Relatório- US 4071

4º Semestre

PROJETO LAPR4 – HELPDESK aService

António Fernandes 1190402

João Pereira 1190742

Rui Soares 1191045

Teresa Pereira Leite 1191072

Âmbito: Sistemas De Computadores

Docentes:

LLF (Luís Lino Ferreira)

PRP (Paulo Rogério Proença)

JPE (Joaquim Peixoto)

JRT (José Reis Tavares)

RFM (Rui Filipe Marques)

Conteúdo

Descrição da US

4071-Como Gestor de Projeto, eu pretendo que seja desenvolvido e integrado no Motor de Fluxos de Atividades algoritmos que distribuam a realização de tarefas automáticas pelas diversas instâncias do Executor de Tarefas Automáticas existentes na infraestrutura instalada.

Algoritmos Implementados

FCFS

```
   public void escalonar () {

      if (emEspera.size() > 0) {// em espera size é a queue com as listas de atividades automáticas
em espera
         imprimir WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA
      }
      for (ExecutorTarefasAutomaticas instancia: this.instancias){ //instancias é a lista que guarda
todas as instancias de executor de atividades automáticas
         if (!instancia.estaOcupada()){
            adicionar a lista temporária //se a instância nao estiver ocupada pode ser usada
                                         //a lista temporária guarda as instâncias disponíveis
         }
      }
      if (temp.isEmpty()) { // se não houverem instancias disponíveis
         WARNING_TUDO_OCUPADO
        adicionar atividade automática à fila de espera (emEspera)- queue
                              WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA

      } else {
         if (emEspera vazia)) {//Se não houver trabalhos em espera na queue
            marcar a instância como ocupada e aumentar a carga
                                                associar esta atividade automática (this.trabalho) à
instancia
            Thread thread = new Thread(temp.get(0)); // Create thread
            thread.start(); // Starts thread running at run()

                                         aguardar fim da execução para evitar que o programa termine
com tarefas por executar

         } else {
            if (this.trabalho != null) {
               adicionar a fila de espera (queue emEspera) a atividade automática (this.trabalho)
porque há atividade automática na fila e tem que ser aplicada a metodologia FCFS

            }
            for (para cada instancia disponível (na lista temp)) {
```

```
            if (emEspera.size() > 0) {se houver trabalhos em espera
               marcar a instância como ocupada e aumentar a carga
                                          associar a primeira atividade automática da queue
(emEspera.poll()) à instância removendo-a da queue

               Thread thread = new Thread(executorTarefasAutomaticas); // Create thread
               thread.start(); // Starts thread running at run()

                                          aguardar fim da execução para evitar que o programa
termine com tarefas por executar

            }
         }
         if (emEspera.size() > 0) {
            WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA;
         }
      }
   }
       limpar o atributo trabalho para evitar erros (this.trabalho=null)
  }
```

Carga e Disponibilidade em Conta

```
public void escalonar () {
    if (emEspera.size() > 0) {// em espera size é a queue com as listas de atividades automáticas
em espera
       imprimir WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA
    }

    for (ExecutorTarefasAutomaticas instancia: this.instancias){ //instancias é a lista que guarda
todas as instancias de executor de atividades automáticas
       if (!instancia.estaOcupada()) {
          adicionar a lista temporária //se a instância nao estiver ocupada pode ser usada
                                       //a lista temporária guarda as instâncias disponíveis
       }
    }

   ordenar as instâncias por ordem crescente de carga


    if (temp.isEmpty()) {
       WARNING_TUDO_OCUPADO
       adicionar atividade automática à fila de espera (emEspera)- List<>
                          WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA

    } else {
       if (emEspera.isEmpty()) {//Se não houver algum com prioridade este pode ser escalonado
          marcar a instância como ocupada e aumentar a carga
                                          associar esta atividade automática (this.trabalho) à
instancia
```

```
            Thread thread = new Thread(temp.get(0)); // Create thread
            thread.start(); // Starts thread running at run()

                                aguardar fim da execução para evitar que o programa termine com
tarefas por executar

          } else {
            if (this.trabalho != null) {
               emEspera.add(this.trabalho);
            }
            ordenar as atividades automáticas por ordem crescente de delay
                                        para assim as mais prioritárias executarem primeiro

            for (int i = 0; i < temp.size(); i++) {
               if (emEspera.size() > 0) {
                  marcar a instância como ocupada e aumentar a carga
                                              associar a primeira atividade automática da queue
(emEspera.remove(0))
                                              à instância removendo-a da List<>

                  Thread thread = new Thread(temp.get(i)); // Create thread
                  thread.start(); // Starts thread running at run()


                  aguardar fim da execução para evitar que o programa termine com tarefas por
executar
               }
            }
            if (emEspera.size() > 0) {
               WARNING_ALGUMAS_ATIVIDADES_EM_ESPERA;
                                              }
          }
       }
                  limpar o atributo trabalho para evitar erros (this.trabalho=null)
    }
```

