



Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA.

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

TEMA: CLASE 11 PLANIFICADOR MLQ

NOMBRES DE LOS ESTUDIANTES:

Padilla Perez Jorge Daray 216584703.

Ernesto Macias Flores 221349941.

Luis Ricardo Díaz Montes 219293947.

NOMBRE DE LA MATERIA: Sistemas operativos

NOMBRE DEL PROFESOR: Ramiro Lupercio Coronel

Table of Contents

Introducción	3
Procesos	4
librerías.....	4
Clase proceso	4
Ventana principal	4
Trigger	5
Lógica del FCFS	6
Conclusiones	6
Padilla Perez Jorge Daray 216584703	6
Ernesto Macias Flores 221349941	6
Luis Ricardo Díaz Montes 219293947	6
General.....	6
Bibliografía	7

Introducción

La planificación mediante colas multinivel (MLQ) es un algoritmo de planificación de procesos en sistemas operativos. Su objetivo es diferenciar entre distintos tipos de trabajos al dividir la cola de procesos preparados en varias colas, una por cada tipo de trabajo. A diferencia de los algoritmos de colas multinivel realimentadas, que permiten el movimiento de trabajos entre colas, la MLQ no permite dicho movimiento, este es como una recopilación de los algoritmos anteriores que hicimos pero en uno solo.

En seguida de esto se realizo el cálculo promedio de los resultados obtenidos junto al documento en formato reporte de este. Todo el código tiene interfaz grafica para mayor y mejor comprensión del tema, además de agregar algunos puntos que se dijeron en la clase.

Procesos

librerías

```
1 import sys
2 import random
3 from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QTableWidget, QTableWidgetItem, QLabel, QHeaderView
4 from PyQt5.QtCore import QThread, pyqtSignal, QTimer
5 from statistics import mean
```

Se importaron las librerías de PyQt5 principalmente para la interfaz, random, sys, y mean.

Clase proceso

```
8  v class ProcesoThread(QThread):
9      proceso_agregado = pyqtSignal(list)
10
11  v     def __init__(self, proceso):
12      super().__init__()
13      self.proceso = proceso
14
15  v     def run(self):
16      # Simulación de operaciones intensivas
17      self.proceso_agregado.emit(self.proceso)
18
```

Ventana principal

```
19  class VentanaPrincipal(QWidget):
20      def __init__(self):
21          super().__init__()
22
23          self.setWindowTitle("Planificador de Procesos")
24          self.setGeometry(100, 100, 1000, 600)
25
26          # Definir nombres de columnas y anchos predeterminados
27          self.encabezados_procesos = ["Proceso", "Tiempo de\nLlegada", "Inicio", "Fin", "Tiempo de\nEjecución",
28          self.anchos_columnas = [100, 100, 100, 100, 100, 100, 100, 200]
29
30          self.iniciarInterfaz()
```

Esta ventana principal se desarrolla con todas las funciones para hacer que funcione.

```

19 < class VentanaPrincipal(QWidget):
31
32     def iniciarInterfaz(self):
33         # Botones
34         hbox_botones = QHBoxLayout()
35         self.nombres_botones = ["Añadir", "Terminar", "Pausar", "Continuar"]
36         for nombre in self.nombres_botones:
37             boton = QPushButton(nombre)
38             hbox_botones.addWidget(boton)
39             boton.clicked.connect(self.on_boton_clickeado)
40
41         # Botón FCFS
42         self.btn_fcfs = QPushButton("FCFS")
43         self.btn_fcfs.clicked.connect(self.iniciar_FCFS)
44
45         # Etiqueta para el ciclo
46         self.etiqueta_ciclo = QLabel("Ciclo: ---")
47         self.hbox_fcfs_ciclo = QHBoxLayout()
48         self.hbox_fcfs_ciclo.addWidget(self.btn_fcfs)
49         self.hbox_fcfs_ciclo.addWidget(self.etiqueta_ciclo)
50
51         # Tabla de procesos
52         self.tabla_procesos = QTableWidget()
53         self.tabla_procesos.setColumnCount(len(self.encabezados_procesos))
54         self.tabla_procesos.setHorizontalHeaderLabels(self.encabezados_procesos)
55

```

Como se explico anteriormente en la clase Ventanaprincipal se inicia la interfaz la que nos permitirá interactuar con nuestro programa.

