Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA.

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

TEMA: MODULO 2 CLASE 9 PLANIFICADOR SRT

NOMBRES DE LOS ESTUDIANTES:

Padilla Perez Jorge Daray | 216584703

Luis Ricardo Díaz Montes | 219293947

Ernesto Macias Flores | 221349941

NOMBRE DE LA MATERIA: Sistemas operativos

NOMBRE DEL PROFESOR: Ramiro Lupercio Coronel

Table of Contents

Introducción

En esta práctica se realizó un planificador SRT (Shortest Remaining Time First) es una variante del algoritmo SJF (Shortest Job First) el cual ya se entregó la semana pasada. En este caso, el procesador asigna el tiempo de procesamiento al proceso que tenga la duración más corta, pero se actualiza constantemente para adaptarse a los cambios en la duración del proceso.

También en términos más sencillos, el planificador SRT prioriza la ejecución de los procesos más cortos en cada momento, lo que ayuda a minimizar el tiempo de espera y a mantener una respuesta ágil ante las variaciones en las necesidades de procesamiento. Es especialmente útil en sistemas multitarea y multiproceso, donde se busca una distribución eficiente del tiempo de CPU entre los procesos disponibles.

Como consecuencia se logro realizar el algoritmo de manera exitosa y como en la actividad pasada, también esta actividad se realizaron botones para Agregar, Terminar, Pausar. Los cuales son puntos clave en la realización de la practica solicitada.

Procesos

librerías

```python
import sys
import random
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QTableWidget, QTableWidgetItem,
from PyQt5.QtCore import QThread, pyqtSignal, QTimer
from statistics import mean
```

En este caso se utilizó la librería sys, random, PyQt5 para la interfaz y por ultimo se hizo uso de statistics.

Interfaz

```python
class ProcesoThread(QThread):
    proceso_agregado = pyqtSignal(list)

    def __init__(self, proceso):
        super().__init__()
        self.proceso = proceso

    def run(self):
        # Simulación de operaciones intensivas
        self.proceso_agregado.emit(self.proceso)

class VentanaPrincipal(QWidget):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Planificador de Procesos")
        self.setGeometry(100, 100, 1200, 600)

        # Definir nombres de columnas y anchos predeterminados
        self.encabezados_procesos = ["Proceso", "Tiempo de\nLlegada", "Inicio", "Fin", "Tiempo de\nEjecución",
        self.anchos_columnas = [100, 100, 100, 100, 100, 100, 100, 200, 100, 100]

        self.iniciarInterfaz()

    def iniciarInterfaz(self):
```

Aquí se crea la clase proceso que incluye al proceso como se indica, y también se crea la ventana principal para la interfaz grafica del programa.

```python
    def iniciarInterfaz(self):
        # Botones
        hbox_botones = QHBoxLayout()
        self.nombres_botones = ["Agregar", "Terminar", "Pausar", "Continuar"]
        for nombre in self.nombres_botones:
            boton = QPushButton(nombre)
            hbox_botones.addWidget(boton)
            boton.clicked.connect(self.on_boton_clickeado)

        # Botón FCFS
        self.btn_fcfs = QPushButton("Shortest Remaining Time")
        self.btn_fcfs.clicked.connect(self.iniciar_FCFS)

        # Etiqueta para el ciclo
        self.etiqueta_ciclo = QLabel("Ciclo: ---")
        self.hbox_fcfs_ciclo = QHBoxLayout()
        self.hbox_fcfs_ciclo.addWidget(self.btn_fcfs)
        self.hbox_fcfs_ciclo.addWidget(self.etiqueta_ciclo)

        # Tabla de procesos
        self.tabla_procesos = QTableWidget()
        self.tabla_procesos.setColumnCount(len(self.encabezados_procesos))
        self.tabla_procesos.setHorizontalHeaderLabels(self.encabezados_procesos)

```

Aquí se aprecia mas el inicio de la interfaz.

