UNA-PUNO

F I M E E S

E. P. DE INGENIERÍA DE SISTEMAS

TAREA 2

Informe del Trabajo Final de la Unidad I

CURSO:

SISTEMAS OPERATIVOS

DOCENTE:

ING. FERNANDEZ CHAMBI MAYENKA

AUTOR:

GUTIERREZ CHAMBILLA RUSSO WILLIAMS

russodx@gmail.com

951 020 703

2023

SEMESTRE I

TRABAJO FINAL

Enunciado de la tarea:

Use esta tarea para presentar el informe correspondiente al trabajo final de la unidad I.

Incluye la URL del video explicativo.

TABLA DE CONTENIDO

1. PARTE 1

Hacer un programa que permita mostrar la ejecución de 4 planificadores vistos en el capítulo de PLANIFICACIÓN DE LA CPU (son opciones del programa). Cada planificador (opción elegida por el usuario) deberá imprimir o mostrar (versión gráfica) la secuencia de ejecución de los procesos y calcular la métrica Tiempo de espera.

El usuario puede ingresar la lista de procesos y otros valores que sean necesarios para la ejecución de cada planificador (desde teclado, GUI, archivo) No es necesario mostrar los cambios de contexto.

1.1. FCFS (First-Come, First-Served)

Definición

Es un algoritmo de planificación de procesos que asigna la CPU al primer proceso que llega y lo ejecuta hasta su finalización.

Algoritmo en lenguaje java

```
    /*1.    FCFS (First-Come, First-Served) - Primero en llegar, primero en ser servido.*/
    public void algoritmo_fsfc(){
        //int[] this.info_tllegada     public void algoritmo_fsfc(){= info_tllegada; // Tiempos de
llegada de los procesos
        //int[] this.info_tejecucion = info_tejecucion; // Tiempos de ejecución de los procesos

        int n = this.info_tllegada.length;
        int[] tiemposEspera = new int[n];
        int[] tiemposRetorno = new int[n];
        int[] tiemposFinalizacion = new int[n];
        int[] tiemposInicio = new int[n];

        // Calcular los tiempos de espera, tiempos de retorno y tiempos de finalización
        int tiempoTotal = 0;
        for (int i = 0; i < n; i++) {
            if (this.info_tllegada[i] > tiempoTotal) {
                tiempoTotal = this.info_tllegada[i];
            }
            tiemposInicio[i] = tiempoTotal;
            tiemposEspera[i] = tiempoTotal - this.info_tllegada[i];
            tiempoTotal += this.info_tejecucion[i];
            tiemposFinalizacion[i] = tiempoTotal;
            tiemposRetorno[i] = tiemposFinalizacion[i] - this.info_tllegada[i];
        }
        // Calcular el tiempo de espera promedio
        double tiempoEsperaPromedio = Arrays.stream(tiemposEspera).average().orElse(0);
        System.out.println("Proceso\t\tLlegada\t\tRáfaga\t\tEspera\t\tRetorno\t\tFinalización");
        for (int i = 0; i < n; i++) {
            System.out.printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, this.info_tllegada[i],
this.info_tejecucion[i],
                    tiemposEspera[i], tiemposRetorno[i], tiemposFinalizacion[i]);
        }

        System.out.println("\nDiagrama de Gantt:");
        for (int i = 0; i < n; i++) {
            System.out.print("| P" + (i + 1) + " ");
        }
        System.out.println("|");
        for (int i = 0; i < n; i++) {
            System.out.print(tiemposInicio[i] + "\t");
        }
        System.out.println(tiemposFinalizacion[n - 1]);
```

```
        System.out.println("\nTiempo de espera promedio: " + tiempoEsperaPromedio);
    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
1
1.      FCFS (First-Come, First-Served)

Proceso         Llegada         R□faga          Espera          Retorno         Finalizaci□n
P1              1               10              0               10              11
P2              2               1               9               10              12
P3              3               2               9               11              14
P4              4               1               10              11              15

Diagrama de Gantt:
| P1 | P2 | P3 | P4 |
1       11      12      14      15

Tiempo de espera promedio: 7.0
////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://www.geeksforgeeks.org/first-come-first-serve-cpu-scheduling-non-preemptive/

https://www.thejavaprogrammer.com/java-program-first-come-first-serve-fcfs-scheduling-algorithm/

https://www.thejavaprogrammer.com/java-program-shortest-job-first-sjf-scheduling/

1.2. **SJF (Shortest Job First)**

Definición

Este algoritmo prioriza la ejecución de los procesos más breves antes que los más largos.

Algoritmo en lenguaje java

