Programação Concorrente

2025/2026 Projeto – Parte B

O processamento computacional de grandes conjuntos de dados é normalmente composto por um conjunto de tarefas de elevada complexidade temporal. No entanto, muitas vezes pode-se tirar partido da existência de múltiplos CPUs ou *Cores* para reduzir o tempo total de processamento.

Na parte A do projecto os alunos implementaram uma aplicação paralela que acelerava a conversão e processamento de imagens usando os vários Cores existentes nos computadores modernos. Na segunda parte do projeto os alunos, para além de paralelismo os alunos irão implementar concorrência entre tarefas, permitindo que a aplicação interaja com o utilizador ao mesmo tempo que processa imagens.

1 Descrição da parte B do projeto

Neste projeto os alunos deverão alterar a parte A do projeto de modo a permitir que durante a execução da aplicação o utilizador dê comandos:

- adição de novas imagens a serem processadas
- impressão das estatísticas.

A aplicação resultante continuará efetuar as 5 transformações a imagens (tal como na parte A), mas é o utilizador que através do teclado (***fgets***) indicará quais as pastas a processar. Esta aplicação chamar-se-á **process-photos-parallel-B**.

Tal como na Parte A do projeto, deverão ser usadas *threads*, mas agora complementadas com *pipes*.

1.1 Funcionamento geral

Quando se executa a aplicação **process-photos-parallel-B** esta fica à espera que o utilizador escreva no teclado nomes de pastas onde se encontram imagens **.jpeg**. Enquanto o utilizador não der um comando a aplicação (e as suas *threads* estão adormecidas). Após a leitura do nome de uma pasta, a aplicação, seleciona as imagens dessa pasta terminadas em .**jpeg** e, para cada uma destas imagens, produz 5 novas imagens (correspondentes às transformações *contrast*, *blur*, *sepia*, *thumb*, *gray*). Esta conversão/transformação será semelhanat à da parte A do projeto e será também paralelizada usando

threads. As imagens resultantes terão o nome da imagem original precedidas do prefixo correspondente à transformação, e serão armazenadas em pastas específicas tal como descrito na Secção 2.3.

1.2 Concorrência

Enquanto a aplicação e *threads* processam essas imagens, o utilizador poderá concorrentemente dar novos comandos, nomeadamente adicionar novas pastas com imagens.

1.3 Paralelização

A aplicação cria inicialmente um determinado número de *threads*, que deverão processar as imagens existentes nas pastas indicadas pelo utilizador. Após o utilizador escrever o nome da pasta, cada uma das *threads* deverá processar um sub-conjunto disjunto dessas imagens. O número de *threads* criadas no início da aplicação é definido pelo utilizador através de um argumento da linha de comandos. O utilizador também define por que ordem as imagens são enviadas para as *threads* (alfabeticamente ou por tamanho crescente).

Depois de saber os nomes das imagens que se encontram na pasta indicada pelo utilizador e ordená-las (por tamanho ou alfabeticamente), o *main* envia para a *threads* a informação necessária para serem processadas.

As *threads* trabalhadoras executam desde o início do programa até à sua terminação, não sabem à priori quais as imagens a processar e têm um ciclo que:

- espera pela informação da próxima imagem a processar
- processa a imagem

O pseudocódigo do ***main*** e das várias *threads* trabalhadoras é o seguinte:

***MAIN*:**

```
Processa argv
Cria threads trabalhadoras
repetir{
   lê comando do teclado
   se (comando == DIR){
      obtém nomes das imagens da directoria
      para cada imagem da directoria{
        envia informação da imagem para as threads trabalhadoras
```

```
      }
   }
   se (comando == STAT){
      imprime estatísticas
   }
}enquanto (comando != QUIT)
imprime estatísticas
exit
```

***threads* trabalhadoras:**

```
repetir{
   recebe informação de uma imagem
   efetua 5 transformações a essa imagem
   guarda imagens resultantes no disco
   actualiza estísticas
   imprime estatísticas
}enquanto (não sair)
```

2 Funcionamento da aplicação paralela

A aplicação deverá ser desenvolvida em **C** e executará em Linux, WSL ou MAC OS X.