Trigger

```python
    def on_boton_clickeado(self):
        sender = self.sender()
        self.pausa = 0
        self.terminar = 0
        if sender.text() == "Agregar":
            proceso_nombre = chr(ord('A') + self.proceso_actual)  # Convertir el índice actual a una letra mayúscula
            tiempo_llegada = str(random.randint(self.ciclo_actual, self.ciclo_actual + 5))  # Generar un tiempo de llega
            tiempo_ejecucion = str(random.randint(1, 9))  # Generar un tiempo de ejecución aleatorio entre 5 y 10
            proceso = [proceso_nombre, tiempo_llegada, "0", "0", tiempo_ejecucion, "0", "0", "0", tiempo_ejecucion, "0"]
            self.agregar_proceso(proceso)
            self.proceso_actual = (self.proceso_actual + 1) % 26  # Incrementar el índice y volver a "A" si alcanza "Z"
        if sender.text() == "Terminar":
            self.eliminar_fila_seleccionada()
        if sender.text() == "Pausar":
            self.pausar_fila_seleccionada()
        if sender.text() == "Continuar":
            self.continuar_fila_seleccionada()

```

Trigger de botón clickeado en el cual se esta esperando que se pulse un botón de los 4 que hay, y dependiendo el botón procede.

Actualizar tabla

```
115     def agregar_proceso(self, proceso):
116         self.procesos.append(proceso)
117         self.agregar_proceso_tabla(proceso)  # Agregar el proceso a la tabla de procesos inmediatament
118 
119     def actualizar_tabla(self, proceso):
120         row_count = self.tabla_procesos.rowCount()
121         self.tabla_procesos.setRowCount(row_count + 1)
122         for col, valor in enumerate(proceso):
123             item = QTableWidgetItem(valor)
124             self.tabla_procesos.setItem(row_count, col, item)
125         # Agregar barra de carga en la columna "Porcentaje"
126         progreso = QProgressBar()
127         progreso.setValue(0)
128         self.tabla_procesos.setCellWidget(row_count, 7, progreso)
129         self.tabla_procesos.sortItems(8)  # Ordenar por la columna 1 (tiempo de llegada)
130         self.calcular_promedio()
```

Aquí se actualiza la tabla en la cual se agrega la barra de carga en la columna, y también ordena por columna los tiempos de llegada.

Lógica botones

```
132     def inicializar_procesos(self):
133         if self.ciclo_actual == 0:
134             # Establecer todos los procesos en la tabla de procesos a "0":
135             # en las columnas "Inicio", "Fin", "Espera", "Paro" y "Porcentaje"
136             for i in range(self.tabla_procesos.rowCount()):
137                 for j in range(2, 7):  # Columnas desde "Inicio" hasta "Porcentaje"
138                     if j != 4:  # Excluir la columna "Tiempo"
139                         item = QTableWidgetItem("0")
140                         self.tabla_procesos.setItem(i, j, item)
141                 self.tabla_procesos.cellWidget(i, 7).setValue(int(0))
142 
143     def eliminar_fila_seleccionada(self):
144         selected_items = self.tabla_procesos.selectedItems()  # Obtener los elementos sele
145         if selected_items:  # Verificar si se ha seleccionado algún elemento
146             row_index = selected_items[0].row()  # Obtener el índice de la fila seleccion
147             self.tabla_procesos.removeRow(row_index)  # Eliminar la fila seleccionada
148 
149     def pausar_fila_seleccionada(self):
150         selected_items = self.tabla_procesos.selectedItems()  # Obtener los elementos sele
151         if selected_items:  # Verificar si se ha seleccionado algún elemento
152             row_index = selected_items[0].row()  # Obtener el índice de la fila seleccion
153             # self.tabla_procesos.removeRow(row_index)  # Eliminar la fila seleccionada
154             self.tabla_procesos.setItem(row_index, 9, QTableWidgetItem("1"))
```

En este caso se implementan funciones para cada botón en los cuales están agregar, eliminar procesos, pausar procesos, inicializar los proceso y falta continuar pero es casi lo mismo que pausa.

Lógica principal FCFS

```python
def logica_FCFS(self):
    # Ordenar la tabla "tabla_procesos" por la columna de tiempo de llegada
    # Lógica específica del algoritmo FCFS
    for i in range(self.tabla_procesos.rowCount()):
        self.proceso = [self.tabla_procesos.item(i, j).text() for j in range(self.tabla_procesos.columnCount())]
        if self.proceso[7] != self.proceso[4] and int(self.proceso[1]) <= self.ciclo_actual and self.proceso[9] != "1":
            if self.proceso[7] == "0":
                self.tabla_procesos.setItem(i, 2, QTableWidgetItem(str(self.ciclo_actual)))
            self.actualizar_progreso(self.proceso, i)
            if self.proceso[7] == self.proceso[4]:
                self.tabla_procesos.setItem(i, 3, QTableWidgetItem(str(self.ciclo_actual + 1)))
            break
        # Conectar la señal itemSelectionChanged a la función eliminar_fila_seleccionada
        # self.tabla_procesos.itemSelectionChanged.connect(self.eliminar_fila_seleccionada)
    self.calcular_promedio()
```