```
    /*2.    SJF (Shortest Job First) - El trabajo más corto primero.*/
    public void algoritmo_sjf() {
        int n = this.info_tllegada.length;
        int[] tiempoInicio = new int[n];
        int[] tiempoFinalizacion = new int[n];
        int[] tiempoEspera = new int[n];
        int[] tiempoRetorno = new int[n];
        int[] tiempoRespuesta = new int[n];

        int tiempoTotalEspera = 0;
        int tiempoTotalRespuesta = 0;

        // Ordenar los procesos por tiempo de ráfaga de menor a mayor
        int[] orden = new int[n];
        for (int i = 0; i < n; i++) {
            orden[i] = i;
        }
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (this.info_tejecucion[j] > this.info_tejecucion[j + 1]) {
                    int temp = this.info_tejecucion[j];
```

```
                    this.info_tejecucion[j] = this.info_tejecucion[j + 1];
                    this.info_tejecucion[j + 1] = temp;
                    temp = this.info_tllegada[j];
                    this.info_tllegada[j] = this.info_tllegada[j + 1];
                    this.info_tllegada[j + 1] = temp;
                    temp = orden[j];
                    orden[j] = orden[j + 1];
                    orden[j + 1] = temp;
                }
            }
        }

        tiempoInicio[0] = this.info_tllegada[0];
        tiempoFinalizacion[0] = tiempoInicio[0] + this.info_tejecucion[0];
        tiempoEspera[0] = 0;
        tiempoRetorno[0] = this.info_tejecucion[0];
        tiempoRespuesta[0] = tiempoEspera[0];
        tiempoTotalEspera += tiempoEspera[0];
        tiempoTotalRespuesta += tiempoRespuesta[0];

        for (int i = 1; i < n; i++) {
            tiempoInicio[i] = tiempoFinalizacion[i - 1];
            tiempoFinalizacion[i] = tiempoInicio[i] + this.info_tejecucion[i];
            tiempoEspera[i] = tiempoInicio[i] - this.info_tllegada[i];
            tiempoRetorno[i] = tiempoFinalizacion[i] - this.info_tllegada[i];
            tiempoRespuesta[i] = tiempoEspera[i];
            tiempoTotalEspera += tiempoEspera[i];
            tiempoTotalRespuesta += tiempoRespuesta[i];
        }

        double tiempoPromedioEspera = (double) tiempoTotalEspera / n;
        double tiempoPromedioRespuesta = (double) tiempoTotalRespuesta / n;

        System.out.println("Proceso\tLlegada\tRáfaga\tInicio\tFinalización\tEspera\tRetorno
\tRespuesta");
        for (int i = 0; i < n; i++) {
            System.out.println((i + 1) + "\t" + this.info_tllegada[i] + "\t" +
this.info_tejecucion[i] + "\t" + tiempoInicio[i]
                    + "\t" + tiempoFinalizacion[i] + "\t\t" + tiempoEspera[i] + "\t" +
tiempoRetorno[i] + "\t\t" + tiempoRespuesta[i]);
        }

        System.out.println("\nTiempo promedio de espera: " + tiempoPromedioEspera);
        System.out.println("Tiempo promedio de respuesta: " + tiempoPromedioRespuesta);

        System.out.println("\nDiagrama de Gantt:");
        System.out.print(" ");
        for (int i = 0; i < n; i++) {
            System.out.print("| P" + (orden[i] + 1) + "\t");
        }
        System.out.println("|");
        System.out.print(tiempoInicio[0]);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < this.info_tejecucion[i]; j++) {
                System.out.print(" ");
            }
            System.out.print(tiempoFinalizacion[i]);
        }
```

```
        System.out.println();
    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////
INGRESA UN OPCION: 2
Ingresa el Numero de DATOS: 4
Ingrese el valor para el Proceso 0: 10
Ingrese el valor para el Proceso 1: 1
Ingrese el valor para el Proceso 2: 2
Ingrese el valor para el Proceso 3: 1
2.      SJF (Shortest Job First) - El trabajo m□s corto primero.

