INTRODUÇÃO À
ARQUITETURA DE COMPUTADORES

2024-2025

Laboratório 7 – Leitura de ficheiros em RISC-V/Ripes

Neste laboratório, iremos explorar o processo de leitura e interpretação de ficheiros de imagem no simulador Ripes, recorrendo a chamadas de sistema (*system calls*) para manipulação de ficheiros.

Este guião laboratorial ajuda-vos também a darem os primeiros passos para resolverem os requisitos de carregamento de ficheiros que o 2º Projeto exige.

Tal como nos laboratórios anteriores, deverá implementar e validar a sua solução no simulador e confirmar os resultados, observando os valores finais dos registos e o conteúdo da memória. Deve igualmente executar uma instrução de cada vez ("step") e/ou colocar pontos de paragem no código para validar a correção da implementação.

Importante: Neste exercício, é fundamental instalar o Ripes na vossa máquina com a versão recomendada na página de IAC no Fénix, uma vez que algumas versões mais antigas não oferecem suporte funcional às chamadas de sistema para leitura de ficheiros.

Exercício 1: Ler conteúdo de ficheiro para *buffer* em memória (função read_file)

1. Abra o **Ripes** e aceda ao menu **Help → System Calls** para consultar os parâmetros das funções **open**, **read** e **close**.

Antes de ler um ficheiro, é necessário abri-lo através da função **open**, que recebe como argumentos uma **string com o caminho de acesso (*path name*) do ficheiro** e **opções (*flags*)** que definem como o ficheiro deve ser aberto.
Sobre estes argumentos, tenha em conta:

- Caso especifique um caminho de acesso relativo, ele será interpretado a partir da diretoria em que o executável do Ripes está a correr. Em alternativa, pode usar um caminho de acesso absoluto.
- Embora existam várias *flags*[1], neste guião e no projeto só precisaremos abrir o ficheiro para leitura, logo basta passar o valor zero (i.e., nenhuma *flag* ativada).

A chamada sistema **open** devolve um **descritor de ficheiro** (*file descriptor*, ou fd), que servirá de identificador nas operações seguintes.
Após obter o descritor de ficheiro, o programa pode finalmente efetuar a leitura de ficheiros através da chamada sistema **read**.
No final, deve fechar o ficheiro usando a chamada sistema **close**.

2. No ficheiro read_file.s, implemente a função read_file. Esta função recebe como argumento o *pathname* de um ficheiro e lê o seu conteúdo para um *buffer* em memória, cujo endereço e tamanho são também passados como argumentos.
   A função deve retornar o número de *bytes* lidos. Caso o conteúdo do ficheiro seja maior que o *buffer*, só deve ler o conteúdo que cabe no *buffer.*

   Para resolver este exercício, deve usar as chamadas sistema descritas acima.

3. Teste a sua função, preenchendo o programa principal (após a diretiva *.text*) de forma a executar os seguintes passos:
   a. Chamar a função read_file("mensagem.txt", buffer, dim_buffer).
   b. Caso a função retorne com sucesso (deve verificar o valor de retorno), imprimir o conteúdo do *buffer* na consola do Ripes chamando a chamada sistema PrintString.
   c. Experimente correr o programa e confirme que o que é impresso na consola do Ripes corresponde ao conteúdo do ficheiro "mensagem.txt".

[1] As *flags* disponíveis podem ser consultadas na definição do *enum Flags* no ficheiro systemio.h do código fonte do Ripes.

Exercício 2: usar a função read_file para ler ficheiros imagens em formato PGM binário.

1. Descarregue e descomprima o ficheiro *classifier-files.zip*, fornecido na página do projeto.
2. Entre na diretoria *input-images/ascii-pgm* e abra um dos ficheiros, usando um visualizador de imagem que suporte o formato PGM.[2]
3. Observe qual dígito é representado nessa imagem.

4. Agora abra o mesmo ficheiro mas usando um editor de texto simples e analise o seu conteúdo.

O formato Portable Gray Map (PGM)[3] pode ser representado em caracteres de texto codificados em ASCII. Por isso é que o pode consultar usando um editor de texto!

