Universidade de Coimbra
Faculdade de Ciências e Tecnologia

DEPARTAMENTO DE ENG. ELETROTÉCNICA E COMPUTADORES
Redes de Computadores

Unix - Comandos e Programação

Ficha 1
Ano Letivo de 2021/2022

---

1. Processos e pipes

Os exercícios de programação desta secção podem ser efetuados num servidor ou na sua própria máquina. Para utilizar uma máquina remotamente, deve ser utilizado um cliente telnet ou ssh.
O código, em linguagem C, deve ser guardado num ficheiro de texto com extensão "***.c*** " (ex.: prog.c). Estes ficheiros podem ser criados utilizando qualquer editor de texto (ex.: **vi** ou **pico**).
Finalmente, o código guardado no ficheiro de texto deve ser compilado utilizando um dos compiladores **cc** ou **gcc**. Por exemplo, para compilar o ficheiro "prog.c", pode executar-se a seguinte instrução na linha de comando:

```
cc prog.c
```

Como resultado desta operação, obtém-se um ficheiro executável com o nome "**a.out**". Se se pretender atribuir um nome específico ao ficheiro executável, resultante da compilação, pode-se utilizar a opção ***–o*** seguida do nome pretendido. O exemplo seguinte mostra como atribuir o nome "***prog***" ao executável resultante da compilação:

```
cc prog.c –o prog
```

Para executar o ficheiro resultante, basta escrever o seu nome (incluindo a extensão) na linha de comando.
Para mais detalhes sobre estes compiladores, pode-se consultar as respectivas *man pages* (ex.: ***man gcc***).

Criação de Processos em UNIX – fork()

Daemons

Em Unix, um ***daemon*** é um processo (programa) que aguarda pedidos para executar determinada acção e a executa. Por exemplo, num servidor de páginas *web* existe um processo (ex: *httpd*) que aguarda pedidos dos utilizadores para mostrar páginas *html*. Quando recebe um pedido, este *daemon* satisfá-lo implementado as regras do protocolo *http*.

No entanto, um *daemon* deve estar sempre apto a receber pedidos dos vários utilizadores e a satisfazê-los, ao mesmo tempo.

Para resolver este problema, o *daemon* cria um processo idêntico a si próprio, um novo processo, para cada pedido recebido. Enquanto o processo original, **processo pai**, continua à espera de novos pedidos, o novo processo, **processo filho**, satisfaz o pedido.

Identificação de processos

Todos os processos têm um identificador – ***pid***. Este identificador é um número inteiro e é utilizado distinguir os processos entre si.

Existem duas funções importantes para a identificação dos processos: ***int getpid()*** e ***int getppid()***.

A primeira, devolve o *pid* do próprio processo. A segunda, devolve o *pid* do seu processo pai.

Criação de processos

Para criar um processo filho, o processo pai recorre à função ***int fork()***. A partir do momento que esta função é chamada, é criado um processo filho absolutamente idêntico ao processo pai. O processo filho herda, para além do código, todo o contexto do processo pai (variáveis, valores, etc.). Imediatamente após a criação do novo processo, ambos os processos (pai e filho) continuam a sua execução na instrução imediatamente a seguir ao *fork()*.

Embora sejam processos absolutamente idênticos, após a criação do processo filho estes processos têm funções distintas: o pai continua a aguardar pedidos do utilizador enquanto que o filho vai satisfazer um pedido em particular.

A distinção na execução do código em cada processo é feita mediante a análise do valor inteiro devolvido pela função *fork()*. Enquanto que ao processo pai é devolvido o identificador do processo filho (*pid*), ao processo filho é devolvido o valor zero.

Em caso de erro, a função *fork()* devolve o valor -1.

<u>Exemplo 1:</u>

```
#include <stdio.h>
int main()
```

```
{
        printf("Olá!\n");
        fork();
        printf(" Eu sou um processo\n");
}
```

Ao executar o exemplo acima, seria escrito no ecrã o seguinte:

```
Olá!
 Eu sou um processo
 Eu sou um processo
```

Na realidade, o processo pai escreveria, apenas, as duas primeiras linhas. O processo filho escreveria a terceira.

O processo pai começa por executar a primeira instrução (printf("Olá!\n);) e escrever a primeira linha (Olá!). De seguida, é criado um processo filho (fork();) e passam a estar em execução dois processos idênticos (pai e filho). A execução de ambos os processo continua com a instrução imediatamente a seguir ao *fork()*. Ambos os processos escrevem " Eu sou um processo".

Exercício 1:

Faça um programa idêntico ao do exemplo acima mas, em que o processo pai escreve "Eu sou o processo pai e o meu pid é X" e em que o processo filho escreve "Eu sou o processo filho, o meu pid é Y e o do meu pai é X".

Comunicação entre processos – Pipes

Um ***pipe*** é um meio de comunicação entre dois processos. Um processo escreve informação no *pipe* enquanto o outro processo a lê essa mesma informação à saída do *pipe*. Por analogia, podemos comparar um *pipe* com um tubo em se pode colocar algo num extremo e retirar no outro extremo. No *pipe*, a comunicação é unidireccional, existindo um identificador para a leitura e outro para a escrita. A informação só pode ser lida pela mesma ordem com que é escrita no *pipe*.

Os *pipes* utilizam as mesmas primitivas do sistema de ficheiros. As primitivas ***read***, ***write*** e ***close*** utilizam-se para os *pipes* exactamente da mesma forma que para os ficheiros.

A **criação de um *pipe*** é efectuada com recurso à primitiva:

```
int pipe (fileids)
        int fileids[2];
```

A primitiva ***pipe*** **devolve** -1 no caso de ocorrer algum erro, caso contrário devolve 0.

