Licenciatura em Eng.ª Informática

Sistemas Operativos - 2º Ano - 1º Semestre

Trabalho Nº 4 – Desenvolvimento de um mini-shell

(baseado no exercício presente em http://www.cs.huji.ac.il/~davidt/course/os/ex10.html)

1. Notas sobre criação de processos

A criação de processos é uma das funções fundamentais do Sistema Operativo, pois é através dela que são lançadas novas aplicações.

Os processos podem ser criados através das interfaces utilizadores (CLI ou GUI), ou directamente a partir da interface de programação do Sistema - *system calls*.

Com este trabalho pretende-se que os alunos utilizem a funcionalidade de criação de processos, usando primeiro a interface utilizador identificando a árvore de processos criada a partir da sessão de *login*.

Seguidamente, através de um programa em C, irão utilizar a API dos *System Calls* para entender o modo como os interpretadores executam os comandos do utilizador.

1.1 Lançamento de uma aplicação a partir da interface de linha de comando (CLI ou Command Line Interface).

Depois de entrar em sessão na sua estação de trabalho, abra um terminal.

Digite um comando para executar um editor de texto qualquer, por exemplo:

```
$ gedit teste.c
```

Através deste comando, foi criado um processo para a execução do editor de texto, e este fica em modo interactivo à espera de comandos.

Vamos agora descobrir o processo correspondente ao editor, listando os processos do sistema que foram criados pelo utilizador <a12345678>. Para isso, vamos executar, **num outro terminal** o seguinte comando:

```
$ ps lU a1234567
```

Este comando lista todos os processos criados pelo utilizador a1234567, indicando para cada um deles o **process_id** (PID) e o **parent_process_id** (PPID).

Começando pelo processo correspondente ao editor de texto que executou, construa a hierarquia de processos até descobrir o processo de *login* inicial.

Não termine ainda o processo que lançou no primeiro terminal.

1.2 Lançamento de uma aplicação a partir da interface gráfica (GUI ou Graphical User Interface)

Execute um editor (gedit, kwrite ou kate) como costuma fazer quando quer editar um ficheiro.

No menu sistema, procure a aplicação KSystemGuard. Uma vez lançada, escolha o Tab *Process Table* que permite visualizar a lista de processos do sistema de forma gráfica. Na

parte inferior da janela escolha a opção Tree, que permite visualizar os processos de forma hierárquica.

Procure o processo correspondente ao editor, identifique o seu PID e o seu processo pai, assim como os processos que estão ao mesmo nível. Identifique também o processo que lançou no ponto anterior a partir do CLI e verifique se a hierarquia de processos que construiu no ponto anterior estava certa.

1.3 Criação de um processo a partir de um programa em C

Utilizando os *system calls* **fork**() e **wait**(), e baseando-se no exemplo apresentado abaixo, escreva um programa em C realize as seguintes funcionalidades:

O processo pai deverá:
1. Deve começar por imprimir o seu PID,
2. Criar um novo processo e imprimir o respectivo PID,
3. Esperar que o processo filho termine e depois terminar.

O novo processo deverá:
1. Imprimir o seu PID e o PID do pai (PPID),
2. Terminar.

```
=============
int main()
{
    pid_t  pid;
    int status;

    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            exit(-1);
    } else if (pid == 0) { /* child process */
            printf ("Child Began\n");
    } else { /* parent process */
            /* parent waits for the child to complete */
            wait (&status);
            printf ("Child Complete\n");
    }
    exit(0);
}
=============
```

1.4 Execução de um comando a partir de um novo processo

Utilizando os *system calls* **fork**(), **exec**() e **wait**(), escreva um programa em C realize as seguintes funcionalidades:
O processo pai deverá:
1. Deve começar por imprimir o seu PID,
2. Criar um novo processo e imprimir o respectivo PID,
3. Esperar que o processo filho termine e depois terminar.

O novo processo deverá:
1. Imprimir o seu PID e o PID do pai (PPID),
2. Executar o comando cujo nome é passado como argumento ao processo pai.

2. Notas sobre Comunicação entre processos

A comunicação entre processos permite que vários processos possam colaborar entre si de forma coordenada e sincronizada.

Uma das formas mais simples de comunicação em Linux é constituída pelos *pipes*, que permitem criar canais monodireccionais entre dois processos.

Nomeadamente, os pipes permitem encadear vários comandos de forma a por exemplo, realizar filtros sobre o output de comandos.

Com este trabalho pretende-se que os alunos utilizem a funcionalidade de criação de pipes, usando primeiro a interface utilizador.

Seguidamente, através de um programa em C, irão utilizar a API dos *System Calls* para entender o modo se criam e utilizam *pipes* entre vários processos.

2.1 Criação de um *pipe* a partir da interface de linha de comando (CLI).

Depois de entrar em sessão na sua estação de trabalho, abra um terminal. Digite o seguinte comando:

```
$ ls -l /home/alunos
```

Este comando lista as pastas de trabalho de todos os alunos inscritos nas aulas práticas do laboratório.

Utilizando um pipe (símbolo "|") e o comando **grep,** liste todos os alunos inscritos na cadeira de sistemas operativos (grupo sisopr) e na cadeira de Sistemas Distribuídos (grupo sisdst).

