Relatório do Trabalho 1 de Sistemas Operacionais I

Maurício Konrath e Nicolas Elias

Simulação de Algoritmos de Escalonamento

Florianópolis

2023-2

Resumo

Este relatório documenta o desenvolvimento e os resultados do Trabalho 1 de Sistemas Operacionais 1. O objetivo deste trabalho era desenvolver os escalonadores vistos em aula de processos em C++ e realizar uma análise de desempenho com base em métricas como turnaround time, tempo médio de espera e número de trocas de contexto.

Introdução

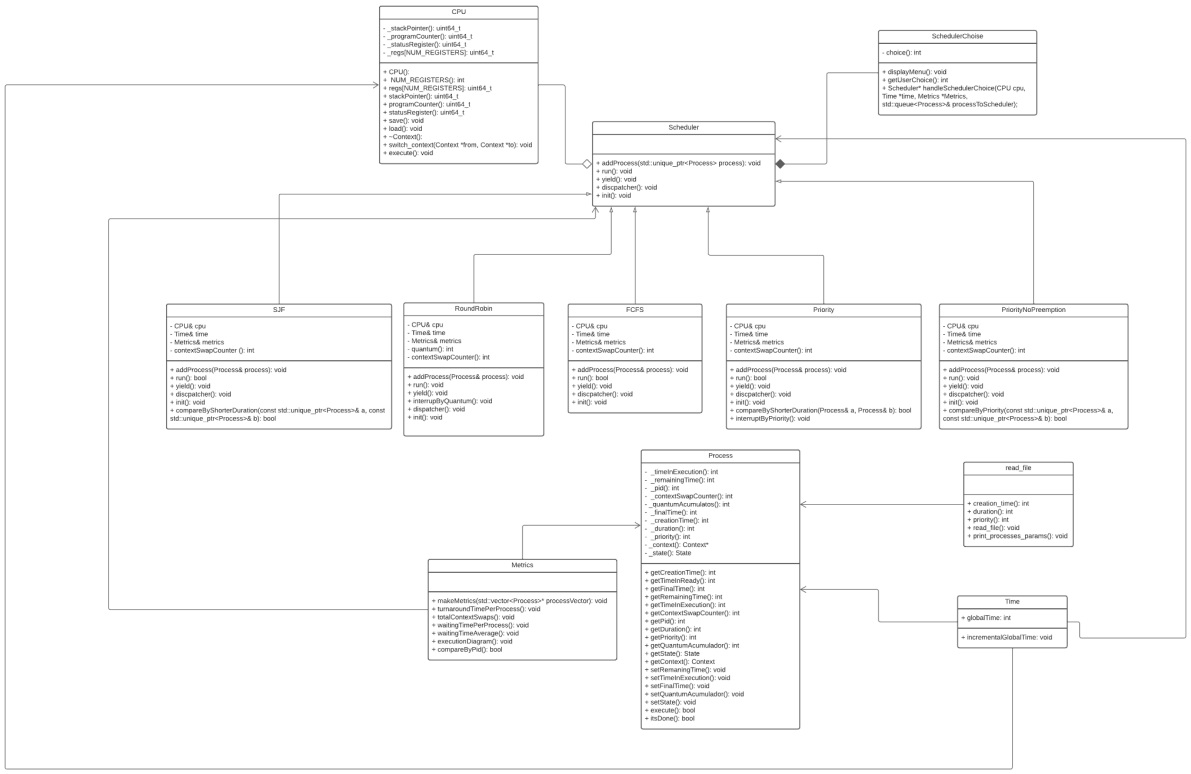
Neste trabalho, fomos desafiados a criar um escalonador de processos em C++ que implementasse vários algoritmos de escalonamento, incluindo FCFS (First-Come, First-Served), SJF (Shortest Job First), Priority with Preemption, Priority without Preemption e Round Robin. A tarefa principal era desenvolver os escalonadores e executá-los em um conjunto de processos de teste, coletando métricas de desempenho para análise.

Metodologia

Desenvolvemos cada um dos escalonadores solicitados em C++, criando classes separadas para cada um deles. A estrutura geral do código incluiu classes para representar processos, escalonadores e a lógica de simulação da CPU.