Após a chamada desta primitiva, o *array fileids* conterá na sua primeira posição, fileids[0], o descritor do extremo de leitura do *pipe*. Na segunda posição, fileids[1], conterá o descritor do extremo de escrita do *pipe*.

Na **comunicação unidireccional entre um processo pai e um processo filho**, o *pipe* deve ser criado no processo pai, antes da criação do processo filho. Quando este é criado, herda o *pipe* com ambos os descritores, leitura e escrita. Antes da utilização do *pipe*, um dos processos deve fechar (*close()*) o descritor de leitura; o outro processo deve fechar o descritor de escrita.
Se se pretender estabelecer uma comunicação bidireccional entre os processos, devem ser criados dois *pipes*, um para cada sentido da comunicação.

Exemplo 2:

```
#include <stdio.h>
#include <string.h>

int main()
{
 char msg[20];
 char buffer[20];
 int fileids[2];

 if (pipe(fileids) == 0)   /* Criou o pipe, sem erros */
  if (fork() == 0){       /* Processo Filho */
    close(fileids[0]);    /* Fecha o descritor de leitura */
    strcpy(msg, "Olá Pai!");
    write (fileids[1], msg, sizeof(msg));
    close(fileids[1]);    /* Fecha o descritor de escrita */
  }
  else{                   /* Processo Pai */
    close(fileids[1]);    /* Fecha o descritor de escrita */
    read (fileids[0], buffer, sizeof(buffer));
    printf("Recebi do pipe a mensagem: %s\n",buffer);
    close(fileids[0]);    /* Fecha o descritor de leitura */
  }
 else
  printf("ERRO na criação do PIPE!\n");

 return(0);
}
```

Ao executar o exemplo acima, seria escrito no ecrã o seguinte:

```
Recebi do pipe a mensagem: Olá Pai!
```

Exercício 2:

Faça um programa idêntico ao do exemplo acima mas, em que o processo pai envia ao processo filho a mensagem "Olá filho, o meu pid é X!" e em que o processo filho envia ao processo pai a mensagem "Olá pai, o meu pid é Y!". Caso o pid do processo pai recebido pelo filho através do pipe seja diferente daquele obtido pela chamada ao getppid(), o processo filho deverá enviar para o ecran uma de mensagem de erro.
Ambos os processos devem escrever no ecran a mensagem que receberam.

---

2 - Comandos de Gestão

De modo a realizar estes exercícios abra uma janela de terminal.
Para obter ajuda sobre qualquer um dos comandos apresentados use o manual *online* do Linux. Por exemplo para ver o manual *online* do comando ping faça man ping.

2.1 - Ficheiros de configuração

As ferramentas gráficas presentes nas várias distribuições Linux editam vários ficheiros de configuração. Embora muitos ficheiros de configuração sejam dependentes da distribuição em questão, existem alguns bastante comuns: /etc/hosts , /etc/resolv.conf

a) Identifique os endereços presentes no ficheiro /etc/hosts.
b) Para que serve o ficheiro /etc/resolv.conf ?

2.2 - Comandos de configuração da rede

hostname

Mostra ou configura o nome da máquina (para configurar é necessário ser o utilizador *root*).

ip

O comando ip (**ifconfig** ou **route** em alguns sistemas mais antigos) permite atribuir endereços IP às interfaces, criar rotas para outras máquinas, apresentar a configuração do TCP/IP.
Ex: /sbin/ip addr

netstat

Mostra conexões de rede, tabelas de roteamento, estatísticas da *interface*.

host

Permite obter endereços IP a partir do nome das máquinas, ou vice-versa. Faz os pedidos ao DNS (*Domain Name Server*). Permite também saber informação de domínios.

ping

Ver se um *host* está ligado à rede (a máquina pode estar configurada para não responder, por motivos de segurança).
(caso o ping esteja configurado para enviar mais de um pacote, interrompa-o com Ctrl+C)

traceroute

Permite ver qual a rota usada por um pacote para determinado *host* destino.
Ex: /usr/sbin/traceroute www.fccn.pt

whois

Ver informação específica do domínio.
Ex: whois uc.pt

nslookup

Programa que permite obter informações do(s) servidor(es) DNS.

Recorrendo aos comandos anteriores responda às perguntas seguintes:

a) Qual o nome da sua máquina?

b) Descubra quais os endereços IP e físicos da sua máquina. Quantas interfaces *ethernet* e Wifi tem a sua máquina?
c) Qual a tabela de routing da sua máquina?

d) Qual o significado da rede 127.0.0.0/8?

e) Qual o endereço IP de *broadcast* da rede a que pertence a interface eth0?

f) Descubra máquinas activas na sua rede local executando o comando ping para o endereço de *broadcast* da sua rede local (para saber este endereço pode usar o comando netstat –ei). Descubra o nome de 3 das máquinas encontradas na alínea anterior.

g) Qual o nome do responsável pelo domínio “pt”?

h) Quais os servidores de DNS para o domínio “uc.pt”?

i) Veja se a máquina “www.dei.uc.pt” está ligada à rede e se responde a comandos ping. Interprete os resultados obtidos.

j) Caso lhe seja possível, faça um ping com 8 requisições de *echo*, tamanho de pacotes de 100 bytes e TTL de 80, à máquina “www.dei.uc.pt”.

k) Verifique o caminho entre a sua máquina e o *host* “www.linux.org”. Interprete os resultados obtidos.

l) Para que serve o TTL no comando traceroute, e qual o seu significado?