Controlador

SISTEMAS OPERATIVOS

2017

Trabalho Prático

Diogo Mendes Gomes a73825
Gonçalo Leal Da Mota Meireles Moreira a73591
Paulo Jorge Machado Guedes a74411

Resumo

Para este trabalho foi pedido para criar um sistema de processamento em série. Este sistema utiliza uma rede de componentes, nodos, para filtrar, modificar e processar um fluxo de eventos. Permite tratar uma grande quantidade de dados explorando a concorrência tanto entre processamentos sucessivos, através do uso de *pipes.*

Conteúdo

Índice de Figuras

Introdução

Para iniciar o sistema, é necessária a criação de componentes iniciais como os nodos da rede, assim como a atribuição de comandos a executar por cada evento. Após a criação de nodos, é necessária a criação dos comandos que este irá executar para processar cada evento.

Assim que criados os componentes iniciais para o funcionamento de um nodo só, é necessário ligar os nodos, redirecionando os eventos de cada nodo para os nodos ligados a este. Para tal, é necessário manter guardadas as ligações que cada nodo tem, de modo direcionar o processamento dos seus eventos para esses mesmos nodos. É necessário garantir que não existe erros na concorrência de processamentos de eventos, de modo a que os eventos não se perturbem uns aos outros.

Componentes Iniciais

Inicialmente, criamos um conjunto de programas que processassem os eventos de cada nodo, sendo eles:

- Const : Este programa reproduz os eventos acrescentando uma nova coluna sempre com o mesmo valor:
- Filter : Este programa reproduz os eventos que satisfazem uma condição indicada nos seus argumentos.
- Window : Este programa reproduz todos os eventos acrescentando-lhe uma nova coluna com o resultado de uma operação calculada sobre os valores da coluna indicada nas linhas anteriores
- Spawn : Este programa reproduz todos os eventos, 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.

Const

Este programa recebe como argumentos o evento a processar seguido da constante a colocar no final, concatenando o evento com a constante, escrevendo o evento processado para o *stdout*. É possível redirecionar o output deste programa, caso seja um nodo sem saída, colocando o nome do ficheiro no final.

Ex: node 0 const 10 const.txt.

```
./const x:10 10
x:10:10
./const x:15 10
x:15:10
./const x:15 15
x:15:15
```

Figura 1 - Programa const

Filter

Este programa recebe como argumentos o evento a processar seguido da primeira coluna a comparar, o operador de comparação e a segunda coluna a comparar.

Assim que recebe o evento, irá separar as colunas, delimitadas por ':' ou '\n', realizando de seguida a comparação das colunas fornecidas como argumento. Para tal, verifica qual o operador fornecido e realiza a comparação, escrevendo o evento caso a comparação seja verdadeira, não o escrevendo caso contrário.

Poderá realizar as operações de '=', '>=', '<=', '>', '<', '!='.

. É possível redirecionar o output deste programa, caso seja um nodo sem saída, colocando o nome do ficheiro no final.

Ex: node 0 filter 2 < 4 filter.txt.

```
./filter a:3:b:4 2 '>' 4

./filter a:5:b:4 2 '>' 4
a:5:b:4
./filter a:5:b:5 2 '>=' 4
a:5:b:5
./filter a:5:b:5 2 '=' 4
a:5:b:5
```

Figura 2 - Programa filter

Window

Este programa recebe como argumento o número de eventos a processar, seguido dos eventos, da coluna a comparar, do tipo de processamento a realizar em cada evento e das linhas anteriores a comparar.

Assim sendo, separa-se cada evento por colunas, delimitadas de ':', '\0' ou '\n', guardando os valores da coluna a comparar (considerado 0 caso não exista essa coluna). De seguida, é calculado o resultado do evento, tendo em conta os anteriores.

Caso não exista eventos anteriores, o resultado será 0. Se não existir eventos anteriores suficientes para satisfazer o número de linhas a utilizar dado como argumento, o resultado será calculado com os eventos anteriores existentes.

