Programação Concorrente

2022/2023

Projeto – Parte B

Na segunda parte do projeto os alunos implementarão um pipeline simples para o processamento das imagens de forma parecida com o da Parte A do projeto.

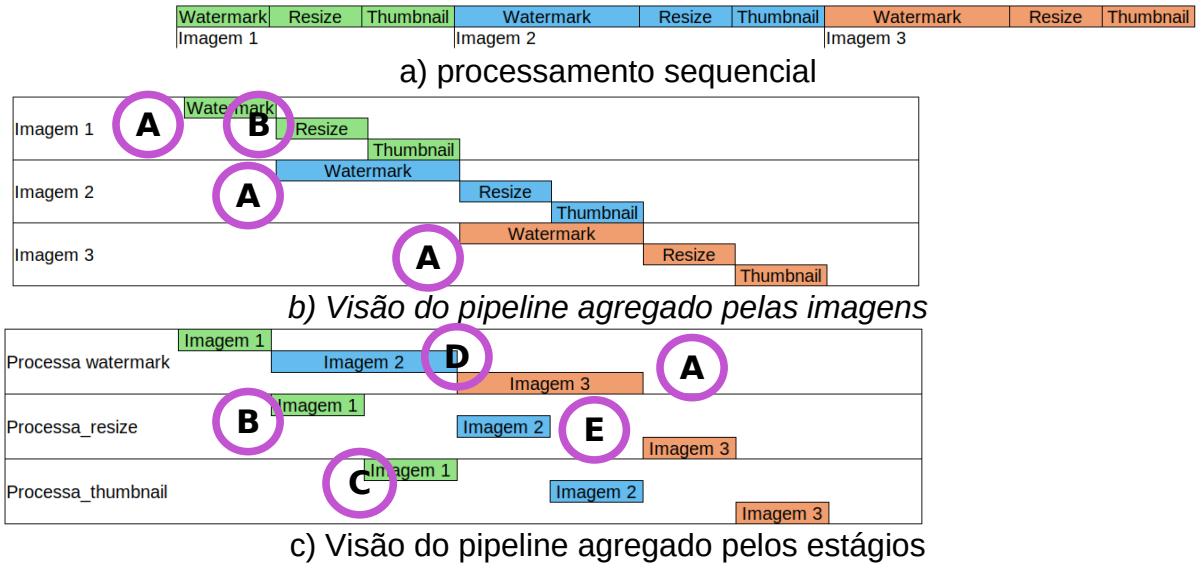
1 Descrição da parte B do projeto (**ap-paralelo-3**)

Neste projeto os alunos deverão implementar uma aplicação paralela que processa imagens, tal como na Parte A do projeto, mas seguindo uma arquitetura pipeline.

A aplicação a desenvolver na **Parte B** do projeto (chamada **ap-paralelo-3**),deverá implementar um pipeline com 3 estágios de processamento diferentes:

- *watermark* → *resize* → *thumbnail*.

A Figura 1, ilustra uma possível evolução do processamento de 3 imagens com as 3 estágios.



a) processamento sequencial

b) *Visão do pipeline agregado pelas imagens*

c) Visão do pipeline agregado pelos estágios

Figura 1: Exemplo de execução no pipeline

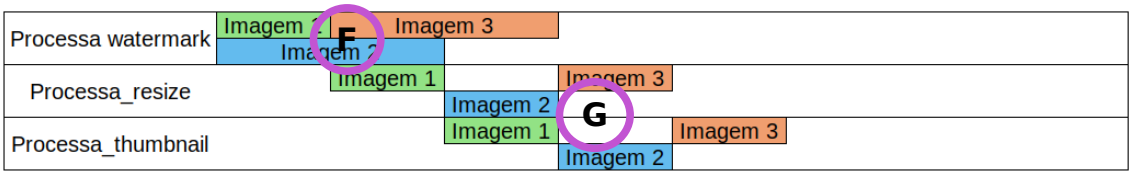
Para cada imagem a *watermark* deverá ser realizada primeiro (**A** na imagem), antes do *resize* e só no fim o *thumbnail* .

Depois de ter sido realizado o *watermark*, o estágio seguinte inicia a realização do *resize* correspondente à mesma imagem (passo **B**) e quando o *resize* terminar o terceiro estágio produz o *thumbnail* (passo **C**) .

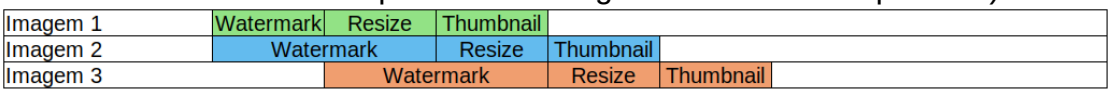
Em cada um dos estágios, após a conclusão do processamento de uma imagem, dever-se-á iniciar o processamento de uma nova imagem que ainda não tenha sido processada (passo **D**) e notificar a *thread* do próximo estágio (se necessário).

