Projeto de Sistemas Operativos 2018-19
`CircuitRouter-AdvShell`
Enunciado do Exercício 3

LEIC-A / LEIC-T / LETI
IST

Antes de lerem este guia, os alunos devem ler primeiro o documento de visão geral do projeto, assim como os enunciados dos exercícios anteriores do projeto.

O objetivo do Exercício 3 é estender a `CircuitRouter-SimpleShell` (daqui em diante, abreviado como `SimpleShell`) com novas funcionalidades avançadas recorrendo aos mecanismos de coordenação/comunicação entre processos estudados recentemente nas aulas teóricas. Neste documento chamaremos `CircuitRouter-AdvShell`, ou simplesmente `AdvShell`, à variante que será desenvolvida neste exercício.

Mais precisamente, as novas funcionalidades resumem-se a:

- Suporte à submissão de pedidos por clientes remotos
- Medição dos tempos de resolução de cada circuito

As secções seguintes descrevem cada requisito em detalhe.

1 Suporte à submissão de pedidos por clientes remotos

O `SimpleShell` recebia os comandos exclusivamente através do `stdin`. O `AdvShell` deverá ser capaz de *também* receber comandos emitidos por outros processos através de um *named pipe*.

Haverá um programa novo, chamado `CircuitRouter-Client` (daqui em diante, abreviaremos para `Client`), que recebe comandos do seu `stdin` e os envia para o *named pipe* onde o `AdvShell` aguarda comandos.

Esse *named pipe* deverá ser associado ao *pathname* composto pelo nome do executável do `AdvShell` e sufixado por `.pipe`. Por exemplo, caso o programa corra a partir de `/tmp/ex3/AdvShell`, o *named pipe* será `/tmp/ex3/AdvShell.pipe`.

Para saber qual *pathname* do *named pipe* descrito acima, o `Client` recebe um argumento de linha de comando obrigatório que indica o *pathname* do *named pipe* para onde os comandos devem ser enviados.

Através do *named pipe*, o `AdvShell` só aceita comandos do tipo `run`, tal como definidos no Exercício 1. Assim que o circuito solicitado seja resolvido, o processo filho que o resolveu deve enviar uma mensagem "Circuit solved" ao `Client`. Ao receber outros comandos (tal como `exit`) por este canal, o `AdvShell` deve simplesmente responder "Command not supported." no `stdout` do `Client`.

Em ambos os casos (comando `run` aceite ou comando não suportado), a mensagem que o processo filho envia ao `Client` deve seguir por um *named pipe* e ser impressa pelo recetor no seu `stdout`.

Notas importantes:

- Uma vez que os *pipes* não são bidirecionais, o *named pipe* referido no início desta secção (para onde o `Client` envia pedidos) não pode ser usado para o `Client` receber as mensagens de resposta aos seus pedidos. Compete a cada grupo definir que outros *named pipes* devem ser criados e como devem ser usados pelos processos que deles dependem.
- A solução deve suportar corretamente a existência de múltiplos processos `Client` em simultâneo. Além disso, o `AdvShell` deve continuar a ser capaz de também receber comandos diretamente pelo seu `stdin`.

2 Medição dos tempos de resolução de cada circuito

No `SimpleShell`, o processo principal apenas tomava conhecimento que cada processo filho havia terminado (quando o limite de filhos simultâneos era esgotado ou quando recebia o comando `exit`).

No `AdvShell`, pretende-se que o processo principal também passe a monitorizar quanto tempo durou a execução de cada processo filho. Tal implica que o processo pai passe a registar, para cada processo filho, os instantes em que este foi criado e em que terminou. Para observar o instante de terminação, o processo pai deve tratar a receção do *signal* `SIGCHLD`.

Assim sendo, a listagem final que o processo pai apresenta após receber o comando `exit` passa a ser mais completa, pois passa a indicar a duração (em segundos) de cada processo filho. De seguida ilustramos o formato esperado:

```
CHILD EXITED (PID=2987; return OK; 65 s)
CHILD EXITED (PID=2945; return NOK; 89 s)
END.
```

Algumas notas importantes:

- A solução implementada para cumprir este requisito não deve implicar qualquer alteração ao código do processo filho; ou seja, este requisito deve ser implementado exclusivamente do lado do processo pai.
- O processo pai não tem controlo sobre quando receberá o `SIGCHLD` e poderá ser interrompido a qualquer momento pela rotina de tratamento do *signal*. Portanto o acesso a estruturas de dados deve ser feito com cuidado acrescido. Adicionalmente, funções não reentrantes não devem ser usadas dentro da rotina de tratamento.[1]
- Como foi discutido nas aulas teóricas, diferentes sistemas operativos UNIX/Linux podem oferecer diferentes semânticas de tratamento de *signals*. A solução desenvolvida deverá, idealmente, ser portável independentemente da semântica de *signals* que o sistema operativo suporta por omissão.

3 Submissão

A submissão deve seguir as regras definidas no enunciado geral do projeto.

[1] http://man7.org/linux/man-pages/man7/signal-safety.7.html