Besturingssystemen: opdracht 2

Academiejaar 2014-2015

1 Doelstelling

Het correct leren aanwenden van de synchronisatiemechanismen uit de cursus om een praktisch probleem op te lossen. Deze opdracht bestaat uit twee delen, waarin respectievelijk een C++ en een Java opdracht moet geïmplementeerd worden.

2 C++ opdracht

2.1 Situatieschets

We willen, bij het implementeren van recording software, de output van een videocamera zowel tonen op het scherm als opslaan op een harde schrijf. We hebben met andere woorden een Producer die 'prentjes' produceert, terwijl meerdere Handlers deze prentjes verwerken.

Uiteraard moet het tonen en opslaan van de output snel gebeuren om de camera, die continue prentjes genereert, te kunnen volgen. Indien een Handler toch niet kan volgen, zal deze enkele frames overslaan. Hij zal met andere woorden altijd de meest recente frame gebruiken.

2.2 De opgave

Jullie opdracht is geïnspireerd op bovenstaand probleem.

Schrijf een thread-safe programma (in C++, met behulp van Pthreads), waarbij een Producer data genereert en 3 Handlers deze data verwerken. De Buffer bevat een constant aantal Data elementen. Wil een Producer data produceren, dan moet deze aan de Buffer een leeg element vragen en dat opvullen. Als er geen lege elementen beschikbaar zijn, moet de Producer wachten tot er één vrijkomt. Wil een Handler data verwerken, dan vraagt deze het **laatst** geproduceerde element aan de Buffer.

Data elementen kunnen op 2 manieren terug leeggemaakt worden. Als de Handlers de data traag verwerken, kan het voorvallen dat het volgende Data element reeds geproduceerd is, terwijl een vorig element nog niet in verwerking is. Gezien we enkel geïnteresseerd zijn in het verwerken van de meest recente data,

zullen we deze 'verouderde' elementen gewoon terug vrijgeven, zonder ooit te zijn verwerkt. Zijn de Handlers daarentegen voldoende snel, dan is het mogelijk dat een element door meerdere Handlers tegelijkertijd verwerkt wordt (de ene Handler toont het element bijvoorbeeld op het scherm, terwijl de andere het opslaat in een bestand). In een dergelijk geval mag het Data element uiteraard pas vrijgegeven worden wanneer al deze Handlers ermee klaar zijn.

Vertrek van de gegeven code en denk eraan: jouw oplossing moet thread-safe zijn en altijd werken, ook in situaties die zelden voorkomen. Bij deze opdracht is argumentatie zeer belangrijk. We vragen daarom om de belangrijkste beslissingen beknopt toe te lichten, via commentaar in de code (dus niet enkel aangeven wat je doet, maar vooral waarom je het zo doet).

3 Java opdracht

3.1 Situatieschets

We zullen een *messaging server* in de aard van twitter.com maken. Hierbij is het belangrijk dat grote hoeveelheden clients en berichten zo snel mogelijk behandeld worden. Clients kunnen berichten achterlaten op de server, alsook de berichten van een bepaalde gebruiker opvragen. Bovendien kunnen we in berichten keywords aanduiden (in de vorm van '`#keyword`'), zodat een client ook alle berichten met een bepaald keyword kan opvragen.

3.2 De opgave

We hebben een framework voorzien met de nodige datastructuren en netwerk code. Jullie opdracht is dit framework verder aan te vullen om het performant en thread-safe te maken. Het is niet de bedoeling om je te verdiepen in de netwerk code. In de bestanden die zeker niet aangepast moeten worden, hebben we dit dan ook expliciet vermeld. De overige bestanden dienen *misschien* wel aangepast te worden.

Argumentatie is opnieuw zeer belangrijk. We vragen daarom om de belangrijkste beslissingen beknopt toe te lichten, via commentaar in de code (dus niet enkel aangeven wat je doet, maar vooral waarom je het zo doet).

3.2.1 Thread pool en indexer

Het multi-threading gedeelte van de opdracht bestaat uit:

- een thread pool met een aantal threads die clients kunnen afhandelen,
- een indexer die in de berichten zal zoeken naar keywords.

In de code is bij elke klasse de nodige uitleg te vinden. **Tip**: Java voorziet in het package java.util[1] een aantal datastructuren die van pas kunnen komen.

3.2.2 Client

We hebben een client voorzien die, via meerdere threads, opdrachten naar de server kan sturen. Als argumenten krijgt dit programma: het aantal threads, een bestand waarin de opdrachten staan en het adres van de server (meestal zal dit 'localhost' zijn).

Berichten worden aangemaakt via "`say gebruikersnaam bericht`", waarbij het bericht keywords kan bevatten. Het opvragen van berichten kan per gebruiker of per keyword: "`ask gebruikersnaam`" zal alle berichten van die gebruiker teruggeven, "`key keyword`" geeft de berichten met een bepaald keyword terug. De server zal antwoorden met een regel tekst; het nummer dat vooraan staat, geeft het aantal regels die nog volgen aan.

4 Vragen

Indien je vragen hebt over deze opdracht, is het aangewezen om gebruik te maken van het discussieforum op blackboard.

5 De deadline

Je kan de opdracht indienen **ten laatste op 24 mei**, via blackboard.

[1] `http://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html`