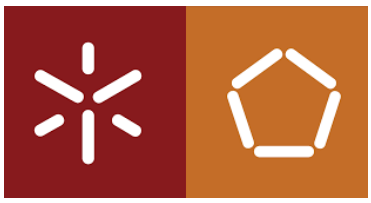


Universidade do Minho
Licenciatura Engenharia e Gestão de Sistemas de Informação
Sistemas Operativos | 2023/2024

Licenciatura em Engenharia e Gestão de Sistemas de Informação

Sistemas Operativos

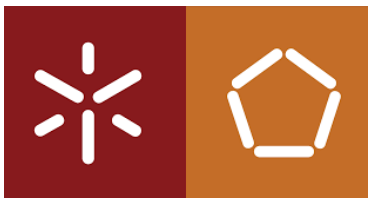
2023/2024

Trabalho Prático

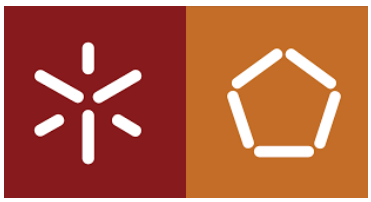
A105109-José Maria Martinho Mendes Godinho

A105557-Martim Pereira Ribeiro

A103985-Rodrigo Manuel da Silva Cunha



Índice



Universidade do Minho
Licenciatura Engenharia e Gestão de Sistemas de Informação
Sistemas Operativos | 2023/2024

Introdução

No âmbito da Unidade Curricular Sistemas Operativos foi-nos proposto desenvolver um projeto onde desenvolvemos um comando/programa que nos permite obter informação sobre os processos ativos no sistema e a execução de comandos Linux com redireccionamento de entrada e saída. Através do desenvolvimento deste projeto pretendemos melhorar e mostrar as competências que fomos adquirindo durante as aulas práticas da Unidade Curricular. Neste relatório iremos mostrar uma visão abrangente do desenvolvimento deste programa, destacando os desafios encontrados, as soluções que adotamos e as lições aprendidas ao longo deste processo.

Objetivos do Trabalho

Este Projeto tem como objetivo a implementação de cinco comandos diferentes através do desenvolvimento de um programa escrito em linguagem 'C' (mycmd). Cada comando terá o seu propósito. Aqui estão os objetivos de cada comando:

Comando 1- Obter Informações sobre processos

- Implementar o comando 'mycmd top' de modo a que seja exibido no ecrã informações como a carga do CPU, um determinado número de processos e detalhes destes mesmo, atualizando esta informação a cada 10 segundos.

Comando 2- Execução de comandos LINUX

- Implementar o comando 'mycmd cmd arg1 … argn' podendo um exemplo ser 'mycmd ls -l'. Neste caso o comando executa um comando LINUX 'ls -l'.

Comando 3- Redireccionamento de saída para ficheiro

- Implementar o comando 'mycmd cmd arg1 … argn ">" file' podendo um exemplo ser 'mycmd ls -l ">" output.txt'. Neste caso o comando irá executar o comando LINUX 'ls -l' e vai redirecionar a sua saída para o ficheiro 'output.txt'.

Comando 4- Redireccionamento de entrada a partir de ficheiro

- Implementar o comando 'mycmd cmd arg1 … argn "<" file' podendo um exemplo ser 'mycmd grep TEXTO "<" input.txt'. Neste caso o comando irá executar o comando LINUX 'grep TEXTO' e vai redirecionar a sua entrada para o ficheiro 'input.txt'.

Comando 5- Redireccionamento da saída de um comando para a entrada de outro

- Implementar o comando 'mycmd cmd1 arg1 … argn "|" cmd2 arg1 … argn' podendo um exemplo ser 'mycmd ps -A "|" grep 12345'. Neste caso os dois comandos serão executados em sequência, redirecionado a saída do primeiro (´ps -A') para a entrada do segundo ('grep 12345').

Durante a implementação destes comandos tivemos atenção ao caso dos comandos 3, 4 e 5, pois caracteres >, < e | devem estar entre aspas, caso contrário, a bash irá assumir o redireccionamento, isto é, os comandos provavelmente vão estar a desempenhar a sua função, mas não da forma correta e pedida no enunciado.

Principais Questões a Considerar e Fundamentos teóricos/tecnológicos subjacentes

Principais Questões a Considerar:

Gestão de Processos e Sistemas de Ficheiros:

- Como obter informações sobre processos ativos usando o sistema de ficheiros /proc?
- Como manipular diretórios e ficheiros no /proc usando chamadas de sistema?

Execução de Comandos em Linux:

- Como executar comandos Linux a partir de um programa em C?
- Como lidar com argumentos passados pela linha de comandos?

Redireccionamento de Saída/Entrada:

- Como implementar o redireccionamento de saída (>) e entrada (<) usando chamadas ao sistema com a função dup2?

Pipes e Comunicação entre Processos:

- Como criar e utilizar pipes para redirecionar a saída de um processo para a entrada de outro (|)?

