UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO ZUMPANGO

INGENIERÍA EN COMPUTACIÓN

SISTEMAS OPERATIVOS

SIMULADOR FCFS (FIRST COME, FIRST SERVED)

DOCENTE:

MARIA GUADALUPE DOMINGUEZ URBAN

NOMBRES DE LOS INTEGRANTES:

MIGUEL ANGEL FLORES URBINA

LUIS GUSTAVO ROJAS PAREDES

SARA BAUTISTA GONZALEZ

FECHA DE ENTREGA: 08 DE MARZO DE 2021

ÍNDICE

Contenido

INTRODUCCIÓN

En el presente trabajo se mostrará la funcionalidad del algoritmo FCFS, las siglas significan en ingles First Come First Served (Primero en llegar, Primero en ser Servido), dentro de los algoritmos de Planificación de la CPU, este es el más sencillo.

La carga de trabajo se procesa simplemente en un orden de llegada. Por no tener en consideración el estado del sistema ni las necesidades de recursos de los procesos individuales, la planificación FCFS puede dar lugar a pobres rendimientos. Este algoritmo posee un alto tiempo de respuesta de la CPU pues el proceso no abandona la CPU hasta no haber concluido pues es un algoritmo Sin Desalojo (No apropiativo). La planificación FCFS elimina la noción e importancia de las prioridades de los procesos.

FIRST COME FIRST SERVED (FCFS)

1.- Los procesos son ejecutados en el orden que llegan a la cola de procesos listos.

2.- La implementación es fácil a través de una cola FIFO.

3.- Es adecuado para sistemas por lotes (batch).

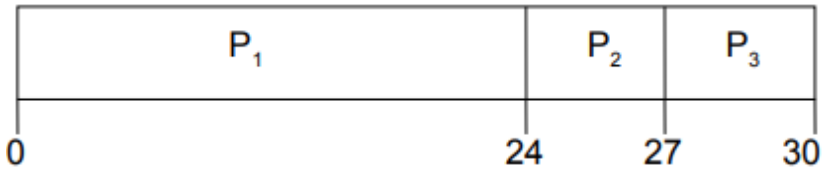
4.- Es un algoritmo no expropiativo: una vez que el procesador le es asignado a un proceso este lo mantiene hasta que termina o se bloquea (por ejemplo, al generar un pedido de E/S).

5.- El tiempo de espera promedio por lo general es alto.

La planificación **FCFS** es justa en el sentido de que una vez que llega una petición, se fija su lugar dentro de la cola de espera. Una petición, se fija su lugar dentro de la cola de espera. Una petición no puede ser desplazada por la llegada de otra con prioridad más alta.

La FCFS es aceptable cuando la carga en un disco es ligera. Pero a medida que crece la carga, la FCFS tiende a saturar el dispositivo y los tiempos de respuesta se incrementan. La FCFS ofrece una varianza pequeña, pero tiene tiempos de espera muy grandes.

Proceso	Burst Time
P1	24
P2	3
P3	3



Tiempo de espera: P1 = 0; P2 = 24; P3 = 27

Tiempo de espera promedio: (0 + 24 + 27)/3 = 17

Para elegir el proceso al cual se le asignara la CPU, se escoge el que lleva más tiempo listo (primero en la cola).

El proceso que se está ejecutando solo cede la CPU por dos razones:

- Que se bloquee voluntariamente en espera de un evento. (Impresora, fichero, etc).
- Cuando termina su ejecución.

Características:

1. Es justa, aunque los procesos largos hacen esperar mucho a los cortos.
2. Predecible.
3. El tiempo medio de servicio es muy variable en función del número de procesos y su duración.

Ventajas

- Optimiza la utilización.
- Muy fácil de implementar (cola FIFO).

Desventajas

- No optimiza el tiempo de espera, retorno y rendimiento.
- Variables en función del orden de llegada y de la duración de
- Intervalos de CPU.
- No adecuado para sistemas interactivos.

REQUERIMIENTOS

- Equipo de cómputo
- Java Netbeans IDE 8.2
- Conocimiento sobre funcionamiento del procesamiento FCFS

