Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas

Sistemas Operativos 2021/2022

Ficha Nº 5 Named pipes. select

Tópicos ->
- Mecanismo de comunicação iterprocesso *named pipes*.
- Aplicações cliente servidor com *named pipes*.
- Cenários exemplo de uso de sinais. Alertas, Coordenação e temporização

Exercícios

Exercício 1: Apresenta os aspectos básicos de *named pipes*.

Exercício 2 e 3: Foca a arquitectura cliente servidor com *named pipes*.

Exercício 4: Mostra o mecanismo *select*.

1. Utilize o comando *mkfifo* para criar um *named pipe*.

I. Num terminal apresente o conteúdo do *named pipe* com o programa cat.

II. Num segundo terminal envie alguma informação para esse *named pipe*.

(ex., *cat > pipe*, *ls > pipe*, echo *"ola mundo" > pipe*).

Escreva algumas linhas e saia com ^D

Repare nos seguintes aspectos de funcionamento:

- No primeiro terminal, o comando *cat* fica a aguardar que haja informação disponível.
- Quando o programa no segundo terminal (*ls*, *echo*, outro) termina, o comando *cat* no primeiro programa detecta que já não haverá mais informação e encerra também.
- Repare que o "cat > pipe" no segundo terminal não sai com ^D se não estiver previamente um programa no primeiro a usar o named pipe. O programa no segundo terminal depende do "final de uso" de pipe no primeiro terminal. O mecanismo de sinais é envolvido nesta questão. Averigúe junto do docente se tiver questões nesta parte.

Verifique que entende os seguintes aspectos (confirme com o docente caso haja dúvidas):

- A relação entre um *named pipe* e o sistema de ficheiros. Verifique o conteúdo da directoria e constate que tem uma nova entrada correspondendo a um ficheiro especial que é o *named pipe*. Identifique a letra indicadora do tipo de ficheiro especial *named pipe*?
- A forma como a informação é enviada através dos *named pipes* (*byte stream*, sem formato específico).
- A forma como funciona a detecção de novos dados disponíveis e *fim de ficheiro*.

Verifique que outras operações habituais de ficheiros podem ser efectuadas sobre o *named pipe*. No final, apague o ficheiro (comando *rm*).

Objectivos do exercício - *Named pipes* ou *FIFOs*.
- Conceito de *named pipes* (FIFOs).
- Funcionamento geral de *names pipes*
- Comando *mkfifo*.

2. Pretende-se um programa "eco" que ecoa no ecrã todas as informações que recebe vinda de outros processos (designados "produtores"). O programa "eco" recebe a informação através de um *named pipe* e mantém-se sempre em funcionamento até receber um ^C ou receber a palavra "sair". Os outros processos recebem a informação de várias fontes, mas por agora pode assumir que é obtida via teclado.

a) Debata a arquitectura a usar neste cenário. Verifique que aborda os tópicos:
   - Qual o modelo de arquitectura a usar.
   - Quem cria o *named pipe* e quem é responsável por o gerir.
   - Como se decide o nome do *named pipe* e como sabem os restantes processos esse nome.
   - Qual o modo de uso mais adequado: bloqueante / não bloqueante.
   - Que tipo de informação passa no *named pipe*: mensagens / *byte-stream*.

b) Construa o programa "eco". Coloque-o a correr. Noutro terminal verifique a existência do *named pipe* no sistema de ficheiros. Usando vários terminais, envie informações para o *named pipe* (exemplo *cat etc > pipe* e *echo etc > pipe*).

c) Construa um programa "produtor" para produzir de informação para o "eco". O produtor faz penas uma pergunta ao utilizador, envia o que tiver sido escrito ao "eco" e encerra logo. Confirme que o "eco" se mantém a funcionar correctamente após o produtor terminar.
   I. Identifique o problema que o "eco" apresenta (depende de como o implementou). A questão tem a ver com o facto de ter deixado de haver processo(s) com o *named pipe* aberto para escrita.
   II. Resolva o problema no servidor. Se não existir problema nenhum, identifique a parte na sua implementação do "eco" que evitou problemas no *named pipe* quando deixa de haver outro processo com o *named pipe* aberto para escrita.

d) Modifique o programa "produtor" de forma a manter-se em execução. Só termina quando recebe a informação (pelo teclado) "termina". Experimente correr vários programas "produtor" em simultâneo (vários terminais) e teste novamente os programas.

e) Experimente correr vários processos "produtor". Num deles mande terminar o "eco" (enviando "sair" ou encerrando-o manualmente com ^C). O que acontece aos processos "produtor"?
   - Proponha e implemente uma forma destes detectarem que o servidor já não está a correr que permita aos produtores encerrarem também de forma ordeira (pode existir várias formas de fazer este ponto).

f) Modifique os programas "eco" e "produtor" de maneira a que a resposta do "eco" seja enviada para o processo que envio a mensagem (em vez de ir simplesmente par ao ecrã). O produtor é que enviará a resposta para o ecrã.
   - Quais são as alterações estruturais ao sistema e modelo de comunicação que está a usar?

- Que *named-pipes* adicionais serão precisos:
    I. Quantos, para que fim, quem os cria, qual a sequência de abertura e em que modo por parte de que processo?
    II. Havendo mais *named pipes*, qual a sua identificação e como sabem os outros processos essa identificação?

