Projecto de programação

Programação Concorrente (CC3037), 2019/20

Eduardo R. B. Marques, DCC/FCUP

Conteúdo

1 Introdução

1.1 Sumário

Neste projecto terá de:

- Programar, validar a correção, e avaliar o desempenho de filas concorrentes com elementos guardados num array, com diversos tipos de aproximações - baseadas em "locks", primitivas atómicas ou STM - e modalidades de implementação - capacidade fixa ou ilimitada.
- Programar um "crawler" concorrente de páginas Web baseado no uso de uma "fork-join pool". Já é fornecido código para um "crawler" sequencial e um servidor HTTP simples de apoio.

Para a realização do projecto é disponibilizado um arquivo ZIP com código base. Veja o ficheiro `README.txt` para um sumário do material disponibilizado.

1.2 Realização, entrega e apresentação

O trabalho pode ser realizado individualmente ou em grupo por 2 alunos e terá de ser entregue até **26 de Junho**, e apresentado em altura a combinar com os alunos na semana a seguir.

No final do trabalho deverá entregar:

- um arquivo (ex. ZIP) **apenas com o conteúdo da pasta** `src` (i.e., o código);
- e um relatório em formato PDF.

Caso não tenha realizado algum dos itens descritos no enunciado a seguir, por favor mencione-o explicitamente no relatório.

1.3 Avaliação

A avaliação terá em conta a qualidade do trabalho desenvolvido em termos da implementação do código, sua correção e avaliação, bem como a exposição do trabalho no relatório e durante a apresentação.

Para notas iguais ou superiores a **17** valores deverá considerar um ou mais pontos do desafio extra de implementação de "deques" apresentado no enunciado, para além é claro de um bom trabalho nos restantes tarefas!

2 Filas concorrentes

2.1 Tipo abstracto de dados

Em `src/pc/bqueue` encontra o interface `pc.bqueue.BQueue` ("blocking queue") para uma TAD fila com as operações `add()`, `remove()` e `size()`.

- Um objecto `BQueue` é bloqueante na operação `remove()`: a operação bloqueia a thread em contexto enquanto a fila estiver vazia.
- Um objecto `BQueue` pode ser também bloqueante na operação `add()` se a fila tiver capacidade fixa e estiver cheia.

2.2 Configuração e execução de testes

- Configure as classes que são objecto de teste em `RunTests.java`.
- Os testes propriamente ditos são expressos em `BQueueTests.java`. Caso seja conveniente desabilitar momentaneamente a execução de algum testes use a anotação `@Ignore` como exemplificado no para `BQueueTests.test9()`.
- Use `ctests.sh` para para executar uma série de testes dados com o Cooperari e semântica cooperativa, e `ptests.sh` para executar testes com o Cooperari com semântica preemptiva.

Exemplos

```
$ ./ctests.sh
Invoking javac
Configuring load-time weaving ...
JAR file for 'pc.bqueue.RunTests' saved to './cdata/pc.bqueue.RunTests-cooperari.jar'
== Cooperari 0.3 - JUnit test execution - mode: cooperative ==
pc.bqueue.MBQueue$Test
  test1                                                [failed: java.lang.NullPointerException]
```

```
    > trials: 1 time: 186 ms coverage: 48.9 % (44 / 90 yp)
    > failure trace: '/Users/edrdo/Desktop/Worklog/2020/aulas/pc/project/pc_projecto/cdata/pc.bqueue
  test2_1                                                [failed: org.cooperari.errors.CWaitDeadloc
    > trials: 2 time: 78 ms coverage: 43.3 % (39 / 90 yp)
    > failure trace: '/Users/edrdo/Desktop/Worklog/2020/aulas/pc/project/pc_projecto/cdata/pc.bqueue
...
```

```
$ ./ptests.sh
Invoking javac
Execution will be preemptive, AspectJ LTW is not active.
== Cooperari 0.3 - JUnit test execution - mode: preemptive ==
pc.bqueue.MBQueue$Test
  test1                                                  [passed]
    > trials: 25 time: 357 ms
  test2_1                                                [failed: java.lang.NullPointerException]
    > trials: 2 time: 9 ms
```

2.3 Implementação

Pretende-se que implementa filas concorrentes suportadas por um array usando 3 técnicas de programação "multi-threaded" distintas:

- Monitores Java;
- STM;
- e primitivas atómicas.