Proceso         Llegada         R□faga          Inicio          FinalizaciOn\ttEspera\ttRetorno              Respuesta
1               2               1               2               3               0               1               0
2               4               1               3               4               -1              0               -1
3               3               2               4               6               1               3               1
4               1               10              6               16              5               15              5

Tiempo promedio de espera: 1.25
Tiempo promedio de respuesta: 1.25

Diagrama de Gantt:
 | P2   | P4    | P3    | P1    |
2 3 4  6          16
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://www.geeksforgeeks.org/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/

https://www.thejavaprogrammer.com/java-program-shortest-job-first-sjf-scheduling/

https://www.guru99.com/shortest-job-first-sjf-scheduling.html

1.3. **RR (Round Robin) - Ronda de turnos.**

Definición

Este algoritmo de planificación de procesos asigna la CPU a los procesos en tiempo limitado de ejecución denominado "quantum" y si no termina su ejecución durante ese tiempo, se pasa al siguiente proceso en la cola.

Algoritmo en lenguaje java

```
    /*3.    RR (Round Robin) - Ronda de turnos.*/
    public void algoritmo_rr() {
        //int[] this.info_tllegada = tllegada; // Tiempos de llegada de los procesos
        //int[] this.info_tejecucion = tejecucion; // Tiempos de ráfaga de los procesos
        //int this.info_quantum = q; // Quantum del algoritmo Round Robin

        int n = this.info_tllegada.length;
        java.util.List<Proceso> colaProcesos = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            Proceso proceso = new Proceso(i + 1, this.info_tllegada[i],
this.info_tejecucion[i]);
            colaProcesos.add(proceso);
        }

        int tiempoTotal = 0;
        int procesosCompletados = 0;
```

```
        while (procesosCompletados < n) {
            for (int i = 0; i < colaProcesos.size(); i++) {
                Proceso proceso = colaProcesos.get(i);

                if (proceso.getTiempoRestante() > 0) {
                    int tiempoEjecucion = Math.min(this.info_quantum,
proceso.getTiempoRestante());
                    proceso.setTiempoRestante(proceso.getTiempoRestante() -
tiempoEjecucion);
                    tiempoTotal += tiempoEjecucion;

                    if (proceso.getTiempoRestante() == 0) {
                        proceso.setTiempoFinalizacion(tiempoTotal);
                        proceso.setTiempoRetorno(proceso.getTiempoFinalizacion() -
proceso.getTiempoLlegada());
                        proceso.setTiempoEspera(proceso.getTiempoRetorno() -
proceso.getTiempoRafaga());
                        proceso.setTiempoRespuesta(proceso.getTiempoEspera() +
proceso.getTiempoRafaga());
                        procesosCompletados++;
                    }
                }
            }
        }

        // Calcular los tiempos promedio
        double tiempoEsperaPromedio = 0;
        double tiempoRespuestaPromedio = 0;

        for (Proceso proceso : colaProcesos) {
            tiempoEsperaPromedio += proceso.getTiempoEspera();
            tiempoRespuestaPromedio += proceso.getTiempoRespuesta();
        }

        tiempoEsperaPromedio /= n;
        tiempoRespuestaPromedio /= n;

        // Mostrar los resultados
        System.out.println("Proceso\t\tLlegada\t\tRáfaga\t\tEspera\t\tRetorno\t\tFinalizaci
ón\tRespuesta");
        for (Proceso proceso : colaProcesos) {
            System.out.printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", proceso.getId(),
proceso.getTiempoLlegada(),
                    proceso.getTiempoRafaga(), proceso.getTiempoEspera(),
proceso.getTiempoRetorno(),
                    proceso.getTiempoFinalizacion(), proceso.getTiempoRespuesta());
        }

        System.out.println("\nDiagrama de Gantt:");