Estrutura

Na imagem 1 pode-se observar o diagrama UML, produzido para o trabalho a fim de se ter mais clareza em relação ao código produzido



(clique aqui para abrir)

Imagem 1 - Diagrama UML

Organização

- *Process*: Define a estrutura e a lógica de um processo dentro do contexto da simulação de escalonamento de processos. Ele mantém informações importantes sobre os processos, como seus estados, tempos e prioridades, e permite a simulação de sua execução durante a simulação. Essa classe é essencial para acompanhar o comportamento dos processos ao longo do tempo durante a simulação.
- *Scheduler*: serve como um ponto de partida para a implementação de algoritmos específicos de escalonamento, como FCFS, SJF, Round Robin, etc. Cada algoritmo

derivado deve implementar esses métodos conforme a sua lógica estabelecida na classe pai.

- *FCFS, SJF, Priority, PriorityNoPreemption, RoundRobin*: Classes que herdam da classe *Scheduler* e implementam a lógica de escalonamento específica para cada algoritmo.
- *CPU*: Abstração de uma cpu, responsável por simular a execução dos processos e a troca de contexto entre eles, garantindo que os processos sejam executados de acordo com a política de escalonamento escolhida.
- *Metrics*: É responsável por calcular métricas de desempenho para processos em um sistema de escalonamento. Ele inclui funções para calcular o tempo de retorno, o tempo médio de retorno, o número total de trocas de contexto, o tempo de espera e o tempo médio de espera para os processos. Além disso, ele gera um diagrama de execução que mostra como os processos são executados ao longo do tempo.
- *SchedulerChoise*: Implementa a escolha de um escalonador a partir de um menu interativo. Ele exibe um menu com cinco opções diferentes de escalonadores e permite ao usuário escolher um deles digitando o número correspondente. O código usa um sistema de seleção com switch para criar a instância correta do escalonador com base na escolha do usuário e imprime uma mensagem indicando qual escalonador foi escolhido.
- *Time*: Usado para rastrear o tempo (valor inteiro) global em uma simulação de escalonamento de processos.

Execução

Executamos os escalonadores em um conjunto de processos de teste. Os processos de teste foram definidos em um arquivo de entrada no formato especificado, contendo informações sobre a data de criação, duração e prioridade de cada processo. Um módulo de leitura de arquivo foi implementado para processar o arquivo de entrada e criar instâncias de objetos *Process* com os detalhes de cada processo. Os processos foram então adicionados aos escalonadores relevantes usando os métodos *addProcess*.

Coleta de Dados

Durante a execução dos escalonadores, coletamos dados de desempenho, incluindo:

- Turnaround time para cada processo.
- Tempo médio de espera para todos os processos.
- Número total de trocas de contexto.

Dificuldades Enfrentadas

A principal dificuldade encontrada foi o uso de ponteiros, uma vez que não estamos acostumados a utilizar a linguagem C++ durante o curso. Além disso, a falta de paralelismo torna o programa inflexível na hora de distribuir a execução, uma vez que precisamos realizar todas as tarefas de forma uniforme. Outro desafio foi a alocação de memória com o uso de unique_ptr, com a expectativa de que a memória seja desalocada automaticamente, o que nem sempre ocorreu como planejado, ao rodar teste de vazamento acusa uns erros onde não obtivemos exito ao resolver. Outra questão que enfrentamos foi a montagem da string para o diagrama de execução, onde o algoritmo não atendeu às nossas expectativas devido à falta de refinamento.

Conclusão

Neste projeto, desenvolvemos um sistema de escalonamento de CPU em C++ que incluiu a implementação de cinco algoritmos de escalonamento: *FCFS, SJF, Priority, Priority sem preempção e Round Robin*. Criou-se uma estrutura de classes, utilizando conceitos de orientação a objetos. O projeto também envolveu a criação de um diagrama UML abrangente para representar a arquitetura do sistema e o cálculo de métricas essenciais, como tempo médio de espera e tempo de retorno. No final, este projeto proporcionou uma compreensão prática dos conceitos fundamentais de escalonamento de CPU em sistemas operacionais.

Referências

Material disponibilizado pelos professores.