DESARROLLO Y PRESENTACIÓN DEL PROGRAMA

ListaDoble.java

En este componente consiste en construir una lista enlazada cuyos nodos se le introducirá información. Las listas nos permiten agregar elementos al inicio, al final o por posición de la lista, modificar elementos, eliminar las listas o los elementos de la lista e imprimir la información introducida.

```
9   public class ListaDoble {
10      private Nodo cabecera;
11      public ListaDoble() {
12          cabecera = null;
13      }
14
15      public boolean esVacio() {
16          return (this.cabecera == null);
17      }
18
19      public int size() {
20          int tamano = 0;
21          Nodo aux = cabecera;
22          if (!esVacio()) {
23              tamano++;
24
25              while (aux.siguiente != null) {
26                  tamano++;
27                  aux = aux.siguiente;
28              }
29          }
30          return tamano;
31      }
32
33      public void insertarCabecera(Proceso dato) {
34          Nodo nuevo = new Nodo(dato);
35          if (esVacio()) {
36              cabecera = nuevo;
37          }
38      }
```

```
39      public void inserta_inicio(Proceso dato){
40        if(!esVacio()){
41              Nodo nuevo=new Nodo(dato);
42              if(cabecera.dato.getTiempo()>nuevo.dato.getTiempo()){
43                  nuevo.siguiente=cabecera;
44                  cabecera.anterior=nuevo;
45                  cabecera=nuevo;
46              }else{
47                  cabecera.siguiente=nuevo;
48                  nuevo.anterior=cabecera;
49              }
50          }
51
52      }
53      public void insertarProcesos_Prioridad(Proceso dato){
54          Nodo nuevo=new Nodo(dato);
55              Nodo aux=cabecera;
56             if(!esVacio()){
57                 if(nuevo.dato.getPrioridad()==0){
58                     inserta_inicio(dato);
59                 }else{
60                     if(aux.siguiente==null){
61                         inserta_inicio(dato);
62                     }else{
63                         while(nuevo.dato.getPrioridad()>aux.dato.getPrioridad()){
64                             if(aux.siguiente==null){
65                                 break;
66                             }
67                             aux=aux.siguiente;
68                         }
```

```
155     public void insertar(int pos,Proceso dato){
156         if(!esVacio()){
157              Nodo aux=cabecera;
158              Nodo nuevo=new Nodo(dato);
159              if(aux.siguiente==null){
160                  inserta_inicio(dato);
161              }else if(aux.siguiente.siguiente==null){
162                  aux.siguiente=nuevo;
163                  nuevo.siguiente=aux.siguiente;
164                  nuevo.anterior=aux;
165                  aux.siguiente.anterior=nuevo;
166              }else{
167
168              }
169          }
170      }
171      public void insertarInicio(Proceso dato) {
172          Nodo nuevo = new Nodo(dato);
173          if (!esVacio()) {
174              nuevo.siguiente = cabecera;
175              cabecera.anterior = nuevo;
176              cabecera = nuevo;
177          }
178      }
179
180      public void insertarFinal(Proceso dato) {
181          Nodo aux = cabecera;
182          Nodo nuevo = new Nodo(dato);
183          if (!esVacio()) {
184              while (aux.siguiente != null) {
```



```
225     public void modificar(int pos, Proceso datos) {
226         Nodo auxiliar = cabecera;
227         int recorrido = 0;
228         if (!esVacio()) {
229             if (pos == 0) {
230                 cabecera.dato = (Proceso) datos;
231             } else {
232                 if (pos == size()) {
233                     get(pos).dato = (Proceso) datos;
234                 } else {
235                     if (pos > 0 & pos < size()) {
236                         Nodo nuevo = new Nodo(datos);
237                         while (recorrido != (pos - 1)) {
238                             auxiliar = auxiliar.siguiente;
239                             recorrido++;
240                         }
241                         nuevo.siguiente = auxiliar.siguiente;
242                         auxiliar.siguiente.dato = nuevo.dato;
243                     } else {
244                         JOptionPane.showMessageDialog(null, "EL ELEMENTO ES MAYOR AL TAMAÑO DE LA LISTA");
245                     }
246                 }
247             }
248         } else {
249             JOptionPane.showMessageDialog(null, "LA LISTA ESTA VACIA");
250         }
251
252     }
253
254     public Nodo get(int posicion) {...9 lines }
```

```java
    public void eliminarLista() {
        if (!esVacio()) {
            cabecera.siguiente = null;
            cabecera = null;
        } else {
        }
    }

    public void eliminarInicio() {
        Nodo aux = cabecera;
        if (cabecera.siguiente != null) {
            cabecera = aux.siguiente;
            cabecera.anterior = null;
            aux.siguiente = null;
        } else {
            cabecera = null;
        }
    }

    public void eliminarFinal() {...14 lines }

    public void eliminarEntreNodos(int pos) {...31 lines }

    public String imprimir() {
        String informacion = "";
        Nodo actual = cabecera;
        System.out.println("DATOS INGRESADOS: ");
        while (actual != null) {
            informacion += actual.dato.toString() + "\n";
            actual = actual.siguiente;
```

Nodo.java

En este componente simplemente se declara el nodo para utilizar la clase, se definen los punteros del proceso dato que guardara el valor del dato y los nodos siguiente y anterior que guardaran el valor null, si el nodo tiene almacenado null la lista está vacía, en caso contrario tiene la dirección del primer nodo de la lista.

```java
public class Nodo {
    public Proceso dato;
    public Nodo siguiente;
    public Nodo anterior;
    public Nodo(Proceso dato) {
        this.dato = dato;
        this.siguiente=null;
        this.anterior=null;
    }

    public Proceso getDato() {
        return dato;
    }

    public void setDato(Proceso dato) {
        this.dato = dato;
    }

    public Nodo getAnterior() {
        return anterior;
    }

    public void setAnterior(Nodo anterior) {
        this.anterior = anterior;
    }
}
```