        for (Proceso proceso : colaProcesos) {
            System.out.print("| P" + proceso.getId() + " ");
        }
        System.out.println("|");
        for (Proceso proceso : colaProcesos) {
            System.out.print(proceso.getTiempoInicio() + "\t");
        }
```

```
        System.out.println(colaProcesos.get(colaProcesos.size() -
1).getTiempoFinalizacion());

        System.out.println("\nTiempo promedio de espera: " + tiempoEsperaPromedio);
        System.out.println("Tiempo promedio de respuesta: " + tiempoRespuestaPromedio);
    }
    static class Proceso {
        private int id;
        private int tiempoLlegada;
        private int tiempoRafaga;
        private int tiempoEspera;
        private int tiempoRetorno;
        private int tiempoFinalizacion;
        private int tiempoRespuesta;
        private int tiempoInicio;
        private int tiempoRestante;

        public Proceso(int id, int tiempoLlegada, int tiempoRafaga) {
            this.id = id;
            this.tiempoLlegada = tiempoLlegada;
            this.tiempoRafaga = tiempoRafaga;
            this.tiempoEspera = 0;
            this.tiempoRetorno = 0;
            this.tiempoFinalizacion = 0;
            this.tiempoRespuesta = 0;
            this.tiempoInicio = 0;
            this.tiempoRestante = tiempoRafaga;
        }

        public int getId() {
            return id;
        }

        public int getTiempoLlegada() {
            return tiempoLlegada;
        }

        public int getTiempoRafaga() {
            return tiempoRafaga;
        }

        public int getTiempoEspera() {
            return tiempoEspera;
        }

        public void setTiempoEspera(int tiempoEspera) {
            this.tiempoEspera = tiempoEspera;
        }

        public int getTiempoRetorno() {
            return tiempoRetorno;
        }

        public void setTiempoRetorno(int tiempoRetorno) {
            this.tiempoRetorno = tiempoRetorno;
        }

        public int getTiempoFinalizacion() {
```

```
            return tiempoFinalizacion;
        }

        public void setTiempoFinalizacion(int tiempoFinalizacion) {
            this.tiempoFinalizacion = tiempoFinalizacion;
        }

        public int getTiempoRespuesta() {
            return tiempoRespuesta;
        }

        public void setTiempoRespuesta(int tiempoRespuesta) {
            this.tiempoRespuesta = tiempoRespuesta;
        }

        public int getTiempoInicio() {
            return tiempoInicio;
        }

        public void setTiempoInicio(int tiempoInicio) {
            this.tiempoInicio = tiempoInicio;
        }

        public int getTiempoRestante() {
            return tiempoRestante;
        }

        public void setTiempoRestante(int tiempoRestante) {
            this.tiempoRestante = tiempoRestante;
        }
    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////
3
3.      RR (Round Robin) - Ronda de turnos.

Proceso         Llegada         R□faga          Espera          Retorno         FinalizaciOn    Respuesta
P1              1               10              3               13              14              13
P2              2               1               0               1               3               1
P3              3               2               0               2               5               2
P4              4               1               1               2               6               2

Diagrama de Gantt:
| P1 | P2 | P3 | P4 |
0       0       0       0       6

Tiempo promedio de espera: 1.0
Tiempo promedio de respuesta: 4.5
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://www.javatpoint.com/round-robin-scheduling-program-in-java

https://mystaridea.com/es/algoritmo-de-programaci%C3%B3n-round-robin-con-ejemplo/

1.4. **SRTF (Shortest Remaining Time First) - El tiempo restante más corto primero.**

Definición

Este algoritmo de planificación de procesos asigna la CPU al proceso con el tiempo de ejecución restante más corto en cada momento.

Algoritmo en lenguaje java

