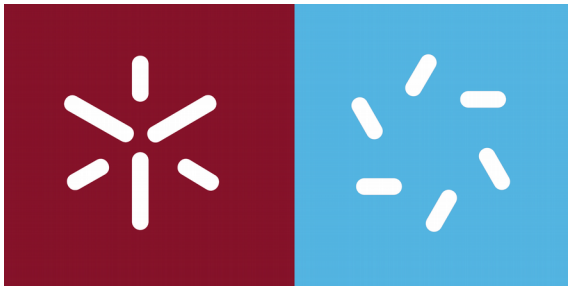
Trabalho de Sistemas Operativos:

Stream Processing



Universidade do Minho

2 Abril de 2017

Grupo 4

Nome:	Numero:
Tiago Costa Loureiro	A71191
Leonel Ferreira Gonçalves	A72305
Luis Rafael Macedo Silva	A72113

Índice

Introdução

Este relatório trata a análise do trabalho prático realizado na disciplina de Sistemas Operativos.

A análise será divida em 2 etapas.

Na primeira etapa, iremos apresentar o problema e perceber qual é o seu objetivo.

Na segunda etapa, será apresentada a proposta de resolução do problema apresentado.

Por fim, será analisada e discutida a resolução apresentada para o problema e se cumpre com que aquilo que é pretendido.

Desta forma, temos como objetivo, a realização do trabalho de forma a consolidar os conhecimentos ensinados em Sistemas Operativos, esperando atingir todos os objetivos propostos.

1-Apresentação do Problema

O problema, foi apresentado, no enunciado, da seguinte forma:

"*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."

Então o trabalho consiste na implementação de um programa que trate um fluxo de dados. O programa deverá contar com diferentes "nodos", onde cada um trata a informação de uma forma especificada.

Assim, o programa receberá um ficheiro de configuração de forma a definir a rede de componentes e um *input* com diversas colunas, por exemplo *informação***:***informação*.

2-Desenvolvimento da Solução

O programa terá um controlador que irá receber e gerir todos os *inputs* de forma a chegarem, aos nodos, em condições que possam ser executados e/ou tratados.

Os vários nodos poderão operar os seguintes programas:

2.1 Componentes

2.1.1 – const <valor>

Neste programa é pedido que, ao ser chamado o programa, adicione, no final de todas as linhas, o argumento dado, como no seguinte exemplo pratico:

```
$ const 100
$ numero:inteiro
numero:inteiro:100
```

Para realizar este programa, usamos o *sprintf* para a criação de uma nova *string*, onde terá como prefixo a *string* passada como *input* e como prefixo a *string* passada como argumento.

2.1.2 – filter <coluna> <operador> <operando>

Este programa filtra o *input* segundo a condição passada como argumento. Caso o *input* satisfaça a condição, será imprimido.

Tendo como operadores para as condições: =, >=, <=, >, <, !=, e os dois argumentos sendo o valor da primeira coluna e o valor da segunda coluna.

Exemplo:

```
$ filter 2 “>”4
$ inteiro:2inteiro:2
```

```
$ filter 2 “>”4
$ inteiro:2inteiro:1
inteiro:2inteiro:1
```

Neste programa, chamamos a função “get_coluna_two” que se situa no ficheiro “funcoes.c” para retornar os inteiro colocados nas colunas das linhas dadas no *input*.

2.1.3 – window <coluna> <operação> <linhas>

Neste programa, recebemos dois argumentos e um operador, onde um dos argumentos indica a coluna onde ocorrerá a operação, e o outro indica as linhas a ter em conta. Os operadores em questão são: avg, max, min e sum.

Exemplo:

```
$ window 4 avg 2
$ numero:3:numero:4
  numero:3:numero:4:0
$ numero:1:numero:10
  numero:1:numero:10:4
$ numero:2:numero:2
  numero:2:numero:2:7
$ numero:5:numero:34
  numero:5:numero:34:6
```

Para o desenvolvimento deste programa, foi criado a função “get_coluna” que está contida no ficheiro “funcoes.c”, que retorna o elemento da coluna onde irá ocorrer a operação. Esse elemento será adicionado a um *array*, onde será aplicada a operação sobre os últimos “n” elementos.

Em seguida, foram criadas 4 funções auxiliares (respetivos operadores): *avg()*, *minimo()*, *maximo()*, *soma()*.

Usando o *sprintf*, foi criado uma nova *string* que tem como prefixo a *string* de *input* e como sufixo o resultado da função auxiliar.

2.1.4 - Spawn <cmd> <args>

Este programa imprime as linhas do *input*, acrescentando, como prefixo, o *exit status* depois de executar, para cada uma, o comando dado como argumento.

Por exemplo,

```
spawn mailx -s $3 x@y.com
$ a:2:w:2
a:2:w:2:0
$ d:5:z:34
d:5:z:34:0
```

Onde w e z são o assunto em mensagens enviadas para x@y.com.

Para o desenvolvimento deste programa, procedemos à criação de um processo filho onde ocorrerá a execução do comando dado como argumento e encerrando-o com a *system call “_exit()”*.

De seguida, é usado um ciclo para receber todos os *exit status* e, usando o sprintf, são colocados como sufixo da respetiva linha de input, imprimindo-a.

2.2 – Controlador

Como enunciado:

"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."

TITO, AJUDA AQUI. DIZ, POR ALTO, COMO FUNCIONA O NOSSO CODIGO DO CONTROLADOR E JUSTIFICA SFF

Os comandos que o controlador deverá aceitar são os seguintes:

2.2.1 - node <id> <cmd> <args...>

O comando "node" permite criar os nodos do programa. Tem como argumentos o "id" que identifica o nodo, o programa que o nodo deve executar e os respetivos argumentos. BRILHA AQUI MIUDO

2.2.2 - connect <id> <ids...>

O comando connect permite criar ligações entre os nodos do programa. Recebe, como argumentos, o "id" de um nodo e a quais deve estar conectado, ou seja, para onde deve ser redirecionado o *output* do nodo "id".

E AQUI

2.2.3 - disconnect <id1> <id2>

Este comando permite desfazer as ligações entre dois nodos.

E AQUI

2.2.4 - inject <id> <cmd> <args...>

Este comando só pode ser usado quando a rede já está definida permite passar o resultado da execução de um comando, com os respetivos argumentos, no nodo "id".

E AQUI

VOU CHORAR :(