Análise dos algoritmos

Os algoritmos foram testados num projeto separado porque foram implementados numa fase em que não existiam as condições necessárias a testar. No entanto, tentou-se simular o máximo de situações que poderiam ocorrer que foi possível, desde a demora, carga baixa, média e pesada conforme apresentado abaixo.

IDE: intellij IDEA

Nº de Processadores Lógicos Usados: 8

FCFS

10 **Trabalhos** (classe que simula as atividades automáticas)

EXEC	1	2	3	4
TEMPO milissegundos	105	90	90	90

TEMPO MÉDIO: 93,75 milissegundos

100 **Trabalhos**

EXEC	1	2	3	4
TEMPO milissegundos	128	109	124	125

TEMPO MÉDIO: 121,5 milissegundos

10000 **Trabalhos**

EXEC	1	2	3	4
TEMPO milissegundos	4309	3962	4070	3965

TEMPO MÉDIO: 4076,5 milissegundos

Carga e Disponibilidade em Conta

10 **Trabalhos** (classe que simula as atividades automáticas)

EXEC	1	2	3	4
TEMPO milissegundos	95	100	85	86

TEMPO MÉDIO: 91,5 milissegundos

100 **Trabalhos**

EXEC	1	2	3	4
TEMPO milissegundos	110	125	109	120

TEMPO MÉDIO: 116 milissegundos

10000 **Trabalhos**

EXEC	1	2	3	4
TEMPO milissegundos	4449	3850	3942	4350

TEMPO MÉDIO: 4 148 milissegundos

Resultados

- ✓ Testes de **pouca carga**: o algoritmo que tinha em conta a disponibilidade e carga de cada uma das instâncias mostrou-se mais eficiente.
- ✓ Testes de **carga média**: novamente o algoritmo que tinha em consideração mostrou-se benéfico relativamente ao FCFS.
- ✓ Testes de **carga elevada**: assimetricamente ao ocorrido anteriormente o algoritmo FCFS mostrou-se mais eficiente tal deve-se ao facto de não fazer tantas ordenações (complexidade O(n)) quanto o segundo algoritmo.

Os resultados encontram-se nas tabelas acima e para obter valores mais fidedignos foram efetuados 4 testes e para fins de comparação foi utilizado o valor médio em milissegundos medido com o algoritmo seguinte:

```
package com.mkyong.time;

import java.util.concurrent.TimeUnit;

public class ExecutionTime {

    public static void main(String[] args) throws InterruptedException {

        long lStartTime = System.currentTimeMillis();

        calculation();

        long lEndTime = System.currentTimeMillis();

        long output = lEndTime - lStartTime;

        System.out.println("Elapsed time in milliseconds: " + output);

    }

    private static void calculation() throws InterruptedException {

        //Sleep 2 seconds
        TimeUnit.SECONDS.sleep(2);

    }
}
```

https://moodle2.isep.ipp.pt/pluginfile.php/304667/mod_resource/content/2/ExecutionTime.java

(LAPR1 2019/2020)

Relatório elaborado por:

Teresa Pereira Leite

Assinatura
Assinado por : **João Pedro Oliveira Pereira**
Num. de Identificação: 30627139
Data: 2021.06.19 17:53:18+01'00'
Localização: Vila do Conde

CHAVE MÓVEL

Assinar Relatório LAPR4
Assinado por : **António Daniel Barbosa Fernandes**
Num. de Identificação: 30335401
Data: 2021.06.19 17:47:15+01'00'
Localização: Souselo, Cinfães

CHAVE MÓVEL

Rui Pedro Magalhães Soares