Proceso.java

En el siguiente componente se declaran los procesos que tendrá el programa, el nombre y el estado serán de tipo string y ráfaga, tiempo, prioridad, estado, tiempo de espera (tiempoespera), ráfaga auxiliar (rafagaaux), tiempo de retorno

(tiemporetorno) y tiempo de respuesta (tiemporespuesta) serán de tipo entero; estos punteros son internos a la lista.

```java
public class Proceso {
    private String nombre;
    private int rafaga;
    private int tiempo;
    private int prioridad;
    private String estado;
    private int tiempoespera;
    private int rafagaaux;
    private int tiemporetorno;
    private int tiemporespuesta;
    public Proceso() {
    }

    public Proceso(String nombre, int rafaga, int tiempo, int prioridad, String estado) {
        this.nombre = nombre;
        this.rafaga = rafaga;
        this.tiempo = tiempo;
        this.prioridad = prioridad;
        this.estado = estado;
    }

    public String getNombre() {
        return nombre;
    }

    public int getRafagaaux() {
        return rafagaaux;
    }

    public int getTiemporespuesta() {
        return tiemporespuesta;
```

Threader.java

Esa clase sirve para distribuir los procesos en un hilo sirve para ejecutar varios procesos simultáneamente sobre la memoria, pero en este caso no los ejecutará simultáneamente porque es FCFS, sino que lo hará secuencialme, uno tras otro Para que de esta forma al momento de generar la interfaz tenga la posibilidad de desplazarse entre los gráficos y las listas dobles con el scroll

```java
public class Threader extends Thread{
    JScrollPane graficos;
    ListaDoble listaDoble=new ListaDoble();
    public Threader(JScrollPane graficos,ListaDoble listaDoble){
        this.graficos=graficos;
        this.listaDoble=listaDoble;
    }
    public void vuelta(){

        try {
            graficos.setViewportView(new Graficos(listaDoble.size(), listaDoble));
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Threader.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Gráficos.java

En esta jframe es para la interfaz gráfica del programa y se creara el panel principal del programa. Insertaremos un panel (JPanel) en el contenedor y el tamaño del componente con el método getSize() con una dimensión de 300 pixeles de ancho y con 20. El contexto grafico del panel tendrá El método un setColor establece el color fondo gris, el método fillRect crea un rectángulo relleno en el color y el método drawString escribe un String en el color negro en las coordenadas especificadas.

```
19  public class Graficos extends javax.swing.JPanel {
20      String nombre;
21      int cont=0;
22      ListaDoble listaDoble=new ListaDoble();
23
24      public Graficos(int cont,ListaDoble listaDoble) {
25          initComponents();
            this.setOpaque(true);
            this.setSize(new Dimension(3000,20));
            this.setBackground(Color.black);
29          this.cont=cont;
30          this.listaDoble=listaDoble;
31
32      }
        public void paint(Graphics g){
34      super.paint(g);
35          g.setColor(Color.gray);
36
37          int disNodo = 50,aumentoNodo = 120;
38
39          int disNombre = 77,aumentoNombre=120;
40
41          for (int i = 0; i < cont; i++) {
42
43              //g.drawImage(img.getImage(), disNodo, 35, 170, 50, this);
44              g.setColor(Color.gray);
45              g.fillRect(disNodo, 35, 100, 50);
46              disNodo += aumentoNodo;
47
48              this.setSize(disNodo+200, 21);
49
```

```
49
50              g.setColor(Color.black);
51              g.drawString(listaDoble.get(i).dato.getNombre(), disNombre, 60);
52
53              this.repaint();
54
55
56              disNombre += aumentoNombre;
57
58          }
59
60      }
```

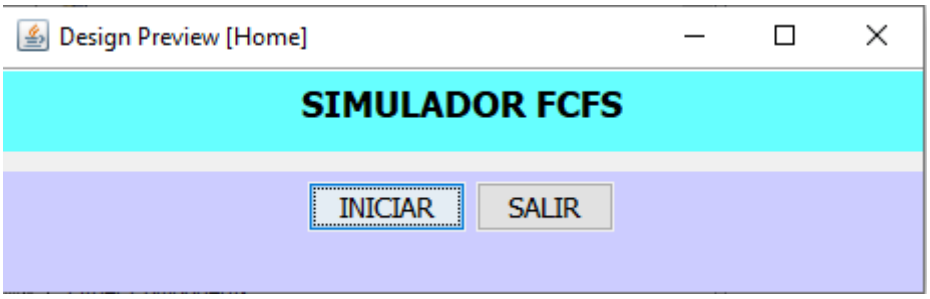
Home.java

Es un JFrame utilizado como menú principal el cual da entrada al simulador con las opciones de Iniciar y Salir. En caso de presionar sobre el primer botón se inicia el programa y se muestra la ventana con los espacios para introducir los datos.

