Projeto de Sistemas Operativos 2024-25
Enunciado da 1ª parte do projeto LEIC-A/LEIC-T/LETI

O objetivo deste projeto é desenvolver o IST "Key Value Store" (IST-KjVS), um sistema de armazenamento de dados que permite a criação, escrita e leitura de dados, sob a forma de um nome (uma chave) associado a um vetor de bytes (um valor), comumente designados como um par chave-valor. Os pares chave-valor serão armazenados numa tabela de dispersão (*hashtable*). Uma tabela de dispersão é composta por um vetor de ponteiros para blocos de dados. No caso do IST-KVS os blocos de dados são os valores dos pares chave-valor. As chaves desses pares permitem encontrar qual a posição da tabela de dispersão onde está armazenado o valor correspondente. A conversão entre a chave e a posição relevante da hashtable é dada por uma função. Por exemplo, podemos considerar uma *hashtable* com tantas posições quantas as 26 letras do alfabeto e a função que converte chaves numa posição da tabela de dispersão ser a função que dá a posição no alfabeto da primeira letra da chave. Naturalmente isto faz com que, neste exemplo, todos os pares chave-valor cujas chaves comecem pela mesma letra sejam armazenados na mesma posição da tabela levando à necessidade de uma solução para estas colisões, como por exemplo cada posição do vetor ter uma lista de pares chave-valor e não só um único.
A tabela de dispersão estará armazenada na memória do programa IST-KVS que recebe os comandos descritos abaixo e armazena dados. Na segunda parte do projecto haverá um programa servidor que permitirá a outras aplicações criarem uma subscrição do serviço de armazenamento de pares chave-valor e serem notificados sempre que haja alterações nas chaves cujos valores lhes interesse acompanhar.

O IST-KVS explora técnicas de paralelização baseadas em múltiplas tarefas de forma a acelerar o processamento de pedidos e recorre a processos para fazer cópias de segurança dos dados de forma não bloqueante. Ao desenvolver o IST-KVS, os alunos aprenderão também como implementar mecanismos de sincronização escaláveis entre tarefas bem como mecanismos de comunicação entre processos (FIFOs e sinais). O IST-KVS irá também interagir com o sistema de ficheiros oferecendo portanto a possibilidade de aprender a utilizar as interfaces de programação de sistemas de ficheiros POSIX.

Código base

O código base fornecido disponibiliza uma implementação sequencial que aceita os seguintes comandos

1. `WRITE [(<c1>,<v1>) (<c2>,<v2>) ...]`
    - Este comando é usado para escrever um ou mais pares chave-valor. Cada par é representado por duas sequências de caracteres entre parêntesis curvos separados por uma vírgula, por exemplo

`(Escritor1,EcaDeQueiroz)` Por simplicidade, pode assumir-se que quer as chaves quer os valores têm um tamanho máximo de 40 caracteres e que não incluem espaços.

- No caso de ser feito um write para uma chave já presente, o valor desta chave deve ser atualizado com o novo valor incluído no comando `WRITE`.
- Sintaxe de uso:
  `WRITE [(umnome,umvalor)(outronome,outrovalor)]`

2. `READ [c1,c2, …]`
   - Permite ler um ou mais valores de uma sequência de chaves fornecida.
   - Este comando devolve um conjunto de pares chave-valor correspondente às chaves fornecidas. Se alguma chave não existir, o valor devolvido será a sequência de caracteres `KVSERROR` (que fica reservada para este efeito).
   - Sintaxe de uso: `READ [Escritor1,Escritor2, …]`
     - Devolveria, por exemplo:
       `[(Escritor1,EcaDeQueiroz)(Escritor2,KVSERROR)]`

