# Feladat ütemezésre szolgáló adatstruktúra C# nyelven

Maráz Márton
Budapesti Műszaki és Gazdaságtudományi Egyetem

2018.12.03.

# Tartalom

# 1. Kivonat

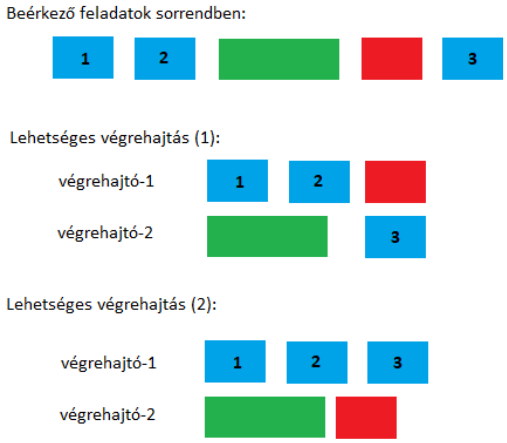
Ez a dokumentum a Budapesti Műszaki és Gazdaságtudományi Egyetem mérnökinformatikus szakán, az AUT tanszék Szoftverfejlesztés specializációjának Témalaboratórium c. tárgyán belül a „Páruzamos programozási módszerek” nevű téma 5. feladatának egy megoldását hivatott bemutatni.

# 2. A feladat leírása

A rendszerbe előre nem ismert feladatokat adnak be párhuzamosan termelők. A feladatokat pár fogyasztó dolgozza fel. A feladatok beérkezési ideje ill. végrehajtási ideje nem ismert. Az ütemező feladata a feladatok fogadása, és a fogyasztók számára mindig egy új feladat kiadása.

Az ütemezésnek meg kell felelnie az alábbi szabályoknak:

1. Egy feladatot csak egy végrehajtónak szabad kiadni.
2. Ha a feladat végrehajtása valamilyen okból meghiúsul (pl. a végrehajtó szál elesik), a feladatot vissza kell tenni a végrehajtási sorba.
3. A feladatok "színezettek": minden feladathoz tartozik egy szín (praktikusan egy szám). Az egész rendszerben egy adott színű feladatból egy időben csak egy lehet végrehajtás alatt.
4. Egy adott színhez tartozó feladatokat a beadás sorrendjében kell végrehajtani.




*Példa végrehajtás*

Az ütemező valójában nem egy algoritmus, hanem egy adatstruktúra. Az ütemezőnek legalább két művelete van: feladat beadása (ezt hívja a termelő), valamint a következő feladat elkérése (ezt hívja a fogyasztó). A rendszerben fix számú (de előre nem rögzített) végrehajtó szál van (vagyis ez a szám induláskor egy bemeneti paraméter, utána a rendszer futása alatt nem változik, de előre nem ismert a szám).

Az ütemezőnek szálbiztosnak kell lennie, mivel egyszerre történik a feladat beadása és végrehajtandó feladatok elkérése.

Javasolt megközelítés:

Az ütemező tekintetében két javasolt megoldás a (3)-as szabály betartása érdekében:

- Egy feladat kezdete esetén "letiltásra" kerül az adott szín, és a feladat befejezésekor vissza kell szólni az ütemezőnek, hogy újból kiadható ilyen színű feladat.
- A színek hozzá vannak rendelve a végrehajtó szálakhoz, így egy adott színt mindig ugyanaz a végrehajtó szál kaphat csak meg. Ekkor viszont azt kell megoldani, hogy új szín érkezése esetén is (tehát olyan feladat, ami egy új, még nem látott színnel rendelkezik) megtörténjen a feldolgozó szálhoz rendelés.

# 3. A megoldásom

(A forráskód ebben a GitHub repository-ban található meg)

## 3.1. Az ütemező működése és felépítése

A kódomban a *TaskScheduler* osztály valósítja meg az ütemező adatszerkezetet. *Task* típusú feladatokat képes fogadni. Lényegesebb funkciói: új feladat hozzáadása kívülről, egy futtatandó feladat kiadása (az ütemezés szabályait betartva)

A feladatok tényleges futtatását és az ütemezőtől való elkérésüket a *TaskConsumer* osztály példányai végzik. A *TaskConsumer* osztály tulajdonképpen egy szál, és a *TaskScheduler* osztály belső (nested) osztálya az objektumorientált egységbezárás érdekében. (Így hozzáfér a *TaskScheduler* private láthatóságú metódusaihoz is, és így a belső funkciók függvényeinek nem kell public minősítő)

Az új feladatokat az adatstruktúrába beadó termelőnek nincs egy kijelölt osztály írva, mivel ez egy kevésbé bonyolult feladat, és a kódban többféleképpen van elvégezve ez a művelet.

