Trabalho prático de Sistemas Operativos
Stream Processing

Grupo de Sistemas Distribuídos
Universidade do Minho

Abril de 2017

Informações gerais

- Conforme indicado no início do ano, o método de avaliação de Sistemas Operativos contempla a realização e discussão de um trabalho prático onde é necessário ter nota mínima de 10 valores.
- Cada grupo de trabalho deve ser constituído por até três elementos. Grupos de dimensão inferior a três terão de ser justificados e autorizados pelos docentes. Quem teve nota positiva no ano imediatamente anterior pode optar por manter a nota anterior (truncada a 15). Essa opção manifesta-se inscrevendo-se num grupo especial, o dos que não querem entregar TP.
- Serão disponibilizados no blackboard muitos grupos (LCC 25, MIEI 100) com capacidade para 3 elementos. Quando o grupo estiver formado, APENAS UM dos elementos escolhe um grupo que esteja vazio e inscreve-se nesse grupo. De seguida comunica aos restantes elementos o número do grupo em que se inscreveu para que os seus colegas se registem também nesse grupo. Essa inscrição irá estar disponível cerca de uma semana, encerrando as inscrições muito antes da entrega do trabalho.
- O trabalho deve ser entregue até às 23:59 do dia 31 de Maio, através do Blackboard. Deve ser enviado um único ficheiro em formato ZIP, contendo um pequeno relatório em PDF, o código fonte, a makefile para compilação de todo o código desenvolvido e eventualmente um script para configuração inicial. Será essa a versão que irão demonstrar no dia da discussão[1].
- A avaliação dos trabalhos deverá ocorrer na semana com início a 5 de Junho, 2ª-feira. Imediatamente após contagem do número de trabalhos submetidos será publicado no blackboard um conjunto de horários (em princípio 1/2 hora por grupo). Cada um destes grupos do Blackboard corresponderá a um determinado slot de 1/2 hora, sendo necessária a presença de TODOS os elementos desse grupo de alunos.
- **Notem por favor que este ano a entrega e a discussão oral terão lugar depois do teste do dia 26 de Junho**. Recomenda-se "vivamente" que o trabalho seja concluído antes do teste, como preparação para o mesmo.

[1] Para evitar percalços e atrasos no setup da demonstração do trabalho, esta poderá ser pedida a qualquer dos elementos do grupo enquanto outro vai explicando as decisões tomadas. Poderá ser realizada num computador qualquer, seja de um dos profs ou colega de grupo; toda a gente tem de saber instalar e correr o software sem comprometer o horário de avaliação.

Resumo

Neste trabalho pretende-se construir um sistema de *stream processing*. Estes sistemas utilizam uma rede de componentes para filtrar, modificar e processar um fluxo de eventos. Tipicamente, permitem tratar uma grande quantidade de dados explorando a concorrência tanto entre etapas sucessivas (*pipelining*) como entre instâncias do mesmo componente. Um exemplo de um sistema de *stream processing* para sistemas distribuídos é o Apache Storm[2].

Neste caso, pretende-se uma implementação para sistemas Unix, explorando a semelhança de *stream processing* com os filtros de texto compostos em *pipeline*. Sendo assim, assume-se que cada evento é uma linha de texto, com campos separados por dois pontos (:), sendo o seu tamanho inferior a `PIPE_BUF`.

Este trabalho tem duas partes: a implementação de um conjunto de componentes, que realizam tarefas elementares; e a implementação do sistema que compõe e controla a rede de processamento.

1 Componentes

Espera-se que cada um dos componentes receba eventos no seu *stdin* e produza eventos no seu *stdout*.

