Universidade do Minho

Serviço de Execução de Tarefas

Licenciatura em Engenharia Informática

Sistemas Operativos

54739 - Hélder Abreu

54786 - Nuno Morais

Braga - Junho 2010

ÍNDICE

OBJECTIVO E MÉTODOS

O objectivo deste trabalho consistiu na criação dum serviço de execução de tarefas, recorrendo à linguagem de programação **C**. Por serviço de execução de tarefas, entende-se um programa que simule o terminal *Unix*, isto é, que permita o uso de comandos isolados ou encadeados por *pipes*.

Um dos passos fundamentais para a concretização deste trabalho passou pelo recurso a ***system calls****,* de forma a requisitar serviços concretos do sistema operativo, como é exemplo o *open*, *close* ou *read*. Foram utilizados vários ***fork()*** de forma a permitir a execução simultânea de várias linhas de instruções. A criação de processos e a execução dos mesmos em ***background*** foi outro passo importante, que permitiram a introdução sequencial de comandos. É ainda dada a possibilidade do utilizador especificar o ficheiro necessário para a execução do primeiro comando, assim como o ficheiro destino do último comando.

RESULTADOS

Função main:

Um dos passos mais importantes desta função é, sem dúvida, aplicação do ***fork()*** à própria função *main*, criando um processo filho que invocará a função ***run***, dando seguimento ao programa. O processo pai, fica assim disponível para a execução de novos comandos, mesmo que o processo filho não tenha terminado.

Função getCommand:

Esta função cria o *prompt* apresentado na própria *shell,* armazena o histórico de comando inseridos e devolve o comando inserido, numa *string*.

Função getArgs:

Tal como o nome indica, esta função obtém os argumentos necessários à execução dos comandos, a partir da *string* introduzida na *shell*. O primeiro passo consiste na divisão da *string* pela ou pelas barras existentes. Na primeira partição é verificada a existência do sinal “>”. Caso seja encontrado, é esperada a existência de um ficheiro de entrada. O resto desta partição é analizado pela função ***wordexp*** que envia o comando com os respectivos argumentos para uma lista de *strings*. As partições seguintes são também tratadas pelo wordexp. Na última partição também é verificada a existência de um ficheiro de saída.

Função run:

Inicialmente cria 2 *arrays* de inteiros com 2 posições (*lpipe* e *rpipe*) onde serão guardados os descritores. Esses *arrays*, são usados para definir posteriormente os *inputs*/*outputs* dos comandos inseridos. Esta função verifica também se o utilizador inseriu um ficheiro *in* e/ou *out*, para serem usados como *input*/*output* do primeiro/último comando. Neste função, é ainda realizado um ciclo com um ***fork_and_pipe*** processa sequencialmente os comandos introduzidos.

Função fork_and_pipe:

Em primeiro lugar faz um ***fork()*** de forma a criar um processo filho da função **run**. Usando este processo, verifica se o *lpipe* e *rpipe* não estão vazios. Caso não estejam, são efectuadas as seguintes alterações no *lpipe* e *rpipe*. No *lpipe*, é fechado o *system output*

e o descritor do *input* do comando actual passa a ser igual ao descritor do *output* do comando anterior. No *rpipe*, é fechado o *system input* e o descritor do *output* actual passa a ser igual ao descritor do comando a ser executado. Feito isto, o comando introduzido, é executado.

CONCLUSÃO

A implementação desta *shell* com algumas das características do terminal UNIX fez-nos entender um pouco mais sobre a forma como são criados e mobilizados os processos dentro do sistema operativo. A interpretação de sequências de comandos em pipelines permitiu-nos perceber como é feita a sua execução sequencial numa única pipeline, assim como o multiprocessamento de instruções (execução de vários comandos em diferentes pipelines).

Em suma, consideramos que foi uma mais-valia no sentido em que nos permitiu aplicar grande parte da matéria abrangida pela unidade curricular, vendo a utilidade da mesma num projecto de maior dimensão, ao invés de exercícios isolados.