O tipo de processamentos disponíveis são:

- avg: realiza a média, da coluna dada como argumento, dos eventos anteriores.
- max: fornece a coluna com o maior valor dos eventos anteriores.
- min: fornece a coluna com o menor valor dos eventos anteriores.
- sum: calcula a soma das colunas, com o índice da coluna dada como argumento, dos eventos anteriores.

É possível redirecionar o output deste programa, caso seja um nodo sem saída, colocando o nome do ficheiro no final.

Ex: node 0 window 4 avg 2 window.txt.

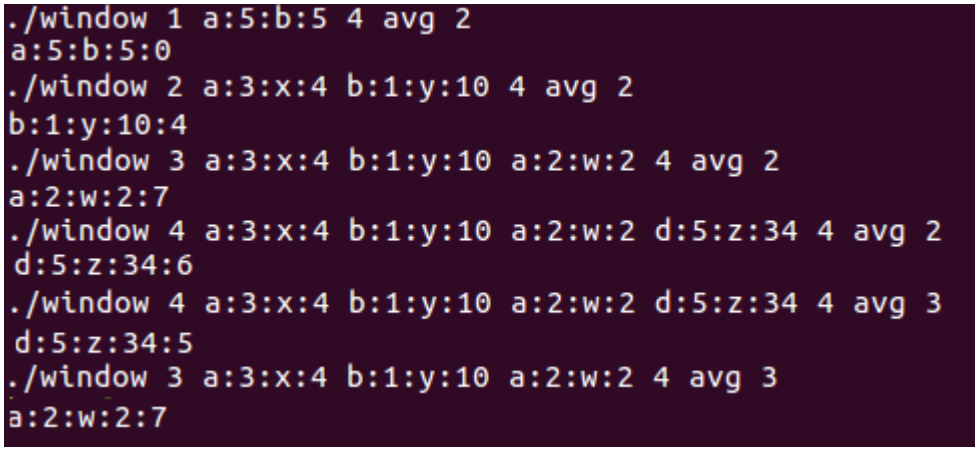

Figura 3 - Programa window

Spawn

Recebe como argumento o evento a processar, assim como o comando a executar. Verifica se algum dos argumentos dados é '$n', caso seja, substitui esse argumento pelo valor da coluna do evento passado como argumento. De seguida executa o comando.

Após o comando executado, é esperado pelo "exit status" do comando, concatenando-o ao evento dado como argumento, escrevendo de seguida para o *stdout*.

É possível redirecionar o output deste programa, caso seja um nodo sem saída, colocando o nome do ficheiro no final antecedido por um '['.

Ex: node 0 spawn echo $3 [spawn.txt.

```
./spawn a:3:x:4 ls
const           enunciado-so-2016-17(1).pdf  fanout.c  log.txt     teste.c
const.c         fanin                        filter    output.txt  window
controlador     fanin.c                      filter.c  spawn       window.c
controlador.c   fanout                       input     spawn.c
a:3:x:4:0
./spawn a:3:x:4 echo '$4'
4
a:3:x:4:0
```

Figura 4 - Programa spawn

Controlador

Inicialmente, definiu-se a estrutura dos nodos como sendo uma estrutura que guarda o seu id, um *array* de inteiros para guardar para quais nodos necessita escrever, o número de nodos que necessita escrever, os comandos a executar por esse modo, e um *pipe* que nos irá permitir comunicar com este.

Node

Aquando a execução do comando "node <id> <cmd>", este irá adicionar ao array que guarda as estruturas dos nodos no índice do array. Decidimos que o utilizador necessitaria de começar por criar os nodes pelo índice 0, e ir avançando de id.

Para iniciar o nodo, preenchemos a estrutura associada a esse nodo com os valores dados como argumento pelo utilizador. De seguida, este criará um novo processo, que irá estar á escuta no seu *pipe*, por eventos, permitindo ao pai que retorne a estar á escuta de eventos no *stdin*. O filho antes de ficar á escuta no seu pipe, irá preencher um array com os comandos necessários a enviar para os programas como argumento, para executar o comando atribuído a esse nodo. Esse array irá ser diferente de comando para comando.

