UNIVERSIDAD MAYOR DE SAN ADRES

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE CIENCIAS DE LA COMPUTACIÓN

PROYECTO INF-151

MATERIA: SISTEMAS OPERATIVOS (INF- 151)

DOCENTE: LIC. RUBEN ALCÓN LOPEZ

UNIVERSITARIO: ALAVE SANJINES CRISTHIAN RODRIGO

LA PAZ – BOLIVIA

2021

Contenido

DOCUMENTACIÓN:

Texto del problema asignado identificando el problema asignado:

1. Administracion de procesos.
2. Administracion de memoria por paginación para el reemplazo de páginas.
3. Administracion de memoria secundaria.

Texto del problema definido. El problema especifico que propone resolver el estudiante, debe incluirse descripciones particulares de la variantes o estrategias consideradas (Que resolver).

Para la parte 1, se tratará de resolver el problema de asignar recursos a los procesos representativos de la petición del usuario de modo que se pueda hacer el seguimiento del mismo hasta su conclusión.

De la parte 2, se tratará de colocar las demandas de tal forma que no se desperdicie el espacio de memoria para ello veremos un algoritmo del reemplazo de página.

De la parte de 3, se tratará de resolver el movimiento del cabezal para buscar alguna x información y que el movimiento del mismo sea mínimo o estar entre los mínimos.

Texto de la estrategia de solución de los problemas definidos (como se resuelve).

De la parte 1, se usará el algoritmo Round Robin (RR), FIFO y Prioridades que como sabemos atiende por turnos o rondas en función del tamaño del quantum(Q), donde Q representa los cambios del reloj o simplemente de cuanto en cuanto le está permitido avanzar además de que busca dar una buena respuesta tanto a procesos cortos como largos, con el FIFO también se verá que se atienden los procesos por orden de llegada, así como el de prioridades que como dice su nombre se atenderán las PRIORIDADES y para ello se nos dará una columna marcando la misma, que además en este caso la alta prioridad son los números mas pequeños, y la más baja prioridad son los números mas altos.

De la parte 2, se usará el algoritmo OPT (Optimo), FIFO y LRU donde se ha de conocer todas las peticiones por anticipado y se implementa este algoritmo porque es el mejor algoritmo para solucionar lo planteado,

donde sabemos que se reemplaza a la referencia más lejana, además de que se necesita conocer por anticipado todas las demandas de manera que aunque es óptimo hablando de la realidad es difícil que se conozcan todas las peticiones de lo que se hará, siempre se van añadiendo peticiones por ello que en la realidad las peticiones varían y no se mantienen constantes, además del FIFO que como sabemos es el primero en la demanda primero en reemplazar, y por último el LRU manejando el concepto de software de pilas, que como sabemos se ayuda de esta para poder dar con el reemplazo indicado.

De la parte 3, se usará el algoritmo del SCHENBACH, FIFO y SSTF dado que hace un recorrido casi lineal si no consideramos los repetidos y por supuesto va de acuerdo a un orden de menor a mayor petición, el de FIFO no requiere presentación más que indicar que las peticiones serán de acuerdo al orden de llegada el primero en entrar es el último en salir, y por último el SSTF que lo que hará es atender las peticiones que requieran el menor esfuerzo del brazo del disco y por su puesto se ha de conocer antes de manera anticipada todas las peticiones y como sabemos necesitamos un punto de partida del cual ira primeramente hacia abajo y luego hacia arriba.

Diseño de la estrategia general de solución.

El plan o la estrategia para dar solución con los problemas ha sido:

- Usar un lenguaje de programación sencillo en la sintaxis como lo es Python.
- Uso de ejercicios resueltos para observar el comportamiento adecuado del algoritmo.
- Análisis de la función de cada algoritmo frente a determinados problemas.
- Uso de programación orientada a objetos para poder aprovechar este paradigma informático en la construcción de un todo.
- Darse un ejemplo hecho en papel y probarlo con el programa.
- En algunos casos se usará el método short que nos ayudará a que se simplifiquen los procesos.
- Se utilizará lo que es la librería que nos permite dibujar funciones a fin de observar claramente su comportamiento, esto para la mayoría de los algoritmos.

- También se hará uso a lo que es la Lista y Pila para poder sobrellevar los algoritmos de manera entendible y fácil.