Az ütemezőt a (3)-as szabály betartása érdekében javasolt megközelítések közül az elsőhöz hasonló módon valósítottam meg. A tervezésnél a fő szempontom a hatékonyság volt, vagyis egy új feladat hozzáadásánál és elkérésénél fellépő ütemezési overhead minimális nagyságrendben való tartása. Ezért arra törekedtem, hogy ha van futtatható feladat, akkor az „egyből” elkérhető legyen, mindenfajta „keresgélés” nélkül. Ennek érdekében bevezettem egy queue-t, amire minden pillanatban igaz, hogy pontosan azokat a színeket tartalmazza, amik éppen tétlenek. (A kódban ez az *idleColors* adattag). A queue típusa egy saját, UniqueQueue nevű speciális implementációm, ami a hagyományos FIFO működést annyival egészíti ki, hogy nem lehet olyan elemet hozzáadni, amit aktuálisan már tartalmaz a queue, így elkerülve a színek duplikálódását. Ezen kívül kulcsfontosságú a színekhez az adott színnel rendelkező task-ok deque-ját hozzárendelő map (a kódban *colorToIdleTaskDequeMap nevű Dictionary*). Ebből tudja az ütemező, hogy ha a tétlen színek queue-jából megtudta, hogy milyen színű task-ot tud futtatni, akkor melyik task-ok közül válogathat. Szóval az előbb említett map-ból egy deque-t kinyerve, megkapja az adott színű task-okat, a beadásnak megfelelő sorrendben. Innentől a deque-n hívott dequeue művelet megadja a következőnek futtatandó task-ot. (Azért deque van a map-ben, mert ha egy task futása meghiúsul, akkor vissza kell tenni az ütemezőbe, viszont fontos hogy a legutóbbi helyére (vagyis a deque vége helyett az elejére) kerüljön vissza). Továbbá van egy a futó színek halmazát tartalmazó halmaz (*runningColors* nevű *HashSet* adattag).

A TaskScheduler osztály következő feladatot elkérő *GetNextTask* függvénye a következőket csinálja:

1. Ellenőrzi, hogy van-e tétlen szín. Ha nincs, akkor várakozik egy Monitor.Pulse-ra, majd újra próbálja.
2. Elkéri a tétlen színeket tartalmazó queue (*idleColors*) végét.
3. Kinyeri a *colorToIdleTaskDequeMap*-ből az adott színnel rendelkező task-ok deque-ját
4. Elkéri a deque végén lévő task-ot
5. Belerakja a színt a futó színek (*runningColors*) halmazába.
6. Visszaadja a task-ot a hívónak

A fogyasztó az ütemezőt annak az *OnTaskFinished(task)* függvényének meghívásával értesíti egy task befejeztéről. Ebben a függvényben visszakerül a tétlen színek közé a task színe, ha van még más ilyen színű task az ütemezőben, valamint a futó színek halmazából kikerül, majd végül egy Monitor.Pulse hívással felébreszt egy az éppen task-ra várakozó fogyasztót, ami ennek hatására megpróbál majd elkérni egy újabb task-ot.

## 3.2 …

## 3.3 Throughput benchmark

Az alábbi ábrán látható a két paraméterrel (fogyasztók és színek száma) futtatott áteresztőképesség-mérés eredménye. A színes cellákba írt számok a két aktuális paraméter mellett 5 másodperc alatt lefuttatott („áteresztett") task-ok számát jelentik. Minél nagyobb egy cellában lévő szám, annál zöldebb a háttere, illetve az annál nagyobb teljesítményt jelent az adott paraméterek mellett.