```
private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Simulador p=new Simulador();
    p.setVisible(true);
    this.setVisible(false);
    dispose();
}

private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}
```

La pequeña ventana de inicio se muestra sencilla para que el usuario se concentre únicamente en estas dos opciones. Salir da por terminada la ejecución del programa.



Simulador.java

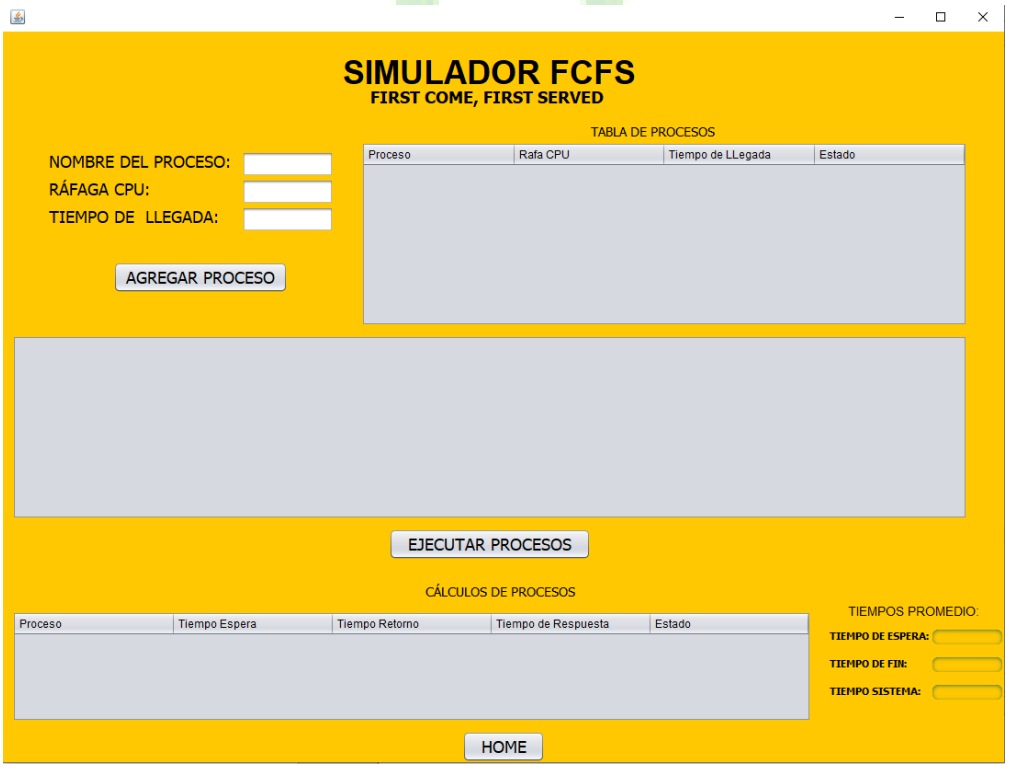
Es el JFrame donde se realiza la simulación de los procesos, en la parte superior izquierda permite introducir los datos sobre los procesos, por lo que requiere de un nombre de proceso, tiempo de ráfaga de procesamiento y un tiempo de llegada específico para el mismo. La introducción de los datos de procesos se guarda en una tabla que se muestra en la parte superior derecha de la ventana al hacer clic sobre el botón Agregar Proceso. Al mismo tiempo que en la parte media se grafican los procesos que se van agregando a la primera tabla, para que al dejar de introducir procesos el simulador prepare los procesos al hacer clic en el botón Ejecutar Procesos, de manera que se ordenan los procesos en el gráfico y la tabla de la parte inferior muestra los resultados para el procesamiento FCFS y los valores de tiempo promedio utilizados.

```
37      DefaultTableModel m;
38      DefaultTableModel aux;
        Vector<Object>listaAlgoritmos;
40      ListaDoble listaDoble=new ListaDoble();
41      public Simulador() {
42          initComponents();
43          getContentPane().setBackground(Color.ORANGE);
            listaAlgoritmos=new Vector<Object>();
            tabla();
46
            tablaResultados();
48  //          bloqueo_cajas_texto();
49  //          bloqueo_botones();
            setLocationRelativeTo(this);
51
52      }
53      Planning f=new Planning(listaDoble);
54       public void tabla(){
55           String cabecera[]={"Proceso","Rafa CPU", "Tiempo de LLegada","Estado"};
56           m=new DefaultTableModel(null,cabecera);
57           tblProcesos.setModel(m);
```