Dificuldades

Como o programa "window" necessita dos eventos anteriores para executar, o array de comandos terá que se adaptar consoante o número de eventos executados anteriores. O tamanho do array também varia consoante o número de eventos anteriores a ter em conta.

Resolução

Resolvemos este problemas considerando que o “window” irá executar tirando partido dos eventos anteriores realizados, alocando memória necessária para guardar o número ideal de eventos anteriores para executar o comando. Para executar o programa “window”, é necessário executar o *execvp* com os seguintes argumentos: “./window”, nº de input, eventos (número variável), coluna, operação, nº de eventos anteriores. Assim, decidiu-se que o tamanho do *array* de comandos seria o nº de eventos anteriores a comparar + 6, para haver espaço para o NULL.

À medida que é executado mais eventos no dado nodo, é adicionado ao *array* de comandos, no índice 2 para a frente, avançando o resto dos comandos uma posição no *array*, até atingir o número de eventos anteriores a ter em conta. No momento em que atinge o número de eventos anteriores a ter em conta, no momento em que recebe um novo evento, o evento guardado á mais tempo é descartado, e o novo é adicionado ao *array*.

Inject

Criados os nodos, é necessário um método para introduzir eventos na rede, visto que os nodos não estão á escuta do *stdin.* Assim, foi criado o método inject <id> <eventos>, no qual é introduzido num certo nodo, um número de eventos.

Assim, é direcionado os eventos para o pipe relativo ao nodo, na qual o nodo irá ler, adicionar o evento ao array de comandos, executando um *execvp* de seguida com o *array* de comandos. Caso o nodo não esteja conectado a nenhum outro nodo, é escrito para um ficheiro “output.txt”, o output do exec. Para efeitos de demonstração, colocamos o output a ser escrito no stdout.

```
node 0 const 10
node 1 filter 2 > 4
node 2 filter 2 != 4
node 3 window 4 avg 2
node 4 window 4 avg 3
node 5 spawn ls
node 6 spawn echo $3
inject 0 x:10
x:10:10
inject 0 x:15
x:15:10
inject 0 x:10 x:15
x:10:10
x:15:10
inject 1 a:3:b:4 a:5:b:4 a:4:b:6
a:5:b:4
inject 2 a:5:b:5 a:3:b:4
a:3:b:4
inject 3 a:5:b:5
a:5:b:5:0
inject 3 b:1:y:10
b:1:y:10:5
inject 3 a:2:w:2
a:2:w:2:7
inject 3 d:5:z:34
d:5:z:34:6
inject 4 a:3:x:4 b:1:y:10 a:2:w:2 d:5:z:34
a:3:x:4:0
b:1:y:10:4
a:2:w:2:7
d:5:z:34:5
inject 5 a:2:b:2
const           enunciado-so-2016-17(1).pdf  fanout.c  log.txt     teste.c
const.c         fanin                        filter    output.txt  window
controlador     fanin.c                      filter.c  spawn       window.c
controlador.c   fanout                       input     spawn.c
a:2:b:2:0
inject 6 a:3:3
3
a:3:3:0
```

Figura 5 - Controlador Inject Node Output

Connect

Criados os nodes, e garantidos que os comandos eram executados corretamente, falta criar um método para criar ligações na rede de nodos. Assim, criou-se o método connect <id> <ids>, que dado vários ids, o nodo com o id <id>

redirecionará o evento processado no seu nodo para os nodos com os ids <ids>. Para tal, foi necessário preencher o array de inteiros da estrutura do nodo com os pipes de escrita dos nodos para o qual o evento era redirecionado.

Após guardados os descritores dos *pipes* para o qual o evento será redirecionado, foi necessário realizar um "fanout" do evento. Para tal, criamos um programa que está a escuta de eventos, guardando-os num *buffer*, escrevendo-o para todos os descritores guardados no array de inteiros do nodo que enviou esses eventos. O evento processado será lido pelos outros nodos, que irão processar o evento e realizar as mesmas etapas até que não nodos conectados ao nodo que está a processar o evento. Se for o caso, o evento será redirecionado para "output.txt", escrevendo também o seu id, para informar qual nodo processou por fim o evento.