1.1 `const <valor>`

Este programa reproduz as linhas acrescentando uma nova coluna sempre com o mesmo valor:

```
const 10
```

tem o seguinte resultado, reproduzindo as linhas em que o valor da segunda coluna é maior ou igual que a quarta coluna:

Input	Output
`a:3:x:4`	`a:3:x:4:10`
`b:1:y:10`	`b:1:y:10:10`
`a:2:w:2`	`a:2:w:2:10`
`d:5:z:34`	`a:2:w:2:10`

1.2 `filter <coluna> <operador> <operando>`

Este programa reproduz as linhas que satisfazem uma condição indicada nos seus argumentos. Por exemplo, executando o seguinte comando:

```
filter 2 < 4
```

tem o seguinte resultado, reproduzindo as linhas em que o valor da segunda coluna é maior ou igual que a quarta coluna:

Input	Output
`a:3:x:4`	`a:3:x:4`
`b:1:y:10`	`b:1:y:10`
`a:2:w:2`	`d:5:z:34`
`d:5:z:34`	

Devem considerar os operadores `=`, `>=`, `<=`, `>`, `<`, `!=`.

[2]Recomenda-se novamente a leitura da documentação disponibilizada no blackboard. VerTutorial de Storm na secção de Conteúdo.

1.3 `window <coluna> <operação> <linhas>`

Este programa reproduz todas as linhas acrescentando-lhe uma nova coluna com o resultado de uma operação calculada sobre os valores da coluna indicada nas linhas anteriores. Por exemplo, executando o seguinte comando:

```
window 4 avg 2
```

tem o seguinte resultado, reproduzindo todas as linhas e acrescentando uma nova coluna com a média dos 2 últimos valores anteriores da quarta coluna:

Input	Output
`a:3:x:4`	`a:3:x:4:0`
`b:1:y:10`	`b:1:y:10:4`
`a:2:w:2`	`a:2:w:2:7`
`d:5:z:34`	`d:5:z:34:6`

Devem considerar os operadores `avg, max, min, sum`.

1.4 `spawn <cmd> <args...>`

Este programa reproduz todas as linhas, executando o comando indicado uma vez para cada uma delas, e acrescentando uma nova coluna com o respetivo *exit status*. Os argumentos que tiverem o formato `$n` devem ser substituídos pela coluna correspondente. Por exemplo, executando o seguinte comando:

```
spawn mailx -s $3 x@y.com
```

tem o seguinte resultado, reproduzindo todas as linhas e enviando mensagens com assunto `w` e com assunto `z` para `x@y.com`:

Input	Output
`a:2:w:2`	`a:2:w:2:0`
`d:5:z:34`	`d:5:z:34:0`

1.5 Outros

Além da implementação destes componentes, devem familiarizar-se com os utilitários do sistema Unix tais como `cut`, `grep`, `tee`.

2 Controlador

O controlador é um programa que permite definir uma rede de processamento de *streams*, com cada nó a executar um componente de transformação, como os acima definidos. O controlador recebe comandos, um por linha. Pode ser especificado um ficheiro de configuração, que é lido no arranque; devem também poder ser enviados comandos mais tarde, por exemplo, através de um *named pipe*. Poderá usar mais do que um processo e vários *pipes* (com ou sem nome) por cada nó.

Nota: esta é a parte interessante do trabalho. Assegure-se que percebe o que se pretende e que consegue partir o problema em subproblemas. Não tem de começar logo a fazer ”um trabalho para 20 valores”!

2.1 Comandos

Os comandos que o controlador deve aceitar são:

2.1.1 `node <id> <cmd> <args...>`

Este comando permite definir um nó da rede de processamento, com o identificador `id`, que irá executar o componente `cmd` com a lista de argumentos `[args]`. Por exemplo, `node 1 window 2 avg 10`, especifica que o nó `1` executa o componente `window` com os argumentos `2 avg 10`.

2.1.2 `connect <id> <ids...>`

Este comando permite definir ligações entre nós. Mais concretamente, especifica que a saída do componente a correr no nó `id` deverá ser enviada para cada um dos nós da lista `[ids]`. Por exemplo, `connect 2 3 5`, especifica que a saída produzida pelo nó `2` deverá ser enviada para o nó `3` e também para o nó `5`.

2.1.3 `disconnect <id1> <id2>`

Este comando também permite desfazer a ligação entre o nó `id1` e o nó `id2`.

