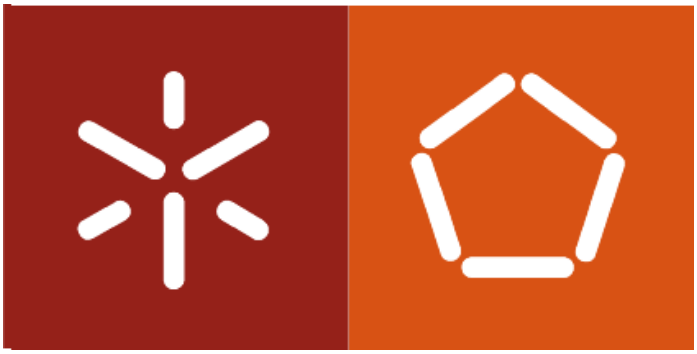

RELATÓRIO DE SO

SO - 2º Semestre 2016

Alexandre Teixeira
(A73547)

Bruno Sousa
(A74330)

Rafael Silva
(A74264)

SO – SOBUSRV

Índice

Resumo

Este trabalho tem como objetivo principal, demonstrar os conhecimentos sobre o tratamento de chamadas ao sistema e comunicação entre processos, isto é, a partir de um programa *cliente* enviar comandos para um servidor, onde este os processa.

Este *programa* só é capaz, como referido no enunciado, de executar no máximo cinco operações em simultâneo, de forma a não sobrecarregar o sistema. Também como referido no enunciado, é possível a execução em simultâneo do *programa*, ou seja, que pode ter vários clientes ligados ao *servidor* desde que estes respeitem a regra acima referida.

1- Estrutura do *software*

Para a resolução do projeto, o grupo considerou que seria melhor a criação de duas aplicações afim de criar o programa pretendido. A primeira consiste na execução de comandos e as respetivas funções pedidas no enunciado e a segunda corresponde ao "contacto" entre os comandos e a execução.

A aplicação "sobucli"(cliente) envia comandos para a aplicação "sobusrv"(servidor) através de um pipe com os dados de utilizador e os respetivos comandos que pretende executar. A aplicação "sobusrv"(servidor) responde enviando sinais para o processo "sobucli"..

2- "sobusrv" (servidor)

O programa "sobusrv" executa as instruções dadas pelo cliente (sobucli) e responde ao cliente através de sinais.

Este é responsável pela execução de todas a operações dadas no argumento do sobucli, sendo tais operações as seguintes: backup e restore.

O programa executa os comando dados a partir de comandos UNIX, tais como: mkrdir, mv, ln, gzip, shasum e rm. Para a execução dos comandos UNIX foi necessária a implementação das funções myBash(), mySystem(), trim() e readln (dadas nas aulas praticas).

2.1- *"mysystem"(Aula)*

Esta função serve para executar várias tarefas de acesso e gravação de dados. Permite a comunicação entre o servidor, a correr em *background*, e o utilizador, onde esta função irá realizar seis operações distintas:

A primeira função consiste num comando dado a função através de um dado ficheiro input, permitindo assim a possibilidade de fornecer dados ao servidor através de ficheiros input tanto como comandos como dados de acesso ao servidor; a segunda operação permite fornecer um dado comando ao servidor e este registar o mesmo comando dado num ficheiro output para futura visualização de comandos que um determinado utilizador executou; a terceira operação tem semelhanças com a primeira, no entanto existe uma diferença, ao contrário de receber um comando esta irá verificar se determinado comando existe num ficheiro input ao programa;

a quarta operação grava os comandos que pertencem ao "standard error" num ficheiro output; a quinta operação é semelhante com a quarta, contudo existe uma pequena diferença quando ocorre um "standard error" e este é gravado num ficheiro output os carateres que diferenciam as cinco operações distintas são: para a primeira temos "<" , para a segunda temos ">", para a terceira temos ">>", para a quarta temos "2>" e por final para a quinta temos "2>>", por final, a sexta operação consiste quando não é fornecido nenhum argumento ou caracter ao sistema este faz *exit*, apos ter feito essa operação a função ira fazer *waitpid* apos isso ela ira retornar o estado do *system*, se consegui abrir um certo executável e executar as operações nele, as operações consistem em fazer *backup* ou *restore* dum certo ficheiro, para poderem ser executadas as operações acima referidas terá primeiro de se executar o *servidor* apos este ser executado para podermos aceder as operações executadas por ele termos de fazer *sobucli (backup ou restore)* e o(s) ficheiro(s) que se pretende efetuar essas operações, devolvendo o servidor como referido acima se a operação foi executado com sucesso ou não. Criamos esta função com a ajuda da ficha 4 das aulas praticas.

2.2- *"mybash"(Aula)*

No processo do servidor, um dos processos principais consiste em poder executar comandos da *bash* para se poder ligar com o servidor, para poderem ser executadas as tarefas propostas no enunciado que são, *backup* e *restore*, os comandos são enviados no formato referido na secção anterior.

Para executar um comando da *bash*, recorre-se á função definida acima escrita, através do ficha 5 das aulas praticas, podemos retirar umas ideias de como iriamos construir a nossa própria *bash.*

A função *mybash* cria um *array* de comandos, esta separa a *string* recebida como argumento da função cada vez que encontra um *pipe* ('|'). Após esta ler os comandos inseridos no *array*, esta cria um processo para executar cada um dos comandos, fazendo uma serie de encadeamentos com os seus inputs e outputs. Cada comando é executado individualmente esperando assim sempre que o comando que esta a ser executado termine de ser executado, fazendo um *waitpid*, ou seja, que esta espera pelo estado do comando que esta a ser executado, caso o comando dado seja invalido ou incorreto esta irá devolver uma *string* a dizer "*Command not found or incorrect arguments",* acabado de executar os comandos pedidos a *"mybash"* esta irá fazer *exit* da mesma.

