CENTRO UNIVERSITÁRIO DA FUNDAÇÃO HERMÍNIO OMETTO

CURSO DE ENGENHARIA DE COMPUTAÇÃO

Gabriel Lisboa Alves RA:101114

Rodrigo Almeida Rocha de Sousa RA: 99227

Vinicius Henrique Otti Masson RA:102678

SIMULADOR DE ESCALONAMENTO DE PROCESSOS

ARARAS/SP

2024

Sumário

1.Introdução

Um escalonador de processos consiste em um componente fundamental para gerenciar a execução dos processos do sistema operacional. Através da lógica do algoritmo implementado, o escalonador é capaz de ordenar a execução e alocar o tempo da CPU para cada processo. O escalonador tem que ser preciso, pois ele garante que o sistema opere de maneira eficiente e justa

2. Metodologia

A implementação do escalonador de processos foi estruturada de forma a simular um ambiente de gerenciamento de processos utilizando diferentes políticas de escalonamento. O objetivo é permitir a criação, manipulação e execução de processos com várias características, bem como comparar o desempenho das diferentes políticas de escalonamento. A seguir, são descritos os principais passos e decisões tomadas durante o desenvolvimento:

2.1 Diagrama de classes

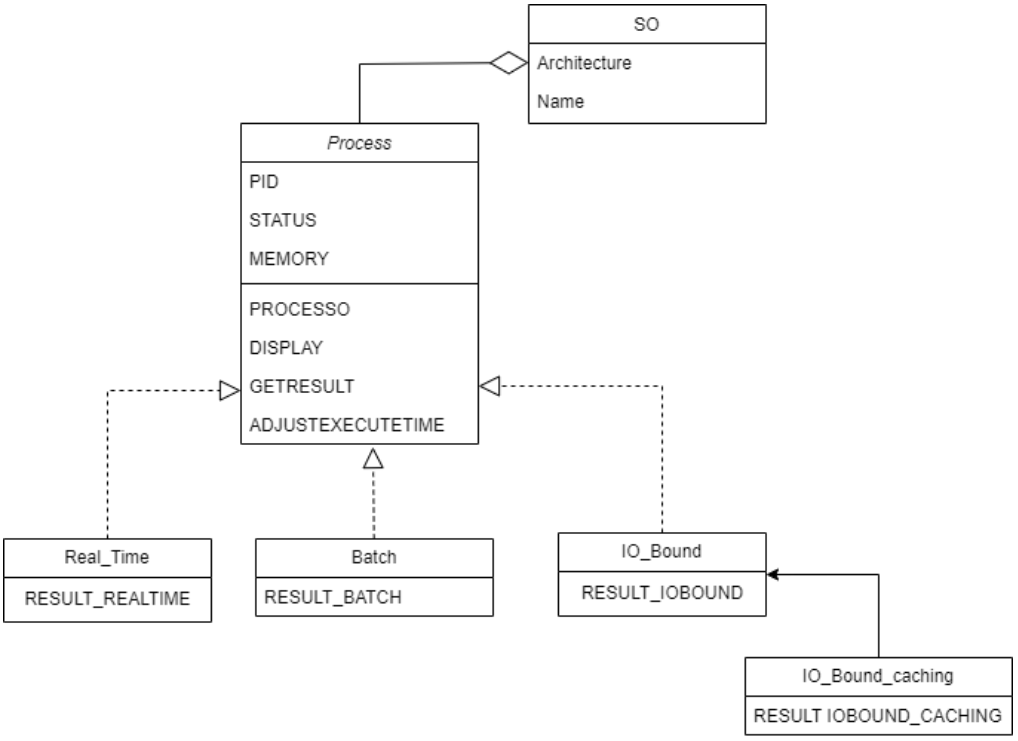

Figura 1 - Diagrama de Classes

2.2. Definição e implementação das classes

Foram definidas várias classes para representar o sistema operacional (SO), processos e suas variações específicas, seguindo o princípio da orientação a objetos.

- Classe **SO**: Representa o sistema operacional com atributos como arquitetura e nome.
- Classe **Processo**: Classe base abstrata que representa um processo genérico com atributos PID, Status e Memory. Inclui os métodos virtuais puro getResult, display, ajustexecutetime que deve ser implementado pelas classes derivadas.
- Classes Derivadas de Processo:
    a. **Real_Time**: Representa processos de tempo real, armazenando o resultado de tempo real.
    b. **Batch**: Representa processos em lote, armazenando o resultado do lote.
    c. **IO_Bound**: Representa processos dependentes de E/S, armazenando o resultado de E/S.
    d. **IO_Bound_Caching**: Estende IO_Bound para incluir resultados específicos de cache. Esta classe foi implementada para simular processos que fazem uso intensivo de operações de entrada/saída e que também possuem operações de cache, o que pode afetar seu desempenho de maneira diferente dos processos IO_Bound comuns.

2.3. Criação e inicialização dos processos

Para simular um ambiente realista, os processos são gerados aleatoriamente com diferentes características:

- Utiliza-se a biblioteca <random> para gerar valores aleatórios para atributos como tipo de processo, PID, memória, status e resultados.
- O número de processos é determinado pelo usuário, que é solicitado a inserir a quantidade desejada de processos a serem criados.

2.4. Visualização dos processos

Foi implementada a função displayProcessList que percorre a lista de processos e chama o método display de cada objeto, exibindo as características dos processos no console.