Aquí se puede apreciar toda la lógica detrás del algoritmo FCFS

Calcular promedio

```python
def calcular_promedio(self):
    proceso_numeros = []
    for i in range(self.tabla_procesos.rowCount()):
        proceso = [self.tabla_procesos.item(i, j).text() for j in range(1, 9)]
        proceso_numeros.append([int(elemento) for elemento in proceso])
    # Transponer la lista para obtener las columnas
    columnas = zip(*proceso_numeros)
    # Calcular el promedio para cada columna
    promedios = [sum(columna) / len(columna) for columna in columnas]
    promedios_formateados = [format(promedio, ".2f") for promedio in promedios]
    for i in range(1, len(self.encabezados_procesos)-1):
        # print(self.encabezados_procesos[i])
        if i == len(self.encabezados_procesos) - 3:
            # print(self.encabezados_procesos[i])
            porcentaje_total = (float(promedios[6])/float(promedios[3])) * 100
            self.tabla_promedios.cellWidget(0, 7).setValue(int(porcentaje_total))
            porcentaje_total = format(porcentaje_total, ".2f")
            self.tabla_promedios.setItem(0, i, QTableWidgetItem(str(porcentaje_total)))
        else:
            self.tabla_promedios.setItem(0, i, QTableWidgetItem(str(promedios_formateados[i-1])))
```

Aquí se aprecia la función calcular promedio que como su nombre lo indica se utiliza para calcular el promedio de las columnas que se fueron agregando.

Conclusión

Padilla Perez Jorge Daray 216584703.

Se implementó el algoritmo SRT para la planificación de procesos en un entorno simulado utilizando PyQt5 en Python. Esta implementación ofrece una oportunidad para explorar y comprender cómo funciona el algoritmo SRT en un sistema operativo, distribuyendo equitativamente el tiempo de CPU entre los procesos activos.

Ernesto Macias Flores 221349941.

La interfaz gráfica desarrollada proporciona una plataforma interactiva para observar el progreso de los procesos en tiempo real. Al mostrar información detallada sobre cada proceso, como su tiempo de llegada, tiempo de ejecución y estado actual, la interfaz ayuda a los usuarios a comprender mejor cómo se administran los recursos del sistema durante la ejecución de múltiples tareas.

Luis Ricardo Díaz Montes 219293947.

A diferencia de otros algoritmos de planificación de procesos, como el FCFS, el algoritmo SRT el cual se parece mucho al SJF garantiza una mayor equidad en la asignación de recursos de CPU al dividir el tiempo de ejecución en pequeños intervalos para cada proceso. Esto mejora la capacidad de respuesta del sistema y reduce el tiempo de espera percibido por los usuarios, lo que resulta en una experiencia de usuario más fluida y satisfactoria.

General

el planificador SRT es una estrategia inteligente para administrar los recursos de la CPU de manera eficiente. Al priorizar los procesos más cortos, logra minimizar el tiempo de espera y mantener una respuesta ágil en sistemas multitarea. La adaptabilidad constante a las condiciones cambiantes de los procesos es su mayor fortaleza. En resumen, el planificador SRT es como un maestro de ceremonias que dirige el espectáculo, asegurando que todos los actos tengan su momento en el escenario.

Bibliografía

1. Tanenbaum, A. S., & Bos, H. (2015). Sistemas operativos modernos (4.ª ed.). Pearson. Sistemas Operativos Modernos de Tanenbaum (comprimido).pdf - Google Drive

2. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10.ª ed.). Wiley. Operating System Concepts 10th : Abraham Silberschatz, Peter B. Galvin, and Greg Gagne : Free Download, Borrow, and Streaming : Internet Archive

3. Stallings, W. (2018). Operating Systems: Internals and Design Principles (9.ª ed.). Pearson. [PDF] Operating Systems: Internals and Design Principles | Semantic Scholar

4. Abraham, S. (2019). Operating Systems: Principles and Practice (2.ª ed.). Recursive Books. Operating system principles : Silberschatz, Abraham : Free Download, Borrow, and Streaming : Internet Archive

5. Deitel, H. M., & Deitel, P. J. (2018). Operating Systems (3.ª ed.). Pearson. Operating systems (Deitel) (3rd edition)(1).pdf - Free Download PDF (kupdf.net)

6. Goscinski, A. M., & Brock, M. T. (2017). Cloud Computing: Principles, Systems and Applications (1.ª ed.). Springer. Cloud Computing: Principles, Systems and Applications | SpringerLink