Como foi referido acima esta função executa cada comando individualmente, parra esse efeito recorre-se a função "*mysystem",* do ficha 4 das aulas praticas.

Caso esta encontre um *token* ('<','>','2>','2>>') de redireccionamento, este são aqueles caracteres referidos na descrição da aplicação *"mysystem"*, não insere no *array*, fazendo a apresentação do respetivo input/output.

A função usa uma função auxiliar chamada *"trim",* que tem como finalidade apagar os espaços no inico e no final duma *string,* para que mesmo quando escrito um comando com espaços incluídos antes e depois deste a função consiga executar o comando sem qualquer problema.

2.3- *"execute"*

A função execute() é responsável por executar das tarefas propostas.

Esta recebe os argumentos e compara o primeiro argumento com os nomes das funções propostas, sabendo assim qual função deve executar.

2.3.1- *"backup"*

Função responsável pelo backup dos dados, para a pasta /home/user/.Backup. Esta trata os ficheiros, utilizado gzip e shasum, isto é, compacta o ficheiro e altera o seu nome para o digest do mesmo.

Para a gravação do backup, tivemos de pegar na diretoria onde se encontra o ficheiro e fazer um strtok() até ficarmos apenas com o nome do utilizador. Além disso, tivemos de criar a pasta /.Backup e as subpastas /data e /metadata. Na data colocamos o ficheiro compactado e com o nome digest, onde, em simultâneo, fazemos um link na /metadata com o nome original do ficheiro, mas, ligado ao seu digest na /data.

Desta maneira conseguimos garantir que o backup dos ficheiros, é executado corretamente e de maneira eficiente.

2.3.2- *"restore"*

Função responsável pela recuperação dos dados guardados na pasta do backup. Esta função trata os ficheiros, utilizando o gzip e shasum, desta vez de maneira inversa ao backup. Primeiro executamos a função readlink, na /metadata, para saber qual o ficheiro correspondente na /data. De seguida, fazemos gzip –d (desecompacta) para a pasta de origem, desta forma não necessitamos de mover o ficheiro descompactado.

2.4- *"strrev"*

A função "strrev" inverte a ordem de uma string, esta utiliza um inteiro com o tamanho da string e copia, por ordem inversa, todos os caracteres desta para uma auxiliar.

2.5- *"get_pid"*

A função "get_pid" devolve o pid do processo executado no cliente (sobucli). Este é inserido no ultimo elemento da string devolvida pelo cliente, logo a função inverte a string (strrev) e remove todos os elementos depois do primeiro espaço (strtok(NULL," ")), ficando assim com o pid invertido. De seguida, volta a inverter a string (strrev) para ficar com o pid na ordem correta.

2.6- *"rem_pid"*

Esta função remove o pid do processo executado no cliente (sobucli). Este é inserido no ultimo elemento da string devolvida pelo cliente, assim a função remove o ultimo elemento invertendo a string (strrev) e tirando o pid (strrev(," ")). Seguidamente, inverte o resto da string (strrev) ficando assim com os comando necessários.

2.7- *"main"*

Na main, é iniciado um ficheiro pipe de modo a que seja feita leitura em FIFO (mkfifo), a partir deste é feita a leitura dos comandos inserido no cliente (readln). De seguida, criamos um processo filho (fork()) que vai executar os comandos dados pelo cliente(execute), onde estes estão limitados a 5 processos(nPid<=5), isto é, se existir mais de 5 processo, estes são obrigados a espera(wait) a conclusão dos anteriores para iniciarem. Seguidamente, o processo termina enviando um sinal ao cliente(kill(pid,SIGRTMAX)). Finalmente, após a inserção de CTRL+C, o fecheiro fecha(close) e termina o programa.

3- *"sobucli"(interface de utilização)*

O programa "sobucli" envia comandos para o "sobusrv" executar, onde este responde através de sinais.

Pela analise de sinais recebidos a partir do servidor "sobusrv", onde a analise destes é responsável pela impressão das seguintes funções: backup, restore, delete e gc.

3.1- *"signalhander"*

Esta função, após receber o sinal do servidor, compara a operação com os diferentes comandos possíveis para executar, onde se responsabiliza pela correta impressão da informação. No entanto, se o sinal devolvido pelo servidor for "SIGTERM", este indica que ocorreu um erro na execução do programa.

3.2- *"main"*

Na main são analisados os argumentos dados, para poderem ser enviados para o servidor, através do *pipe* (open(pipe,)). De seguida, é utilizada uma função que escreve no standard output o conteúdo inserido nos argumentos.
Após o tratamento de dados no servidor, é analisado o sinal enviado por este (signal(SIGRTMAX, signalhandler)). Mediante o sinal recebido, é efetuada a sua impressão, por fim, o programa termina.

4- Conclusão

Após a realização deste programa, cada elemento do grupo conseguiu aperfeiçoar os conhecimentos que adquiriu no ano anterior relativamente à linguagem de programação em C e reter novos noções.

Este programa tinha como objetivo a criação de um servidor cuja finalidade é receber dados a partir de um cliente, tendo em atenção que o cliente não executa, apenas recebe os dados e o servidor é responsável por executar.

Para que este programa fosse bem sucedido, foi fundamental a participação do grupo nas aulas práticas e a leitura cuidadosa e detalhada dos guiões disponibilizados, assim conseguiu-se assimilar a informação necessária para a construção do mesmo.

Notamos também, o quanto é complexo criar um *programa* que trata com restauro e backup de dados, pois para executar estas aplicações requer uma grande capacidade e conhecimento acerca dos servidores e de gestão de dados.