```
    /*4.    SRTF (Shortest Remaining Time First) - El tiempo restante más corto primero.*/
    public void algoritmo_srtf(){
        //int[] this.info_tllegada = tllegada; // Tiempos de llegada de los procesos
        //int[] this.info_tejecucion = tejecucion; // Tiempos de ráfaga de los procesos

        int n = this.info_tllegada.length;
        int[] tiemposEspera = new int[n];
        int[] tiemposRetorno = new int[n];
        int[] tiemposFinalizacion = new int[n];
        int[] tiemposInicio = new int[n];
        int[] tiemposRespuesta = new int[n];

        // Copiar los tiempos de ráfaga a un nuevo array para no modificar el original
        int[] tiempos_Rafaga_Copia = Arrays.copyOf(this.info_tejecucion, n);

        boolean[] procesosCompletados = new boolean[n];
        int tiempoTotal = 0;
        int procesosCompletadosCount = 0;

        while (procesosCompletadosCount < n) {
            int procesoActual = -1;
            int rafagaMinima = Integer.MAX_VALUE;

            // Buscar el proceso con la ráfaga más corta que no haya sido completado
            for (int i = 0; i < n; i++) {
                if (!procesosCompletados[i] && this.info_tllegada[i] <= tiempoTotal &&
tiempos_Rafaga_Copia[i] < rafagaMinima) {
                    rafagaMinima = tiempos_Rafaga_Copia[i];
                    procesoActual = i;
                }
            }

            // Si no se encontró ningún proceso válido, incrementar el tiempo total
            if (procesoActual == -1) {
                tiempoTotal++;
                continue;
            }

            // Calcular el tiempo de inicio del proceso
            if (tiemposInicio[procesoActual] == 0) {
                tiemposInicio[procesoActual] = tiempoTotal;
            }

            // Ejecutar el proceso durante un tiempo unitario
            tiempos_Rafaga_Copia[procesoActual]--;
            tiempoTotal++;

            // Verificar si el proceso ha finalizado
```

```
            if (tiempos_Rafaga_Copia[procesoActual] == 0) {
                tiemposFinalizacion[procesoActual] = tiempoTotal;
                tiemposRetorno[procesoActual] = tiemposFinalizacion[procesoActual] -
this.info_tllegada[procesoActual];
                tiemposEspera[procesoActual] = tiemposRetorno[procesoActual] -
this.info_tejecucion[procesoActual];
                tiemposRespuesta[procesoActual] = tiemposEspera[procesoActual] +
tiemposInicio[procesoActual];
                procesosCompletados[procesoActual] = true;
                procesosCompletadosCount++;
            }
        }

        // Calcular los tiempos promedio
        double tiempoEsperaPromedio = Arrays.stream(tiemposEspera).average().orElse(0);
        double tiempoRespuestaPromedio =
Arrays.stream(tiemposRespuesta).average().orElse(0);

        // Mostrar los resultados
        System.out.println("Proceso\tLlegada\tRáfaga\tEspera\tRetorno\tFinalización\tRespue
sta");
        for (int i = 0; i < n; i++) {
            System.out.printf("P%d\t%d\t%d\t%d\t%d\t%d\t\t%d\n", i + 1,
this.info_tllegada[i], this.info_tejecucion[i],
                    tiemposEspera[i], tiemposRetorno[i], tiemposFinalizacion[i],
tiemposRespuesta[i]);
        }

        System.out.println("\nDiagrama de Gantt:");
        for (int i = 0; i < n; i++) {
            System.out.print("| P" + (i + 1) + " ");
        }
        System.out.println("|");
        for (int i = 0; i < n; i++) {
            System.out.print(tiemposInicio[i] + "\t");
        }
        System.out.println(tiemposFinalizacion[n - 1]);

        System.out.println("\nTiempo promedio de espera: " + tiempoEsperaPromedio);
        System.out.println("Tiempo promedio de respuesta: " + tiempoRespuestaPromedio);

    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////
4
4.  SRTF (Shortest Remaining Time First) - El tiempo restante m□s corto primero.