Como se observa na imagem anterior, mesmo que um estágio esteja disponível para processar uma nova imagem, tal pode não acontecer imediatamente, visto não existir nenhuma imagem que ainda não tenha completado os estágios anteriores (**E** na imagem).

No exemplo anterior apenas uma *thread* executada o código de cada estágio. Na **Parte B** do projeto será possível ao utilizador definir quantas *threads* replicadas executarão cada um dos estágios.



*a)*Visão do pipeline agregado pelos estágios (observa-se que no estágio da *watermark* as duas threads processam imagens diferentes em paralelo)



b) Vis*ão do p*ipeline agregado pelas imagens

Figura 2: Exemplo de execução no pipeline com duas threads por estágio

Com duas *threads* (exemplificado na Figura 2) será possível em cada estágio ter duas imagens a ser processadas em simultâneo. Todos os outros constrangimentos (uma imagem só poder ser processada num estágio depois de ter terminado o processamento no estágio anterior, e cada *thread* só poder processar uma imagem em cada instante) mantêm-se.

Na Figura 2, no estágio *watermark*, as duas *threads* processam em paralelo (marca **F**) duas imagens diferentes. Devido ao tempo de processamento das diversas imagens e seu encadeamento poderá haver momentos em que apenas uma *thread* de um estágio processa uma imagens (exemplificado neste exemplo com a marca **G**).

1.1 Funcionamento geral

Para além da implementar um pipeline, o objetivo global é muito semelhante à **Parte A** do projeto: várias *threads* transformam concorrentemente imagens de modo a acelerar o processamento do conjunto elevado de imagens.

Tal como na Parte A do projeto, a lista de imagens a processar será fornecida às aplicações através do acesso a um ficheiro de texto que contem em cada linha o nome de uma imagem a ser processada. Depois de ler os nomes das imagens do ficheiro de configuração, o **main()** deverá introduzir no pipeline cada uma das imagens a processar. No fim da execução do pipeline, cada uma das imagens terá o resultado correspondente:

- imagem original com o símbolo do IST aplicado como marca de água (produzida no primeiro estágio).
- versão reduzida da imagem com o símbolo do IST aplicado como marca de água (produzida no segundo estágio.
- *thumbnail* da imagem com símbolo do IST aplicado como marca de água (produzida no terceiro estágio)

Na **Parte A** do projeto a comunicação e transferência de informação entre o **main()** e as outras *threads* (essencialmente nomes das imagens) era feito recorrendo a variáveis globais/partilhadas e a distribuição de trabalho era definida por um algoritmo:

- na **Parte A** era usada memória partilhada para acesso aos nomes dos ficheiros a processar
- o algoritmo definia de antemão que imagens seriam processadas por cada *thread*.

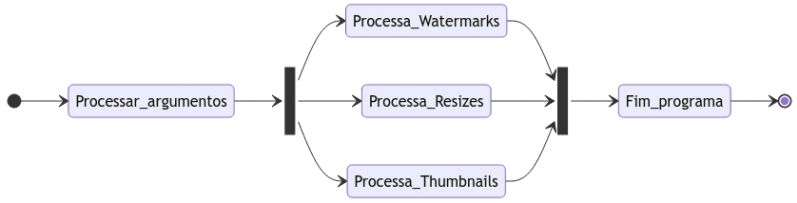
Na **Parte B** do projeto, para implementar a comunicação entre os vários estágios do pipeline, o envio de informação **entre as várias *threads* será feita através de pipes** e a distribuição das imagens será dinâmica e dependente do tempo de processamento das imagem nos estágios anteriores.

Quando já não houver mais imagens para serem processadas por cada estágio, as diversas *threads* correspondentes deverão terminar ordeiramente.

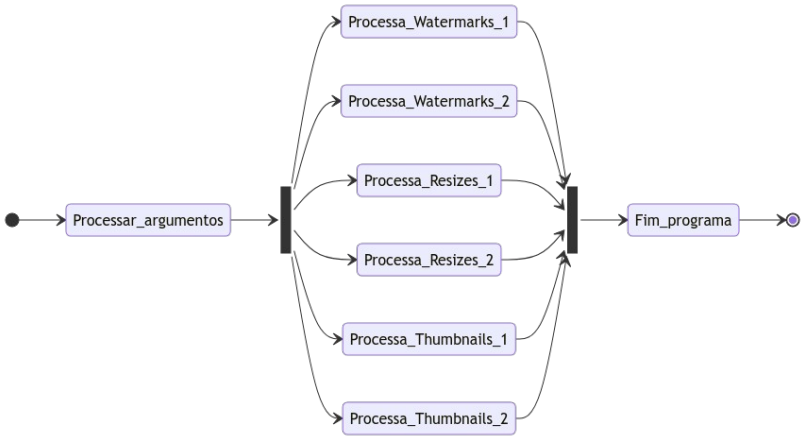
1.2 Threads

A aplicação a desenvolver na Parte B do projeto (chamada **ap-paralelo-3**), terá 3 tipos de *threads* distintas. Cada *thread* faz uma transformação específica correspondente a um dos estágios do pipeline (como ilustrado na Figura 3):