Para cada uma das aproximações:

1. É dada uma implementação inicial para filas com capacidade fixa, mas com "bugs"! Deverá analisar em detalhe o que está errado pelo código, resultados de testes, e "logs" de execução gerados pelo Cooperari. Exponha a sua análise com cuidado no relatório em cada caso.
2. Posteriormente deverá acertar o código e validar os seus acertos re-executando os testes. Em cada caso, faça no relatório um sumário das alterações feitas ao código.
3. Terá depois, novamente em cada caso, de definir uma implementação de filas sem capacidade limitada, a partir do código no passo anterior (para filas com capacidade fixa). No caso da implementação baseada em primitivas atómicas, modifique também o código por forma a suportar um esquema de "back-off" exponencial. No relatório faça um sumário do novo código para cada uma das implementações.

2.3.1 Filas baseada em monitores

1. Em `MBQueue` é dada uma implementação de uma fila com capacidade fixa baseada no uso do suporte "built-in" em Java para monitores/locks. A implementação contém bugs em `add()` e `remove()` que:

- usam 2 blocos `synchronized` quando deveria ser usado apenas um;
- empregam `notify()` em vez de `notifyAll()`.

2. Faça a análise do problema no relatório com a ajuda dos "logs gerados" pelo Cooperari para os testes que falham. Comece por resolver o "bug" do uso de 2 blocos `synchronized` e depois analise o que acontece na execução dos testes antes de substituir `notify()` por `notifyAll()`. Verifique no final que (todos) os testes passam.

3. Defina em `MBQueueU.java` uma variante de `MBQueue.java` por forma a suportar filas sem limite de capacidade. Para tal, o método `add()`, quando o array actual estiver cheio e antes de colocar na fila um novo elemento, deverá criar um novo array com o dobro do tamanho do actual, em vez de bloquear a thread em contexto. Os elementos do array anterior deverão é claro passar para o novo array, e o estado interno deverá também ser actualizado de forma consistente. Para tal deverá bastar definir `MBQueueU` simplesmente como subclasse de `MBQueue`: observe que precisa de redefinir `add()`, mas não `remove()` ou `size()`.

2.3.2 Filas baseada em STM

1. Em `STMQueue` é dada uma implementação de filas de capacidade fixa baseada no uso de STM. O código tem "bugs" relativamente óbvios. Explique no relatório quais são e o seu possível efeito na execução de testes em `BQueueTest`. Pode executar os testes em modo preemptivo (usando `cjunitp.sh`), devendo com boa probabilidade levar à observação de testes falhados na execução, mas não cooperativo (usando `cjunit.sh`); o Cooperari **não tem suporte** para executar de forma cooperativa código STM (que poderá entrar facilmente numa situação de deadlock).
2. Acerte o código e verifique que os testes passam.
3. Em `STMQueueU.java` defina uma variante `STMQueueU` de `STMQueue` por forma a suportar filas sem capacidade fixa. A estratégia deverá ser similar à empregue em `MBQueueU` em termos de redimensionamento do array. Precisa no entanto de considerar a definição do campo `array` como tendo o tipo `Ref.View<TArray.View<E>>`, como ilustrado já no esqueleto de código dado.

