ACH2044 - Sistemas Operacionais

Prof. Norton Trevisan Roman

Relatório EP 01

Felipe Oliveira do Espirito Santo – 11925242

Fernando José Germinio Fenley - 11207630

Guilherme Jimenes - 11911021

Jalmir Iago Alves da Silva - 11808999

1. Definição das classes do projeto

Como decisão de projeto, foram implementadas as seguintes classes para o projeto:

BCP

Responsável por gerenciar as informações gerais do processo, como nome, estado, código (que consiste em um array de strings), registradores de uso específico (CP) e de uso geral (X, Y), além de um atributo privado “espera” que armazena o tempo de espera de um processo bloqueado que realizou uma instrução de E/S até que ele possa entrar na fila de processos prontos. Ainda, esta classe possui um atributo estadoProcesso que consiste em um Enum de estados do processo, em que 0 corresponde a Executando, 1 corresponde a Pronto e 2 corresponde a Bloqueado.

Com isso, a classe possui os seguintes atributos:

- private String nomeProcesso;
- private int CP;
- private int X;
- private int Y;
- private EstadoProcesso estadoProcesso;
- private String[] codigo;
- private int espera;
- private EstadoProcesso estadoProcesso.

Além dos getters e setters para os atributos, a classe possui o seguinte método:

- public void aumentaCP(): responsável por incrementar o registrador CP que corresponde ao contador de programa, ou seja, posição no array do código do processo da próxima instrução a ser executada.

Processo

Classe responsável por representar um processo a ser executado pelo escalonador. Contém como atributo uma instância da classe **BCP**. Como métodos, possui getters e setters.

TabelaDeProcessos

Classe responsável por armazenar uma lista com todos os processos a serem executados. Para isso, contém os seguintes atributos:

- private List<Processo> processos;

Os métodos da classe, além de getters e setters, são:

- public void addProcesso(Processo processo): responsável por adicionar um novo processo à lista de processos a serem executados.
- public void removeProcesso(Processo processo): responsável por remover um processo da lista de processos a serem executados.

LogWriter

Responsável por criar arquivos de logs, bem como escrever mensagens nesses arquivos. Por isso, alguns métodos podem lançar exceções do tipo IOException. De forma geral, a classe contém os seguintes métodos:

- public void criaArquivo(int quantumLido): responsável por criar os arquivos de log com o nome correspondente ao valor do quantum que foi especificado no arquivo quantum.txt
- private void escreveString(String str) throws IOException: responsável por escrever a String str recebida como parâmetro no arquivo de log.
- public void escreveCarregando(String nomeProcesso) throws IOException: responsável por solicitar a escrita de mensagens quando um arquivo com as informações e instruções dos processos está sendo lido.
- public void escreveES(String nomeProcesso) throws IOException: responsável por solicitar a escrita de mensagens quando o escalonador interpreta uma instrução de E/S.
- public void escreveInterrupcao(String nomeProcesso, int instrucoesExecutadas) throws IOException: método responsável por solicitar a escrita de mensagens quando ocorrer uma interrupção no escalonador.
- public void escreveFinalizou(String nomeProcesso, int regX, int regY) throws IOException: responsável por solicitar a escrita de mensagens quando um processo finalizar sua execução.
- public void escreveExecutando(String nomeProcesso) throws IOException: método responsável por solicitar a escrita de mensagens quando um processo estiver sido escalonado para execução no escalonador.
- public void escreveMediaTrocas(float mediaTrocas) throws IOException: método responsável por solicitar a escrita do valor da média de trocas de processos após o término da execução de todos os processos.
- public void escreveMediaIntrucoesPorQuantum(float mediaIntrucoesPorQuantum) throws IOException: método responsável por solicitar a escrita do valor da média de instruções realizados por quantum após o término da execução de todos os processos.