Dificuldades

Quando um nodo é criado, é criada uma tabela de descritores nova, copiada do processo pai, assim como copiadas todas as variáveis utilizadas até esse momento. No entanto, as modificações efetuadas na tabela de descritores ou nas variáveis no processo pai após criar o processo filho, não serão feitas também no processo filho. Tendo em conta este fenómeno, caso atualizemos o array de inteiros do nodo, as alterações não serão efetuadas no processo filho, visto que modificamos as variáveis no processo pai, após criarmos o filho.

Para a realização do fanout, houve a mesma complicação, na qual o processo que está a escuta de eventos não sabia qual nodo enviava os eventos, e mesmo sabendo, caso algum nodo fosse conectado após a criação do processo que executa o fanout, este não continha as modificações ao array de inteiros de escrita do nodo.

Resoluções

Assim sendo, os ids dos nodos que serão ligados ao nodo em questão são enviados através do pipe do nodo. Para o processo distinguir os dados recebidos, de

um evento, foi criado um parse no processo filho. Caso o primeiro caracter dos dados lidos seja um '+', será para adicionar descritores ao array de inteiros de escrita do nodo. Caso seja um '-', será retirado algum descritor do array de inteiros de escrita do nodo (iremos discutir mais à frente quando é que este método é executado). Qualquer outro caso, será um evento a ser processado como demonstrado em cima. Assim sendo, os ids dos nodos que serão ligados ao nodo em questão são enviados através do pipe do nodo. Para o processo distinguir os dados recebidos, de um evento, foi criado um parse no processo filho. Caso o primeiro caracter dos dados lidos seja um '+', será para adicionar descritores ao array de inteiros de escrita do nodo. Caso seja um '-', será retirado algum descritor do array de inteiros de escrita do nodo (iremos discutir mais à frente quando é que este método é executado). Qualquer outro caso, será um evento a ser processado como demonstrado em cima.

Para resolver a complicação do fanout, foi definido que o processo que executa o fanout é terminado e criado um novo processo a executar um novo fanout, sempre que existe a criação de um novo nodo, para este processo estar atualizado sobre os nodos existentes. Em adição, foi criado um pipe global (pipeM), na qual o processo que realiza o fanout irá estar á escuta. Os dados enviados a esse pipeM será o id na qual o fanout irá estar á escuta de eventos, e os ids para o qual o fanout irá escrever os eventos processados. Assim sendo, antes dos nodos realizarem o processamento dos eventos, caso tenham descritores no array de inteiros de escrita, enviam pelo pipeM o seu pipe de leitura, assim como os pipes para o qual deseja que o fanout redirecione os eventos. Caso não tenha nenhum descritor para redirecionar o output, é apenas redirecionado o output do processo para o file "output.txt".

Para efeitos de demonstração, o output final será o stdout.