Desarrollo de la aplicación

Concluida la programación debe adjuntarse en la documentación la impresión de:

Código del programa principal.

```
from FIFO import FIFO
from PRIORIDADES import Prioridades
from RR import RoundRobin

from OPT import OPT
from FIFO1 import FIFO1
from LRU import LRU

from SCHENBACH import SCHENBACH
from FIFO2 import FIFO2
from SSTF import SSTF


#1. Administracion de procesos.
f1 = FIFO(Proc=['A', 'B', 'C', 'D', 'E', 'F'], Tini=[3, 1, 0, 6, 3, 9], Tcpu=[3, 2, 4, 2, 3, 1])
P1 = Prioridades(Proc=['A', 'B', 'C', 'D', 'E'], Tini=[7, 8, 8, 10, 12], Tcpu=[3, 4, 2, 3, 2], Prioridades=[3, 2, 1, 4, 5])
RR1 = RoundRobin(Proc=['A', 'B', 'C', 'D', 'E'], Tini=[0, 1, 3, 9, 12], Tcpu=[3, 5, 2, 5, 5], Quantum=10)
#2. Administracion de memoria por paginación para el reemplazo de páginas.
O1 = OPT(Demanda=[2, 3, 4, 2, 1, 3, 5, 2, 4, 3, 2, 1], Marcos=3)
F11 = FIFO1(Demanda=[3, 2, 1, 4, 6, 5, 3, 2, 4, 3], Marcos=4)
L1 = LRU(Demanda=[2, 4, 6, 2, 3, 4, 5, 2, 1, 2, 3, 4], Marcos=3)
#3. Administracion de memoria secundaria
FF21 = FIFO2(Peticiones=[10, 40, 5, 20, 15, 35, 40, 5, 10, 25, 10, 35, 10, 5, 20], inicio=20)
S1 = SCHENBACH(Peticiones=[10, 15, 10, 30, 15, 10, 20, 30, 10, 40, 20, 30])
SS1 = SSTF(Peticiones=[10, 40, 5, 20, 15, 35, 40, 5, 10, 25, 10, 35, 10, 5, 20], inicio=20)

f1.solucion(); P1.solucion(); RR1.solucion()

O1.solucion(); F11.solucion(); L1.solucion()

FF21.solucion(); S1.solucion(); SS1.solucion()
```

Código de las rutinas/procedimientos de los algoritmos implementados o utilizados.

Para el algoritmo de FIFO:

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
#funciona
class FIFO:
    def graficar(self):
        x = np.array(self.TiempoInicial)
        y = np.array(self.TiempoFinal)
        plt.plot(x, y, 'og--', mfc = 'y', mec = 'y')
        plt.title('FIFO')
        plt.xlabel('Tiempo inicial')
        plt.ylabel('Tiempo final')
        plt.show()
        #plt.show()#para mostrar pero se queda esperando hasta cerrar la ventana
    def __init__(self, Proc, Tini, Tcpu):
        self.Proceso = Proc
        self.TiempoInicial = Tini
        self.TiempoCpu = Tcpu
        self.T = []; self.E = []; self.I = []; self.TiempoFinal = []
        self.contador = 0
        self.numeroProcesos = len(self.Proceso)
    def mostrar(self):
        nombres = ['Proceso', 'Tini', 'Tcpu', 'Tfin', 'T', 'E', 'I']
        dataFrame = pd.DataFrame(list(zip(self.Proceso, self.TiempoInicial, self.TiempoCpu,
                                          self.TiempoFinal, self.T, self.E, self.I)), columns=nombres)
        print(dataFrame)
        print(f'T = {round(sum(self.T) / self.numeroProcesos, 2)}\n'
              f'E = {round(sum(self.E) / self.numeroProcesos, 2)}\n'
              f'I = {round(sum(self.I) / self.numeroProcesos, 2)}')
    def calcularT(self):
        for i in range(self.numeroProcesos):
            self.T.append(self.TiempoFinal[i] - self.TiempoInicial[i])
```

```
    def calcularE(self):
        for i in range(self.numeroProcesos):
            self.E.append(self.T[i] - self.TiempoCpu[i])
    def calcularI(self):
        for i in range(self.numeroProcesos):
            self.I.append(round(self.TiempoCpu[i] / self.T[i], 2))
    def ordenar(self):
        #ordena Tini, Tcpu y Proc para poder hacer el FIFO
        for i in range(self.numeroProcesos):
            for j in range(self.numeroProcesos - 1):
                if self.TiempoInicial[j] > self.TiempoInicial[j + 1]:
                    self.TiempoInicial[j], self.TiempoInicial[j + 1] = self.TiempoInicial[j + 1], self.TiempoInicial[j]
                    self.Proceso[j], self.Proceso[j + 1] = self.Proceso[j + 1], self.Proceso[j]
                    self.TiempoCpu[j], self.TiempoCpu[j + 1] = self.TiempoCpu[j + 1], self.TiempoCpu[j]

    def calcularTfin(self):
        aux = 0
        for i in range(self.numeroProcesos):
            if i == 0:#inicialmente obtendra el valor aux
                aux = self.TiempoInicial[i] + self.TiempoCpu[i]
            else:
                aux += self.TiempoCpu[i]
            self.TiempoFinal.append(aux)
```

