Sistemas Operativos

Relatório do Projeto MP1

André Lino dos Santos (up201907879)
João André Silva Roleira Marinho (up201905952)
Miguel Boaventura Rodrigues (up201906042)
Tiago Caldas da Silva (up201906045)

2020/2021

Conteúdo

1 Introdução

No âmbito da cadeira de Sistemas Operativos, desenvolvemos uma versão adaptada do comando: **chmod**. Este comando, **xmod**, oferece-nos a possibilidade de alterarmos as permissões de determinados diretórios/ficheiros/*symbolic links*.

Deste modo, aquando da sua chamada temos a possiblidade de decidir quais as permissões que devem ser dadas, adicionadas ou removidas ao conteúdo sob o qual estamos a atuar, tendo ainda a possiblidade de decidir sobre que grupo queremos aplicar este comando (*user*, *group*, *other*, *all* (*default*)).

2 Ficheiros/Módulos

2.1 xmod

O ficheiro ***xmod.c*** é o ficheiro principal do programa. Neste ficheiro está contida a função **main()** que recebe os argumentos passados pela *shell* ou por uma chamada de sistema **exec()**.

No ínicio da execução, o programa verifica e prepara uma série de aspetos antes da chamada ao **chmod()** onde efetivamente as premissões irão ser alteradas. Mais abaixo serão explicadas com mais detalhe todas essas preparações.

Com tudo pronto, será tempo de seguir um de vários caminho de execução possíveis. O mais simples será se o *input* se tratar de um ficheiro regular, onde, independentemente do modo recursivo estar ativo ou não, basta alterar as suas permissões e terminar com o programa. Todavia, se o *input* for um diretório haverá uma panóplia de possíveis direções de execução. A primeira, mais simples - sem recursão - basta que o próprio diretório seja alterado e de seguida terminar a execução. A segunda, bastante mais complexa, é executar a primeira e ainda iterar sobre o elementos presentes nele mesmo. Quando encontrado um sub-diretório deveremos colonar o processo atual, pela chamada à função ***fork()*** e no processo recém criado, processar de forma recursiva esse diretório e os respetivos conteúdos lá contidos.[1]

2.2 Mode

É nesta componente do projeto que é feito o tratamento do modo. O utilizador pode introduzir as permissões requeridas de 2 formas diferentes. A primeira trata-se da escrita das permissões no modo octal (4 dígitos), atríbuindo integralmente ao ficheiro as novas permissões. O principal problema será apenas testar a validade do número.

A segunda acaba por ser mais complexa pois consiste em poder adicionar, remover, ou atribuir um conjunto de permissões específicas, seja de leitura (r), escrita (w), execução (x) ou de qualquer combinação das três (funciona para o *user* (u), *group* (g), *others* (o) e em simultâneo (a)). Neste caso, para além de ser necessário testar se a string é válida e adaptá-la para o modo octal, é preciso ler o estado atual das permissões do ficheiro em questão, pois uma soma, por exemplo, pode ter efeito aditivo (permissão por atribuir) ou neutro (permissão já atríbuida).

2.3 Logs

Neste módulo encontramos todas as funções relacionadas com os registos que o programa deve tratar se a variável de ambiente - "LOG_FILENAME" - estiver previamente definida, pelo utilizador. Os registos estão em conformidade com aquilo apresentado no enunciado.

Em cada linha temos uma coluna com o instante de occorrência de um dado evento em relação a um instante arbitrário de tempo - o ínicio da execução do processo líder, processo gerado pela *shell*. Na segunda coluna, o ID do processo responsável pelo evento. Depois, uma breve descrição do evento em conjunto com um conjunto de outras informações; sempre seguindo os requisitos arquiteturais expressos no enunciado do projeto.

[1] No caso recursivo, à semelhança do que acontece no **chmod**, o **xmod** não altera os ficheiros apontados por *symbolic links*.

2.4 Option

Responsável pelo processamento do input das diferentes flags, estas que podem representar modo recursivo (R), modo verboso (v) ou modo de change (c). O programa faz a distinção da flag "v" e "c", conforme a sua ordem na lista de argumentos, ou seja, caso o modo "v" seja inserido após a opção "c", a informação será apresentada consoante a última flag inserida, neste exemplo "v" - à semelhança daquilo que o **chmod** faz. Além disso, é apresentada na *shell* a informação do ficheiro/diretório com as permissões anteriores e com novas permissões, conforme as opções passadas pelos argumentos.

