Trabalho Prático do Grau B

Descrição

O problema proposto é uma simulação de execução de um pool de processos. O pool consiste em numa fila de processos que é incrementada pelo usuário e que serão executados num segundo momento, também por opção do usuário.
Os processos são inseridos pelo usuário um-a-um e sempre no final da fila dos processos (portanto, na próxima posição livre do array). Os processos também devem ser executados na ordem em que foram cadastrados, a menos que o usuário solicite um específico pelo seu pid. Neste caso, portanto, um processo pode ser executa antes do que o próximo a ser executado de direito. Quando isto acontece os demais processos, sucessores do escolhido devem "andar" uma posição a frente no array.
Existem 4 tipos específicos de processos. Cada tipo implementa um algoritmo para execução. Eles são especializados nos seguintes tipos: processo de gravação, processo de leitura, processo de impressão e processo de cálculo. Veja abaixo o que deve ser executado no método executar de cada tipo.
Quanto aos atributos, **todo** processo (na sua forma mais geral) possui um **pid** e tem o método **execute()**. A superclasse processo não deve implementar o execute() apenas deve declará-la. Cada subclasse específica de processo deve fazer a sua implementação do método **execute()**.
Abaixo seguem as especificações sobre o que cada método **execute** de cada subtipo de processo deve fazer quando for solicitado pelo usuário:

- Processo de cálculo (*ComputingProcess*): executa o cálculo de uma expressão e imprime o resultado do cálculo. Uma expressão é formada por dois operandos e uma operação (que pode ser +, -, * ou /). A expressão deve saber como se executar. Por exemplo, se é uma soma, então o resultado do cálculo é a soma dos dois operandos.
- Processo de gravação (*WritingProcess*): executa a gravação de uma expressão em um arquivo de processos (chamado computation.txt). Note que, se já existem expressões gravadas no arquivo, o processo de gravação não deve sobrescrevê-las.
- Processo de leitura (*ReadingProcess*): deve ler completamente o arquivo de computações (computation.txt) e, para cada registro lido do arquivo deve criar um objeto de processo de cálculo (*ComputingProcess*) e **adicioná-lo** na lista de processos do sistema. Ao final da leitura, o processo deve "limpar" o arquivo. Vale lembrar que cada linha do arquivo é uma expressão aritmética.
- Processo de impressão (*PrintingProcess*): tem por objetivo simplesmente imprimir na tela o pool de processos a serem executados. Imprimindo o pid, o tipo do processo e atributos relacionados, se for o caso.

Atividades que devem ser implementadas no trabalho:

- Implementar as classes relacionadas a processo.
- Implementar as seguintes opções de menu do **sistema**:
  - **Criar processo**: permite ao usuário criar um processo de um dos quatro tipos específicos. Portanto, deve solicitar ao usuário o tipo de processo a ser criado e os dados necessários para tipo de processo. Por exemplo, para um processo do *ComputingProcess*, deve ser lida e montada a expressão a ser executada. Esta opção do sistema apenas cria um objeto e o adiciona no final do array.
  - **Executar próximo**: executa o próximo processo na ordem em que foram criados. Deve também remover o processo executado e atualizar a lista. Portanto, executar sempre o processo do índice zero do array (o primeiro), depois de executado removê-lo e, por fim, trazer os sucessores uma posição a frente no array de processos.
  - **Executar processo específico**: deve solicitar ao usuário o pid do processo a ser executado, procura o processo pelo pid informado e, caso tenha encontrado, executa este processo mesmo que ele não seja o primeiro da fila/array. Após a execução o processo deve ser removido e os seus sucessores devem ocupar uma posição a frente no array.
  - **Salvar a fila de processos**: salvar em arquivo o estado atual da fila de processos em Arquivo.
  - **Carregar do arquivo a fila de processos**: inicializar o sistema com um array de processos do arquivo.

Dicas

- Vamos assumir um tamanho físico máximo estático para o array. Por exemplo, 100 elementos.
- A lista de processos não deve ficar com "buracos". Isto significa que quando um processo é removido os processos seguintes devem ser movidos uma posição à frente. Por exemplo, se o primeiro processo da lista (o do índice zero do array) é removido, o segundo deve ser movido para a primeira posição, o terceiro para a segunda e assim por diante; até o fim da lista.
- Sugiro colocar um parâmetro no construtor da classe *ComputingProcess* para receber uma expressão como uma string e quebrá-la em seus termos (operandos e operador).
- Sugiro passar por parâmetro no construtor das classes *ReadingProcess* e *PrintingProcess* uma referência para lista de processos do sistema. Uma vez que eles devem interagir com o array.

Considerações finais

- Será avaliada como qualidade de código a correta separação das responsabilidades de cada classe.
- Trabalho pode ser feito em duplas.
- **É de responsabilidade do aluno buscar maiores esclarecimentos e ajuda no desenvolvimento do trabalho com o professor. Normalmente, os melhores trabalhos são mostrados desde o início ao professor, que assim, pode corrigir em tempo hábil, eventuais desvios de andamento.**
- A entrega consiste em: arquivos de projeto e arquivo de modelagem.
- **O prazo de entrega é 02/07.**
    - Enviar somente os arquivos fonte do projeto para a atividade aberta no Canvas **até às 19h30min do dia 02/07/2024.** Apenas um integrante do grupo envia os arquivos. Importante: não sendo entregue até a data prevista a nota passa a ser ZERO.

Avaliação

- **Modelagem do problema (10%)**
- **Classe *Process* e subclasses (30%)**
- **Salve em arquivo (20%)**
- **Classe Sistema (40%). Que se divide em:**
    - Criar processo (10%)
    - Executar próximo (10%)
    - Executar um processo específico (10%)
    - Funcionalidades em geral (10%)