```
def solucion(self):
    try:
        self.ordenar()
        self.calcularTfin()
        self.calcularT()
        """print(self.Proceso)
        print(self.TiempoInicial)
        print(self.TiempoCpu)
        print(self.TiempoFinal)"""
        self.calcularE()
        self.calcularI()
        self.mostrar()
        self.graficar()
    except Exception as error:
        print(error)
```

Para el algoritmo de Round Robin:

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
#funciona
class RoundRobin:
    def __init__(self, Proc, Tini, Tcpu, Quantum):
        self.Proceso = Proc
        self.TiempoInicial = Tini
        self.TiempoCpu = Tcpu
        self.contador = 0
        self.numeroProcesos = len(self.Proceso)
        self.quantum = Quantum
        self.T = []; self.E = []; self.I = []; self.TiempoFinal = [0 for i in range(self.numeroProcesos)]
    def graficar(self):
        x = np.array(self.TiempoInicial)
        y = np.array(self.TiempoFinal)
        plt.plot(x, y, 'og--', mfc = 'y', mec = 'y')
        plt.title('Round Robin(RR)')
        plt.xlabel('Tiempo inicial')
        plt.ylabel('Tiempo final')
        plt.show()
    #plt.show()#para mostrar pero se queda esperando hasta cerrar la ventana

 def calcularT(self):                                                                    31
     for i in range(self.numeroProcesos):
         self.T.append(self.TiempoFinal[i] - self.TiempoInicial[i])
 def calcularE(self):
     for i in range(self.numeroProcesos):
         self.E.append(self.T[i] - self.TiempoCpu[i])
 def calcularI(self):
     for i in range(self.numeroProcesos):
         self.I.append(round(self.TiempoCpu[i] / self.T[i], 2))
 def ordenar(self):
     #ordena Tini, Tcpu y Proc para poder hacer el FIFO
     for i in range(self.numeroProcesos):
         for j in range(self.numeroProcesos - 1):
             if self.TiempoInicial[j] > self.TiempoInicial[j + 1]:
                 self.TiempoInicial[j], self.TiempoInicial[j + 1] = self.TiempoInicial[j + 1], self.TiempoInicial[j]
                 self.Proceso[j], self.Proceso[j + 1] = self.Proceso[j + 1], self.Proceso[j]
                 self.TiempoCpu[j], self.TiempoCpu[j + 1] = self.TiempoCpu[j + 1], self.TiempoCpu[j]

def mostrar(self):
    print('**********************Round Robin*****************************')
    nombres = ['Proceso', 'Tini', 'Tcpu', 'Tfin', 'T', 'E', 'I']
    dataFrame = pd.DataFrame(list(zip(self.Proceso, self.TiempoInicial, self.TiempoCpu,
                                      self.TiempoFinal, self.T, self.E, self.I)), columns=nombres)
    print(dataFrame)
    print(f'T = {round(sum(self.T) / self.numeroProcesos, 2)}\n'
          f'E = {round(sum(self.E) / self.numeroProcesos, 2)}\n'
          f'I = {round(sum(self.I) / self.numeroProcesos, 2)}\n')

def calcularTfin(self):
    TiempoCpu = self.TiempoCpu.copy()
    tiempo = self.TiempoInicial[0]
    ronda = 0
    i = 0
    while sum(TiempoCpu) != 0:#controla la salida
        if i < self.numeroProcesos:
            if TiempoCpu[i] == 0:
                i += 1
```

```
            elif self.TiempoInicial[i] > tiempo:
                i = 0
            else:
                if TiempoCpu[i] - self.quantum >= 0:
                    TiempoCpu[i] -= self.quantum
                    tiempo += self.quantum
                elif TiempoCpu[i] < self.quantum and TiempoCpu[i] !=
                    tiempo += TiempoCpu[i]
                    TiempoCpu[i] = 0
                if TiempoCpu[i] == 0:
                    self.TiempoFinal[i] = tiempo

                i += 1
        else:
            ronda += 1
            i = 0

def solucion(self):
    try:
        self.ordenar()
        self.calcularTfin()
        self.calcularT()
        """print(self.Proceso)
        print(self.TiempoInicial)
        print(self.TiempoCpu)
        print(self.TiempoFinal)"""
        self.calcularE()
        self.calcularI()
        self.mostrar()
        self.graficar()
    except Exception as error:
        print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo de PRIORIDADES (PR):

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
#Funciona
class Prioridades:
    def graficar(self):
        x = np.array(self.TiempoInicial)
        y = np.array(self.TiempoFinal)
        plt.plot(x, y, 'og--', mfc = 'y', mec = 'y')
        plt.title('PRIORIDADES(PR)')
        plt.xlabel('Tiempo inicial')
        plt.ylabel('Tiempo final')
        plt.show()
        #plt.show()#para mostrar pero se queda esperando hasta cerrar la ventana
    def __init__(self, Proc, Tini, Tcpu, Prioridades):
        self.Proceso = Proc
        self.TiempoInicial = Tini
        self.TiempoCpu = Tcpu
        self.T = []; self.E = []; self.I = []; self.TiempoFinal = []; self.Prioridades = Prioridades
        self.contador = 0
        self.numeroProcesos = len(self.Proceso)
    def mostrar(self):
        print('**********************Prioridades**************************')
        nombres = ['Proceso', 'Tini', 'Tcpu', 'Prio', 'Tfin', 'T', 'E', 'I']
        dataFrame = pd.DataFrame(list(zip(self.Proceso, self.TiempoInicial, self.TiempoCpu, self.Prioridades,
                                          self.TiempoFinal, self.T, self.E, self.I)), columns=nombres)
        print(dataFrame)

        print(f'T = {round(sum(self.T) / self.numeroProcesos, 2)}\n'
              f'E = {round(sum(self.E) / self.numeroProcesos, 2)}\n'
              f'I = {round(sum(self.I) / self.numeroProcesos, 2)}\n')

    def calcularE(self):
        for i in range(self.numeroProcesos):
            self.E.append(self.T[i] - self.TiempoCpu[i])
    def calcularI(self):
        for i in range(self.numeroProcesos):
            self.I.append(round(self.TiempoCpu[i] / self.T[i], 2))
    def ordenar(self):
        #ordena Tini, Tcpu y Proc para poder hacer el FIFO
        for i in range(self.numeroProcesos):
            for j in range(self.numeroProcesos - 1):
                if self.TiempoInicial[j] > self.TiempoInicial[j + 1]:
                    self.TiempoInicial[j], self.TiempoInicial[j + 1] = self.TiempoInicial[j + 1], self.TiempoInicial[j]
                    self.Proceso[j], self.Proceso[j + 1] = self.Proceso[j + 1], self.Proceso[j]
                    self.TiempoCpu[j], self.TiempoCpu[j + 1] = self.TiempoCpu[j + 1], self.TiempoCpu[j]
                elif self.TiempoInicial[j] == self.TiempoInicial[j + 1] and self.Prioridades[j] > self.Prioridades[j + 1]:
                    self.TiempoInicial[j], self.TiempoInicial[j + 1] = self.TiempoInicial[j + 1], self.TiempoInicial[j]
                    self.Proceso[j], self.Proceso[j + 1] = self.Proceso[j + 1], self.Proceso[j]
                    self.TiempoCpu[j], self.TiempoCpu[j + 1] = self.TiempoCpu[j + 1], self.TiempoCpu[j]

    def calcularTfin(self):
        aux = 0
        for i in range(self.numeroProcesos):
            if i == 0:#inicialmente obtendra el valor aux
                aux = self.TiempoInicial[i] + self.TiempoCpu[i]
            else:
                aux += self.TiempoCpu[i]
            self.TiempoFinal.append(aux)


    def solucion(self):
        try:
            self.ordenar()
            self.calcularTfin()
            self.calcularT()
            self.calcularE()
            self.calcularI()
            self.mostrar()
            self.graficar()
        except Exception as error:
            print(error)
```