2.3.3 Fila baseada em primitivas atómicas

1. Em `LFQueue` é dada uma implementação de uma fila com capacidade fixa baseada no uso de primitivas atómicas. Novamente a implementação tem "bugs". Executando os testes dados em modo cooperativo e analisando os "logs" do Cooperari nos casos de falha, poderá perceber que:
   - Não há problema se várias threads estiverem a executar apenas `add()`, apenas `remove()` ou ainda apenas `size()` de forma concorrente, como ilustrado pela execução de `BQueueTest.test1()`.
   - No entanto a fila poderá não funcionar correctamente quando duas ou mais threads executam pelo menos 2 destes 3 métodos de forma concorrente.
2. No relatório faça uma análise dos "bugs" antes de proceder ao acerto do código. Para este último propósito deverá recorrer à classe auxiliar `Rooms`. Esta classe provém a abstração de um conjunto de "quartos" de tal forma que, como exemplificado no programa `RoomsDemo`:
   - Inicialmente nenhum "quarto" está ocupado, até que uma thread entre num quarto `r` via `enter(r)`.
   - Estando o "quarto" `r` ocupado, não há limite para o número de threads que podem entretanto entrar também em `r`.
   - Threads a tentar entrar em "quartos" `r' != r` irão bloquear enquanto `r` estiver ocupado. Estas outras threads poderão ter a chance de aceder apenas quando todas as threads no quarto ocupado `r` saírem via `leave(r)`.

   A ideia será usar um objecto `Rooms` com **3 "quartos"**, em correspondência às operações `add()`, `remove` e `size()`.
3. Em `LFBQueueU.java` defina uma variante `LBFQueueU` de `LFBQueue` por forma a suportar filas sem capacidade fixa. Deverá fazer uso de "quartos" para operações distintas. Para lidar

com o redimensionamento do array sugere-se a estratégia simples de uso de um objecto `AtomicBoolean` que funcione como "flag" de exclusão mútua no acesso ao array e possível redimensionamento do mesmo.

4. Modifique a classes `LFQueue` / `LFQueueU` por forma a que o parâmetro de construção `back-off` habilite um esquema de "back-off" exponencial provido pela classe utilitária `Backoff` similar à que consideramos nas aulas. Deverá modificar o código por forma a que pontos de espera activa sejam mitigados pelo uso de "back-off". Note que a classe `Rooms` tem também um parâmetro de `backoff` e já a lógica de suporte associada.

2.4 Análise de execução linearizável

Faça uma análise da história de operações implícitas em `BQueueTest.test9`.

Apresente no relatório:

- as precedências das operações sobre a fila e registos `a`, `b`, `c`, e `d`;
- as possíveis linearizações, e sua análise em termos da evolução do estado da fila e dos registos.

Complete o código de `test9` em função da análise feita (veja um esqueleto possível em comentários). Inversamente o estado da execução observado durante os testes também podem ajudar a guiar a análise!

2.5 Avaliação de desempenho

Use o programa `BQueueBenchmark` para comparar as várias implementações de filas **sem capacidade fixa** que desenvolveu. Para `LFBQueueU` considere a avaliação das duas variações do parâmetro de "back-off" (habilitado ou não). Para resultados fiáveis, execute o "benchmark" em condições de baixa carga computacional, por exemplo sem janelas gráficas abertas além do terminal da linha de comandos.

Atendendo a possíveis variações entre execuções, repita as execuções 5 vezes e detalhe no relatório os valores observados bem como o valor médio para cada implementação e variante na forma de uma tabela, possivelmente complementada por um gráfico se quiser. Indique também características básicas do ambiente em que foram executados os testes: sistema operativo, número e tipo de CPUs/"cores" e memória RAM disponível. No relatório faça também uma apreciação geral dos resultados em termos de comparação entre as implementações e escalabilidade das mesmas à medida que aumenta o número de threads.

2.6 Desafio extra

Considere a implementação de filas de duplo sentido (chamadas "double-ended queues" ou simplesmente "deques") sem capacidade fixa `LFDeque` e `STMDeque`, implementando o interface `BDeque` (ver `BDeque.java`):

- partindo das implementações base em `LFQueueU` e `STMQueueU`;
- programando testes para as mesmas, tentando maximizar o "yield" point coverage" do Cooperari no caso de `LFDeque`, e validando o correcto funcionamento das operações em concorrência ou sequência;
- e avaliando o seu desempenho, da mesma forma que a implementação de filas concorrentes, considerando "back-off" habilitado ou não no caso de `LFDeque`.

Descreva o trabalho feito no relatório.

