Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia de Computação e Automação

DCA0107 – Sistemas Operacionais - Threads

EDSON AUGUSTO DIAS GOMES – 20230001561

GABRIEL SOUTO LOZANO BARBOSA - 20220080955

NATAL/RN - 2023

1. DESCRIÇÃO DE LINGUAGEM E BIBLIOTECAS

Na realização dessa atividade foi utilizado a linguagem C e as bibliotecas ‘OpenCV’ para manipular a imagem proposta e ‘pthreads’ para criação e gerenciamento de threads. As funções principais que fazem uso das threads são a ‘calcular_borda_x’ e ‘calcular_borda_y’, que calculam as bordas da imagem em direções diferentes, assim como demonstrado no pseudocódigo da atividade. Essas funções são executadas concorrentemente em threads separadas para melhorar o desempenho do processamento de imagens. As threads são criadas e controladas com as funções ‘pthread_create’ e ‘pthread_join’, permitindo a execução paralela das operações de cálculo de borda nas direções x e y. Ao final, a imagem de saída é gerada a partir das bordas calculadas e exibidas utilizando o comando ‘imshow’ do OpenCV.

2. EXPLICAÇÃO DO PROGRAMA

Inicialmente, temos as funções as responsáveis por calcular as bordas nas direções x e y, utilizando os operadores de Sobel, sendo criado dois threads, no qual cada um será responsável por realizar o cálculo em cada direção.

Figura 1: Funções responsáveis por calcular as bordas na direção x e y.

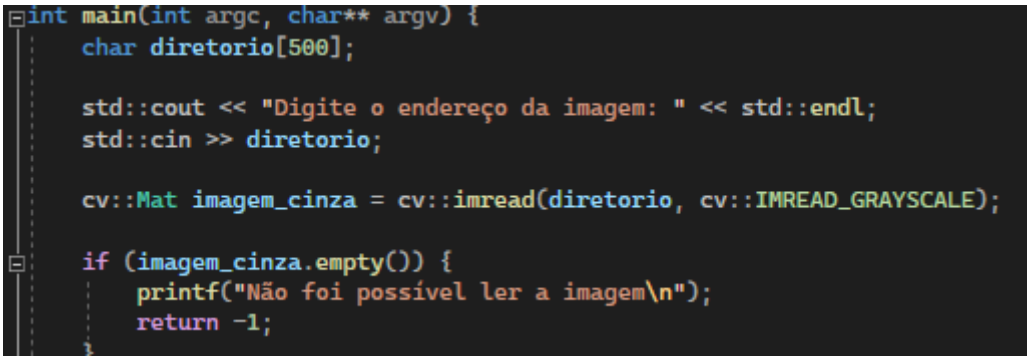
```
// Função para calcular a borda na direção x
void* calcular_borda_x(void* arguments) {
    ThreadArgs* args = (ThreadArgs*)arguments;
    cv::Mat imagem = args->imagem;
    cv::Mat Gx = args->Gx;

    for (int i = 1; i < imagem.rows - 1; ++i) {
        for (int j = 1; j < imagem.cols - 1; ++j) {
            Gx.at<uchar>(i, j) = abs(imagem.at<uchar>(i + 1, j - 1) + 2 * imagem.at<uchar>(i + 1, j) +
                imagem.at<uchar>(i + 1, j + 1) - imagem.at<uchar>(i - 1, j - 1) -
                2 * imagem.at<uchar>(i - 1, j) - imagem.at<uchar>(i - 1, j + 1));
            Gx.at<uchar>(i, j) = std::min(255, std::max(0, (int)Gx.at<uchar>(i, j)));
        }
    }

    pthread_exit(NULL);
}

// Função para calcular a borda na direção y
void* calcular_borda_y(void* arguments) {
    ThreadArgs* args = (ThreadArgs*)arguments;
    cv::Mat imagem = args->imagem;
    cv::Mat Gy = args->Gy;

    for (int i = 1; i < imagem.rows - 1; ++i) {
        for (int j = 1; j < imagem.cols - 1; ++j) {
            Gy.at<uchar>(i, j) = abs(imagem.at<uchar>(i - 1, j + 1) + 2 * imagem.at<uchar>(i, j + 1) +
                imagem.at<uchar>(i + 1, j + 1) - imagem.at<uchar>(i - 1, j - 1) -
                2 * imagem.at<uchar>(i, j - 1) - imagem.at<uchar>(i + 1, j - 1));
            Gy.at<uchar>(i, j) = std::min(255, std::max(0, (int)Gy.at<uchar>(i, j)));
        }
    }

    pthread_exit(NULL);
}
```

Fonte: Autores

Utilizando do OpenCV para conseguirmos trabalhar com imagens, dessa forma conseguimos fazer a leitura da imagem de entrada utilizando a função imread, que recebe o parâmetro do endereço da imagem e como vai realizar a

leitura da imagem. O segundo parâmetro “IMREAD_GRAYSCALE” serve para ler toda imagem de entrada em tons de cinza.

Figura 2: Leitura da imagem de entrada.

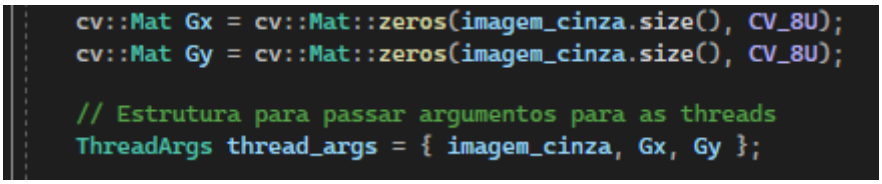


```
int main(int argc, char** argv) {
    char diretorio[500];

    std::cout << "Digite o endereço da imagem: " << std::endl;
    std::cin >> diretorio;

    cv::Mat imagem_cinza = cv::imread(diretorio, cv::IMREAD_GRAYSCALE);

    if (imagem_cinza.empty()) {
        printf("Não foi possível ler a imagem\n");
        return -1;
    }
```

Fonte: Autores.