- As *threads* **Processa_watermarks** aplicam uma marca de água às imagens por si processadas;
- As *threads* **Processa_resizes** fazem o redimensionamento às imagens já com a *watermark* aplicada
- As *threads* **Processa_Thumbnails** criam *thumbnails* das imagens já com a *watermark* aplicada.



a) 1 thead por estágio



b) 2 threads por estágio

Figura 3: Criação e terminação de threads associadas a cada estágio

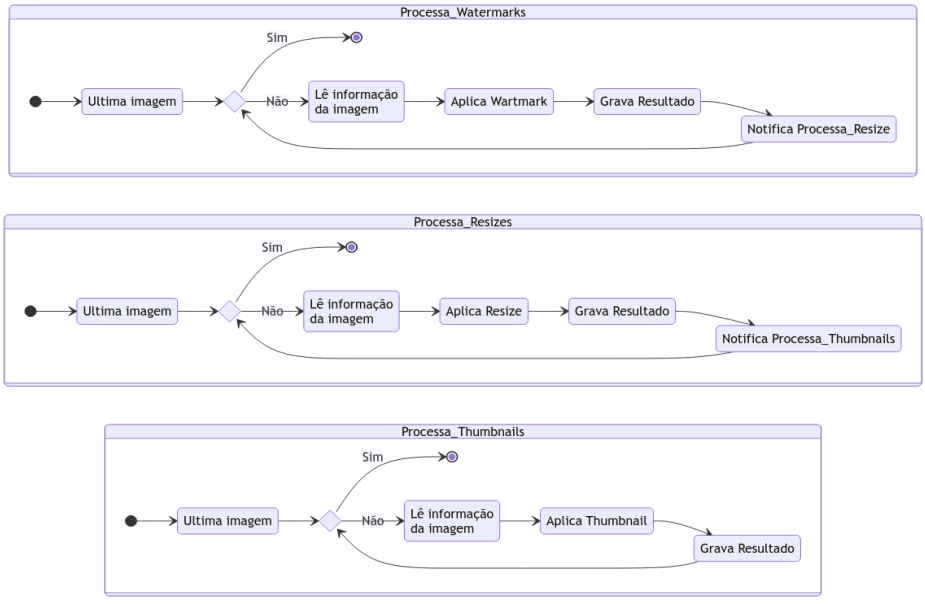


Figura 4: Funcionamento das threads

Para cada estágio de processamento (*watermark*, *resize* e *thumbnail*) poderão existir diversas *threads* a executar concorrentemente, em que cada *thread* processa conjunto disjunto de imagens das outras *threads* do seu estágio.

Todas as *threads* de todos os estágios são iniciadas no princípio do programa e devem terminar automática e ordeiramente quando não houver mais imagens para o estágio correspondente.

Apesar de todas as *threads* estarem a executar constantemente, haverá alturas em que estão bloqueadas à espera de imagens. Esta espera é conseguida através da leitura de um **pipe**, de onde será lido o nome/referência da imagem a processar. Os alunos deverão decidir quantos e quais a finalidades dos **pipes** a usar no projeto.

A leitura e escrita no(s) pipe(s) deverá seguir o esquema apresentado na Figura 4. Cada *thread* espera por dados no pipe; depois de ler a referência de uma imagem, a *thread*, processa-a, grava o resultado e notifica o estágio seguinte (no caso das *threads* do estágio *watermark* e *resize*).

As *threads* devem terminar ordeiramente quando já não houver imagens a ser processadas no estágio correspondente.

1.3 Otimizações

O processamento de cada um dos estágios deverá executar durante o mais curto espaço de tempo. Assim, o código deve ser otimizado de forma a não repetir código ou operações desnecessárias.

Os alunos poderão definir as estruturas de dados que considerarem convenientes, garantindo sempre que a referência/nome da imagem a processar em cada iteração das *threads*/estágio é lida a partir de pipes.

Se desejarem, os aluno poderão comparar a solução do pipeline proposto (*thumbnail* só executado depois do *resize*) com uma solução em que que o *resize* e *thumbnail* executem em paralelo imediatamente a seguir ao *watermark*.