3. `DELETE [c1,c2, …]`
   - Permite eliminar um ou mais pares chave-valor identificados por uma lista de chaves.
   - Se alguma chave não existir, deve devolver a sequência de caracteres `KVSMISSING` (que fica reservada para este efeito).
   - Sintaxe de uso: `DELETE [ChaveAEliminar1,ChaveAApagar2]`
     - Devolveria, por exemplo:
       `[(ChaveAApagar2,KVSMISSING)]`

4. `SHOW`
   - Imprime o estado de todos os pares chave-valor armazenados com uma ordenação alfabética crescente das chaves.
   - Sintaxe de uso: `SHOW`
     - Imprime uma lista, por exemplo:
       `(Achave,Ovalor)`
       `(Chave,V)`
       `(Xave,grandeValor)`

5. `WAIT <delay_ms>`
   - Introduz um atraso na execução dos comandos, útil para testar o comportamento do sistema sob condições de carga.
   - Sintaxe de uso: `WAIT 2000`
     - Adiciona um atraso de 2000 milissegundos (2 segundos) do próximo comando.
6. `BACKUP`
   - Faz uma cópia de segurança do estado atual dos dados armazenados usando para o efeito um processo à parte (usado somente a partir do exercício 2).
7. `HELP`
   - Fornece informações sobre os comandos disponíveis e como usá-los.

Comentários no Input:
Linhas iniciadas com o carácter **'#'** são consideradas comentários e são ignoradas pelo processador de comandos (úteis para os testes).

- Exemplo: `# Isto é um comentário e será ignorado`

1ª parte do projeto

A primeira parte do projeto consiste em 3 exercícios.

Exercício 1. Interação com o sistema de ficheiros

O código base recebe pedidos apenas através do terminal (*standard input*). Nesse exercício pretende-se alterar o código base de forma que passe a processar pedidos em lotes obtidos a partir de ficheiros.

Para este efeito o IST-KVS deve passar a receber como argumento na linha de comando o caminho para uma diretoria, onde se encontram armazenados os ficheiros de comandos e um segundo parâmetro inteiro indicando quantos *backups* podem ser executados concorrentemente (ver exercício 2). O IST-KVS deverá obter a lista de ficheiros com extensão `.job` contidos na diretoria indicada no primeiro parâmetro. Estes ficheiros contêm sequências de comandos que respeitam a mesma sintaxe aceite pelo código base.

O IST-KVS processa todos os comandos em cada um dos ficheiros `.job`, criando um correspondente ficheiro de output com o mesmo nome e extensão `.out` que contém o resultado de todos os comandos contidos nos ficheiros `.job`.

O acesso e a manipulação de ficheiros deverão ser efetuados através da interface POSIX baseada em descritores de ficheiros, e não usando a biblioteca stdio (carregada através do ficheiro stdio.h) e a abstração de *FILE stream.*

Ficheiro de entrada exemplificativo `/jobs/test.job`*:*

```
#Leitura de chave inexistente
READ [c1]
#Criação e escrita de 2 chaves
WRITE [(c1,v1)(chave2,valor-exemplo)]
#Leitura de chave existente
READ [chave2]
#Escrita de uma chave nova e outra existente
WRITE [(chave2,valor-exemplo2)(umaChave,umValor)]
# backup
BACKUP
# Eliminação de chave existente e de chave não existente
DELETE[c1,chaveAleatoria]
```

```
#Espera de um segundo
WAIT 1000
# Impressão de toda a tabela
SHOW
```

Resultado da execução do ficheiro `test.job` acima (contido no ficheiro `/jobs/test.out`):

```
[(c1,KVSERROR)]
[(chave2,valor-exemplo)]
[(chaveAleatoria,KVSMISSING)]
(chave2, valor-exemplo2)
(umaChave, umValor)
```