Quantos alunos estão inscritos em cada uma dessas cadeiras? Que comando utilizou para obter os resultados? Dica: consulte o manual do grep através de "man grep", de forma a validar se existe alguma opção que permita contar o nº de linhas do resultado.

2.2 Criação de um *pipe* num programa em C.

Utilizando os *system calls* **fork**() e **pipe**(), escreva um programa em C que realize as seguintes funcionalidades:

O processo pai deverá:
4. Criar um *pipe*,
5. Criar um novo processo e imprimir o respectivo PID,
6. Entrar num ciclo de leitura de linhas de input do teclado para um buffer interno
7. Escrever o conteúdo do buffer no descritor de escrita do *pipe*.

O processo filho deverá:
3. Imprimir o seu PID, ler a mensagem

4. Entrar num ciclo em irá ler a partir do descritor de leitura do *pipe*.
5. Imprimir os dados recebidos no *standard output*, com a indicação ">FILHO: <mensagem>"

Para realizar a leitura de linhas do teclado poderá ser utilizada a função **fgets** que permite ler uma linha de texto até receber um ENTER.

3. Projecto a realizar

Desenvolver uma aplicação chamada **mshell** (de mini-shell), que permite executar comandos de terminal. Devem ser suportados os seguintes comandos:

1. Comandos simples: program [parâmetros]
   Exemplo: ls –la

2. Execução de comandos em background: program [parâmetros] &

3. Pipes: program [parâmetros] | program [parâmetros] | ... | program [parâmetros]

4. Redirecionamento do "standard input": program [parâmetros] < abc.txt

5. Redirecionamento do "standard output": program [parâmetros] > abc.txt (cria ou sobrepõe em abc.txt_file) e program [parâmetros] >> abc.txt (cria ou complementa abc.txt).

6. Implementação de variáveis de contexto
   Exemplo: criadas através de comandos como "dir=/usr/include/sys", e depois utilizados em outros comandos através de "ls $dir/*.h". O símbolo '$' deve ser interpretado como um caracter especial que significa o ínicio de uma variável de contexto.

7. Quaisquer combinações, como: cat < abc.txt | fgrep a | sort > efg.txt.

Pressupostos:

1. A aplicação mshell lê comandos do "standard input", podendo obviamente receber comandos de um ficheiro caso seja este redireccionado ("mshell < testes.txt")

2. A mshell deverá aceitar espaços em branco à volta de caracteres especiais (como o '>' de redirecionamento). Como exemplo, "ls>a.txt" e "ls > a.txt" são duas formas correctas da mesma série de comandos.

3. Se o comando fôr executado em foreground, a mshell aguardará que este termine antes de executar o próximo comando. Cado seja executado em background, a mshell aguardará e executará o próximo comando de imediato.

4. Quando um erro for encontrado (por exemplo, ficheiro não existente), deve ser impressa uma mensagem de erro para o standard error, e aguardar o comando seguinte.

5. A aplicação mshell terminará quando encontrar o caracter de fim de ficheiro (Ctrl+D no terminal)

6. Devem ser criados processos filhos utilizando o fork(), gerir indirecções através das funções open() e dup2(), e receber avisos de terminação de processos através de signal() e wait() ou waitpid().

7. Cada processo tem uma lista de variáveis associadas ao seu conexto (a sua tabela de contexto). Esta tabela contém uma lista de strings no formato "Nome = Valor", que é transmitida aos processos filho, pelo que deve ser utilizada, em detrimento de uma lista interna à aplicação mshell, através das funções putenv, getenv, setenv e unsetenv

4. Modelo de Entrega do Trabalho

O trabalho deverá ser realizado obrigatoriamente por **grupos de 2 alunos**, e entregue num prazo de três semanas (**até dia 2/Dez, domingo**). Trabalhos entregues após esta data serão penalizados em dois (2) valores por cada dia de atraso.

O trabalho deve ser enviado para o email so2007.lusofona@gmail.com no seguinte formato:
Subject: a<nº aluno1>/a<nº aluno2>-<dia aula>-<hora-aula>-trab4
(exemplo: Os alunos 20171234 e 20175678, com aula na terça-feira às 14h, enviarão o subject a2017134/a20175678-3-14-trab4)

Anexos:
• Relatório descrevendo as opções tomadas na realização do trabalho (limite de 5 páginas, fonte mínima 10).

• Ficheiros com o código fonte da mini-shell e o Makefile que permite a geração da aplicação mshell.

Critérios de avaliação:
70% - Funcionalidade
20% - Relatório
10% - Organização e legibilidade do código

5. Links e Referências

1 *Programação em C e System Calls*
2 http://www.cs.cf.ac.uk/Dave/C
3
4 *Linux System Calls Table*
5 http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html
6
7 *"The C Programming Language, 2nd Edition (Ansi C)"* de by Brian W. Kernighan, Dennis M. Ritchie, ed. Prentice Hall. http://cm.bell-labs.com/cm/cs/cbook

8 "*The C Book,* second edition, by Mike Banahan, Declan Brady and Mark Doran, ed. Addison Wesley
9 http://publications.gbdirect.co.uk/c_book

Manual de Referência (man pages) Linux on-line: http://www.sourcentral.org/man/SUSE101