Implementação com visualização gráfica e duas linguagens de programação

Isabele Santos Scherdien

Especificação: Implementar uma aplicação gráfica (sugestão: fractal de Mandelbrot, ray tracing, simulação de partículas) com apoio de duas linguagens de programação: Python e C (ou C++). O desafio consiste em realizar o uso conjunto de duas linguagens de programação, sendo o desafio especificamente, realizar o uso conjunto das linguagens. Python deve ser utilizado para oferecer uma interface com o usuário e apresentar a imagem gerada. A linguagem C (ou C++) deve ser utilizada para implementar o serviço de cálculo desejado.

---

1. Introdução

Este projeto mostra a integração entre diferentes linguagens de programação, utilizando a linguagem C para implementar algoritmos de ordenação e Python para a visualização desses algoritmos. O foco principal é no algoritmo de ordenação por inserção (*Insertion Sort*), que é implementado em C e visualizado em tempo real usando Python e a biblioteca matplotlib. Os algoritmos de ordenação já foram implementados para uma cadeira anterior.

2. Estrutura do Projeto

O projeto é estruturado da seguinte forma:

- **biblioteca.c**: Este arquivo contém as implementações dos algoritmos de ordenação, incluindo insertion_step, além de funções auxiliares como PopularSqc e embaralhar.
- **main.py**: O script Python que interage com o código C. Ele utiliza a biblioteca ctypes para chamar as funções de ordenação e mostra o processo utilizando gráficos.
- **biblioteca.h e Makefile.**

3. Implementação em C

3.1 Algoritmo de Ordenação por Inserção

A função insertion_step implementa um único passo do algoritmo de ordenação por inserção. Isso permite a chamada repetitiva da função para demonstrar cada etapa do processo de ordenação:

```c
void insertion_step(int* vet, int vet_size, int step) {
    if(step < 1 || step >= vet_size)
        return;

    int key = vet[step];
    int j = step - 1;

    while (j >= 0 && vet[j] > key) {
        vet[j + 1] = vet[j];
        j = j - 1;
    }
    vet[j + 1] = key;
}
```

3.2 Funções Auxiliares

- PopularSqc: Popula um vetor com valores sequenciais para facilitar a visualização ordenada.
- embaralhar: Embaralha o vetor para simular uma lista não ordenada.

4. Visualização em Python

4.1 Interação com C usando ctypes

O script Python carrega a biblioteca C e define os tipos de argumentos das funções a serem chamadas. Isso é necessário para garantir que os dados sejam passados corretamente entre C e Python.

```python
def main():
    #Carrega a biblioteca compartilhada do C
    lib = ctypes.CDLL('build/biblioteca.so')

    vet_size = 100
    vet = (ctypes.c_int * vet_size)()

    # Define os tipos de argumentos da função C
    lib.insertion_step.argtypes = (ctypes.POINTER(ctypes.c_int), ctypes.c_int, ctypes.c_int)

    # Chama a função C para popular e embaralhar o vetor
    lib.PopularSqc(vet, vet_size)
    lib.embaralhar(vet, vet_size)

    # Chama a função para plotar o gráfico
    plot(vet, vet_size, lib)
```

4.2 Visualização

A visualização é realizada passo a passo. O script limpa o gráfico anterior, chama insertion_step para o próximo passo, atualiza o gráfico e repete até que o vetor esteja ordenado:

```python
def plot(vet, vet_size, lib):
    plt.ion()  # Habilita o modo interativo do matplotlib

    for step in range(1, vet_size):
        plt.clf()  # Limpa o gráfico

        # Chama a função C para realizar um passo na ordenação
        lib.insertion_step(vet, vet_size, step)

        vet_list = [vet[i] for i in range(vet_size)]

        # Re plota o grafico no estado atual
        plt.bar(range(vet_size), vet_list)
        plt.draw()
        plt.pause(0.01)

    plt.ioff()
    plt.show()
```

5. Compilação do Programa

5.1 Flags de Compilação

-shared: Esta opção indica ao GCC que o arquivo de saída deve ser uma biblioteca compartilhada (também chamada de "shared object" ou .so). As bibliotecas compartilhadas podem ser carregadas em tempo de execução por executáveis ou outras bibliotecas, e são utilizadas para economizar memória e espaço em disco, uma vez que múltiplos processos podem compartilhar a mesma instância da biblioteca em memória.

-fPIC: Esta opção instrui o compilador a gerar código que pode ser executado em qualquer endereço de memória, o que é crucial para bibliotecas compartilhadas. Como as bibliotecas compartilhadas podem ser carregadas em diferentes localizações de memória em diferentes processos, o código gerado deve ser independente de sua localização.

Diferença do Output com uma Compilação "Normal" Quando você compila um programa usando gcc sem as opções -shared e -fPIC, o compilador geralmente gera um executável estático ou dinâmico que é vinculado no momento da compilação (com gcc arquivo.c -o executavel). Esse executável tem um código que espera ser carregado em endereços de memória específicos e geralmente inclui todo o código necessário (ou depende de bibliotecas compartilhadas que já estão presentes no sistema).

Código Posicionalmente Independente (-fPIC): Para uma biblioteca compartilhada, o código gerado com -fPIC pode ser carregado em qualquer endereço de memória. Isso é diferente de um executável "normal", que geralmente é gerado para ser carregado em um endereço de memória fixo, especificado pelo linker.

Com -shared, o GCC gera uma biblioteca que é dinamicamente vinculada. Isso significa que quando um programa que usa essa biblioteca é executado, a ligação (linkagem) entre o programa e a biblioteca acontece em tempo de execução, permitindo que várias instâncias do programa compartilhem o mesmo código. Sem -shared, o GCC pode gerar um executável que tem todo o código necessário incluído nele, o que é conhecido como linkagem estática.

Documentação:

https://gcc.gnu.org/onlinedocs/gcc-14.2.0/gcc.pdf