Seguindo essa linha, deve-se inicializar as matrizes responsáveis por enviar valores necessários para as funções “calcular_borda”, para isso usa-se a função “zeros” da biblioteca do OpenCV, onde inicia as matrizes zeradas e recebem os tamanhos da imagem enviada. Esses parâmetros, junto com a imagem, são passados como argumentos para as threads.

Figura 3: Inicialização das estruturas para passar como argumento.

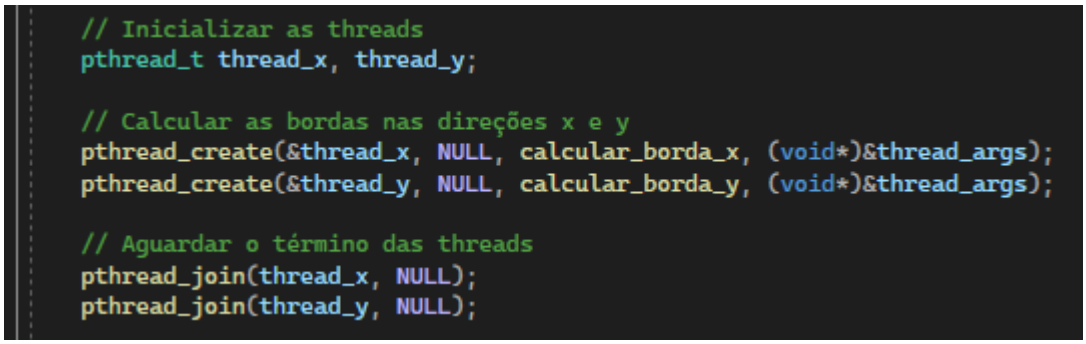


```
    cv::Mat Gx = cv::Mat::zeros(imagem_cinza.size(), CV_8U);
    cv::Mat Gy = cv::Mat::zeros(imagem_cinza.size(), CV_8U);

    // Estrutura para passar argumentos para as threads
    ThreadArgs thread_args = { imagem_cinza, Gx, Gy };
```

Fonte: Autores.

Concluindo, temos a criação e execução das threads, as quais serão as grandes responsáveis por realizar o cálculo das bordas, chamando as funções de calculo de borda, descritas anteriormente. A criação das threads se dá pela função pthread_create e utiliza-se a pthread_join para garantir que as threads vão conseguir terminar os cálculos das borda de forma paralela, garantindo um bom desempenho computacional.

Figura 4: Criação e execução das threads.



```
    // Inicializar as threads
    pthread_t thread_x, thread_y;

    // Calcular as bordas nas direções x e y
    pthread_create(&thread_x, NULL, calcular_borda_x, (void*)&thread_args);
    pthread_create(&thread_y, NULL, calcular_borda_y, (void*)&thread_args);

    // Aguardar o término das threads
    pthread_join(thread_x, NULL);
    pthread_join(thread_y, NULL);
```

Fonte: Autores

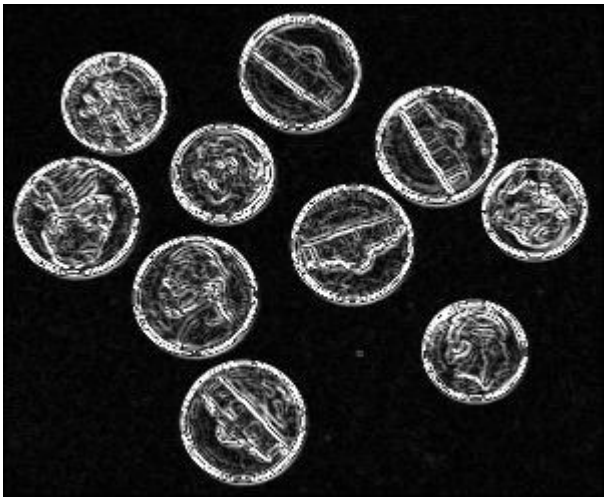
Por fim, utiliza-se a função ‘add’ do OpenCV, para adicionar os resultados na imagem final e mostra o resultado a operação pela ‘imshow’ e salva o resultado pela ‘imwrite’.

Figura 5: Final do código

```
// Calcular a imagem de saída G
cv::Mat imagem_saida;
cv::add(Gx, Gy, imagem_saida);

// Exibir a imagem de saída
cv::imshow("Imagem de Saída", imagem_saida);
cv::imwrite("ImagemSaida.png", imagem_saida);
cv::waitKey(0);
```

Fonte: Autores

Figura 6: Resultado do código



Fonte: Autores

3. CONCLUSÃO

Durante o desenvolvimento deste projeto, foi-se notado a vantagem do uso da programação multithreading, pois a thread é conhecida como processo leve, em que a ideia é alcançar o paralelismo, dividindo um processo em várias threads.

Utilizando da biblioteca sugerida na turma visual e seguindo os arquivos disponibilizados, foi possível adquirir o conhecimento e tirar as dúvidas sobre a biblioteca.

4. LINK GITHUB

https://github.com/KempesBarbosa/Sistemas-Operacionais/blob/main/Thread/Threds.cpp

REFERÊNCIAS

LABORATORY, B. B., Lawrence Livermore National. **POSIX Threads Programming**. Disponível em: <https://hpc-tutorials.llnl.gov/posix/>.

BRITO, Agostinho. **Introdução ao processamento digital de imagens com OpenCV**. Disponível em: <https://agostinhobritojr.github.io/tutorial/pdi/>.