Report

Responsável por gerenciar as informações necessárias para cálculo das médias para os arquivos de logs. Assim, possui os seguintes atributos:

- private int qntTrocaProcesso: responsável por armazenar a quantidade de trocas de processos feitas pelo escalonador.
- private int qntIntrucoesPorQuantum: responsável por armazenar a quantidade de instruções realizados em cada quantum dos processos em execução.
- private final int qntProcessos: responsável por armazenar a quantidade de processos executados pelo escalonador.

Destacam-se os seguintes métodos:

- public void instrucaoExecutada(): incrementar o valor do atributo qntIntrucoesPorQuantum.
- public void trocaProcesso(): incrementar o valor do atributo qntTrocaProcesso.
- public float mediaIntrucoesPorQuantum(): calcular a média de instruções executadas por quantum.
- public float mediaTrocaDeProcessos(): calcular a média de troca de processos feitas pelo escalonador.

Escalonador

Classe principal, responsável por implementar a política de interrupções, bem como interpretar as instruções dos processos. A classe possui os seguintes atributos:

- private int quantum: atributo responsável por armazenar a quantidade de instruções que corresponde a um quantum do escalonador.
- private final TabelaDeProcessos tabProcessos: atributo responsável por armazenar uma instância única da classe TabelaDeProcessos
- private final List<Processo> processosProntos: atributo responsável por armazenar uma lista de objetos processo cujo estado corresponde a pronto para execução.
- private final List<Processo> processosBloq: atributo responsável por armazenar uma lista de objetos processo cujo estado corresponde à bloqueado.
- private final Processo processoExec: atributo responsável por armazenar a instância do processo em execução no escalonador.
- private final LogWriter logWriter: atributo responsável por armazenar uma instância da classe LogWriter para geração dos arquivos de log.

Para os métodos da classe, é válido destacar os seguintes métodos:

private void trataProcessosBloqueados(): responsável por decrementar o tempo de espera dos processos na fila de bloqueados bem como adicioná-los na fila de prontos quando a espera terminar. Este método é executado sempre que uma interrupção é gerada pelo escalonador.

private int carregaProcessosRAM() throws IOException(): responsável por ler os dados dos arquivos dos processos bem como inicializar o objeto BCP correspondente a cada processo e armazená-los na tabela de processos. Pode lançar uma exceção do tipo IOExepction caso haja algum erro na leitura dos arquivos. Retorna o valor do quantum lido no arquivo quantum.txt

public void execute() throws IOException(): método responsável por executar todo o projeto em si, ou seja, escalonar um processo da fila de prontos para execução, controlar a quantidade de instruções executadas pelo processo em execução a fim de não extrapolar seu quantum, adicionar processos na fila de prontos e bloqueados. Além disso, esse método instancia um objeto da classe **Report** que será responsável por gerenciar as informações necessárias para o cálculo das médias para os arquivos de log. Pode lançar uma IOException pois invoca o método carregaProcessoRAM().

2. Execução do projeto

Para executar o projeto de forma correta, realize os seguintes passos:

1. Descompacte a pasta 11925242.zip;
2. Com o terminal aberto no diretório que contém o arquivo Escalonador.java, execute os seguintes comandos:

   javac Escalonador.java

   java Escalonador

3. Análises e escolha do n_com

Para as análises, e conseguinte escolha do n_com, foram realizados os cálculos das médias para os seguintes valores de quantum:

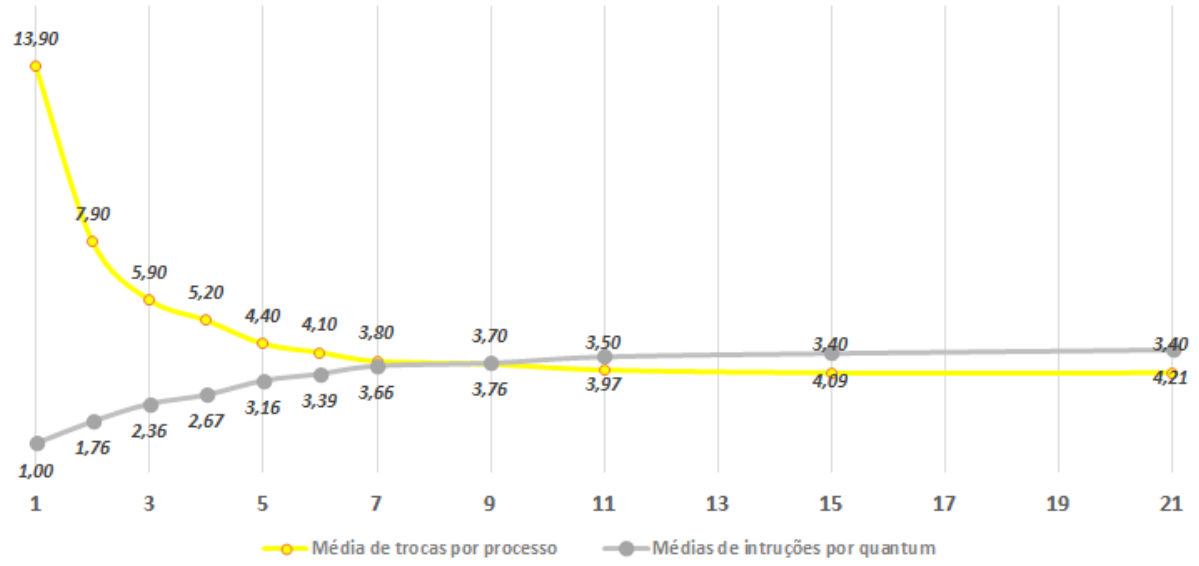
1	2	3	4	5	6	7	9	11	15	21

Assim, obtivemos os seguintes dados para as médias de troca por processo bem como de instruções por quantum:

Quantum	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**9**	**11**	**15**	**21**
Média trocas	13,90	7,90	5,90	5,20	4,40	4,10	3,80	3,70	3,50	3,40	3,40
Média instruções	1,00	1,76	2,36	2,67	3,16	3,39	3,66	3,76	3,97	4,09	4,21

Tais valores podem ser verificados nos arquivos de log gerados que estão contidos na pasta logs.

Por fim, geramos o seguinte gráfico para as médias quando cruzadas em relação ao quantum:



Para realizarmos a análise dos resultados obtidos, podemos começar pelos extremos. Pelo gráfico, para valores de quantum de 1 até 3, por exemplo, podemos observar uma intensa troca de processos feita pelo escalonador. O fato das médias de troca por processo serem 13,90, 7,90, 5,90 para esses valores de quantum, respectivamente, indica que o escalonador está constantemente realizando trocas, tornando a execução de programas ineficiente. Desse modo, não há um equilíbrio entre o esforço necessário para escalonar um processo para execução e o número de instruções que ele executa justamente quando está em execução.

Por outro lado, a análise do gráfico indica que quanto maior o valor do quantum oferecido aos processos, menor é a quantidade de trocas. Porém, a partir de certos valores de quantum, como superiores a 11 por exemplo, observa-se uma leve estagnação nos valores das médias, pois os processos que não realizam operações de E/S tendem a monopolizar o processador, pelo fato de sofrerem menos interrupções do escalonador, transformando a execução praticamente em um sistema batch o que não é o objetivo quando o intuito é trabalhar com multiprogramação. Assim, é

possível identificar que simplesmente aumentar a taxa de instruções, ou seja, o aumento do valor do quantum não garante eficiência depois de um certo valor.

Por fim, conclui-se que é necessário um equilíbrio justamente entre a taxa de instruções por processo e a quantidade de vezes em que ele é escalonado para execução, de forma a maximizar a eficiência do processo. Portanto, a partir dos resultados obtidos e das análises expostas, **o intervalo de 7-9 para o valor de n_com** é o mais recomendado.