```
75      public void tablaResultados(){
76          String cabecera[]={"Proceso","Tiempo Espera", "Tiempo Retorno","Tiempo de Respuesta","Estado"};
77          aux=new DefaultTableModel(null,cabecera);
78          tblResultados.setModel(aux);
79      }
80
81      int CPU;
82      int tiempo;
83      int prioridad;
84      int numeroProcesos=0;
85      void fifo(){
86
87          CPU=Integer.parseInt(txtRafaga.getText());
88          tiempo=Integer.parseInt(txtTiempo.getText());
89
90          String estado="Preparado";
91          String Datos[]= new String[5];
92          Datos[0]=txtProceso.getText();
93          Datos[1]=String.valueOf(CPU);
94          Datos[2]=String.valueOf(tiempo);
95          Datos[3]=String.valueOf(estado);
96          m.addRow(Datos);
97          Proceso p=new Proceso(txtProceso.getText(), CPU, tiempo, prioridad, estado);
98          listaDoble.insertarPrincipio(p);
99
100         jspGraficos.setViewportView(new Graficos(listaDoble.size(), listaDoble));
101     }
```

```
309     private void btnEjecutarProcesosActionPerformed(java.awt.event.ActionEvent evt) {
310         // TODO add your handling code
            Planning f=new Planning(listaDoble);
312         f.FCSC();
313         jspGraficos.setViewportView(new Threads(listaDoble.size(), listaDoble));
314         tiempoProcesos_FIFO();
315         bloqueo_botones();
316         bloqueo_cajas_texto();
317         f.FCSC();
318         lblTPE.setText(String.valueOf(f.tiempoEspera()));
319         lblTPRet.setText(String.valueOf(f.tiempoRetorno()));
320         lblTPResp.setText(String.valueOf(f.tiempoRespuesta()));
```

La ventana vacía se visualiza de la siguiente manera:



Threads.java

Esta es una clase controla los gráficos de los procesos mediante rectángulos que muestra cada proceso como un objeto. Cada rectángulo se introduce al espacio destinado sobre el JFrame Simulador.java para después ser ordenado y colocado en el lugar que le corresponde una vez que el procesamiento FCFS ha ocurrido y se han calculado los tiempos. Se trata de la formación de un hilo de ejecución gráfico para observar los tiempos necesarios.

```
22    public class Threads extends javax.swing.JPanel {
23
24        /**
25         * Creates new form Hilos
26         */
27         String nombre;
28        int cont=0;
29        ListaDoble listaDoble=new ListaDoble();
30        Planning f=new Planning(listaDoble);
31
32        public Threads(int cont,ListaDoble listaDoble) {
33            initComponents();
34            this.setOpaque(true);
35            this.setSize(new Dimension(3000,20));
36            this.setBackground(Color.black);
37            this.cont=cont;
38            this.listaDoble=listaDoble;
39        }
40
42         public void paint(Graphics g){
42        super.paint(g);
43        g.setColor(Color.white);
44            g.setColor(Color.gray);
45
46            int disNodo = 50,aumentoNodo = 120;
47
48            int disNombre = 77,aumentoNombre=120;
49
50            for (int i = 0; i < cont; i++) {
51
52                //g.drawImage(img.getImage(), disNodo, 35, 170, 50, this);
53                g.setColor(Color.gray);
54                g.fillRect(disNodo, 80, 100, 50);
55                disNodo += aumentoNodo;
```

```
57                this.setSize(disNodo+200, 21);
58
59                g.setColor(Color.black);
60                g.drawString(listaDoble.get(i).dato.getNombre(), disNombre, 100);
61
62    //              g.setColor(Color.orange);
63    //              g.drawString(String.valueOf(tiempoespera), disNombre, 190);
64                this.repaint();
65
66                disNombre += aumentoNombre;
```

Planning.java

La clase Planning.java es la que realiza todos los cálculos relacionados con la vista gráfica del simulador. Los cálculos se insertan mediante una lista donde se introducen los tiempos en los que ocurren los procesos para ser transferidos hacia el JFrame donde son tomados para introducirlos a una tabla mediante el método FCFS(). El método tiempoEspera() realiza el cálculo de los tiempos de espera en base a los datos introducidos por el usuario en el JFrame con otra lista que requerirá el formulario de simulación para visualizar los resultados.

Así mismo, los métodos de tiempoRetorno() y tiempoRespuesta() implementarán otra lista cada uno para ir calculando los procesos en su orden de llegada para retornar los resultados finales hacia el usuario tras la ejecución de los procesos. Al final se pueden presentar los procesos con el método Presentar() para obtener las listas con los resultados anteriores.

