Ministerul Educatiei si Cercetarii Republicii Moldova

Universitatea Tehnica a Moldovei

Departamentul ISA

Raport

La lucrarea de laborator № 2 & 3

Disciplina “Programare in retea”

Tema: Programare multi-threading si clientul HTTP

A efectuat: st. gr. TI-153 Cobîlaș V.

A controlat: prof. univ. Gavrisco A.

Chisinau 2018

Tema: Programare multi-threading si clientul HTTP

Scopul lucrării: Realizarea firelor de execuţie în C#. Proprietăţile firelor. Stările unui fir de execuţie. Lansarea, suspendarea şi oprirea unui fir de execuţie. Grupuri de Thread-uri. Elemente pentru realizarea comunicării şi sincronizării. Luarea datelor prin HTTP client cu concurenta.

Aplicația trebuie să ofere următoarele funcționalități:

Scoate datele dupa un interval de date
Afisarea listei categoriilor disponibile
Analiza și validarea datelor primite
Date agregate
Afișarea rezultatelor către utilizator
Cache-ing la primirea datelor la nivel local

Există încă câteva cerințe:

Aplicația trebuie să aibă un interfață prietenoasă (Command line interface)
Când aplicația este deschisă, trebuie să se încarce datele cache dacă este disponibilă
Toate operațiile I/O nu trebuie să blocheze interfața utilizator
Solicitările trebuie efectuate simultan ori de câte ori este posibil
Agregarea datelor trebuie efectuată concurent
Aplicația trebuie să afișeze rezultatele și să le cacheze simultan

Sarcina: De creat o aplicație HTTP Client care are ar putea trimite sau primi cereri/răspunsuri prin metoda GET către o resursa online, și agregheaza datele concurent.

Considerații teoretice

Principiul de multithreading presupune execuția mai multor thread-uri în același pipeline, fiecare având propria secțiune de timp în care este menit să lucreze. Odată cu creșterea capabilităților procesoarelor au crescut și cererile de performanță, asta ducând la solicitarea la maxim a resurselor unui procesor. Necesitatea multithreading-ului a venit de la observația că unele procesoare puteau pierde timp prețios în așteptarea unui eveniment pentru o anumită sarcină.

Avantajele multithreading-ului

Ca și aplicabilitate, multithreading-ul poate fi folosit pentru sporirea eficienței atât în cadrul multiprogramării sau a sarcinilor de lucru pe mai multe fire de execuție, cât și în cadrul unui singur program. Astfel, un fir de execuție poate rula în timp ce alt fir de execuție așteaptă un anumit eveniment. În ziua de azi, capacitățile oferite de acest tip de procesare sunt folosite până și la nivelul sistemului de operare.

Pentru a putea beneficia de avantajele multithreading-ului, un program trebuie să poată fi despărțit în secțiuni ce pot rula independent și în mod paralel, fiind foarte greu de utilizat în codul reprezentat de o înșiruire foarte lungă de instrucțiuni. Gestiunea firelor de execuție este controlată de sistemul de operare.

Conceptul de multithreading poate fi folosit și pentru accelerarea unui singur thread prin utilizarea, atunci când procesorul nu este foarte solicitat, a execuției speculative pe mai multe căi și a firelor de execuție ajutătoare.

Metodele disponibile sunt

1. **GET**: este cea mai folosită metodă, fiind utilizată atunci când serverului i se cere o resursă.
2. **HEAD**: se comportă exact ca metoda GET, dar serverul returnează doar antetul resursei, ceea ce permite clientului să inspecteze antetul resursei, fără a fi nevoit să obțină și corpul resursei.
3. **PUT**: metoda este folosită pentru a depune documente pe server, fiind inversul metodei GET.
4. **POST**: a fost proiectată pentru a trimite date de intrare către server.
5. **DELETE**: este opusul metodei PUT.
6. **TRACE**: este o metodă folosită de obicei pentru diagnosticare, putând da mai multe informații despre traseul urmat de legătura HTTP, fiecare server proxy adăugându-și semnătura în antetul Via.
7. **OPTIONS**: este folosită pentru identificarea capacităților serverului Web, înainte de a face o cerere.
8. **CONNECT**: este o metodă folosită în general de serverele intermediare