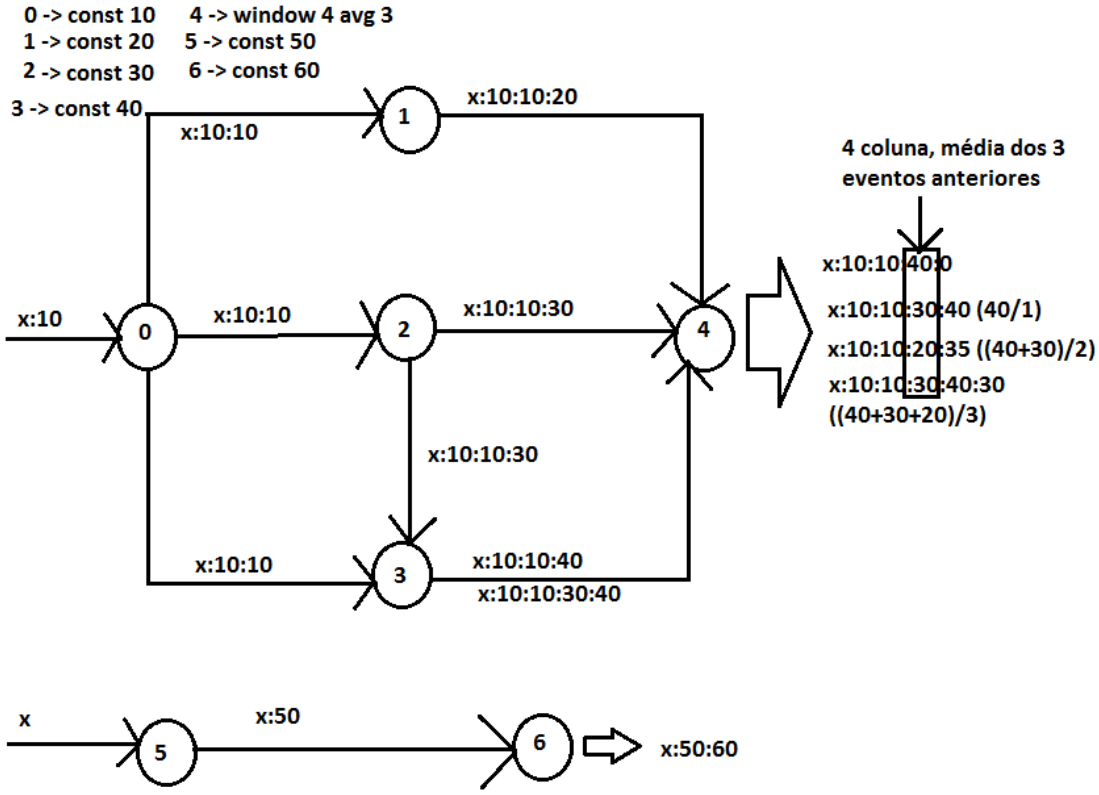

Figura 6 - Connect Rede

```
node 0 const 10
node 1 const 20
node 2 const 30
node 3 const 40
node 4 window 4 avg 3
connect 0 1 2 3
connect 1 4
connect 2 3
connect 2 4
connect 3 4
inject 0 x:10
x:10:10:40:0
x:10:10:30:40
x:10:10:20:35
x:10:10:30:40:30
node 5 const 50
node 6 const 60
connect 5 6
inject 5 x
x:50:60
```

Figura 7 - Connect Demonstração

Disconnect

Após estabelecidas as conexões, poderá ser necessário desfazer uma ligação entre nodos. Para tal, foi necessário criar o método disconnect <id> <id>, na qual dado dois id de nodos, ela retira o pipe do segundo id do array de descritores do nodo, caso este exista.