Como pode perceber ao analisar o conteúdo dos ficheiro .pgm contidos na diretoria *input-images/ascii-pgm*, estes apresentam um cabeçalho com os seguintes campos:

- Tipo de ficheiro: **P2** (PGM Plain) representa um formato PGM em texto, onde os valores de cinza são escritos em texto.
- Largura (w) e altura (h) da imagem.

  Valor máximo de intensidade de cinza (em que 0 é a cor preta e o valor máximo é a cor branca).

A seguir ao cabeçalho, surgem os valores das cores de cada pixel, num total de w x h inteiros.

Uma alternativa é PGM representado em binário, que corresponde ao tipo P5 (PGM Raw). A sua sintaxe é a mesma do tipo P2 (PGM Plain), mas os números passam a ser representados em binário. No caso dos valores de cada pixel, são armazenados em inteiros de 8 bits (1 byte) desde que o valor máximo de valores de cinza caiba em 8 bits. Note que os 2 primeiros bytes do ficheiro, que especificam o tipo ("P5"), continuam a ser codificados em ASCII.

Como vantagens, este formato é mais compacto e, se carregado em memória, fica pronto para ser processado pelas operações aritméticas do RISC-V (tal como o 2º projeto fará!).
Como desvantagem, o formato binário é mais difícil de ler por humanos.

4. Utilize o script Python pgm_ascii_to_bin.py para converter a imagem anterior para o formato raw PGM, que é binário.

[2] Por exemplo, https://jumpshare.com/viewer/pgm.
[3] https://en.wikipedia.org/wiki/Netpbm#File_formats

5. Experimente usar o programa composto no exercício 1 para ler este ficheiro em formato binário. No entanto, neste caso não o deve imprimir na consola (pois o *buffer* não será carregado com caracteres de texto). Em vez disso, use o Ripes para, no final do programa, verificar o conteúdo do *buffer* em memória e confirmar que é idêntico ao conteúdo do ficheiro.
   Nota: Para visualizar o conteúdo do ficheiro binário, pode usar o comando *hexdump* (Linux/MacOS) ou *format-hex* (Windows PowerShell).

Além deste guião laboratorial...

No 2º Projeto, depois de ler o conteúdo do ficheiro de entrada (com a imagem a classificar) para um *buffer*, no passo 2 da função *classify* precisará de passar esse *buffer* como como um dos argumentos da função *matmul.*

No entanto, há alguns desafios a ultrapassar:

- Embora o conteúdo carregado do ficheiro inclui o cabeçalho e o corpo do PGM, só o corpo deve ser passado à função *matmul*.
- A função *matmul* espera receber matrizes de inteiros de 32 bits mas o *buffer* que carregaram a partir do ficheiro é de inteiros de 8 bits (1 byte).

Pode aproveitar a aula laboratorial para, com o vosso docente, discutir com ele como ultrapassar estes desafios.

Exercício 3: Usar a função read_file para carregar as matrizes de peso.

Vamos agora proceder à leitura dos pesos da rede neuronal do 2º Projeto, aproveitando de novo a função read_file.

Antes, começamos por descrever o formato que usaremos para os ficheiros com as matrizes de pesos no projeto.

- Cada peso é um inteiro em binário, de 32 bits.
- A todos os valores guardados no ficheiro, adicionámos previamente o valor 32. Como os valores dos pesos originais estão necessariamente contidos no intervalo [-32, 31], este ajuste (+32) implica que, no ficheiro, nenhum inteiro é negativo.
- Portanto, após ler do ficheiro para memória, cada inteiro deve ser corrigido subtraindo-lhe o valor 32.

1. Estenda o programa principal do exercício anterior para, depois de ter carregado o ficheiro PGM, ler uma matriz de pesos a partir do ficheiro *weight-matrices/m0.bin*. Use também a função *read_file* para esse propósito.

2. Após carregar o conteúdo do ficheiro em memória, não se esqueça de corrigir os valores da matriz.

3. Repita as alíneas anteriores, mas estendendo o programa para ler outra matriz a partir do ficheiro *m1.bin*.

Se chegou aqui, já tem o passo 1 da função *classify* praticamente completado! Próximos passos: pensar no passo 2 e seguintes da mesma função.

Bom trabalho!