Desfășurarea lucrării:

Am efectuat urmatorii pasi:

1. Creare entry point care va primi parametrii

```
static void Main(string[] args)
{
    args = new string[2];
    args[0] = "20/1/2017";
    args[1] = "20/4/2018";
    if (args.Length == 0)
    {
        tt(@"
e: dd.exe from to
ple: dd.exe 20/1/2017 20/4/2018

    }
    wr("start");
    DateTime from = DateTime.Now, to = from;
    try
    {
        from = DateTime.ParseExact(args[0], "d/M/yyyy",
                                   CultureInfo.InvariantCulture);
        to = DateTime.ParseExact(args[1], "d/M/yyyy",
                                 CultureInfo.InvariantCulture);
        if (from > to)
        {
            throw new ArgumentException("from should not be greater than to");
        }
    }catch(Exception c)
    {
        tt(c.ToString());
    }
    wr("after validation");
    var category = ConfigurationManager.AppSettings["CategoryUrl"];
    var values = ConfigurationManager.AppSettings["ValuesUrl"];
    var key = ConfigurationManager.AppSettings["Key"];
    var cat = new ApiConnector(category, values, key, from, to);
    wr("starting getting values");
    var worker = new Worker(cat);
    worker.GetData();
```

Fig. 1 – Entry point care primeste valorile intervalului pentru date

2. Crearea APIConnector care va face apleurile catre webserviciu

```
public ApiConnector(string category, string values, string key, DateTime from, DateTime to)
{
    this.category = category;
    this.values = values;
    this.from = from;
    this.to = to;
    httpClient = new HttpClient();
    httpClient.DefaultRequestHeaders.Accept.Add(new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("text/csv"));
    httpClient.DefaultRequestHeaders.Add("X-API-Key", key);
}

public IEnumerable<Category> GetCategories()
{
    var task = httpClient.GetAsync(category).GetAwaiter().GetResult();
    string v = task.Content.ReadAsStringAsync().GetAwaiter().GetResult();
    var o = Read<Category>(v);

    return o;
}
public IEnumerable<Order> GetValues()
{
    var url = values +
        "?start=" +
        from.ToString("yyyy-M-dd") +
        "&end=" + to.ToString("yyyy-M-dd");

    var task = httpClient.GetAsync(url).GetAwaiter().GetResult();
    string v = task.Content.ReadAsStringAsync().GetAwaiter().GetResult();
    var o = Read<Order>(v);

    return o;
}
```

Fig. 2 – APIConnector care face conexiunea cu webserviciu

3. Parsing CSV

```
53      public IEnumerable<T> Read<T>(string v) where T : class
54      {
55          var csv = new CsvReader(new StringReader(v));
56          var ct = new List<T>();
57          csv.Read();
58          csv.ReadHeader();
59          while (csv.Read())
60          {
61              var record = csv.GetRecord<T>();
62              ct.Add(record);
63          }
64          return ct;
65      }
66  }
```

Fig. 3 – Parsarea CSV in obiecte POCO

4. Luarea datelor, implicind concurenta

```
67
68      internal void GetData()
69      {
70          wr("started GetData");
71          Task.Run(() =>
72          {
73              wr("start GetCategories");
74              _cats = cat.GetCategories();
75              wr("end GetCategories");
76              CheckForCompletness();
77          });
78
79          Task.Run(() =>
80          {
81              wr("start GetValues");
82              _values = cat.GetValues();
83              wr("end GetValues");
84              CheckForCompletness();
85          });
86          wr("end GetData");
87
88      }
89
90      private void CheckForCompletness()
91      {
92          lock (sync)
93          {
94              wr("start CheckForCompletness");
95              if (_values.Any() && _cats.Any())
96              {
97                  DataReceived?.Invoke(this, null);
98              }
99              wr("end CheckForCompletness");
```

