Explicaţie - Non-synchronized Threads

Petculescu Mihai-Silviu

De la programul dat, modificaţi timpii ceruţi şi explicaţi rezultatele.

Observaţie
Operaţiile pe thread-uri şi-au câştigat o reputaţie destul de controversată în rândul programatorilor datorită aparentei lor impredictibilităţi. Deoarece în execuţia unui thread intervin numeroşi factori externi, ca de exemplu disponibilitatea timpului pe procesor, unele situaţii din practică pot avea rezultate total diferite faţă de cele dintr-un mediu de testare. Un alt dezavantaj pentru unii ar fi lipsa unei posibilităţi solide pentru debuging, întrucat nu sunt expluse situaţiile în care rezultatele returnate de program să difere implicit prin utilizarea unei astfel de soluţii, baza codului nefiind atinsă.

Descriere program
Clasa `Model` cuprinde o variabilă privată `x`, initializată cu 0, o metoda `get_x()` cu ror de getter şi o metodă `increments()` care primeşte ca parametru un timp de aşteptare. In ultima metodă, se creează o variabilă locală `temp` careia îi este atribuită valoarea lui `x` prin linia `temp=x`, urmată de o incrementare cu o unitate `temp=temp+1` şi in final alterarea valorii cu `x` prin noua variabilă `x=temp`. Între toate aceste operaţii se execută funcţia `Thread.sleep(sleeping_time)` prin care tredul curent cedează timp pe procesor pentru o durată de `sleeping_time`.

Clasa `MyThread` este compusă dintr-un constructor ce primeşte ca parametru un obiect de tip `Model` si un întreg reprezentând timpul de aşteptare, ambele salvate în variabile private specifice instanţei create. Funcţia `public void run()` este o suprascriere a funcţiei specifică clasei `Thread`, una dintre puţinele de altfel şi singura care trebuie overloaded pentru ca şi clasa să poată fi executată pe mai multe procese. În interiorul funcţiei `run()` sunt plasate instrucţiunile care se doresc a fi executate la rularea unui nou thread de tipul clasei respective, în cazul nostru `md.increments(sleep_time)` inclus în blocul `try...catch` pentru capturarea eventualelor excepţii produse de clasa `Model`.

Clasa `Main` conţine funcţia `main()` în care este creată o instanţă a clasei `Model` şi două threduri de timpi diferiţi (de tip `MyThread`). Pornim "aproape" concomitent ambele threduri `t1.start(); t2.start();` şi apoi menţinem programul activ cât timp acestea continuă să se execute `while(t1.isAlive() || t2.isAlive()){}`. La final afişăm valoarea `x` specifică modelului creat `System.out.println(mins.get_x())`.

`sleeping1 = 100 ; sleeping2 = 100`

```
1
```

`sleeping1 = 200 ; sleeping2 = 100`

`sleeping1 = 100 ; sleeping2 = 200`

```
1
```

`sleeping1 = 500 ; sleeping2 = 100`

`sleeping1 = 100 ; sleeping2 = 500`

```
2
```

`sleeping1 = 1 ; sleeping2 = 11`

`sleeping1 = 11 ; sleeping2 = 1`

```
2
```

După cum se poate observa mai sus, pentru ca un thread să poată modifica cu succes variabila x, acesta trebuie să aştepte aproximativ 3x `sleeping_time`. Astfel, în cazul în care timpul de aşteptare al unui thread este de minim 3 ori cât acela al altui thread, atunci valoarea rezultată în urma executării programului va fi 2 (întrucât până când thread-ul cu timpul de aşteptare mai mare va putea copia valoarea variabilei `x` în `temp` aceasta fusese deja alterată de thread-ul cu timpul de aşteptare mai scurt).

În orice alt scenariu, se va afişa valoarea 1, întrucât niciun thread nu e suficient de rapid pentru al putea surclasa pe celălalt, deci ambele îşi vor copia în memorie valoarea standard a variabilei `x` care este 0, ambele o vor inclementa, iar la final variabila instanţei `Model` va fi rescrisă de două ori cu noua valoare, adică 1.