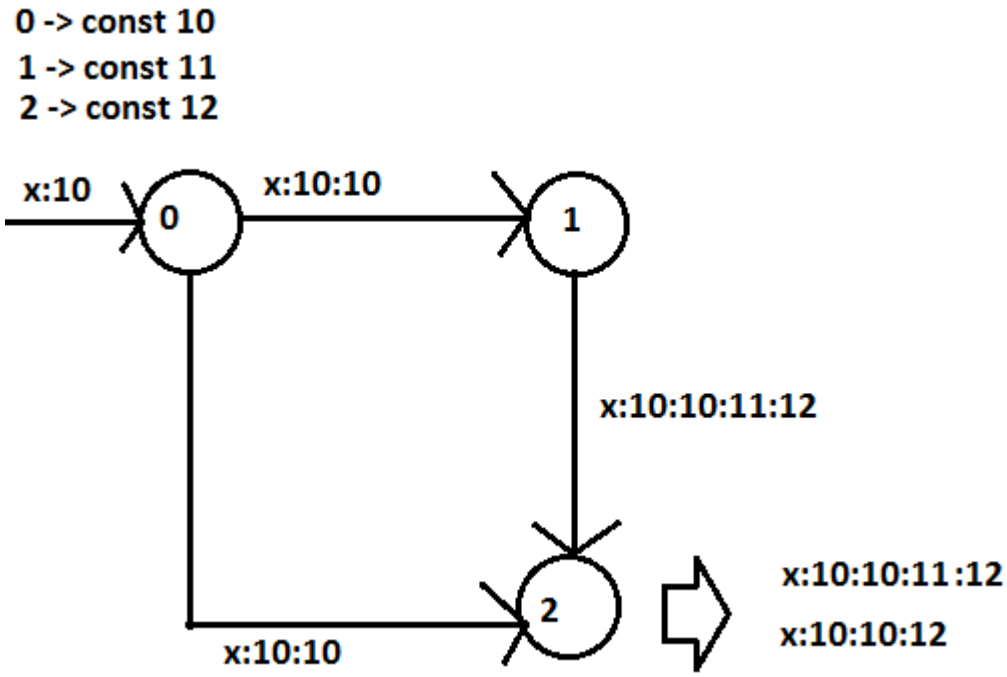

Figura 8 - Rede Antes do Disconnect

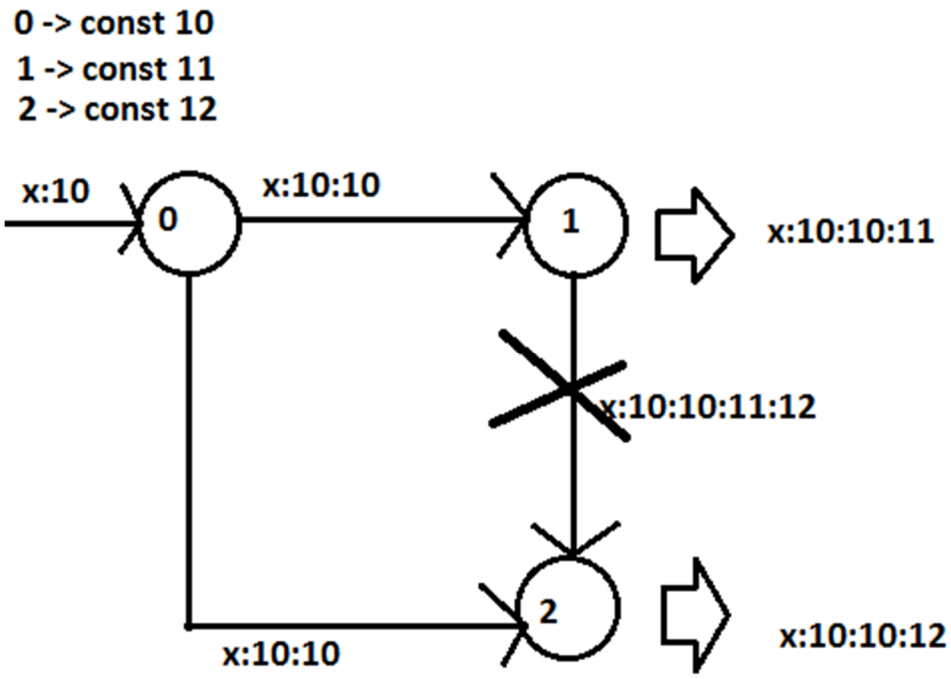

Figura 9 - Rede Após o Disconnect

```
node 0 const 10
node 1 const 11
node 2 const 12
connect 0 1 2
connect 1 2
inject 0 x:10
x:10:10:12
x:10:10:11:12
disconnect 1 2
inject 0 x:10
x:10:10:11
x:10:10:12
```

Figura 10 - Controlador Disconnect

Conclusão

Após a realização do projeto, é possível criar uma rede que processe eventos em cadeia, através de várias funcionalidades da linguagem C, nomeadamente, *pipes*, processos, etc. Assim, conseguimos criar uma rede bastante elaborada de modo a que os eventos sejam redirecionados para os nodos aos quais estão ligados, conseguindo processar eventos de diversas formas, de modo a que os nós de uma rede continuem a existir até o término do programa, garantiu-se a concorrência nas escritas no mesmo nó, sendo mais eficiente sem nunca afetar os eventos. Por pré--definição, os eventos são redirecionados para o ficheiro “output.txt”, caso nada seja dito em contrário. A rede vai sendo atualizada á medida que se vai adicionando ligações, notificando os processos através de *pipes* sobre as alterações relevantes ao processo. No futuro, poderia ser implementado a opção de remover nodos da rede, não sendo possível de momento, podendo também ser implementado a alteração de componentes da rede.