Para el algoritmo de OPT:

```
class OPT:
    def __init__(self, Demanda, Marcos):
        self.Demanda = Demanda
        self.Marcos = Marcos
        self.Pila = []#trabaja como memoria
        self.Falla = []
        self.numeroFallos = 0
        self.n = len(self.Demanda)
        self.contadorFifo = 0
        self.anteriorSwFifo = 0

    def PilaVacia(self):
        if len(self.Pila) == 0:
            return True
        return False

    def PilaLlena(self):
        if len(self.Pila) == self.Marcos:
            return True
        return False
```

```
def buscarReferenciaLejana(self, Di, x):
    posicion = -1; c = 0; posPila = 0
    for i in self.Pila:
        if (Di + 1 < self.n) and (i in self.Demanda[Di + 1: ]):
            if posicion == -1:
                posicion = self.Demanda[Di + 1:].index(i)
                posPila = c
            else:
                cand = self.Demanda[Di + 1:].index(i)
                if cand > posicion:
                    posicion = cand
                    posPila = c
        c += 1
    if posicion != -1:
        self.Pila[posPila] = x
        return True
    return False
```

```
def FIFO(self, x, contador):
    if self.PilaVacia() or (not self.PilaLlena()):
        self.Pila.append(x)
    elif self.PilaLlena():
        self.Pila[contador] = x
```

```
def OPT(self):
    Di = 0
    for x in self.Demanda:
        if self.PilaVacia() or (not self.PilaLlena() and not x in self.Pila):
            self.Pila.append(x)
            self.numeroFallos += 1
            self.Falla.append('x')
        elif self.PilaLlena():
            if x in self.Pila:
                #si se encuentra x en la memoria + 1
                self.Falla.append('-')
            else:
                if self.buscarReferenciaLejana(Di, x):
                    # suponemos de que existe la referencia mas lejana aqui se hace uso
                    # del OPT
                    self.numeroFallos += 1
                    self.Falla.append('x')
                    self.anteriorSwFifo = 1

                else:
                    # aqui se usa FIFO porque no encuentra una referencia lejana
                    if self.anteriorSwFifo:
                        self.contadorFifo = 0
                    else:
                        if self.contadorFifo + 1 < self.Marcos:
                            self.contadorFifo += 1
                        else:
                            self.contadorFifo = 0
                    self.FIFO(x, self.contadorFifo)
                    self.Falla.append('x')
                    self.numeroFallos += 1
                    self.anteriorSwFifo = 0
        Di += 1


def mostrarFallas(self):
    print('**********************OPT*****************************')
    print('D',*self.Demanda, '\n'
                             ' ',*self.Falla, sep='  ')
    print(f'NFallos = {self.numeroFallos}\n'
          f'NBienes = {self.n - self.numeroFallos}'
          f'\n')

def solucion(self):
    try:
        self.OPT()
        self.mostrarFallas()
    except Exception as error:
        print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo del FIFO:

```
class FIFO1:  # PARA REEMPLAZO DE PAGINAS
    def __init__(self, Demanda, Marcos):
        self.Demanda = Demanda
        self.Marcos = Marcos
        self.Pila = []  # trabaja como memoria
        self.Falla = []
        self.numeroFallos = 0
        self.n = len(self.Demanda)
        self.contadorFifo = 0

    def PilaVacia(self):
        if len(self.Pila) == 0:
            return True
        return False

    def PilaLlena(self):
        if len(self.Pila) == self.Marcos:
            return True
        return False

    def Asignacion(self, x, contador):
        self.Pila[contador] = x
        self.contadorFifo += 1

    def FIFO(self):
        for x in self.Demanda:
            if self.PilaVacia() or (not self.PilaLlena() and not x in self.Pila):
                self.Pila.append(x)
                self.numeroFallos += 1
                self.Falla.append('x')
            elif self.PilaLlena():
                if x in self.Pila:
                    # si se encuentra x en la memoria se mantiene
                    self.Falla.append('-')
                else:
                    # aqui se usa FIFO porque no encuentra una referencia lejana
                    if not self.contadorFifo < self.Marcos:
                        self.contadorFifo = 0
                    self.Asignacion(x, self.contadorFifo)
                    self.Falla.append('x')
                    self.numeroFallos += 1

    def mostrarFallas(self):
        print('**********************FIFO*****************************')
        print('D', *self.Demanda, '\n'
                                  ' ', *self.Falla, sep='  ')
        print(f'NFallos = {self.numeroFallos}\n'
              f'NBienes = {self.n - self.numeroFallos}'
              f'\n')

    def solucion(self):
        try:
            self.FIFO()
            self.mostrarFallas()
        except Exception as error:
            print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo del LRU:

```
class LRU:
    def __init__(self, Demanda, Marcos):
        self.Demanda = Demanda
        self.Marcos = Marcos
        self.PilaMemoria = []#la memoria
        self.Pila = []#la pila que servira para hacer el reemplazo
        self.Falla = []
        self.numeroFallos = 0
        self.n = len(self.Demanda)
        self.contadorFifo = 0

    def PilaVacia(self, Pila):
        if len(Pila) == 0:
            return True
        return False

    def PilaLlena(self, Pila):
        if len(Pila) == self.Marcos:
            return True
        return False

    def PilaSubirElemento(self, x):
        elem = self.Pila.pop(self.Pila.index(x))
        self.Pila.append(elem)

    def PilaApilarElemento(self, x):
        reemplazar = self.Pila.pop(0)
        self.Pila.append(x)
        self.PilaMemoria[self.PilaMemoria.index(reemplazar)] = x
```

```
    def LRU(self):
        for x in self.Demanda:
            if self.PilaVacia(self.PilaMemoria) or (not self.PilaLlena(self.PilaMemoria)):
                self.PilaMemoria.append(x)
                self.Pila.append(x)
                self.Falla.append('x')
                self.numeroFallos += 1
            else:
                if x in self.PilaMemoria:
                    self.Falla.append('-')
                    self.PilaSubirElemento(x)
                else:
                    self.PilaApilarElemento(x)
                    self.Falla.append('x')
                    self.numeroFallos += 1

    def mostrarFallas(self):
        print('**********************LRU**************************')
        print('D',*self.Demanda, '\n'
                                 ' ',*self.Falla, sep='  ')
        print(f'NFallos = {self.numeroFallos}\n'
              f'NBienes = {self.n - self.numeroFallos}'
              f'\n')

    def solucion(self):
        try:
           self.LRU()
           self.mostrarFallas()
           print(self.PilaMemoria)
           print(self.Pila)
        except Exception as error:
            print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo del FIFO:

```
import numpy as np
from matplotlib import pyplot as plt
class FIFO2:
    def __init__(self, Peticiones, inicio):
        self.Pistas = []
        self.Peticiones = Peticiones
        self.Movimientos = 0
        self.inicio = inicio

    def graficar(self):
        self.Peticiones.insert(0, self.inicio)
        y = np.array(self.Peticiones)
        x = np.array([i for i in range(1, len(self.Peticiones) + 1)])
        plt.plot(x, y, 'og--', mfc='y', mec='y')
        plt.title('FIFO')
        plt.xlabel('Peticiones')
        plt.ylabel('Pistas')
        plt.show()

    def FIFO(self):
        for x in range(len(self.Peticiones)):
            if x == 0:
                self.Movimientos += abs(self.Peticiones[x] - self.inicio)
            else:
                self.Movimientos += abs(self.Peticiones[x] - self.Peticiones[x - 1])

    def solucion(self):
        try:
            self.FIFO()
            print('**********************FIFO**************************')
            print(f'numero de movimientos = {self.Movimientos}')
            self.graficar()
        except Exception:
            print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo del SCHENBACH:

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
class SCHENBACH:
    def __init__(self, Peticiones):
        self.Pistas = []
        self.Peticiones = list(sorted(set(Peticiones)))
        self.Movimientos = 0

    def graficar(self):
        x = np.array(self.Peticiones)
        y = np.array(self.Peticiones)
        plt.plot(x, y, 'og--', mfc='y', mec='y')
        plt.title('SCHENBACH')
        plt.xlabel('Peticiones')
        plt.ylabel('Pistas')
        plt.show()
```

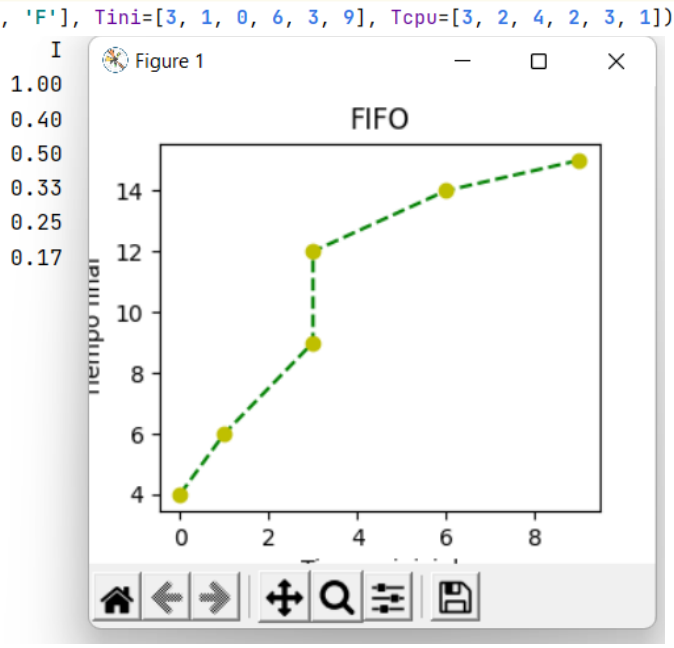
```
def Scheenbach(self):
    self.Movimientos += self.Peticiones[0]
    for i in range(len(self.Peticiones) - 1):
        self.Movimientos += abs(self.Peticiones[i] - self.Peticiones[i + 1])

def solucion(self):
    try:
        self.Scheenbach()
        print('**********************SCHENBACH*****************************')
        print(f'numero de movimientos = {self.Movimientos}')
        self.graficar()
    except Exception:
        print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

Para el algoritmo del SSTF:

```
import numpy as np
from matplotlib import pyplot as plt
class SSTF:
    def __init__(self, Peticiones, inicio):
        self.Pistas = []
        self.Peticiones = Peticiones
        self.Movimientos = 0
        self.inicio = inicio

    def graficar(self):
        self.Peticiones.insert(0, self.inicio)
        y = np.array(self.Peticiones)
        x = np.array([i for i in range(1, len(self.Peticiones) + 1)])
        plt.plot(x, y, 'og--', mfc='y', mec='y')
        plt.title('SSTF(Shortes seek time fast first)')
        plt.xlabel('Peticiones')
        plt.ylabel('Pistas')
        plt.show()

    def ordenar(self):
        auxiliar = []
        auxiliar2 = []
        for x in self.Peticiones:
            if x <= self.inicio:
                auxiliar.append(x)
            else:
                auxiliar2.append(x)
        self.Peticiones.clear()
        self.Peticiones.extend(sorted(auxiliar)[::-1])
        self.Peticiones.extend(sorted(auxiliar2))

    def SSTF(self):
        self.ordenar()
        for x in range(len(self.Peticiones)):
            if x == 0:
                self.Movimientos += abs(self.Peticiones[x] - self.inicio)
            else:
                self.Movimientos += abs(self.Peticiones[x] - self.Peticiones[x - 1])

    def solucion(self):
        try:
            self.SSTF()
            print('**********************SSTF*****************************')
            print(f'numero de movimientos = {self.Movimientos}')
            self.graficar()
        except Exception:
            print("OCURRIO ALGUN ERROR INESPERADO!!!")
```

