Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Sistemas Operativos

Trabalho prático 2

Realizado por:

António Ramos: 23906

Gonçalo Abreu: 22996

Leandro Dias: 23028

Viseu, 2022

Índice

Introdução

O presente relatório é relativo ao trabalho prático 2 da Unidade Curricular de Sistemas Operativos.
Neste será desenvolvido um código para uma aplicação multiprocessos que permita a criação de hierarquias de processos.

Função main()

Inicialmente, criamos a função "**main()**" que contem o menu que dará acesso às 4 funções acessórias: "**void** criarFicheiro()", "**void** mostrarValores()", "**void** eliminarFicheiros()", "**void** terminar()".

Para tal, usou-se a estrutura *switch…case* para se concretizarem as opções (menu da aplicação).

Opção 1- Criar Ficheiros

Para esta opção, iniciamos por criar o processo pai e filho 1.1, como descrito na hierarquia do enunciado, usando a função "**fork()**".

Usou-se de novo a função "**fork()**" para criar um processo filho do 1.1, o processo 1.1.1.

Recorreu-se à estrutura if… else para se chegar ao processo filho 1.1.1., fazendo "**if (pid1_1 == 0)**" em que, se o pid1_1 for 0, significa que se está no processo filho do 1.1.
Neste, recorrendo à função "**gravarFicheiro(getppid(),getpid())**" escrevemos no ficheiro criado o PID do processo pai "**(getppid())**" e o PID do processo filho "**(getpid())**".
Após um "**else**", usou-se a função "**wait()**" de maneira que se esperasse que o processo filho 1.1.1. terminasse.

De seguida, criou-se um processo filho 1.1.2.

Usando a mesma estratégia para o processo filho 1.1.1, escreveu-se no ficheiro criado o PID do processo pai "**(getppid())**" e o PID do processo filho "**(getpid())**".

O mesmo se fez para o processo filho 1.2.

Função "*void gravarFicheiro(pid_t pidPai, pid_t pidFilho)*":

Esta função foi criada para se conseguir criar o ficheiro em cada processo e escrever neles.

Os argumentos são do tipo pid_t, declarados na função "**criarFicheiro()**".

Nesta função é criado um ficheiro com o nome "**PID.pso**" recorrendo à função "**sprintf(str, "%d.pso",pidFilho)**".

É de seguida aberto o ficheiro, com o intuito de se escrever nele, como descrito na linha "**f=fopen(str, "w")**".

Nas funções "**fprintf()**" são escritos no ficheiro o pid do processo pai, o pid do processo filho e ainda "**P2G1**" (correspondente à turma prática 2, grupo 1), por esta ordem.

Opção 2- Mostrar Valores

Nesta função, deu-se uso às funções incluídas na biblioteca "**<dirent.h>**".

O objetivo era percorrer a diretoria para obter o nome dos ficheiros e apresentar sob a forma "Ficheiro: nome XX nº de bytes".

Inicialmente declararam-se as variáveis e estruturas necessárias.

De seguida, usou-se uma estrutura do…while, para que o programa percorresse todos os diretórios existentes até não haver diretório encontrado, como se declarou "**while (dp != NULL)**", ou seja, enquanto dp não devolvesse NULL, voltava-se a correr os diretórios.

Dentro da estrutura acabada de apresentar, usou-se uma outra estrutura if…else, que vai avaliar se o diretório atual apresenta algum ficheiro criado. Caso contrário apresenta a frase "**Não existem ficheiros .pso na pasta**".

Usou-se funções como "**readdir()**" para permitir ler o diretório atual e avaliar se este se encontra criado ou não e, caso exista, é lido e de seguida devolvido a frase acima apresentada na linha "**printf("Ficheiro: %s %ld bits\n",dp->d_name,f.st_size)**".

Opção 3- Eliminar Ficheiros

Nesta opção, o objetivo é eliminar os ficheiros criados na opção 1.

Nesta função, a estrutura é idêntica à do menu 2, porém apresenta a função "**remove(dp->d_name)**".
O conceito da programação deste menu é o mesmo que o da opção 2, ou seja, esta estrutura vai permitir percorrer os diretórios todos existentes, avaliar se em cada um deles existe algum ficheiro criado e, em caso afirmativo, este será removido pela função "**remove()**".

Opção 4- Terminar

O objetivo desta opção será terminar o programa.

Nesta opção, criou-se um processo. No processo filho "**if (pid == 0)**", envia-se um sinal ao processo pai.
No processo pai, fez-se o tratamento do sinal **SIGINT** "**(kill(pid, SIGINT))**" e esperou-se o processo filho terminar "**(pid = wait(&estado))**" e, de seguida, termina-se o programa com "**kill(pid, SIGKILL)**".

Conclusão

Ao longo deste trabalho houve uma certa dificuldade em interpretar o que se pretendia na opção 4 terminar. Da maneira como foi interpretada pelos membros do grupo, houve alguma dificuldade em usar os sinais da comunicação entre processos, tanto na questão da estrutura, como na questão de entender o conceito na prática.