2.1 Argumentos da linha de comandos

Para a execução da aplicação **process-photos-parallel-B** o utilizador deverá sempre indicar através dos argumentos da linha de comandos o seguinte (pela ordem indicada):

- número de *threads* trabalhadoras a criar no início do programa
- **-name** ou -**size** que indicam a ordem pela qual as imagens são organizadas antes de serem enviadas para as *threads*

como exemplificado de seguida:

```
./process-photos-parallel-B 4 -size
./process-photos-parallel-B 8 -name
./process-photos-parallel-B 1 -name
```

O número de *threads* deve ser um qualquer número inteiro positivo e indica quantas *threads* trabalhadoras serão criadas e efetuarão o processamento das imagens. Se, por exemplo, o utilizador

indicar 1 como o número de *threads* a criar, o programa utilizará apenas uma *thread* para além do ***main()*** para efetuar o processamento de todas as imagens.

Os parâmetros **-name** e **-size** configuram por que ordem as imagens são organizadas antes de serem processadas/enviadas para a *threads* trabalhadoras.

2.2 Leitura dos comandos

Nesta aplicação é o utilizador que, durante a execução da aplicação, indica que pasta deverá ser processada. Para isso a aplicação deverá ler do teclado os comandos dados pelo utilizador. Mesmo que a aplicação se encontra a processar algumas imagens , o utilizador poderá dar comandos.

Os comandos que o utilizador pode introduzir são os seguintes: **DIR**, **STAT** e **QUIT** como se ilustram no seguinte exemplo (texto escrito pelo utilizador a negrito):

```
> ./process-photos-parallel-A 4 -size
Foram criadas 4 threads
Qual o comando: STAT
0 imagens – 0.0s tempo médio
Qual o comando: DIR ./dataset_A
A 10 imagens na pasta ./dataset_A serão processadas pelas 4 threads
Qual o comando: DIR ./dataset_B
A 10 imagens na pasta ./dataset_B serão processadas pelas 4 threads
Qual o comando: QUITA
comando inválido
Qual o comando: QUIT
Despois de todas a imagens serem processadas o program temrinara
 . .
```

O utilizador poderá concorrentemente dar novos comandos através do teclado, mesmo que o programa esteja a processar/transformar imagens.

2.2.1 Comando DIR

Para que sejam processadas imagens existentes numa pasta, o utilizador deverá escrever o comando **DIR** seguido no nome da pasta onde se encontram imagens do tipo **.jpeg**.

```
Qual o comando: DIR ./dataset_A
A 10 imagens na pasta ./dataset_A serão processadas pelas 4 threads
Qual o comando: DIR ./dataset_B
```

`A 10 imagens na pasta ./dataset_B serão processadas pelas 4 threads`

A pasta onde se encontram as imagens pode ser uma relativa ao local onde o programa é executado (começando por **./**) ou absoluta se começar por **/home/jnos/pconc/** (por exemplo).

O programa deverá obter o nome de todas as imagens cujo extensão seja **.jpeg** que se encontrem na pasta indicada no primeiro argumento e enviar essa informação para as *threads* trabalhadoras.

Depois das imagens serem enviadas para as *threads* o *main* deverá ficar pronto para ler um novo comando, independentemente do tempo que as imagens demorarem a ser processadas.

2.2.2 Comando STAT

Quando o utilizador dá este comando, o programa deverá imprimir o numero de imagens que já foram processadas e o tempo médio de processamento (criação das imagens transformadas) dessas imagens.

A informação a ser apresentada no ecrã está descrita na secção 2.3.1.

2.2.3 Comando QUIT

Quando o utilizador dá o comando **QUIT**, o programa deverá terminar, mas só após todas as imagens em fila de espera terem sido processadas. Quando o utilizador der o comando QUIT e já não há imagens à espera de serem processadas o programa poderá sair imediatamente, mas se ainda houver imagens que não foram transformadas (devido a comandos **DIR** anteriores) o programa só pode terminar quando essas imagens forem transformadas.

Imediatamente antes do *main* terminar deverá imprimir as estatísticas como descrito na secção 2.3.1.