Proceso Llegada R□faga  Espera  Retorno Finalizaci□n   Respuesta
P1      1       10      4       14      15              5
P2      2       1       0       1       3               2
P3      3       2       0       2       5               3
P4      4       1       1       2       6               6

Diagrama de Gantt:
| P1 | P2 | P3 | P4 |
1       2       3       5       6

Tiempo promedio de espera: 1.25
Tiempo promedio de respuesta: 4.0
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://www.geeksforgeeks.org/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/

https://www.thejavaprogrammer.com/java-program-shortest-job-first-sjf-scheduling/

https://gist.github.com/sschakraborty/cc63fba0ad19e0fc7a6bf43fdd9d01c2

https://stackoverflow.com/questions/12679943/shortest-remaining-time-first-java-multithreading

1.5. **Priority Scheduling - Planificación por prioridad(fija-dinamica)**

Definición

Este algoritmo de planificación de procesos asigna la CPU según la prioridad asignada a cada proceso.

Algoritmo en lenguaje java

```java
    /*5.    Priority Scheduling - Planificación por prioridad(fija-dinamica)*/
    public void algoritmo_Priority_Schedulingint() {
        int n = this.info_tllegada.length;
        int[] tiemposEspera = new int[n];
        int[] tiemposRetorno = new int[n];
        int[] tiemposFinalizacion = new int[n];
        int[] tiemposInicio = new int[n];
        int[] tiemposRespuesta = new int[n];

        // Copiar los tiempos de ráfaga a un nuevo array para no modificar el original
        int[] tiempos_Rafaga_Copia = Arrays.copyOf(this.info_tejecucion, n);

        boolean[] procesosCompletados = new boolean[n];
        int tiempoTotal = 0;
        int procesosCompletadosCount = 0;

        while (procesosCompletadosCount < n) {
            int procesoPrioridadMinima = -1;
            int prioridadMinima = Integer.MAX_VALUE;

            // Buscar el proceso con la prioridad más baja que no haya sido completado
            for (int i = 0; i < n; i++) {
                if (!procesosCompletados[i] && this.info_tllegada[i] <= tiempoTotal &&
this.info_prioridad[i] < prioridadMinima) {
```

```
                    prioridadMinima = this.info_prioridad[i];
                    procesoPrioridadMinima = i;
                }
            }

            // Si no se encontró ningún proceso válido, incrementar el tiempo total
            if (procesoPrioridadMinima == -1) {
                tiempoTotal++;
                continue;
            }

            // Actualizar el tiempo de inicio si el proceso es diferente al anterior
            if (tiemposInicio[procesoPrioridadMinima] == 0) {
                tiemposInicio[procesoPrioridadMinima] = tiempoTotal;
            }

            // Ejecutar el proceso durante un ciclo
            tiempos_Rafaga_Copia[procesoPrioridadMinima]--;
            tiempoTotal++;

            // Verificar si el proceso ha finalizado
            if (tiempos_Rafaga_Copia[procesoPrioridadMinima] == 0) {
                tiemposFinalizacion[procesoPrioridadMinima] = tiempoTotal;
                tiemposRetorno[procesoPrioridadMinima] =
tiemposFinalizacion[procesoPrioridadMinima] - this.info_tllegada[procesoPrioridadMinima];
                tiemposEspera[procesoPrioridadMinima] =
tiemposRetorno[procesoPrioridadMinima] - this.info_tejecucion[procesoPrioridadMinima];
                tiemposRespuesta[procesoPrioridadMinima] =
tiemposEspera[procesoPrioridadMinima] + tiemposInicio[procesoPrioridadMinima];
                procesosCompletados[procesoPrioridadMinima] = true;
                procesosCompletadosCount++;
            }
        }

        // Calcular los tiempos promedio
        double tiempoEsperaPromedio = calculateAverage(tiemposEspera);
        double tiempoRespuestaPromedio = calculateAverage(tiemposRespuesta);

        // Mostrar los resultados
        System.out.println("Proceso\t\tLlegada\t\tRáfaga\t\tPrioridad\tEspera\t\tRetorno\t\
tFinalización\tRespuesta");
        for (int i = 0; i < n; i++) {
            System.out.printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1,
this.info_tllegada[i], this.info_tejecucion[i],
                    this.info_prioridad[i], tiemposEspera[i], tiemposRetorno[i],
tiemposFinalizacion[i], tiemposRespuesta[i]);
        }

        System.out.println("\nDiagrama de Gantt:");
        for (int i = 0; i < n; i++) {
            System.out.print("| P" + (i + 1) + " ");
        }
        System.out.println("|");
        for (int i = 0; i < n; i++) {
            System.out.print(tiemposInicio[i] + "\t");
        }
        System.out.println(tiemposFinalizacion[n - 1]);
```

```
        System.out.println("\nTiempo promedio de espera: " + tiempoEsperaPromedio);
        System.out.println("Tiempo promedio de respuesta: " + tiempoRespuestaPromedio);
    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////
5
5.      Priority Scheduling - Planificaci□n por prioridad(fija-dinamica)

Proceso         Llegada         R□faga          Prioridad       Espera          Retorno         Finalizaci□n    Respuesta
P1              1               10              3               1               11              12              2
P2              2               1               1               0               1               3               2
P3              3               2               3               9               11              14              21
P4              4               1               4               10              11              15              24

Diagrama de Gantt:
| P1 | P2 | P3 | P4 |
1       2       12      14      15

Tiempo promedio de espera: 5.0
Tiempo promedio de respuesta: 12.25
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://programmerclick.com/article/6982137934/

https://programmerclick.com/article/97541980291/

https://www.delftstack.com/es/howto/java/java-priority-queue/

1.6. Planificación prioritaria con Round-Robin

Definición

Este algoritmo es una combinación de la planificación por prioridad y el algoritmo Round Robin. Los procesos se ejecutan de acuerdo a su prioridad

Algoritmo en lenguaje java