Trigger

```

95     def on_boton_clickeado(self):
96         sender = self.sender()
97         self.pausa = 0
98         self.terminar = 0
99         if sender.text() == "Añadir":
100             proceso_nombre = chr(ord('A') + self.proceso_actual) # Convertir el índice actual a una letra mayúscula
101             tiempo_llegada = str(random.randint(self.ciclo_actual, self.ciclo_actual + 5)) # Generar un tiempo de llegada aleatorio entre 5 y 10
102             tiempo_ejecucion = str(random.randint(0, 10)) # Generar un tiempo de ejecución aleatorio entre 0 y 10
103             proceso = [proceso_nombre, tiempo_llegada, "0", "0", tiempo_ejecucion, "0", "0", "0"] # Ejemplo de proceso
104             self.agregar_proceso(proceso)
105             self.proceso_actual = (self.proceso_actual + 1) % 26 # Incrementar el índice y volver a "A" si alcanza "Z"
106         if sender.text() == "Terminar":
107             self.terminar = 1
108         if sender.text() == "Pausar":
109             self.pausa = 1
110         if sender.text() == "Continuar":
111             self.pausa = 0
112

```

Aquí se realizo un disparador para cuando se presionen los botones con sus respectivas acciones.

Lógica del FCFS

```
147     def logica_FCFS(self):
148         # Ordenar la tabla "tabla_procesos" por la columna de tiempo de llegada
149         # Lógica específica del algoritmo FCFS
150         self.calcular_promedio()
151         for i in range(self.tabla_procesos.rowCount()):
152             proceso = [self.tabla_procesos.item(i, j).text() for j in range(self.tabla_procesos.columnCount())
153             if self.pausa == 1:
154                 proceso[5] = str(int(proceso[5]) + 1)
155                 self.tabla_procesos.setItem(i, 5, QTableWidgetItem(str(int(proceso[5]))))
156                 proceso[6] = str(self.ciclo_actual)
157                 self.tabla_procesos.setItem(i, 6, QTableWidgetItem(str(int(proceso[6]))))
158                 #break
159             elif self.terminar == 1:
160                 self.eliminar_fila_seleccionada()
161                 self.terminar = 0
162                 #break
163             elif proceso[7] != proceso[4] and int(proceso[1]) <= self.ciclo_actual: # Si el proceso no ha comenzado
164                 if proceso[7] == "0":
165                     self.tabla_procesos.setItem(i, 2, QTableWidgetItem(str(self.ciclo_actual)))
166                     self.actualizar_progreso(proceso, i)
167                 if proceso[7] == proceso[4]:
168                     self.tabla_procesos.setItem(i, 3, QTableWidgetItem(str(self.ciclo_actual + 1)))
169                     #break
170             # Conectar la señal itemSelectionChanged a la función eliminar_fila_seleccionada
171             # self.tabla_procesos.itemSelectionChanged.connect(self.eliminar_fila_seleccionada)
```

Como se menciona en el título aquí se desarrolla la lógica del FCFS para ordenar los procesos, está hecho con otros códigos existentes.

Conclusiones

Padilla Perez Jorge Daray 216584703.

Para concluir la actividad quiero decir que Las colas (MLQ) es como la versión extendida de nuestras prácticas o mas bien dicho el recopilatorio de estas, permite usar estas al momento de la planificación y se me hace una mejor función que los otros algoritmos que aunque buenos, este aplica cada uno de ellos.

Ernesto Macias Flores 221349941.

En esta práctica aprendí a resolver problemas relacionados a el procesamiento de un FCFS y aunque anteriormente ya había desarrollado un sistema de simulación por lotes que es bastante parecido a este programa no es lo mismo, por lo que integrar los resultados de los 2 programas realizados mejoró mi manera de entender estas simulaciones de mejor manera.

Luis Ricardo Díaz Montes 219293947.

Las Colas Multinivel (MLQ): Son importantes ya que cuentan con. Criterios de Planificación: Incluyen prioridad fija con o sin requisito y prioridad variable, para gestionar la ejecución de procesos y evitar monopolio de la CPU. Gestión de Procesos: Utiliza mecanismos como el tiempo de permanencia y envejecimiento para asignar prioridades y asegurar una ejecución justa.

General

El funcionamiento de este algoritmo consiste en ejecutar los procesos de la cola de prioridad más alta, a continuación, se pasan a ejecutar los procesos de la siguiente cola y así sucesivamente. Con esta distribución, los procesos con ráfagas cortas se ejecutarán de forma rápida sin necesidad de

llegar muy lejos en la jerarquía de colas de listos. Mientras que los procesos con ráfagas largas irán degradándose gradualmente.

Bibliografía

GeeksforGeeks-1 y Barcelona Geeks (Traductores). Título: Planificación mediante colas multinivel

https://www.wikiwand.com/es/Planificaci%C3%B3n_mediante_colas_multinivel

Webplusvalencia. (2023). Algoritmos de Planificación FCFS, SJF, SRTF, ROUND ROBIND.

Recuperado de [!\[\]\(83f22ed94ec5517769dd76d702c6bfd8_img.jpg\) Algoritmos de Planificación FCFS, SJF, SRTF, ROUND ROBIND !\[\]\(58518edde73d42d67a35a8ed26134c7b_img.jpg\)](#)
[Webplusvalencia](#)