2 Funcionamento das aplicações

A aplicação será desenvolvida em **C** e executará em Linux/WSL (ou Cygwin).

Para cada execução, o utilizador deverá indicar através dos argumentos da linha de comando o seguinte (pela ordem indicada):

- a diretoria onde se encontram as imagens originais,
- o número de *threads* para cada estágio do pipeline

Os programas aplicarão as transformações atrás descritas, armazenando as imagens num conjunto de pastas pré-definido. Na diretoria onde se encontram as imagens originais deve existir o ficheiro `image-list.txt` .

Para a execução da paralelização 3 (**ap-paralelo-3**) o utilizador deverá indicar a diretoria onde se encontram as imagens e o número de *threads* por estágio, como exemplificado de seguida:

```
ap-paralelo-1 dir-1 4
ap-paralelo-1 dir-2 8
app-parallel-1 . 1
```

O primeiro argumento corresponde à diretoria e o segundo ao número de *threads*.

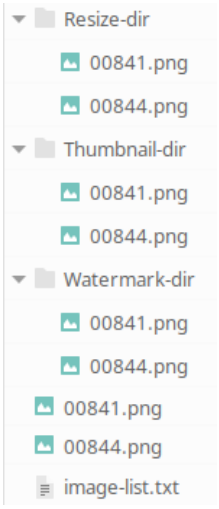
O número de *threads* deve ser um qualquer número inteiro positivo. Se, por exemplo, o utilizador indicar **1** como o número de *threads* a criar, o programa utilizará apenas uma *thread* para cada estágio do pipeline.

O ficheiro `image-list.txt`, contém a lista das imagens que se encontram na diretoria e que devem ser processadas. Este ficheiro contêm um nome de ficheiro por linha. As aplicações deverão ler este ficheiro e só as imagens aí listadas serão processadas. Assim, a diretoria indicada pelo utilizador poderá conter mais imagens do que aquelas a serem processadas.

Apenas deverão ser processadas imagens do formato PNG.

2.1 Resultados

A execução das aplicações produz um conjunto de novas imagens, correspondentes à transformações de cada uma das imagens iniciais.



O nome das novas imagens será o mesmo da imagem original, mas cada imagem será colocada numa sub-diretoria específica:

- ***Resize-dir*** – sub-diretoria que contém os resultado da redução das imagens
- ***Thumbnail-dir*** – sub-diretoria que contém os *thumbnails* produzidos
- ***Watermark-dir*** – sub-diretoria que contém as imagens com a marca de água aplicada

Estas sub-diretorias deverão ser criadas pelas aplicações a desenvolver, na diretoria indicada na linha de comando pelo utilizador e onde se encontram as imagens originais.

2.2 Interrupção de execução

Se as aplicações forem interrompidas a meio do processamento dos ficheiros, apenas parte dos resultados terão sido produzidos e guardados no disco.

Se o utilizador voltar a executar a aplicação, não deverá ser necessário voltar a produzir os ficheiros resultado já existentes. A aplicação só deverá processar e gastar tempo na criação dos ficheiros em falta.

Para verificar se um ficheiro existe os alunos poderão usar as funções *`access()`* como no seguinte exemplo:

```
if( access( nome_fich, F_OK ) != -1)
   printf("%s encontrado\n", nome_fich);
else
   printf("%s nao encontrado\n", nome_fich);
```

3 Submissão do projeto

O prazo para submissão da resolução da Parte B do projeto é dia **13 de Janeiro de 2023 às 19h00** no FENIX.

Os alunos deverão submeter um ficheiro **.zip** contendo o código da aplicação numa diretoria chamada **ap-paralelo-3/**. Os alunos deverão entregar também uma `Makefile` para a compilação da aplicação e verificar cuidadosamente que a mesma executa corretamente.

Os alunos deverão submeter um pequeno documento/relatório (chamado **pconc-relatorio-B.pdf**). O modelo do relatório será fornecido posteriormente, mas deverá listar as funcionalidades implementadas e apresentar os seguintes resultados para várias execuções:

- Numero de *threads*
- Tempo total de execução da aplicação
- Throughput
- Características do Computador

Para obter o tempo total de execução, os alunos devem usar o output produzido pelo comando **time**:

time(1) - Linux manual page - man7.org

Estes resultados deverão ser recolhidos executando a aplicação no mesmo ambiente onde executaram as aplicações da parte A e com as várias combinações de:

- conjuntos de dados (a ser fornecidos pelos docentes),
- 1, 2, 4, 8 número de *threads*.

4 Avaliação do projeto

A nota para esta parte do projeto será dada tendo em consideração o seguinte:

- Número de aplicações e funcionalidades implementadas
- Modo de gestão das *threads* e recursos
- Estrutura e organização do código
- Tratamento de erros
- Comentários
- Relatório