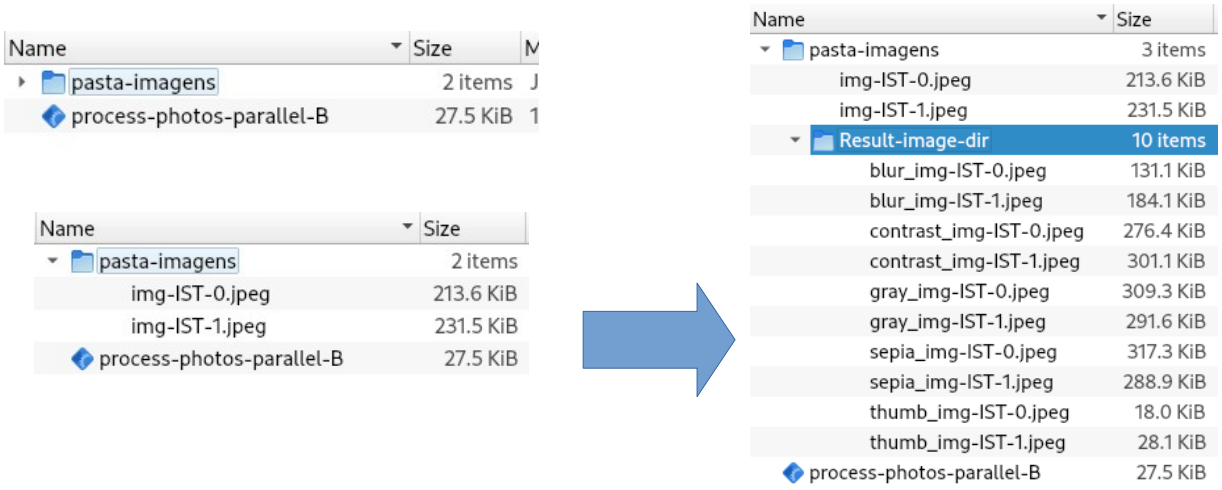
2.3 Resultado do processamento das imagens

Sempre que o utilizador der o comando **DIR**, a aplicação deverá transformar as imagens que se encontram da pasta indicada pelo utilizador.

Todas as imagens resultantes das transformações deverão ser colocadas numa nova pasta, chamada **Result-image-dir** a ser criada dentro da pasta onde se encontram as imagens originais. O processamento/transformação a aplicar às imagens será o mesmo da parte A do projeto (*contrast*, *blur*, *sepia*, *thumb*, *gray*) e o nome das novas imagens será o mesmo da imagem original mas com um prefixo correspondente à transformação, tal como no exemplo **process-photos-serial.c** (da Parte A do projecto).

A imagem seguinte exemplifica a estrutura dos ficheiros e pastas antes e depois de executar a aplicação com os seguintes argumentos e comando:

```
./process-photos-parallel-B 3 -size
Qual o comando: DIR ./pasta-imagens
A 2 imagens na pasta ./pasta-imagens serão processadas pelas 3 threads
   . . .
```



2.3.1 Estatísticas

O programa deverá ser capaz de calcular o tempo médio de processamento das imagens. Sempre que o processamento/transformação de uma imagem termina, o programa deverá armazenar toda a informação necessária para que seja possível imprimir o tempo médio de processamentos:

```
Numero total de imagens processadas – 10
Tempo médio de processamento – 13.29s
```

Esta informação deverá ser impressa nas seguintes situações:

Término de processamento de cada imagem

Após o processamento de cada imagem, a *thread* trabalhadora responsável deverá imprimir o nome da imagem, o tempo que essa imagem demorou a ser processada/transformada e tempo médio de processamento de todas as imagens:

```
thread 1 processou img-IST-1.jpeg em 10.87s
Numero total de imagens processadas – 2
Tempo médio de processamento – 9.73s
```

Comando STAT do utilizador

Quando o utilizador escreve no teclado o comando **STAT** o programa deverá imprimir as estatísticas globais do processamento das imagens.

```
Qual o comando: STAT
Numero total de imagens processadas – 1
```

```
Tempo médio de processamento – 8.59s
```