Objectivos do exercício - Sistemas "caixa de correio" e "cliente-servidor" com *named pipes*.
- Entender o modelo de comunicação "caixa de correio"
- Entender os aspectos de implementação relativos ao modelo "caixa de correio"
- Entender o modelo de comunicação "cliente servidor"
- Entender os aspectos de implementação relativos ao modelo "cliente servidor"
- Uso de *named pipes* e do API relacionado com este mecanismo de comunicação.
- Entender e resolver os aspectos práticos envolvidos no uso de *named pipes* no contexto do modelo cliente-servidor.
- Funções *mkfifo*, *unlink*, *open*, *close*, *read*, *write.*

3. Na sequência do exercício anterior e usando os conhecimentos que adquiriu, Construa um sistema cliente-servidor que permita efectuar cálculos. O sistema é constituído por um programa "servidor" que aguarda pedidos de operações, e por clientes que pedem os dados e operações ao utilizador, enviam o pedido ao servidor, aguardam a resposta, e depois apresentam o resultado no ecrã. O cliente termina quando se pede para fazer a operação ".". O servidor encerra apenas quando recebe ^C. Quando o servidor encerra, os clientes também encerram.

a) Proponha uma arquitectura para a resolução do problema.

b) Defina as mensagens que irão ser trocadas entres os vários processos.

c) Faça o sistema pedido de forma robusta (o servidor nunca entra em funcionamento "em vazio" quando os restantes processos terminam, os clientes detectam quando o servidor não está a correr, etc.). Use o modelo de comunicação mais adequado à natureza do problema proposto e implemente-o de forma correcta e racional.

d) Faça com que o servidor liberte a consola ao ser lançado (sem usar o caracter & na linha de comandos).

e) Acrescente a funcionalidade de "autenticação". OS clientes apenas poderão usar a funcionalidade do servidor após enviarem um nome e uma password. Existe um ficheiro de texto com nomes e *password* (um par nome password por linha). Este ficheiro é consultado pelo servidor e o nome do ficheiro é
    I. Dado como argumento na linha de comandos.
    II. Dado na variável de ambiente PASSFILE.

f) Acrescente a seguinte funcionalidade: O servidor pode ser encerrado quando recebe um sinal. Ao encerrar avisa os clientes. Implemente esta notificação aos clientes com sinais.

Objectivos do exercício - Sistemas cliente-servidor com *named pipes*.
- Rever e treinar o modelo de comunicação cliente-servidor.
- Treinar os aspectos de implementação relativos ao modelo cliente servidor.
- Uso de *named pipes* e do API relacionado com este mecanismo de comunicação.
- Uso de estruturas como conteúdo a enviar através de *named pipes*.
- Treinar a resolução de aspectos práticos envolvidos no uso de *named pipes* no contexto do modelo cliente-servidor.
- Rever a obtenção de informação por argumento da linha de comandos e por variáveis de ambiente.
- Funções *mkfifo*, *unlink*, *open*, *close*, *read*, *write. Funções biblioteca C-standard várias.*

4. Escreva um programa "eco2" que crie dois *named pipes* (pipe_a e pipe_b). O programa irá repetir no ecrã tudo aquilo que lhe for enviado em cada um dos *named pipes* e mantém-se em funcionamento. Não existe nenhuma sequência prevista quanto às mensagens. A qualquer instante pode chegar informação por cada um dos *named pipes*.

- O programa "eco2", com as características enunciadas acima representa a parte essencial do exercício.
- Faça também um programa "envia" para usar em conjunto com o "eco2". Este programa consiste num ciclo: pede *string*, envia ao "eco2", repete. Este programa deve receber (linha d comandos, utilizador, outra) a indicação de qual o *named pipe* do eco2 para onde irá enviar a informação.

  Para melhor testar a funcionalidade deste exercício, pode ter dois "envia" em execução em simultâneo, uma para cada *named pipe* do "eco2".
- Faça os programas prevendo uma forma de os terminar de forma ordeira (ex., ^C no "eco2", "sair" no "envia".

a) Assuma que vai resolver o problema apenas com base nos conhecimentos usados até aqui nos exercícios anteriores. Qual a maior dificuldade na resolução deste problema? Esta alínea é para planear e debater com o docente e não envolve código.

b) Construa os programas "eco2" e "envia" recorrendo ao uso de sinais: o programa "envia" notifica o "eco2" quando lhe envia algo.

   Nota: pode usar a forma sigaction/sigqueue para indicar qual o *named pipe* onde o eco2. Se usar apenas signal/kill, como faria para descobrir qual o *pipe* a ler? No caso mais restritivo de usar apenas um sinal com signal/kill, como seria a estratégia a usar pelo eco2?

c) Considere agora que o eco2 também pretende ler informação do teclado em simultâneo com os dois *named pipes*. Continua a não se saber por que ordem as mensagens vão chegar.

   i. Adapte o programa de forma a cumprir este novo requisito.

   ii. Refaça os programas de forma a não ser necessário o uso de sinais. Use o mecanismo *select*.

Objectivos do exercício - Leitura de diversas fontes de dados em simultâneo com *select* e com sinais
- Compreender o problema na leitura de diversas fontes de dados em simultâneo.
- Perceber como o mecanismo de sinais pode ajudar nestas situações.
- Entender e usar o mecanismo select para leitura de diversas fontes de dados em simultâneo.
- Funções *close*, *open*, *read, write, mkfifo, unlink, signal, kill, select*

5. Volte ao exercício 3 e modifique-o da seguinte forma: o servidor não liberta a consola: em vez disso mantêm-se atendo ao *input* via teclado, sendo a ordem para o encerrar agora dada pela palavra "encerra" escrita pelo utilizador. O servidor mantem-se alerta aos clientes. Use apenas *select* nesta nova funcionalidade.

Objectivos do exercício - Leitura de diversas fontes de dados em simultâneo com *select*
- Consolidar o uso do mecanismo *select*.