Párhuzamos programozás témalabor

5. feladat: feladat ütemezésre szolgáló adatstruktúra

Székely Ádám

E0W8ZI

2018-12-05

Bevezető

Alábbiakban egy termelő/fogyasztó feladattal foglalkozunk. Különböző színű feladatokat generál a termelő, amit a fogyasztók hajtanak végre. Egyes színekből a beérkezési sorrendben kell végre hajtani a feladatokat. Egy időben egyszerre csak egy fajta színből, csak egyet lehet végrehajtani. Ezenkívül, ha valamilyen okból meghiúsulna az éppen feldolgozás alatti feladat, azt vissza kell helyezni a végrehajtási sorba.

Adatstruktúra

Az adatstruktúra ütemezőként viselkedik. Egyes színekhez külön sorok tárolják a feladatokat, amit egy osztály valósít meg (`ColorTasks`). Egészeket tartalmazó listában tárolja a feladatokt. A lista sorként működik, az új feladatokat a sor végére kerülnek, feladatot mindig a sor elejéről kerül ki. Ez a struktúra biztosítja, hogy sorrendhelyesen hajtódnak végre a feladatok.

Zár (`lock`) biztosítja a kölcsönös kizárást, egy időben csak egy szál tudjon hozzáférni a listához a termelő szél és a konkurens fogyasztó szálak közül. `ColorTasks.Get` és a `ColorTasks.Add` függvényekben található a zár ezzel kizárva, hogy azonos időben két szál férjen hozzá a listához.

Egyes soroknak van kettő jelzésre alkalmas szinkronizációs konstrukciója (*`EventWaitHandle`*[1]), egy „`haveTask`" (`ManualResetEvent`), ami jelzi, hogy van-e feladat az adott sorban vagy, hogy éppen elfogyott. A másik a „`colorNotWorking`" (`AutoResetEvent`), aminek az a felelősége, hogy jelezze, hogy éppen dolgozik-e valaki az adott színen vagy sem. `ColorTasks.Get` függvényben *várakozunk két fentebb említett eseményekre*[2] 50ms-ig, majd utána van a korábban említett zár. Ha várakozás alatt nem következik be a két esemény akkor a függvény hamis értékkel visszatér.

Változó számú szín lehetséges ezért, van egy osztály szintű változó (`nextColor`), ami segíti inicializálni egy új színnek a példányát. Egy sornak a színét egy privát változót tárolja (`currColor`), amit egy tulajdonságon (`Property - CurrColor`) keresztül lehet lekérdezni.

Végrehajtó szál

Az egyes szín sorokon iterálva a `ColorTasks.Get` függvényét hívja, amivel próbál feladatot kérni. Ha kap feladatot, akkor végre hajtja, majd jelzi az adott sornak, hogyha végzett. Ha netalán elesne a szál végrehajtás közben, akkor visszakerül a feladat az adott sorba. Maga a feladat kérés és a végrehajtás egy `try block`-ban van, aminek van egy `finally block`-ja is. A végrehajtó szál elmenti egy-egy változóban (`flag`), hogy sikeresen kapott egy feladatot (`gotOne`), és hogy azt sikeresen végrehajtotta (`done`). A `finally block`-ban a két `flag` állapota alapján kerül vissza a sorba a feladat, ha meghiúsult volna a szál végrehajtása.

---

[1] Két leszármazottja `ManualResetEvent és AutoResetEvent`

[2] `AutoResetEvent` esetén a WaitOne(50) függvénynél, amikor tovább fut automatikusan nem jelzettre állítja az eseményt. `ManualResetEvent`-nél viszont nem történik meg.

Termelő szál

Véletlen időközönként generál egy véletlen nehézségű feladatot. Termelő szálat megvalósító osztályban (`Logic`) lehet paraméterezni a véletlen generálást konstansok segítségével.

Logic osztály

Az osztály neve kicsit csalóka, mert az osztály felelősége az az, hogy a végrehajtó szálakat és a termelő szálat implementálja és azokat elindítja. Ezen kívül még az áteresztőképesség méréshez biztosít egy függvényt, aminek a hívás hatására kiírja indítás óta megvalósított áteresztőképesség az konzolra (*`ThthroughPut(Boolean text)`*[3]).

A feladat futtatásához ezt az osztályt kell példányosítani, majd lehetőség van inicializálni egy függvénnyel, ami egy paraméter listát vár (`Init(string[] args)`). Első paraméternek a szálak számát, másodiknak a színek számát lehet megadni. A különböző *szálak futtatása*[4] `StartConsumers és StartProducer` függvényekkel lehetséges.

Mérés

Első mérésben a termelő szál 25-750ms késleltetésben generált véletlen feladatokat. A fogyasztó szálak feladatok elvégzési ideje ~250-750ms között volt. Az áteresztőképesség mérése 30sec-ig tartott, amit ötször végeztem el, majd átlagoltam.

Áteresztőképesség (Hz)		Szín (db)			
		1	**2**	**3**	**4**
Szál (db)	**1**	1,99	1,94	2,02	1,89
	2	2,02	2,63	2,55	2,67
	3	2,03	2,64	2,49	2,64
	4	1,99	2,48	2,55	2,61
	5	2	2,48	2,45	2,6
	6	2,01	2,64	2,41	2,63
	7	2,05	2,55	2,42	2,49
	8	1,99	2,56	2,63	2,57

1. mérés

[3] A függvény paramétereként meg lehet adni, hogy kiírjon-e szöveget is az adatok mellé
[4] Szálak futtatása háttérszál ként működik

Második mérésben a termelő szál 25ms késleltetéssel generált feladattokat. A feladatok nehézsége szinte azonosak voltak ~140ms.

Áteresztőképesség (Hz)		Szín (db)			
		1	**2**	**3**	**4**
Szál (db)	**1**	7	7,03	7,07	7,07
	2	7,13	10,4	10,4	14,1
	3	7,1	14,1	20,3	16,9
	4	7,13	13,6	20,6	20,5
	5	7,13	14,2	20,6	22,8
	6	7,13	14,2	21	23,2
	7	7,13	14,1	21	23,3
	8	7,13	14,2	20,9	23,6

2. mérés

A mérések eredményén jó látszik, hogyha színek számát meghaladja a szálak száma akkor, nem fog emelkedni az áteresztőképesség, mivel egy időben egy színből csak egy feladat futhat. Ugyan ez igaz fordítva is, ha a szálak számát meghaladja a színek száma, akkor úgy szintén nem nő az áteresztőképesség. Első mérésben kicsit nagyobb a szórás, mivel a termelő szál nem kötött időközönként termel új feladatokat és a feladatok sem azonos nehézségűek.