Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER HUMANA.

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

TEMA: Practica3

NOMBRE DE LOS ESTUDIANTES:

Padilla Perez Jorge Daray

Juan Jesús Sámano Juárez

Ernesto Macias Flores

NOMBRE DE LA MATERIA: Seminario Sistemas Operativos

SECCION: D04

CICLO ESCOLAR: 2024-A

NOMBRE DEL PROFESOR: Julio Esteban

Contenido

Introducción.

En este documento que se muestra cómo se realizan acciones mediante teclas, dichas funciones como I para interrumpir procesos, E para marcar error, P para pausar el proceso y C para continuar el proceso pausado, esta actividad funciona junto con la practica2 sobre lotes así mismo se lleva una continuidad que también se refleja plasmada en este documento.

También se presentan capturas de las ejecuciones del programa funcionando de manera correcta de los momentos que creemos cruciales, el código fuente que se utilizó para el desarrollo del programa, y por ultimo las conclusiones personales de cada integrante del equipo.

Aparte de lo anteriormente visto en la práctica, en este caso se amplió la practica anterior agregándole y cambiándole las interrupciones, así como algunos detalles a la hora de hacer el sistema, también en estas interrupciones se hacen mas solidas ya que se piden más cosas por interrupción haciendo mas interactiva la practica y a la vez mas complica ya que requiere de mas cosas.

Capturas de pantalla.

Código fuente.

Main():

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


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, 1000, 600)
```

Class proceso que sirve para crear los procesos automáticos y la venta también sale la clase.

```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("FCFS")
        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)
```

Inicio de interfaz grafica y su enlazamiento.

```python
    def on_boton_clickeado(self):
        sender = self.sender()
        self.pausa = 0
        self.terminar = 0
        if s
            (variable) tiempo_llegada: str
                                        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 llegada aleatorio e
            tiempo_ejecucion = str(random.randint(10, 15))  # Generar un tiempo de ejecución aleatorio entre 10 y 15
            proceso = [proceso_nombre, tiempo_llegada, "0", "0", tiempo_ejecucion, "0", "0", "0"]  # Ejemplo de proceso
            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.terminar = 1
        if sender.text() == "Pausar":
            self.pausa = 1
        if sender.text() == "Continuar":
            self.pausa = 0

```

Trigger para los botones al momento de estar en ejecución los procesos.

```python
    def agregar_proceso(self, proceso):
        self.procesos.append(proceso)
        self.agregar_proceso_tabla(proceso)  # Agregar el proceso a la tabla de procesos inmediatamente

    def actualizar_tabla(self, proceso):
        row_count = self.tabla_procesos.rowCount()
        self.tabla_procesos.setRowCount(row_count + 1)
        for col, valor in enumerate(proceso):
            item = QTableWidgetItem(valor)
            self.tabla_procesos.setItem(row_count, col, item)
        # Agregar barra de carga en la columna "Porcentaje"
        progreso = QProgressBar()
        progreso.setValue(0)
        self.tabla_procesos.setCellWidget(row_count, 7, progreso)
        self.tabla_procesos.sortItems(1)  # Ordenar por la columna 1 (tiempo de llegada)
        self.calcular_promedio()
```

Agregar procesos y actualizar tabla aparecen para que se siga ciclando.

```python
    def logica_FCFS(self):
        # Ordenar la tabla "tabla_procesos" por la columna de tiempo de llegada
        # Lógica específica del algoritmo FCFS
        self.calcular_promedio()
        for i in range(self.tabla_procesos.rowCount()):
            proceso = [self.tabla_procesos.item(i, j).text() for j in range(self.tabla_procesos.columnCount())]
            if self.pausa == 1:
                proceso[5] = str(int(proceso[5]) + 1)
                self.tabla_procesos.setItem(i, 5, QTableWidgetItem(str(int(proceso[5]))))
                proceso[6] = str(self.ciclo_actual)
                self.tabla_procesos.setItem(i, 6, QTableWidgetItem(str(int(proceso[6]))))
                break
            elif self.terminar == 1:
                self.eliminar_fila_seleccionada()
                self.terminar = 0
                break
            elif proceso[7] != proceso[4] and int(proceso[1]) <= self.ciclo_actual:  # Si el proceso no ha completado su tiempo
                if proceso[7] == "0":
                    self.tabla_procesos.setItem(i, 2, QTableWidgetItem(str(self.ciclo_actual)))
                self.actualizar_progreso(proceso, i)
                if proceso[7] == 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
```

La lógica del FCFS se representa aquí.

Faltaría realizar las interrupciones, y también faltaría realizar la terminación de interrupciones.

Conclusiones:

Padilla Perez Jorge Daray:
Para concluir con esta actividad interesante ya que es raro tener que simular que una computadora actual tarde tanto en hacer ciertos procesos y más cuando es algo tan sencillo, el tema de tener varias ventanas fue todo un reto que el compañero Ernesto soluciono, la comunicación en equipo fue importante para la realización de esta actividad, aprendí juntos a mis compañeros y espero que podamos seguir así de comunicados.

Juan Jesús Sámano Juárez:
Al empezar a realizar esta actividad me costó trabajo entender los requerimientos, con apoyo de mis compañeros pude entender de mejor manera, en un principio realicé un código que cumplía con la mayoría de lo requerido, después con el equipo se decidió reunir 3 trabajos para hacer uno solo, aprendí sobre el manejo de hilos y a trabajar mediante GitHub.

Ernesto Macias Flores:
El desarrollo de un simulador de lotes en Python es una tarea valiosa para comprender los conceptos de planificación de procesos y la ejecución de tareas en un entorno de procesamiento por lotes, como una buena introducción a lo que se viene después, y aunque es una simulación es útil para comparar y despejarnos un poco de lo que es lo normal en la programación.