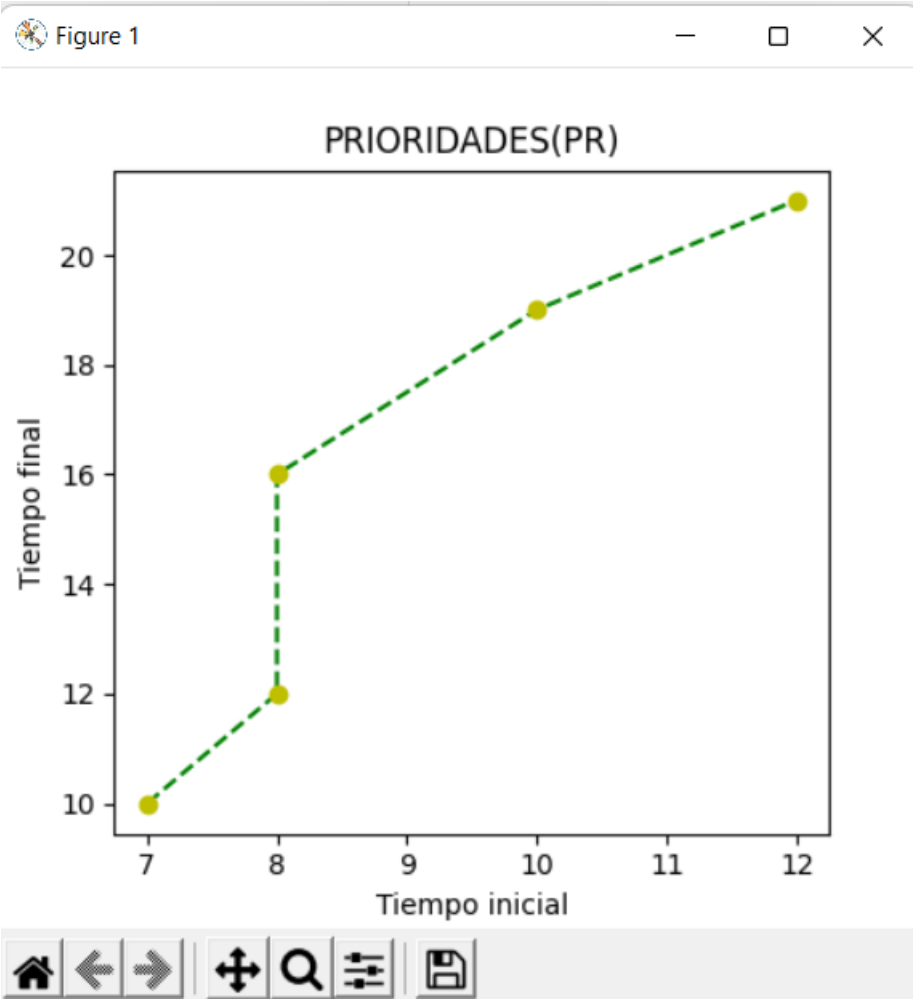
Pruebas de corrida (no de escritorio). Dada una entrada x, el programa produce una salida y que corresponde al algoritmo considerado. Ejecutado la aplicación desarrollada se establece que representa adecuadamente los conceptos de la temática.

```
f1 = FIFO(Proc=['A', 'B', 'C', 'D', 'E', 'F'], Tini=[3, 1, 0, 6, 3, 9], Tcpu=[3, 2, 4, 2, 3, 1])
   Proceso  Tini  Tcpu  Tfin  T  E     I
0        C     0     4     4  4  0  1.00
1        B     1     2     6  5  3  0.40
2        A     3     3     9  6  3  0.50
3        E     3     3    12  9  6  0.33
4        D     6     2    14  8  6  0.25
5        F     9     1    15  6  5  0.17
T = 6.33
E = 3.83
I = 0.44
```



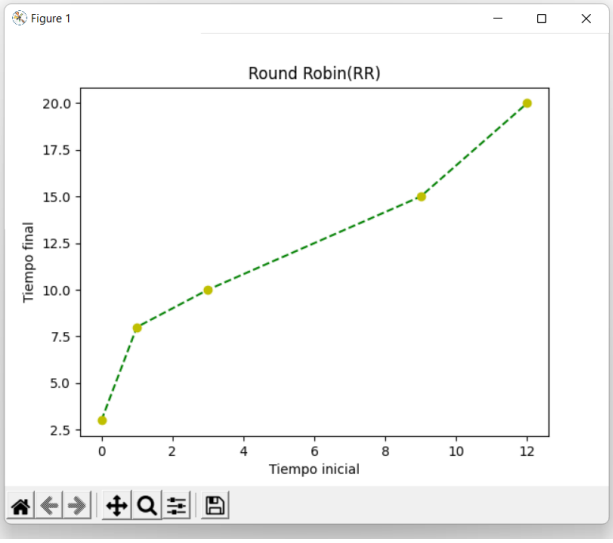
```
P1 = Prioridades(Proc=['A', 'B', 'C', 'D', 'E'], Tini=[7, 8, 8, 10, 12], Tcpu=[3, 4, 2, 3, 2], Prioridades=[3, 2, 1, 4, 5])
```

```
**********Prioridades**********
  Proceso  Tini  Tcpu  Prio  Tfin  T  E     I
0       A     7     3     3    10  3  0  1.00
1       C     8     2     2    12  4  2  0.50
2       B     8     4     1    16  8  4  0.50
3       D    10     3     4    19  9  6  0.33
4       E    12     2     5    21  9  7  0.22
T = 6.6
E = 3.8
I = 0.51
```



```
RR1 = RoundRobin(Proc=['A', 'B', 'C', 'D', 'E'], Tini=[0, 1, 3, 9, 12], Tcpu=[3, 5, 2, 5, 5], Quantum=10)
```