Exercício 2. Cópia de segurança não bloqueante

Após terem realizado o Exercício 1, os alunos devem estender o código criado de forma a que sempre que num ficheiro `.job` se encontre um comando `BACKUP` seja lançada um processo que realiza uma cópia integral do conteúdo da *hashtable* (semelhante ao comando `SHOW`) para um ficheiro no estado em que a *hashtable* se encontra após os comandos de escrita que antecedem o comando de `BACKUP`. O ficheiro deverá chamar-se `<nome-ficheiro>-<num-backup>.bck` em que `<nome-ficheiro>` é o nome do ficheiro de entrada com o comando `BACKUP` e `<num-backup>` é o contador de cópias de segurança realizadas pela aplicação, dentro do mesmo ficheiro, e deverá ser guardado na mesma diretoria que o ficheiro `.job` correspondente.
Por exemplo, uma diretoria com dois ficheiros, onde o primeiro, chamado `input1.job`, faz dois *backups*, e o segundo, chamado `input2.job`, faz apenas um, a listagem dessa diretoria depois de chamar o programa usando-a como directoria de entrada deverá ser:

```
input1.job
input2.job
input1.out
input2.out
input1-1.bck
input1-2.bck
input2-1.bck
```

Este procedimento deve ser feito usando o mecanismo de `fork` de forma a garantir que o processo pai possa continuar a executar a sequência de comandos a seguir ao comando `BACKUP` enquanto o processo filho grava o backup no ficheiro.

O número de cópias de segurança a poderem ser executadas simultaneamente está definido pelo 3º parâmetro do programa. Se, por exemplo, o parâmetro tiver o valor 2, isto

significa, que, caso se encontrem 2 cópias de segurança em execução e se pretenda executar um novo comando `BACKUP`, dever-se-á aguardar que termine um deles antes de se iniciar a execução deste novo `BACKUP`.

Exercício 3. Paralelização usando múltiplas tarefas

Neste exercício pretende-se tirar partido da possibilidade de paralelizar o processamento de diferentes ficheiros `.job` usando múltiplas tarefas. Isto é, cada tarefa processa um ficheiro `.job` diferente em paralelo.

O número de tarefas a utilizar para o processamento dos ficheiros `.job`, MAX_THREADS, deverá ser especificado como último parâmetro da linha de comando no arranque do programa. Serão valorizadas soluções de sincronização no acesso ao estado das tabelas de dispersão que maximizem o grau de paralelismo atingível pelo sistema. Mais ainda, o resultado da execução concorrente deverá garantir que a execução de cada instrução seja *atómica*, ou seja cada operação num ficheiro deverá aparecer como executada de forma instantânea

Idealmente, este exercício deveria ser realizado a partir do código obtido após a resolução do exercício 2. Contudo, não serão aplicadas penalizações se a solução deste exercício for realizada a partir da solução do exercício 1.

Submissão e avaliação

A submissão é feita através do Fénix **até ao dia 13/12/2024 às 23h59**.

Os alunos devem submeter um ficheiro no formato `zip` com o código fonte e o ficheiro `Makefile`. O arquivo submetido não deve incluir outros ficheiros (tais como binários). Além disso, o comando `make clean` deve limpar todos os ficheiros resultantes da compilação do projeto.

Recomendamos que os alunos se assegurem que o projeto compila/corre corretamente no cluster *sigma*. Ao avaliar os projetos submetidos, em caso de dúvida sobre o funcionamento do código submetido, os docentes usarão o cluster *sigma* para fazer a validação final.

O uso de outros ambientes para o desenvolvimento/teste do projeto (e.g., macOS, Windows/WSL) é permitido, mas o corpo docente não dará apoio técnico a dúvidas relacionadas especificamente com esses ambientes.

A avaliação será feita de acordo com o método de avaliação descrito no site da cadeira.

Os alunos não podem partilhar código e ou soluções com outros grupos. O código submetido tem de ser o resultado do trabalho original de cada grupo. A submissão de código com grande grau de semelhança com outros grupos ou realizado recorrendo a entidades externas ao grupo levará à reprovação dos grupos envolvidos e ao reporte da situação à coordenação da LEIC e ao Conselho Pedagógico do IST.