Fig. 4 – Taskurile pentru efectuarea call-urilor concurent

5. Agregarea datelor concurent

```
127
128     var me = new oo { ChildCategories = ccx.ToList(), Cat = new Category { name = "ROOT" } };
129     var stack = new Stack<Order>(_values);
130     while (stack.Count != 0)
131     {
132         var single = stack.Pop();
133         var cat = me.FindNode(single.category_id.Value);
134         cat?.Orders.Add(single);
135     }
136
137     wr(new string('-', 20));
138     wr(Environment.NewLine);
139
140     var xk = me.ChildCategories.Select(k => k).ToList();
141     xk.ForEach(c =>
142     {
143         waiter.TryAddCount();
144         Task.Run(() =>
145         {
146             wr("started GetData tid: " + Thread.CurrentThread.ManagedThreadId + ": " + c.Cat.name);
147             Extension.CalculateTotalPerNode(c, Extension.OnCalculationDone);
148             wr("stopped GetData tid: " + Thread.CurrentThread.ManagedThreadId + ": " + c.Cat.name);
149
150             waiter.Signal();
151         });
152     });
153     waiter.Signal();
154     waiter.Wait();
```

Fig. 5 – Agregarea datelor prin intermediul taskurilor si CountdownEvent (.NET event)

Rezultatul:

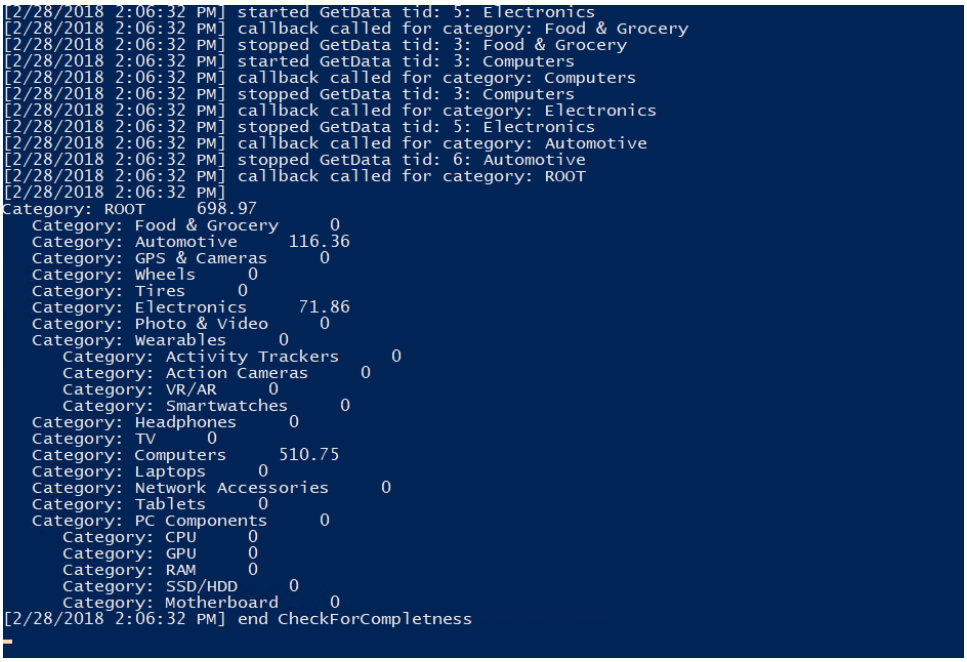

Fig. 6 – Agregarea concurent si afisrea datelor in CLI interface

Concluzie

Datorita faptului ca aceasta lucrare este concentrată asupra firelor de execuție, se poate de spus că aceasta este un mecanism foarte puternic utilizat de calculatoare, dar nu întocmai ușor de realizat. Creare mai multor fire de execuție într-un program softwere poate duce la anumite probleme, cea mai majoră fiind nedeterminismul. Pe lîngă acest fapt în zilele noastre este nevoie de cît mai multă eficiență în prelucrare resurselor, ceea ce duce la implementarea mecanismului de multithreading.

În general se poate de spus că firele sunt foarte utile în programle software dar totodată cu aceasta aduce un șir de probleme care dezvoltatori trebuie să le studize și implemnteze cu atenție.

Anexa

Tot codul sursa pentru toate laboratoarele se poate gasi aici:
https://github.com/VasileCobilas/PR