Temporização e Atualização Periódica:

- Como implementar a temporização para atualizar as informações dos processos a cada 10 segundos no comando mycmd top?
- Como lidar com a entrada do utilizador para interromper o processo?

Fundamentos Teóricos/Tecnológicos Subjacentes:

Chamadas ao Sistema:

- Compreensão das chamadas ao sistema relacionadas à manipulação de processos, ficheiros e pipes em sistemas Linux.

Sistema de Ficheiros /proc:

- Familiaridade com a estrutura do sistema de ficheiros /proc e sua utilidade para obter informações sobre processos.

Pipes Anónimos e dup2:

- Conhecimento profundo sobre o uso de pipes para comunicação entre processos.
- Entendimento da chamada ao sistema da função dup2 para redireccionamento de entrada/saída.

Gestão de Processos em C:

- Habilidades para manipular processos em C, incluindo a execução de novos processos e a obtenção de informações sobre eles.

Temporização em Programas C:

- Conhecimento sobre como implementar temporização em programas C, possivelmente usando funções relacionadas ao tempo.

Descrição da implementação dos comandos

Comando 1 (mycmd top)

Para a implementação do primeiro comando, desenvolvemos duas funções: printTopInfo () e executeCommandTop().

- printTopInfo ()

  - Nesta função desenvolvemos a interface do comando 'top' de acordo com o que é pedido no enunciado. Para tal, vamos utilizar a diretoria '/proc' onde esta montado o sistema de ficheiros proc que disponibiliza as informações necessárias recorrendo à leitura dos ficheiros.
  Para obtermos as informações relativas à carga média do CPU iremos abrir o ficheiro /proc/loadavg no modo leitura de modo a retirarmos a informação relativa à carga média nos últimos 1 min, 5 min e 15min.

  -Através da criação de uma estrutura (struct dirent *entry) que guarda informações relativas ao diretório, bem como o nome de diretórios ou ficheiros presentes nesta, e da implementação de um ciclo 'while' percorremos todos os diretórios de processos e verificamos se estes são processos através do seu nome/PID e fazemos assim a contagem do total de processos. Após esta verificação criamos o caminho para o ficheiro 'status' de cada processo. Após a verificação se este ficheiro realmente existe retiramos a informação relativamente ao estado do processo do ficheiro '/proc/[PID]/status' e se este for um processo que esteja 'running' incrementamos à variável 'running_processes'. No final do ciclo 'while' apresentamos o total de processos e os que se encontram em execução.

  -Voltamos a recorrer aos mesmos métodos referidos acima para abrir o ficheiro '/proc/[PID]/status' e através do estado de cada processo retiramos a informação relativa ao User ID (UID) de cada processo. Depois convertemos o UID para obter o nome de usuário de cada processo. Definimos como 'unknow' os nomes de usuários em que 'pwd é NULL'.

  - Ainda no ciclo ´while´ do ponto acima, criamos o caminho para o ficheiro 'cmdline' de cada processo. Após a verificação da existência deste ficheiro obtemos a informação de cada processo relativa a estado, *username* e a linha de comando. O ciclo 'while' acaba quando acabarem os diretorios de processos ou quando a soma de os processos em execução mais os restantos for maior que 20, desta forma só aparecem 20 processos no ecrã quando executamos o comando 'mycmd top'.

- executeCommandTop()

  - Nesta função criamos um ciclo que começa por limpar o ecrã, executa de seguida a função anterior printTopInfo (), depois aguarda 10s e depois apresenta uma mensagem para se o utilizador quiser terminar a apresentação das informações do comando e reinicia o ciclo. Ou seja, será apresentado as informações do comando top relativa aos processos aparecendo sendo só 20 processos e atualizará esta informação a cada 10s podendo este comando ser interrompido pelo utilizador quando este quiser.

Estas funções foram implementadas com as devidas verificações assim como as mensagens de erro necessárias para o entendimento do utilizador se acontecer algum erro.

Comando 2 (mycmd cmd arg1 … argn)

Para a implementação do segundo comando apenas desenvolvemos uma função executeCommand(char *cmd, char *args[]).

- executeCommand(char *cmd, char *args[])

  - Nesta função recorremos ao uso de processos pai e filho através de fork(). Criamos então um processo pai e filho onde no processo filho é executado o comando recebido como argumento assim como o respetivo número de argumentos através da função ‘execvp(cmd, args)’. O processo pai espera sempre que o processo filho seja executado e só depois é executado(‘waitpid()’).Se o ‘execvp’ falhar aparecerá uma mensagem de erro no ecrã.

Comando 3 (mycmd cmd arg1 … argn “>” file’)

Para a implementação do terceiro comando apenas desenvolvemos uma função executeCommandWithOutputRedirection(char *cmd, char **args, char *outputFile).