2.1.4 `inject <id> <cmd> <args...>`

Este comando permite injetar na entrada do nó `id` a saída produzida pelo comando `cmd` executado com a lista de argumentos `args`. Como tal, permite definir uma fontes de eventos. Este comando é usado quando uma rede já está definida, permitindo exercitar a rede.

2.2 Exemplo

Considere o seguinte ficheiro de configuração que define uma rede:

```
node 1 window 2 avg 10
node 2 window 2 max 10
node 3 filter 2 <= 3
node 4 const cooling
node 5 filter 2 > 3
node 6 const warming
node 7 filter 2 > 4
node 8 tee warm-log.txt
node 9 const 100
node 10 filter 2 > 5
node 11 spawn shutdown -n
node 12 tee log.txt
connect 1 2
connect 2 3 5
connect 3 4
connect 4 12
connect 5 6
connect 6 7 12
connect 7 8 9
connect 9 10
```

```
connect 10 11
```

Esta rede assume que são injetados eventos com duas colunas contendo o tempo e uma temperatura, por exemplo:

```
1020:25
1021:25
1021:26
1022:28
1023:17
...
```

2.3 Funcionalidades básicas

2.3.1 Permanência de uma rede

Os nós de uma rede, depois de criados, continuam a existir até serem explicitamente removidos (funcionalidade avançada, abaixo). A rede deve ficar recetiva a processar *streams* de eventos que parem (e.g., devido a *end-of-file*) e recomecem mais tarde, quando um novo comando `inject` é executado.

2.3.2 Concorrência nas escritas no mesmo nó

Ao especificar uma rede, vários nós podem ser ligados a um mesmo nó, ou várias instâncias do comando `inject`, com outras ainda a decorrer. Como tal, o sistema deve garantir que cada evento (linha de texto) chega corretamente ao destino, não corrompido, havendo apenas intercalamento de eventos de diferentes origens.

2.3.3 Nós sem saída definida

Os nós cuja saída não esteja ligada a nenhum outro devem ter a sua saída descartada. (Mas devem executar normalmente, pois podem estar, por exemplo, a escrever para um ficheiro, passado como argumento.)

2.3.4 Especificação incremental da rede

O servidor vai recebendo comandos, devendo atuar perante cada um, atualizando a rede (acrescentando nós ou conexões), independentemente de comandos futuros. Tal leva a que não se saiba à partida, por exemplo, se um nó vai ter descendentes, tendo que ser possível adicionar conexões depois de um nó ter sido criado. Isto deve ser possível, mesmo usando vários comandos `connect`: cada um especifica ligações a acrescentar. Caso a rede esteja a processar eventos quando recebe um pedido de nova conexão, tal não deve levar a que se percam eventos; i.e. todos devem chegar às conexões já existentes. Deve, portanto, ser evitada a terminação e o re-arranque de processos.

2.4 Funcionalidades adicionais (avançadas)

2.4.1 Casos de teste

É valorizada a apresentação de redes de processamento que usem diferentes combinações dos componentes propostos e dos comandos Unix para resolver problemas concretos.

2.4.2 Alteração de componentes

Deve poder ser alterado o componente a correr num nó. Mais uma vez, não devem ser perdidos eventos: cada evento que chega ao nó deve ser processado ou pelo componente atual ou (a partir de determinado momento) pelo novo componente.

2.4.3 Remoção de nós

O controlador deve permitir um comando `remove <id>`, que permita remover um nó da rede. A rede resultante deve ter uma ligação entre cada nó que ligava ao removido e cada nó a que o nó removido ligava.

Notas finais

Este trabalho vale apenas 1/3 da classificação final. É relativamente simples para quem foi resolvendo os exercícios propostos nos guiões e serve essencialmente para averiguar o grau de preparação para o teste e/ou exame de recurso, onde não vai existir o auxílio externo nem dos restantes elementos do grupo. Todos os elementos do grupo terão de estar à vontade para responder às questões durante a discussão pública, justificando as opções tomadas e mostrando conhecimento do código desenvolvido, dos testes efetuados, da configuração do sistema, etc. Pretende-se que cada um mostre o que aprendeu e que estará à vontade no teste/exame.