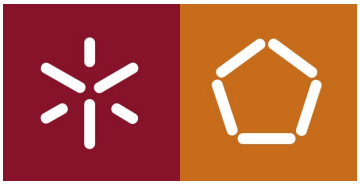

Sistemas Operativos
Trabalho Prático

Catarina Morales (a93319)
Joaquim T. Roque (a93310)
Tiago Carneiro (a93207)

29 de maio de 2022

1 Introdução

No âmbito do trabalho prático da Unidade Curricular de Sistemas Operativos, é proposto que se desenvolva um serviço que permita cifrar e/ou comprimir ficheiros. Para tal, os alunos deverão aplicar os conhecimentos adquiridos nas aulas práticas, tendo em vista a familiarização com as *system calls* do *Unix*, bem como conceitos de concorrência, arquitetura cliente-servidor e ainda a sensibilização para a programação de baixo nível.

2 Comunicação

Um dos maiores desafios iniciais foi pensar numa forma de o servidor comunicar apenas com um único cliente de cada vez, de forma a não receber *bytes* de outros clientes. Para tal, pensamos numa solução que envolve a criação de três *fifos*.

O primeiro *fifo*, **GATE_FIFO**, é aberto pelo servidor, ficando aqui bloqueado até que um cliente abra este mesmo *fifo*. Assim que os dois extremos são abertos, tanto o servidor como o cliente fecham os seus extremos. Após isto, o cliente já sabe que é o único a escrever, ou a ler, para o servidor e, assim, não há perda de informação nem do lado do cliente nem do lado do servidor.

Assim que o *fifo* anteriormente mencionado é aberto tanto pelo servidor como um cliente, estes utilizam outros dois *fifos* para comunicarem entre si:

- **SERVER_CLIENT_FIFO** é utilizado para o servidor comunicar a sua informação para o cliente.
- **CLIENT_SERVER_FIFO** utilizado para o servidor receber a informação providenciada pelo cliente.

Adicionalmente, existe um outro *fifo* (**BYTES_INFO**), cujo único propósito é comunicar ao cliente, aquando da finalização do pedido, o número de *bytes* dos ficheiros de entrada e saída, tendo em vista a concretização de um dos requisitos adicionais propostos.

3 Arquitetura de processos

Para solucionar o problema da organização dos processos, foi criada a estrutura *linkedlistProcess*. Esta estrutura contem a seguinte informação:

- **int pid_client** - Este é o *process id* do cliente que pediu o processamento
- **int pid_child** - Este é o *process id* do filho que vai realizar o processamento
- **int task_number** - O número de *tasks* a ser executadas
- **char* input_file** - *String* que contêm o *path* e o nome do ficheiro de *input*

- **char* output_file** - *String* que contêm o *path* e o nome do ficheiro de *output*
- **int commandsCount** - Número de comandos a serem executados
- **char** commands** - Lista de comandos a serem executados
- **int priority** - Nível de prioridade
- **linkedlistProcess* next** - Apontador para o próximo processo

Esta estrutura foi utiliza para criar duas listas ligadas. Uma inicial, para guardar os processos que estão na fila de espera para serem processados, e uma segunda, contendo os processos que estão atualmente a serem processados pelo servidor.

Na lista de processos pendentes, estes são organizados pelo seu nível de *priority*, sendo 5 a maior prioridade e 0 a menor. Isto leva a que seja possível que haja processos que sofram de *starvation*, já que podem ter um nível de prioridade inferior aos novos processos que possam aparecer constantemente e assim nunca ser atendido pelo servidor.

Para cada nodo na lista de processos a serem executados, é designado um processo filho do processo principal do servidor. Este filho, por sua vez, cria n filhos, sendo n o número de comandos a serem executados ao ficheiro. Cada um destes filhos aplica um comando diferente, utilizando *pipes* para comunicarem a informação entre si.

4 Servidor

O servidor necessita de dois argumentos. Primeiramente, necessita do *path* para o ficheiro com a informação dos limites dos vários comandos (isto é, o ficheiro de configuração do servidor), e de seguida necessita do *path* onde os executáveis correspondentes a estes comandos se encontram.

Assim que o servidor inicia, este lê o ficheiro com a informação dos comandos e organiza esta numa *llCommand*.

Esta estrutura, por sua vez, é uma lista ligada da estrutura *Command*, que contêm a seguinte informação:

- **char* type** : *string* com o tipo do comando, que é utilizada pelo servidor para executar o mesmo
- **int max** : capacidade máxima de processos que podem estar a utilizar este comando ao mesmo tempo
- **int running** : quantidade atual de processos que estão a utilizar este comando de momento

De forma a atualizar a lista de pedidos pendentes do servidor, este utiliza um alarme de 1 em 1 segundo. A função que processa o sinal do alarme verifica os processos atualmente atendidos já acabaram de ser processados, assim como verificar se o próximo elemento da lista de espera já pode ser atendido, colocando-o na lista de processos em processamento se assim for.

O servidor utiliza também ferramentas da biblioteca *sys/stat* para analisar a informação dos ficheiros que trata. Assim que um processamento de um ficheiro é terminado, o servidor manda o signal **SIGUSR2** para o cliente, informando o mesmo de que o processamento terminou e para que o cliente leia a informação do tamanho do ficheiro inicial e do tamanho do ficheiro final através do fifo **BYTES_FIFO**.

Para além destas funcionalidades, o servidor é também capaz de receber o sinal **SIGTERM**. Caso o mesmo aconteça, o servidor deixa de receber novos pedidos de processamento, recebendo apenas pedidos de consulta, e espera até todos os processos acabarem, encerrando após o mesmo acontecer.

5 Cliente

Passando agora ao cliente, este pode fazer dois tipos de pedidos ao servidor. O primeiro comando, ***status*** indica ao servidor para enviar a informação atual do servidor. O mesmo escreve os processos pendentes, os que estão atualmente a correr, a capacidade máxima de um comando assim como a atualmente utilizada e o *process id* do processo pai do servidor, tudo para o **SERVER_CLIENT_FIFO** de onde o cliente lê esta mesma informação.

O segundo comando consiste da seguinte lista de argumentos: ***proc-file (-p) (priority) (inputFile) (outputFile) (Command 1) ...***. Com este comando, o cliente faz um pedido ao servidor para que a lista de comandos seja aplicada ao ficheiro no caminho *inputFile* e que o resultado seja colocado no caminho *outputFile*. É de notar que a *flag -p* é opcional, sendo assim o campo da prioridade também. Caso não seja colocada esta mesma *flag* na lista de argumentos, o servidor assume por defeito como sendo 0 a prioridade deste pedido.

Após o pedido de processamento ao servidor, o cliente fica a par do estado do seu pedido através de sinais que o servidor vai mandando para o processo do cliente, ficando este a saber em que fase do processamento o seu processo se encontra.

6 Conclusão

Para finalizar, ficamos contentes com aquilo que conseguimos alcançar, tendo conseguido encontrar soluções para todas as dificuldades com que nos deparamos, assim como corrigir todas as perdas de memória do trabalho. Embora houvesse mais funcionalidades que podíamos ter implementado, conseguimos concluir o trabalho na sua totalidade.