Encerramento do programa

Após o utilizador dar o comando **QUIT** e todas as imagens que se encontravam em fila de espera forem processadas, imediatamente antes do *exit*, o programa deverá imprimir as estatísticas globais do processamento das imagens.

```
Qual o comando: QUIT
   . . .
Numero total de imagens processadas – 2
Tempo médio de processamento – 9.73s
```

Exemplo

No caso do exemplo anterior o programa poderá produzir (entre outros) o seguinte resultado:

```
./process-photos-parallel-B 3 -size
Qual o comando: DIR ./pasta-imagens
thread 1 processou img-IST-0.jpeg em 8.59s
Numero total de imagens processadas – 1
Tempo médio de processamento – 8.59s
Qual o comando: STAT
Numero total de imagens processadas – 1
Tempo médio de processamento – 8.59s
thread 0 processou img-IST-1.jpeg em 10.87s
Numero total de imagens processadas – 2
Tempo médio de processamento – 9.73s
Qual o comando: STAT
Numero total de imagens processadas – 2
Tempo médio de processamento – 9.73s
Qual o comando: QUIT
Numero total de imagens processadas – 2
Tempo médio de processamento – 9.73s
```

A mensagem **Qual o comando:** poderá aparecer desformatada no ecrã.

2.4 *threads* trabalhadoras

As *threads* trabalhadoras são responsáveis pelo processamento/transformação das imagens. Ao contrário da part A do projeto, na parte B estas *threads* não sabem *a priori* quais a imagens a si atribuídas. Durante a sua execução cada uma destas *threads* é repetidamente informada de qual a imagem que deverá ser processada de seguida.

Estas *threads* são criadas logo no início do programa, emn número dependente do *argv*, e terminam apenas após o utilizador dar o comando **QUIT**.

Quando as *threads* trabalhadoras recebem a informação acerca da próxima imagens a ser processada, não necessitam de verificar se as imagens resultantes já existem. O tempo de processamento de cada imagem terá de ser contabilizado de modo a ser produzida uma estatística geral.

3 Comunicação entre *threads*

É responsabilidade do ***main*** ler o nome da pasta onde se encontram as imagens e comunicar a cada *thread* trabalhadora a informação referente a cada imagem. Esta transferência de informação deverá ser implementada usando um **pipe**. Os alunos deverão definir que informação é transmitida entre o ***main*** e cada uma das *threads* trabalhadoras.

Para a comunicação do tempo de processamento das imagens (para efeitos de estatísticas), poder-seá usar um outro *pipe* ou variáveis partilhadas.

4 Submissão do projeto

4.1 Prazo de submissão

O prazo para submissão da resolução da Parte B do projeto é dia **9 de Janeiro de 2026 às 19h00** no FENIX.

Antes da submissão, os alunos devem criar grupos de dois alunos e registá-los no FENIX.

4.2 Ficheiros a submeter

Os alunos deverão submeter um ficheiro **.zip** contendo:

- todo o código da aplicação **process-photos-parallel-B**
- a **Makefile** para a compilação da aplicação
- Relatório referente à parte A do projeto em formato PDF
- Relatório referente à parte B do projeto em formato PDF

Não incluir no .zip as imagens dos *datasets* utilizadas.

4.3 Relatórios

Os alunos deverão escrever dois relatórios simples, um para cada parte do projeto.

Serão fornecidos modelos para esses relatórios que apresentam a informação a apresentar.

Os relatórios deverão ser submetidos em formato PDF no mesmo zip do código da Parte B.

5 Avaliação do projeto

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

- Funcionalidades implementadas
- Modo de gestão das *threads* e recursos
- Estrutura e organização do código
- Tratamento de erros
- Comentários

O formato e conteúdo dos relatórios também contará para a nota final do projeto.

Anexos

6 Código para leitura do teclado

Para efetuar a leitura dos comandos do utilizador aconselha-se o uso do seguinte código:

```
char palavra_1[100], palavra_2[100];
fgets(linha, 100, stdin);
int n_palavras = sscanf(linha, "%s %s", palavra_1, palavra_2);
```

Desta forma facilmente se sabe se o utilizador escreveu uma ou duas palavras (variável n_palavras) e quais (**palavra_1** e **palavra_2**).

Um exemplo de utilização deste código foi disponibilizada com este enunciado.