3 Crawler

3.1 Código disponibilizado

É disponibilizado o código para um "crawler" de páginas web que opera de forma sequencial, e ainda para um servidor HTTP simples de conteúdos estáticos. Ao encontrar conteúdo HTML, o "crawler" pesquisa no ficheiro "links" do tipo (`< ahref="..." >`) para páginas ou outros ficheiros no mesmo servidor, tendo o cuidado de não descarregar ficheiros repetidos.

Note que o "crawler" cria apenas ficheiros temporários que são apagados no fim da sua execução. O intuito não é guardar os ficheiros em si mas implementar o "crawler" de forma concorrente (e correcta) e avaliar o seu desempenho.

O código encontra-se na pasta `src/pc/crawler` com duas classes: o "crawler" sequencial em `SequentialCrawler` e o servidor em `WebServer`.

3.1.1 Execução do servidor

O servidor pode ser executado usando o script `wserver.sh`:

```
./wserver.sh [path_for_files [port [threads]]]
```

onde:

- `path_for_files` é a pasta "root" para conteúdos do servidor (por omissão a pasta contendo a documentação Javadoc do Cooperari);
- `port` é a porta TCP/IP que o servidor usará (por omissão 8123);
- `threads` é o número de threads para uma "work-stealing pool" usada pelo servidor (por omissão 4);

Exemplo

```
$ ./wserver.sh
27 May 2020 11:37:21 GMT | Home: cooperari-0.3/doc/javadoc
27 May 2020 11:37:21 GMT | Port: 8123
27 May 2020 11:37:21 GMT | Threads: 4
27 May 2020 11:37:21 GMT | Starting server ...
27 May 2020 11:37:21 GMT | Server started /0:0:0:0:0:0:0:0:8123
27 May 2020 11:37:41 GMT | 1 | Request for '/'
27 May 2020 11:37:41 GMT | 1 | Listing directory
27 May 2020 11:37:41 GMT | 1 | Sending / (text/html, 964 bytes)
27 May 2020 11:37:41 GMT | 2 | Request for '/constant-values.html'
27 May 2020 11:37:41 GMT | 2 | Sending /constant-values.html (text/html, 12024 bytes)
27 May 2020 11:37:41 GMT | 3 | Request for '/overview-tree.html'
27 May 2020 11:37:41 GMT | 3 | Sending /overview-tree.html (text/html, 37016 bytes)
27 May 2020 11:37:41 GMT | 4 | Request for '/index.html'
27 May 2020 11:37:41 GMT | 4 | Sending /index.html (text/html, 2971 bytes)
27 May 2020 11:37:41 GMT | 5 | Request for '/overview-frame.html'
...
```

3.1.2 Execução do crawler sequencial

O servidor pode ser executado usando o script `scrawl.sh`:

```
$ ./scrawl.sh [url]
```

onde `url` deve ter a forma `http://127.0.0.1:<porta do servidor>/path` e é por omissão definido como `http://127.0.0.1:8123`.

Exemplo

```
$ ./scrawl.sh
27 May 2020 11:41:07 GMT | Starting at http://127.0.0.1:8123/
27 May 2020 11:41:07 GMT | 1 | http://127.0.0.1:8123/ | 200 | 964 bytes | text/html
27 May 2020 11:41:07 GMT | 1 | http://127.0.0.1:8123/ | 964 bytes received
27 May 2020 11:41:07 GMT | 2 | http://127.0.0.1:8123/constant-values.html | 200 | 12024 bytes | text
27 May 2020 11:41:07 GMT | 2 | http://127.0.0.1:8123/constant-values.html | 12024 bytes received
27 May 2020 11:41:07 GMT | 3 | http://127.0.0.1:8123/overview-tree.html | 200 | 37016 bytes | text/h
...
27 May 2020 11:41:10 GMT | 354 | http://127.0.0.1:8123/org/cooperari/core/aspectj/class-use/CAgent.h
27 May 2020 11:41:10 GMT | 354 | http://127.0.0.1:8123/org/cooperari/core/aspectj/class-use/CAgent.h
27 May 2020 11:41:10 GMT | Done: 354 transfers in 3277 ms (108.03 transfers/s)
```

