, ellenben a `signal`-modulban meghatározott jeleket csak a program elsődleges szála.



A Python-szálak ugyanúgy működnek minden POSIX-szálakat (pthreads) támogató operációs rendszer alól, a Windowst, a Solarist és a Linuxot is beleértve. Ellenben előfordulhat, hogy a rendszerhez adott Python-csomagot az adott rendszeren szálak támogatása nélkül fordították, ebben az esetben a Python parancsértelmezőjét nekünk kell újrafordítanunk. A Debiant használók helyzete ismét könnyű, hiszen esetükben a rendszerhez egy teljes, rendkívül zsúfolt Python jár, mely természetesen a szálak támogatását is tartalmazza.

Ismerkedés a szálak gyakorlati megvalósításával

A Pythonban a szálak kezelésére két modul is a rendelkezésünkre áll, az egyik a `thread`, a másik a `threading`. Ebben a cikkben az utóbbival fogunk foglalkozni, mely magasabb szintű szálkezelési felületet biztosít. A `threading`-modul – mely valójában maga is a `thread`-modulra épül – a `Thread` nevű osztályra épül. Ez az osztály az egész modulnak a lelke, az ebből származtatott új objektumok lesznek a programunk egyes szálai.
A `Thread` osztály a következő kezdeti értékekkel hívható:

```
Thread(group=None, target=None, name=None,
➥args=(), kwargs={})
```

A `group` értéknek a `ThreadGroup` használata során lesz majd szerepe, melynek megvalósítása még várat magára. Egyelőre az értéke kötelezően `None`. A `target` érték határozza meg a létrejövő szál által hívott első függvényt, melynek értéke alapesetben csak akkor lehet `None`, ha a `Thread` osztályt egy leszármaztatott osztályban használjuk és a `run()` eljárását felülírjuk. Ennek a `run()` eljárásnak a dolga lenne egyébként a `target` érték által meghatározott függvény meghívása.
A `name` érték értelemszerűen a szál nevét határozza meg, míg az `args` értékben egy `tuple`-ot, vagyis listát kell megadni, amely a `target` függvény értékeit határozza meg. A `kwargs` az `args`-hoz hasonló, azzal a különbséggel, hogy ebben nevesített kulcsszóparamétereket adhatunk át a `target` függvénynek.
Az *1. listában* egy egyszerű példaprogram látható, mely jól

1. *lista* Ismerkedés a szálakkal (thread1.py)

```
1. #! /usr/bin/python
2.
3. import threading
4. import time
5. import random
6.
7. def thrFunc(par):
8.     print "KezdØs: %d (i: %d)" % (par, i)
9.     time.sleep(random.choice((range(1,9))))
10.    print "    VØg: %d (i: %d)" % (par, i)
11.
12. for j in range(0, 15):
13.    t = threading.Thread(target =
       thrFunc, args = (j,))
14.    t.start()
15.
16. for i in range(0, 10):
17.    print "--- i: %d ---" % i
18.    time.sleep(1)
19.
20. print "V GE"
```

illusztrálja a szálak egyidejűségét, ugyanakkor egy közös változó használatával az is jól látható, hogy egy másik szál által megváltoztatott változó a közös adatterület miatt az összes szál változóira hatással van.
A program futása akkor ér véget, miután minden szál befejezte a tevékenységét. Ezt a viselkedést egyszerűen leellenőrzihetjük, ha a programból eltávolítjuk a „`for i`”-vel kezdődő bekezdést. Így az elsődleges szál futása azonnal végetér, de a program mindaddig nem lép ki, míg a többi szál nem végzett. Ha a 13. sor után beszúrunk a `t.start()` elé egy `t.setDaemon(1)` sort, akkor a létrejövő szál „Daemon” üzemmódban fog futni, vagyis az így létrejövő szálakra a program nem fog várakozni. Ebben az esetben programunk futása a VÉGE felirat megjelenítése után azonnal véget ér, anélkül, hogy az egyes szálak futása befejeződne. Ez azért van így, mert a program futása az elsődleges szálhoz kötött, így annak befejeződése egyben a program befejeződését is jelenti.