2.5. Execução e comparação dos algoritmos de escalonamento

O código permite a execução dos processos utilizando diferentes algoritmo para escalonamento:

- SJF (Shortest Job First): Os processos são ordenados pelo menor tempo de execução primeiro.
- SRTF (Shortest Remaining Time First): Similar ao SJF, mas simula interrupções e reordenação de processos.
- FIFO (First In, First Out): Os processos são executados na ordem em que foram criados, sem reordenação.
- RR (Round Robin): Round Robin foi implementado basicamente para simular fatias de tempo (time slices)

A função executeProcesses é responsável por ordenar os processos de acordo com a política selecionada e calcular o tempo total de execução e o tempo de espera de cada processo. Este cálculo é feito para cada política individualmente, permitindo comparações posteriores.

2.6. Interface do usuário

Foi desenvolvida uma interface simples em modo texto para interação com o usuário:

- O usuário pode escolher entre mostrar a lista de processos, selecionar uma política de execução, comparar os tempos de execução ou sair do programa.
- As escolhas do usuário são tratadas em um loop que permite múltiplas interações até que o usuário decida sair.

2.7. Limpeza de memória

Para evitar vazamento de memória, todos os objetos de processos criados dinamicamente são deletados ao final da execução do programa.

3.Fluxo de execução

A. Inicialização: Solicita ao usuário o número de processos a serem criados e gera processos com atributos aleatórios.

B. Interação com o Usuário: Oferece opções para visualizar processos, executar processos com diferentes políticas de escalonamento e comparar resultados.
C. Execução do algoritmo: Ordena e executa processos de acordo com o algoritmo selecionado, calculando tempos de execução e espera.
D. Comparação e Exibição: Compara os tempos de execução das diferentes algoritmos(após todas terem sido executadas) e exibe os resultados.
E. Finalização: Limpa a memória alocada dinamicamente e encerra o programa.

4.Conceitos Utilizados

- **Classes e Objetos:** Definição de Classes: As classes são definidas para modelar o comportamento de diferentes tipos de processos (SO, Processo, Real_Time, Batch, IO_Bound, IO_Bound_Caching).Instanciação de Objetos: Objetos das classes derivadas são instanciados e armazenados em um vetor de ponteiros para a classe base Processo.
- **Abstração:** Classe Abstrata: A classe Processo é uma classe abstrata devido ao método puramente virtual getResult, que deve ser implementado pelas classes derivadas.Simplificação da Complexidade: A abstração é usada para simplificar a complexidade do programa, fornecendo uma interface comum para diferentes tipos de processos.
- **Destrutores:** Destrutor Virtual: O destrutor na classe Processo é declarado como virtual para garantir que o destrutor apropriado da classe derivada seja chamado, permitindo a correta liberação de recursos.
- **Manipulação de Ponteiros e Memória Dinâmica:** Uso de Ponteiros: Ponteiros para a classe base Processo são usados para armazenar objetos de classes derivadas.Alocação Dinâmica: new é usado para criar objetos dinamicamente, e delete é usado para liberar a memória.
- **Polimorfismo**: A classe base Processo define métodos virtuais que são sobrecarregados nas classes derivadas (Real_Time, Batch, IO_Bound, IO_Bound_Caching). Isto permite que o código trate objetos de diferentes tipos de processo de maneira uniforme, chamando métodos como display e getResult de forma polimórfica.
- **Herança**: A classe IO_Bound_Caching herda da classe IO_Bound, estendendo sua funcionalidade ao adicionar um resultado específico de cache. A herança

permite a reutilização de código e a criação de hierarquias de classes que representam processos com características adicionais ou especializadas.

- **Vector**: Nesse projeto o vetor está sendo usado por agregação. Onde é caracterizada pela relação de "tem um", onde os objetos contidos (neste caso, objetos do tipo Processo* dentro do vector<Processo*>) podem existir independentemente do objeto que os contém

5.Bibliotecas aparte utilizadas

Neste tópico será abordado as bibliotecas que não foram vistas em aula, porém foram essenciais para o desenvolvimento do projeto de escalonamento de processos.

- **<algorithm>** utilizada na implementação para simplificar a manipulação e ordenação de coleções, especificamente o vetor de processos, o uso desta biblioteca foi fundamental para implementar as políticas de escalonamento.

- **<random>** foi utilizada para gerar valores aleatórios que simulam características variadas dos processos, como tipo, PID, memória, status e tempo de execução. Isso garante uma simulação realista e diversificada, essencial para testar e analisar as políticas de escalonamento em condições similares às de um sistema operacional real.

6.Conclusão

O desenvolvimento deste código de simulação de processos de um sistema operacional ilustra a aplicação prática dos conceitos fundamentais de programação orientada a objetos. Ao longo do desenvolvimento, foram utilizados princípios como encapsulamento, herança, polimorfismo e abstração, cada um contribuindo de maneira significativa para a estrutura e funcionalidade do programa. Este projeto tem aplicações práticas em áreas como simulação de sistemas operacionais, ensino de conceitos de OOP e desenvolvimento de software orientado a objetos. Com algumas modificações, o sistema poderia ser expandido para incluir funcionalidades mais avançadas, como escalonamento de processos em tempo real, análise de desempenho e visualização gráfica.