```
12  public class Planning {
13      ListaDoble listaDoble=new ListaDoble();
14      public double tiempoespera;
15      public double tiemporetorno;
16      public double tiemporespuesta;
17      public Planning(ListaDoble listaDoble){
18          this.listaDoble=listaDoble;
19      }
20      public void FCSC(){
21          for(int i=1;i<listaDoble.size();i++){
22              for(int j=0;j<listaDoble.size()-1;j++){
23                  if(listaDoble.get(i).dato.getTiempo()<listaDoble.get(j).dato.getTiempo()){
24                     Proceso aux=listaDoble.get(j).getDato();
25                     listaDoble.modificar(j, listaDoble.get(i).dato);
26                     listaDoble.modificar(i, aux);
27                  }
28              }
29          }
30      }
31      public double tiempoEspera(){
32          double t = 0;
33          double r=0;
34          String resultado;
35          for(int i=0;i<listaDoble.size();i++){
36                  if(i==0){
37                    tiempoespera+=listaDoble.get(i).dato.getRafaga()+listaDoble.get(i).dato.getTiempo();
38                    t=listaDoble.get(i).dato.getTiempo();
39                    r+=t;
40                    System.out.println(listaDoble.get(i).dato+"="+t);
41                  }else{
42                      t=tiempoespera-listaDoble.get(i).dato.getTiempo();
43                      r+=t;
44                      tiempoespera+=listaDoble.get(i).dato.getRafaga();
45                      System.out.println(listaDoble.get(i).dato+"="+t);
46                  }
```

```
52      public double tiempoRetorno(){
53          double t = 0;
54          double r=0;
55          for(int i=0;i<listaDoble.size();i++){
56                  if(i==0){
57                    tiemporetorno+=listaDoble.get(i).dato.getRafaga()+listaDoble.get(i).dato.getTiempo();
58                    t=listaDoble.get(i).dato.getTiempo()+listaDoble.get(i).dato.getRafaga();
59                    r+=t;
60                    System.out.println(listaDoble.get(i).dato+"="+t);
61                  }else{
62                      t=tiemporetorno+listaDoble.get(i).dato.getRafaga();
63                      r+=t;
64                      tiemporetorno+=listaDoble.get(i).dato.getRafaga();
65                      System.out.println(listaDoble.get(i).dato+"="+t);
66                  }
```

```
72      public double tiempoRespuesta(){
73          double t = 0;
74          double r=0;
75          for(int i=0;i<listaDoble.size();i++){
76                  if(i==0){
77                    tiemporespuesta+=listaDoble.get(i).dato.getRafaga()+listaDoble.get(i).dato.getTiempo();
78                    t=listaDoble.get(i).dato.getTiempo();
79                    t=t+listaDoble.get(i).dato.getRafaga();
80                    r+=t;
81                    System.out.println(listaDoble.get(i).dato+"="+t);
82                  }else{
83                     t=tiemporespuesta-listaDoble.get(i).dato.getTiempo();
84                     t=t+listaDoble.get(i).dato.getRafaga();
85                     r+=t;
86                     tiemporespuesta+=listaDoble.get(i).dato.getRafaga();
87                     System.out.println(listaDoble.get(i).dato+"="+t);
88                  }
89
90          }
91          System.out.println("Tiempo Promedio:");
92          return r/listaDoble.size();
93      }
94      public String presentar(){
95          String respuesta="";
96          for(int i=0;i<listaDoble.size();i++){
97              respuesta+=String.valueOf(listaDoble.get(i).dato);
98          }
99          return respuesta;
```

Prioridad.java

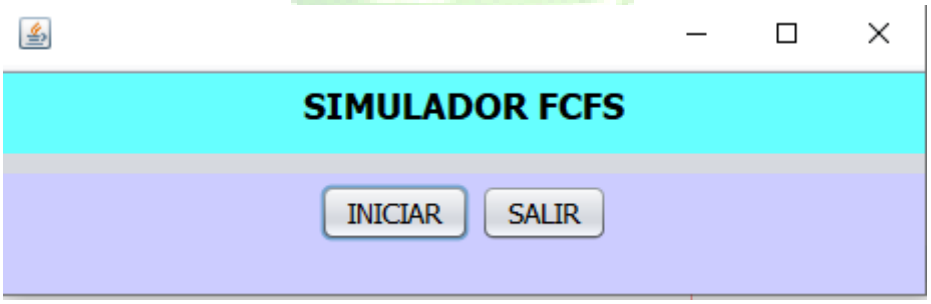
En el procesamiento no existe una prioridad muy estricta como en otros algoritmos para la ejecución de procesos, pero si es necesario comprobar que los procesos se ejecuten en el mismo orden en que fueron introducidos, ésa es la función de esta clase: comprobar el procesamiento desde el que entró en los primeros segundos o cualquier unidad de tiempo que se considere para la ejecución de los procesos, hasta el que ingresó al procesador en los últimos segundos de recibimiento. En todo momento se respeta el tiempo de ráfaga de cada uno, en tanto que habrá una espera dependiendo de la duración asignada a los procesos, y el orden de ejecución se conserva.