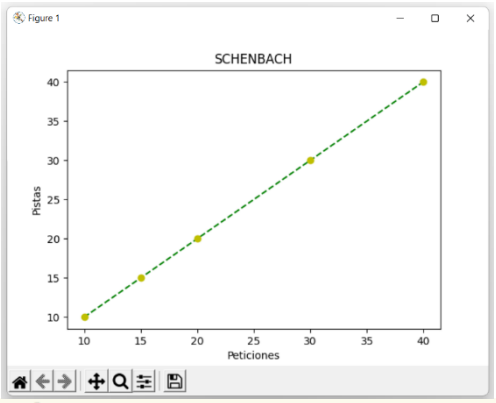
```
"C:\Users\sanjc\ocb nivel 2\Scripts\python.exe" "G:/Cursos por mi cuenta/pycharm proyectos/ocb nivel 2/INF 151/PROYECTO/Main
 .py"
**********Round Robin**********
  Proceso  Tini  Tcpu  Tfin  T  E     I
0       A     0     3     3  3  0  1.00
1       B     1     5     8  7  2  0.71
2       C     3     2    10  7  5  0.29
3       D     9     5    15  6  1  0.83
4       E    12     5    20  8  3  0.62
T = 6.2
E = 2.2
I = 0.69
```



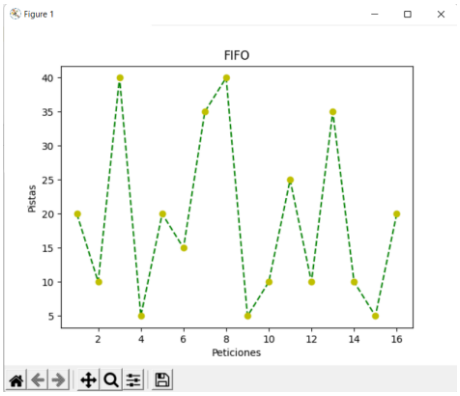

```
O1 = OPT(Demanda=[2, 3, 4, 2, 1, 3, 5, 2, 4, 3, 2, 1], Marcos=3)

**********OPT**********
D  2  3  4  2  1  3  5  2  4  3  2  1
   x  x  x  -  x  -  x  -  x  -  x  x
NFallos = 8
NBienes = 4

F11 = FIFO1(Demanda=[3, 2, 1, 4, 6, 5, 3, 2, 4, 3], Marcos=4)

 **********FIFO**********
 D  3  2  1  4  6  5  3  2  4  3
    x  x  x  x  x  x  x  x  x  -
 NFallos = 9
 NBienes = 1

L1 = LRU(Demanda=[2, 4, 6, 2, 3, 4, 5, 2, 1, 2, 3, 4], Marcos=3)

 **********LRU**********
 D  2  4  6  2  3  4  5  2  1  2  3  4
    x  x  x  -  x  x  x  x  x  -  x  x
 NFallos = 10
 NBienes = 2
```

```
S1 = SCHENBACH(Peticiones=[10, 15, 10, 30, 15, 10, 20, 30, 10, 40, 20, 30])
```

```
**********SCHENBACH**********
numero de movimientos = 40
```

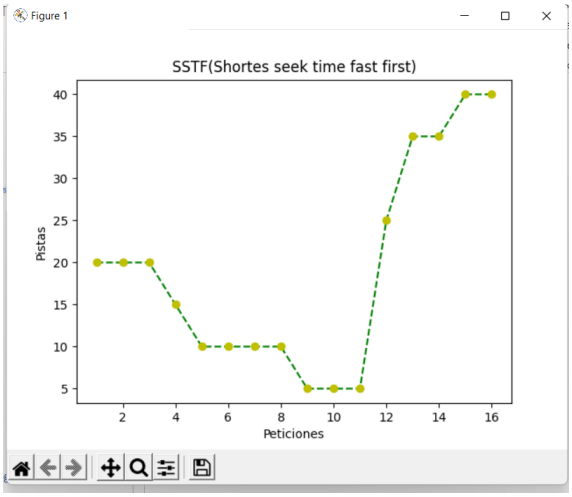



```
FF21 = FIFO2(Peticiones=[10, 40, 5, 20, 15, 35, 40, 5, 10, 25, 10, 35, 10, 5, 20], inicio=20)
```




```
**********FIFO**********
numero de movimientos = 260
```

```
SS1 = SSTF(Peticiones=[10, 40, 5, 20, 15, 35, 40, 5, 10, 25, 10, 35, 10, 5, 20], inicio=20)
```




```
**********SSTF**********
numero de movimientos = 50
```

Conclusiones respecto al producto desarrollado. ¿Se logro lo propuesto?

Se logro lo propuesto, viendo mediante gráficos, además de datos de suma importancia de algoritmos escogidos y mostrando, así como funcionan de manera correcta.