Összehangolás

Az egyes szálak közötti összehangolást az úgynevezett `mutex` objektumok, a `threading` modul használata esetén egyszerűen a `Lock()` függvény által visszaadott objektumok felhasználásával végezhetjük el. Összehangolásra akkor van szükség, ha egy változót egy több szál használta adatterületen módosítunk, ilyen módon elkerülhetjük programunk megzavarodását, mivel pontosan meghatározhatjuk, hogy egy változón melyik szál mikor és hogyan módosíthat.
A `Lock` objektumok használata rendkívül egyszerű. A példányosítást követően, vagy a `Lock()` függvény által visszatért objektumot felhasználva a következő metódusokat érhetjük el:

```
acquire([blocking = 1])
```

Az eljárásnak van egy nem kötelező értéke, amely alapértelmezésben 1, vagyis ha a `Lock` objektumot már lefoglalta valaki, akkor addig vár, míg fel nem szabadul. Ha ennek értéke `0`, és a `Lock` foglalt, akkor hamis értékkel azonnal visszatér.

```
release()
```

Ezzel az eljárással egy már lefoglalt `Lock`-ot szabadíthatunk fel. Programjainkban a kényes részeket e két utasítás között kell elhelyeznünk, így biztosítva, hogy egyszerre csak egy szál dolgozhat rajta.
A `Lock` objektummal nagyjából megegyező működése az `RLock` objektum, azzal a különbséggel, hogy az ezzel létrehozott objektumok többször is lefoglalhatók, vagyis az `acquire()` eljárás egy adott szálból több alkalommal is meghívható. Hogy az `RLock` felszabaduljon, a `release()` eljárást ugyanannyiszor kell meghívni, mint a lefoglalásért felelős párját.

Ugyanez Tkinterből

Grafikus felületek készítése esetén ügyelnünk kell arra, hogy a grafikus objektumok frissítéséért felelős `mainloop()` függvénynek a program elsődleges szálából kell futnia. Ha így járunk el, sok kellemetlenségtől kímélhetjük meg magunkat.
A Tkinter használatával a szálak kezelése lényegében ugyanúgy működik. A 2. listában (33. CD Magazin/Tkinter könyvtár) szereplő példánkban a külső szálak semmilyen hatással nincsenek a grafikus felületre, mivel a grafikus objektumok frissítése az elsődleges szálon belülről, segédváltozók segítségével történik.
A programból a ***Bezár*** gombra kattintva lehet kilépni. Amennyiben a programban a létrejövő szálak esetében a Daemon módot elmulasztjuk megadni (27. sor), abban az esetben a program ablakának bezárása esetén (ha nem a ***Bezár*** gombra kattintva akarunk kilépni), a program futása nem szakad meg, csak a program ablakai tűnnek el, de a program tovább fut.
A legtisztább kilépési mód, ha a ***Bezár*** gombra kattintunk, így a `self.ok` változó – mely a szálak futását vezérli – nullára vált, tehát a külső szálak sorra megszakadnak.

Összegzés

Ebben a részben a Python szálainak kezelésével kapcsolatos módszereket vettük át. Ezek ismeretében már belekezdhetünk saját több szálon futó programunk írásába is. A cikkben ismertetett módszereken kívül még egyéb lehetőségek is rendelkezésre állnak, melyek megismerésében a leírás nagy segítséget jelenthet.

A cikkhez tartozó példaprogramok megtalálhatók a lemezmellékleten, a 33. CD Magazin/Tkinter könyvtárában.

Gludovátz Gábor
(ggabor@sopron.hu) Szeret nyelveket tanulni, jól beszél angolul és németül, egy cseppet franciául, és ha már a nyelveknél tartunk, kedvencei közé tartozik még a Java, a C++ és a Python is.

Kapcsolódó címek

A Python honlapja ➲ http://www.python.org/
Tkinter-tanfolyam
➲ http://www.pythonware.com/library/tkinter/introduction/