A mérést egy 6 magos processzorral rendelkező gépen futtattam. A mérés alapján kijött optimumot jelöltem vastagabb szegélyű cellákkal az ábrán.

| Színek száma \ Fogyasztók száma | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2257 | 3556 | 3281 | 3494 | 3475 | 3381 | 3290 | 3183 | 3214 | 3127 | 2797 | 2747 |
| 2 | 2203 | 4255 | 5888 | 6381 | 6783 | 6859 | 6499 | 6305 | 6003 | 5719 | 5658 | 5470 |
| 3 | 2148 | 3661 | 5817 | 7996 | 9142 | 9978 | 9373 | 9341 | 8538 | 8723 | 8322 | 8414 |
| 4 | 1862 | 3566 | 5931 | 7704 | 9372 | 13199 | 12626 | 12383 | 12623 | 12410 | 12143 | 12096 |
| 5 | 1978 | 4075 | 5693 | 7645 | 8281 | 11494 | 13357 | 13811 | 15362 | 15469 | 16148 | 16231 |
| 6 | 1959 | 3586 | 5936 | 8255 | 11342 | 12442 | 13167 | 15444 | 16222 | 16349 | 17316 | 18527 |
| 7 | 1976 | 3918 | 5895 | 7445 | 10708 | 12050 | 12479 | 12991 | 14233 | 16754 | 19220 | **19879** |
| 8 | 1915 | 3911 | 5752 | 6912 | 10205 | 11787 | 12876 | 13301 | 13737 | 14957 | 18482 | 18870 |
| 9 | 1901 | 3939 | 5870 | 7395 | 10976 | 12301 | 13162 | 13444 | 13610 | 14623 | 14964 | 18930 |
| 10 | 2013 | 3436 | 5537 | 7858 | 8923 | 11532 | 12044 | 13128 | 13123 | 13114 | 14054 | 15241 |
| 11 | 1907 | 3485 | 5087 | 6744 | 8842 | 12091 | 12904 | 13363 | 13404 | 13059 | 14256 | 15634 |
| 12 | 1835 | 3759 | 5675 | 7508 | 9104 | 10096 | 12378 | 13158 | 13017 | 13115 | 14158 | 14036 |
| 13 | 1993 | 3895 | 5562 | 7272 | 8287 | 12067 | 13061 | 13615 | 13399 | 13191 | 12740 | 15533 |
| 14 | 1957 | 3780 | 5428 | 6701 | 8970 | 11255 | 13244 | 12936 | 13228 | 12776 | 13519 | 14251 |
| 15 | 1870 | 3780 | 5717 | 7429 | 10579 | 10886 | 11841 | 13104 | 14052 | 12408 | 12545 | 14361 |
| 16 | 1777 | 3603 | 4883 | 7520 | 8694 | 11227 | 12620 | 12171 | 12951 | 13681 | 14637 | 14269 |
| 17 | 1983 | 3542 | 5828 | 7898 | 10926 | 10670 | 12126 | 13314 | 12604 | 12523 | 13593 | 14279 |
| 18 | 1972 | 3602 | 5551 | 7574 | 8717 | 11589 | 12331 | 13605 | 12568 | 13586 | 11942 | 14739 |
| 19 | 2014 | 3930 | 5587 | 7126 | 10497 | 12474 | 12067 | 13492 | 13431 | 13409 | 13938 | 14862 |
| 20 | 1954 | 3797 | 5699 | 6867 | 8874 | 12061 | 12213 | 12792 | 13491 | 13981 | 13787 | 15271 |
| 21 | 1942 | 3233 | 5529 | 7488 | 8946 | 11683 | 12362 | 13676 | 13294 | 14097 | 13335 | 14755 |
| 22 | 1873 | 3697 | 5522 | 7594 | 10287 | 11120 | 12570 | 13578 | 13346 | 13070 | 13178 | 14204 |
| 23 | 1914 | 3924 | 5493 | 6759 | 11369 | 12450 | 13100 | 13503 | 12719 | 13397 | 12909 | 14222 |
| 24 | 1921 | 3450 | 5471 | 7538 | 10355 | 12680 | 10875 | 13188 | 13321 | 13508 | 13848 | 15225 |

A gép adatai, amivel a mérést végeztem:

- CPU
  - AMD Phenom II X6 1075T
  - **6 mag** (a magok nincsenek logikai magokra szétosztva)
  - 3 Ghz
- 16GB RAM
- Windows 7 Professional

## 3.4 …