- executeCommandWithOutputRedirection(char *cmd, char **args, char *outputFile)

  - Nesta função voltamos a recorrer ao uso de um fork() para criarmos um processo pai e outro filho. No processo filho abrimos o ficheiro que foi passado como argumento para a função no modo de leitura. De seguida, utilizamos a ferramenta dup2 para redirecionar a saída para o ficheiro, e depois executamos o comando que foi recebido como argumento assim como o número total de argumentos que este comando tem. Para executar o comando voltamos a recorrer à função ‘execvp (cmd, args)’. Na abertura do ficheiro, no redireccionamento e na função execvp foram implementadas mensagens de erro, caso ocorra alguma falha. Após a processo filho acabar, o processo pai inicia já que utilizamos a função waitpid(), e fazemos a devida verificação para saber se o processo filho ocorreu sem problemas, caso contrário é mostrada uma mensagem de erro, igualmente para se existir problemas no fork().

Comando 4 (mycmd cmd arg1 … argn “<” file’)

Para a implementação do quarto comando apenas desenvolvemos uma função executeCommandWithInputRedirection(char *cmd, char **args, char *inputFile).

- executeCommandWithInputRedirection(char *cmd, char **args, char *inputFile)

  - Nesta função utilizamos exatamente o mesmo código que a função do comando 3, com a pequena alteração para o uso da ferramenta dup2(), já que pretendemos redirecionar a entrada do ficheiro, usamos um ‘STDIN’ invés de um ‘STDOUT’.
  Fazemos sempre a devida verificação de erros, e é sempre uma mensagem no ecrã do utilizador caso alguma falha aconteça.

Comando 5 (mycmd cmd1 arg1 … argn "|" cmd2 arg1 … argn')

Para a implementação do quinto comando apenas desenvolvemos uma função executeCommandWithPiping(char *cmd1, char **args1, char *cmd2, char **args2).

- executeCommandWithPiping(char *cmd1, char **args1, char *cmd2, char **args2)

  - A função começa por criar um pipe usando a chamada do sistema pipe. O pipe tem dois extremos: um para leitura (pipefd[0]) e outro para escrita (pipefd[1]). A função realiza um fork para criar um processo filho para o primeiro comando (cmd1).O processo filho fecha a extremidade de leitura do pipe (pipefd[0]) porque ele só irá escrever no pipe. De seguida, redireciona 'stdout' para o 'write' no fim do pipe, e a seguir fecha a extremidade de escrita do pipe (pipefd[1]). Após estes procedimentos o primeiro comando passado como argumento para a função será executado usando a função 'execvp' como nos comandos anteriores. A função realiza outro fork para criar um segundo processo filho para o segundo comando (cmd2). O processo filho fecha a extremidade de escrita do pipe (pipefd[1]) porque ele só irá ler do pipe. De seguida, redireciona 'stdin' para o 'read' no fim do pipe, e a seguir fecha a extremidade de leitura do pipe (pipefd[0]). Após estes procedimentos o segundo comando passado como argumento para a função será executado usando a função 'execvp' como nos comandos anteriores. No processo pai são fechadas ambas as extremidades do pipe, pois ele não usará diretamente o pipe. O processo pai espera pelos dois processos filhos completarem a execução e acaba a função. A função contém as devidas verificações e mensagens de erro caso alguma falha ocorra.

Universidade do Minho
Licenciatura Engenharia e Gestão de Sistemas de Informação
Sistemas Operativos | 2023/2024

Manual dos comandos

1. Comando top:

- Este comando exibe informações sobre a carga média do CPU, o total de processos e a lista detalhada dos 20 principais processos em execução ou em outros estados.
- O usuário pode pressionar 'q' para sair do comando top.
- Exemplo de utilização:./mycmd top

2. Comando Padrão (Sem Redirecionamento ou Pipe):

- Qualquer comando que não tenha operadores de redirecionamento ou pipe será executado pela função executeCommand().
- Exemplo de utilização: ./mycmd ls -l

3. Redirecionamento de Entrada (<):

- Permite redirecionar a entrada de um comando a partir de um arquivo.
- Exemplo de utilização: ./mycmd grep TEXTO “<” arquivo.txt

4. Redirecionamento de Saída (>):

- Permite redirecionar a saída de um comando para um arquivo.
- Exemplo de utilização: ./mycmd ls -l “>” lista_arquivos.txt

5. Pipe (|):

- Permite encadear a saída de um comando como entrada para outro comando.
- Exemplo de utilização: ./mycmd ps -A “|” grep 12345

Conclusão

O desenvolvimento do programa "mycmd" representou um grande desafio para compreender melhor e aplicar as competências adquiridas durante as aulas de Sistemas Operativos. Tentamos desenvolver um programa robusto e de fácil entendimento por parte do utilizador através de constantes verificações, mensagens de erro e através da utilização dos anexos e de informações disponibilizadas pelos docentes como o /proc por exemplo. Em resumo, o trabalho não apenas ampliou nossos conhecimentos técnicos, mas também fortaleceu competências essenciais para o desenvolvimento de software em sistemas operativos, destacando a importância da colaboração e do entendimento prático desses conceitos.