2.5 Signal

O módulo signal presente no ficheiro ***signal.c*** contém todas as funções que tratam sinais que os processos em execução possam receber, é neste ficheiro que estão declarados os *signal handlers*.

Relativamente aos *signal handlers* tomamos especial atenção às chamadas a funções que não são *signal safe*, ou seja, funções que pelas as suas caraterísticas podem apresentar inconsistências perante eventos assíncronos. A maioria das funções *buffered/IO* da biblioteca padrão do C não respeitam essa caraterística. Não obstante, com as restrições impostas no que toca à apresentação dos registos, vimo-nos forçados a usar a função **sprintf()** que não se encontra listada em *man signal-safety(7)*.

É importante referir que a gestão de processos e amostragem da informação quando é premido *CTRL-C* ficou atribuída ao sinal *SIGUSR1*.

Ainda apontar que, no ínicio da execução do programa é executada a função **setup_signals()**, que como o nome já deixa antever atribuirá a cada tipo de sinal o respetivo *handler*.

2.6 File

O módulo File, composto pelo ficheiro ***file.c*** é provavelmente o mais simples. É aqui que occorrem uma série de verificações como garantir que o processo em execução tem acesso e/ou permissões para efetuar as operações necessárias no ficheiro/diretório passado como argumento.

Destaque ainda para a função **process_node()** que combina, a cada iteração sobre o diretório passado como argumento do programa, o nome do próprio com o nome ficheiro/diretório nele contido. No ficheiro é possível entender melhor o propósito desta função pela sua documentação.

3 Conclusão

Para terminar, todos os elementos do grupo enfrentaram os mais diversos de desafios, seja pela busca das "soluções" nas páginas do manual ou por umas boas horas em frente ao ecrã em sessões de *debugging*, é unânime que este tipo de mini-projetos é algo muito positivo e enriquecedor. Outro aspeto que o grupo aponta como algo benéfico são os desafios que mini-projetos como este implicam. Desta forma consideramos que cada elemento contribuiu em 25.0% para este.

Talk is cheap. Show me the code.
Linus Trovalds.

4 Apêndice

```
instant ; pid ; event ; info
6.85 ; 31956 ; PROC_CREAT ; -Rv 0775 folder-0
7.01 ; 31956 ; FILE_MODF ; folder-0 : 0775 : 0775
7.47 ; 31957 ; PROC_CREAT ; -vR 0775 folder-0/folder-1
8.49 ; 31957 ; PROC_CREAT ; -vR 0775 folder-0/folder-1
8.61 ; 31957 ; FILE_MODF ; folder-0/folder-1 : 0775 : 0775
8.78 ; 31957 ; FILE_MODF ; folder-0/folder-1/file-1 : 0664 : 0775
8.88 ; 31957 ; SIGNAL_SENT ; SIGCHLD : 31956
8.93 ; 31957 ; PROC_EXIT ; 0
9.18 ; 31956 ; SIGNAL_RECV ; SIGCHLD
9.52 ; 31958 ; PROC_CREAT ; -vR 0775 folder-0/folder-2
10.56 ; 31958 ; PROC_CREAT ; -vR 0775 folder-0/folder-2
10.68 ; 31958 ; FILE_MODF ; folder-0/folder-2 : 0775 : 0775
10.80 ; 31958 ; FILE_MODF ; folder-0/folder-2/file-1 : 0664 : 0775
10.87 ; 31958 ; SIGNAL_SENT ; SIGCHLD : 31956
10.91 ; 31958 ; PROC_EXIT ; 0
11.16 ; 31956 ; SIGNAL_RECV ; SIGCHLD
11.51 ; 31959 ; PROC_CREAT ; -vR 0775 folder-0/folder-3
12.50 ; 31959 ; PROC_CREAT ; -vR 0775 folder-0/folder-3
12.66 ; 31959 ; FILE_MODF ; folder-0/folder-3 : 0775 : 0775
12.81 ; 31959 ; FILE_MODF ; folder-0/folder-3/file-1 : 0664 : 0775
12.87 ; 31959 ; SIGNAL_SENT ; SIGCHLD : 31956
12.90 ; 31959 ; PROC_EXIT ; 0
13.14 ; 31956 ; SIGNAL_RECV ; SIGCHLD
13.24 ; 31956 ; FILE_MODF ; folder-0/file-1 : 0664 : 0775
13.33 ; 31956 ; SIGNAL_SENT ; SIGCHLD : 5436
13.37 ; 31956 ; PROC_EXIT ; 0
```

Exemplo simbólico do estado de um ficheiro definido pela variável de ambiente "LOG_FILENAME" após a execução do programa xmod.