```
15  public class Prioridad {
16
17      ListaDoble listaDoble=new ListaDoble();
18      public double tiempoespera;
19      public double tiemporetorno;
20      public double tiemporespuesta;
21      public Prioridad(ListaDoble listaDoble){
22          this.listaDoble=listaDoble;
23      }
24      public void FCSC(){
25          for(int i=1;i<listaDoble.size();i++){
26              for(int j=0;j<listaDoble.size()-1;j++){
27                  if(listaDoble.get(i).dato.getTiempo()<listaDoble.get(j).dato.getTiempo()){
28                     Proceso aux=listaDoble.get(j).getDato();
29                     listaDoble.modificar(j, listaDoble.get(i).dato);
30                     listaDoble.modificar(i, aux);
31                  }
32              }
33          }
34      }
35      public void ordena(){
36          listaDoble.Prioridad();
37      }
38      public void algoritmoPrioridad(){
39          Proceso base;
40          int rafaga = listaDoble.get(0).dato.getRafaga() + listaDoble.get(0).dato.getTiempo();
41          for (int i = 0; i < listaDoble.size()-1; i++) {
42              base = listaDoble.get(i + 1).dato;
43              for (int j = i + 1; j < listaDoble.size()-1; j++) {
44
45                  if (listaDoble.get(j + 1).dato.getPrioridad() < base.getPrioridad()) {
46                      if (listaDoble.get(j + 1).dato.getTiempo() <= rafaga) {
47                          base = listaDoble.get(j + 1).dato;
```