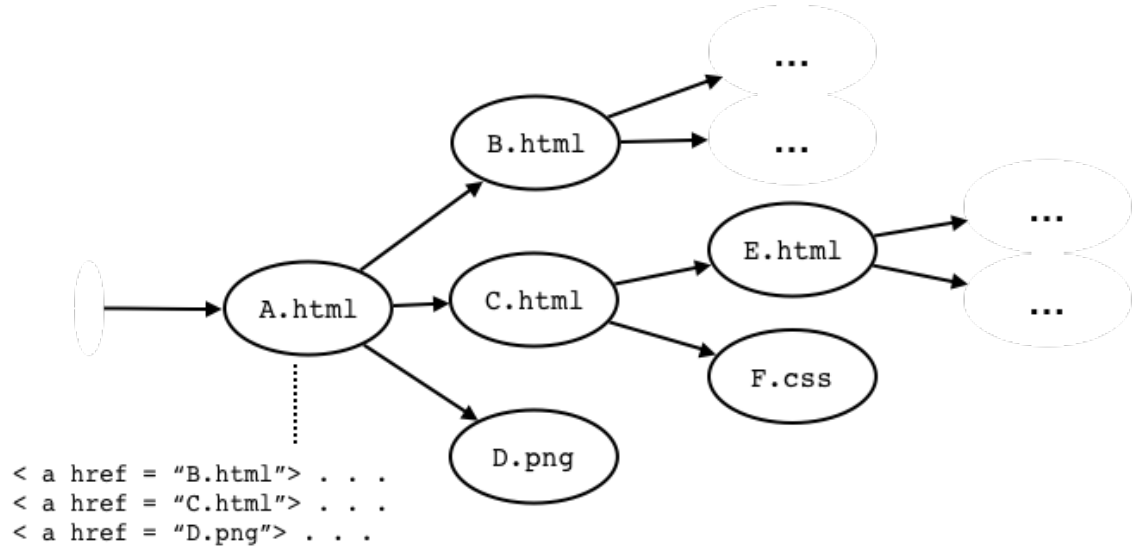
3.2 Implementação

A ideia é que programe um "crawler" concorrente baseado no uso de uma "fork-join pool" (ForkJoinPool), discutida anteriormente nas aulas, com um número de threads configuradas à partida. O "crawler" deverá ser programado por forma a que:

- a uma tarefa lançada na "pool" corresponda a uma transferência;
- uma transferência de conteúdo HTML deve levar ao lançamento de novas tarefas em função de "links" (para novos conteúdos) encontrados na página, da forma ilustrada na figura abaixo;
- a lógica inerente ao estado partilhado entre tarefas, por exemplo para determinar conteúdo já descarregado, deverá ser programada cuidadosamente para um funcionamento correcto do programa.

Pode usar e adaptar o código já dado da forma que achar mais conveniente. É sugerido um esqueleto inicial em `ConcurrentCrawler.java`.

No relatório apresente uma descrição da sua implementação, em particular da lógica de execução concorrente e manipulação de estado partilhado entre tarefas.



Operação do crawler

3.3 Teste e avaliação de desempenho

Pode testar e avaliar inicialmente o seu "crawler" sobre a documentação Javadoc do Cooperari ou sobre o directório raíz do projecto. Posteriormente será fornecido um exemplo de uma pasta com grande número de páginas HTML para avaliação.

Faça uma avaliação o desempenho do seu "crawler" concorrente, desligando a opção de "verbose output", e medindo o tempo de 5 execuções do crawler concorrente:

- variando o número de threads da "fork-join pool" do "crawler" de 1 a 16;
- lançando o servidor com igual número de threads em alternativa apenas metade delas (ex. 4 se o "crawler" tiver 8 threads);
- habilitando ou desabilitando a "flag" `WORK_STEALING_POOL` no código de `WebServer`.

Descreva no relatório os resultados de forma análoga à que fez para filas concorrentes.