```java
    public String presentar(){
      String respuesta="";
      for(int i=0;i<listaDoble.size();i++){
          respuesta+=String.valueOf(listaDoble.get(i).dato);
      }
      return respuesta;
    }
    public void presentar2(){
        for(int i=0;i<listaDoble.size();i++){
            System.out.println(listaDoble.get(i).dato.getNombre()
        }
    }
  public static void main(String[] args) {
   ListaDoble l=new ListaDoble();
   Proceso p1=new Proceso("p1", 3, 2, 2, null);
   Proceso p2=new Proceso("p2", 1, 4, 3, null);
   Proceso p3=new Proceso("p3", 3, 0, 1, null);
   Proceso p4=new Proceso("p4", 4, 1, 3, null);
   Proceso p5=new Proceso("p5", 2, 3, 4, null);
   l.insertarCabecera(p1);
   l.insertarInicio(p2);
   l.insertarInicio(p3);
   l.insertarInicio(p4);
   l.insertarInicio(p5);
   Prioridad p=new Prioridad(l);
   p.FCSC();
   p.algoritmoPrioridad();
   System.out.println(p.presentar());
   p.presentar2();
  }
```

PRUEBAS DE FUNCIONAMIENTO

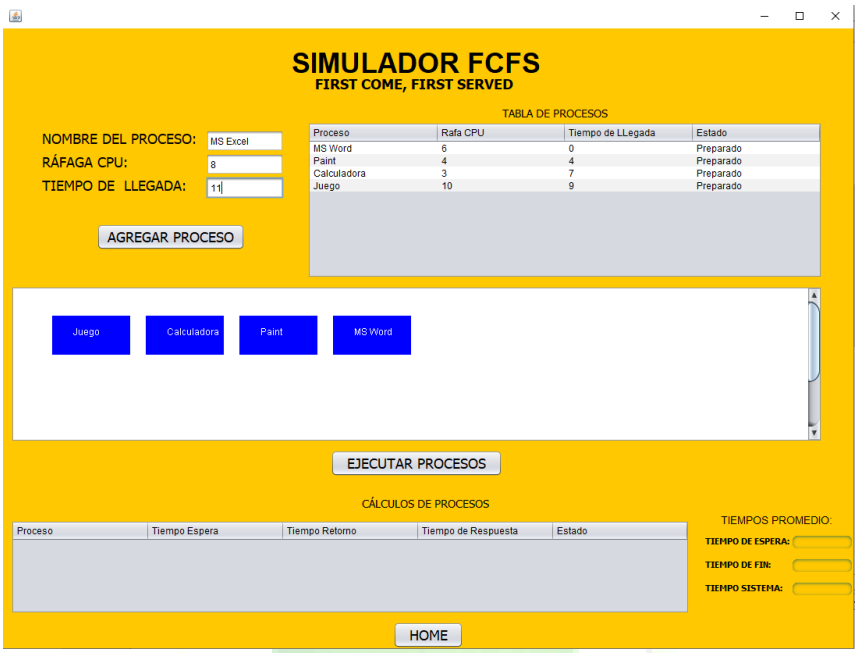
Accedemos al simulador, y la primera interfaz gráfica que se muestra es la de inicio:



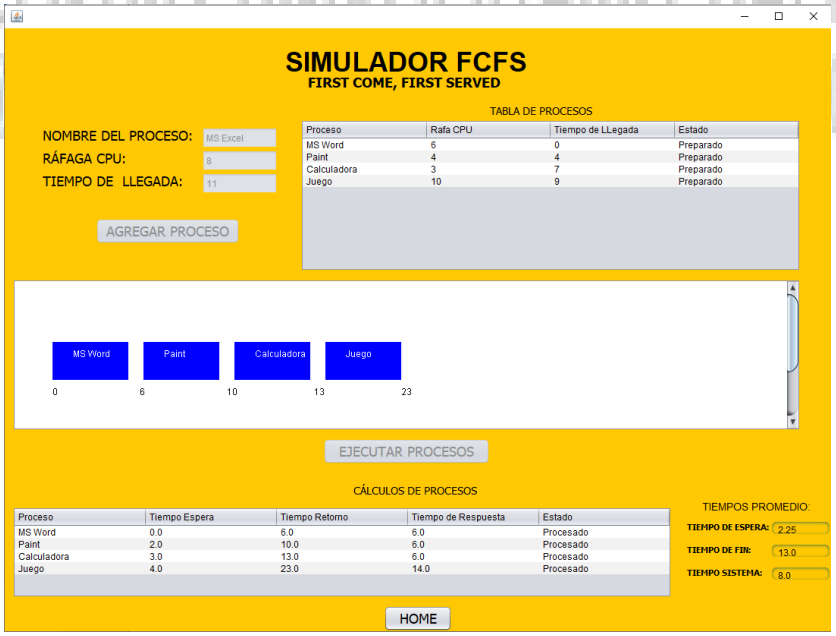
Hacemos clic en Iniciar y nos muestra el JFrame donde especificaremos los siguientes procesos:

PROCESO	RÁFAGA CPU	TIEMPO DE LLEGADA
MS Word	6	0
Paint	4	4
Calculadora	3	7
Juego	10	9
MS Excel	8	11

Los datos de los procesos los agregamos a la tabla de registro de Procesos que se muestra en la parte superior derecha del JFrame de simulación y por cada proceso introducido hacer clic en Agregar Proceso. También en la parte media de la vista gráfica se encuentran los procesos representados por rectángulos en el espacio de ejecución, se muestran en el orden en que fueron introducidos por el usuario.



Con los procesos ya preparados para la ejecución en secuencia ordenada, al hacer clic en el botón Ejecutar Procesos, comenzará el procesamiento de los elementos, el espacio de representación gráfica de procesos ordenará los procesos y dará a conocer los tiempos de término de ejecución entre cada objeto o proceso. En la tabla inferior se muestran los valores calculados para los procesos y se considera que su ejecución ha finalizado.



Para obtención de una información generalizada sobre los valores de procesamiento calculados en el simulador, se muestran los promedios de tiempo de espera, de respuesta y de retorno que arrojó el procesador.

Teóricamente los valores de tiempo de espera se calculan con la resta del tiempo de inicio de ejecución del proceso y el tiempo en que ingresó al procesador; para el tiempo de retorno se toma la marca de tiempo en que el proceso terminó de ejecutarse; y finalmente el tiempo de respuesta es el tiempo en que el proceso permaneció dentro del sistema.

Si hemos finalizado con el procesamiento, se puede reiniciar el sistema mediante el botón Home, el cual nos dirige a la pequeña ventana inicial mostrada al principio de las pruebas, y al ingresar nuevamente se muestra la interfaz gráfica del simulador vacía y lista para recibir nuevos procesos.

CONCLUSIÓN

El planificador FCFS no es muy eficiente, ya que este genera gran tiempo de espera, y lo que se conoce como efecto convoy, efecto que ocurre al tener varios procesos limitados por E/S y pocos limitados por CPU, donde en determinados momentos el CPU o los dispositivos de E/S quedan inactivos. Pero es un planificador básico que tiene que estar.

Para su implementación se podría haber utilizado una lista, pero eso generaría mucha sobrecarga al sistema operativo sobre todo porque habría que seguir manteniendo su coherencia a medida que se cambia a otros planificadores.

Como posible mejora se le podría atribuir prioridades. Es decir, que el algoritmo seleccione como debe, y además teniendo en cuenta las prioridades de los procesos. De esta forma, el proceso que haya estado primero en estado Listo tenga a la vez mayor prioridad, será el que pasará a manos del procesador.

REFERENCIAS

Anonimo. (15 de Junio de 2017). *GESTION DE PROCESOS INFORMATICOS*. Obtenido de Políticas de planificación de procesos. Ventajas y desventajas: http://gestion-de-procesos-informaticos.blogspot.com/2017/06/politicas-de-planificacion-de-procesos.html

Mendoza, C. (12 de Enero de 2014). *Curso 2014*. Obtenido de Sistemas Operativos: https://www.fing.edu.uy/inco/cursos/sistoper/recursosTeoricos/6-SO-Teo-Planificacion.pdf

Pozo, D. (14 de Septiembre de 2016). *CIENCIA DE LA COMPUTACIÓN*. Obtenido de Procesos: Algoritmos de Planificación: https://danielpozoblog.wordpress.com/2016/09/14/procesos-algoritmos-de-planificacion/

Valerio, M. A. (20 de Febreo de 2019). *ISSUU*. Obtenido de Algoritmo FCFS: https://issuu.com/marcocastillo97/docs/instituto_tecnologico_superior_de_l_bdb0f3363df4a1