```
    /*6.    Planificación prioritaria con Round-Robin*/
    public void algoritmo_PrioridadRoundRobin(){
        int n = this.info_tllegada.length;
        int[] tiemposEspera = new int[n];
        int[] tiemposRetorno = new int[n];
        int[] tiemposFinalizacion = new int[n];
        int[] tiemposInicio = new int[n];
        int[] tiemposRespuesta = new int[n];

        // Copiar los tiempos de ráfaga y this.info_prioridad a nuevos arrays para no
modificar los originales
        int[] tiempos_Rafaga_Copia = Arrays.copyOf(this.info_tejecucion, n);
        int[] prioridad_Copia = Arrays.copyOf(this.info_prioridad, n);

        int tiempoTotal = 0;
        int procesosCompletadosCount = 0;
        boolean[] procesosCompletados = new boolean[n];

        Queue<Integer> colaProcesos = new LinkedList<>();
        for (int i = 0; i < n; i++) {
            colaProcesos.add(i);
```

```
        }

        while (procesosCompletadosCount < n) {
            int procesoActual = colaProcesos.poll();

            // Ejecutar el proceso durante el Quantum o hasta que se complete su ráfaga
            int tiempoEjecucion = Math.min(this.info_quantum,
tiempos_Rafaga_Copia[procesoActual]);
            tiempos_Rafaga_Copia[procesoActual] -= tiempoEjecucion;
            tiempoTotal += tiempoEjecucion;

            // Actualizar el tiempo de inicio si es el primer ciclo del proceso
            if (tiemposInicio[procesoActual] == 0) {
                tiemposInicio[procesoActual] = tiempoTotal - tiempoEjecucion;
            }

            // Verificar si el proceso ha finalizado
            if (tiempos_Rafaga_Copia[procesoActual] == 0) {
                tiemposFinalizacion[procesoActual] = tiempoTotal;
                tiemposRetorno[procesoActual] = tiemposFinalizacion[procesoActual] -
this.info_tllegada[procesoActual];
                tiemposEspera[procesoActual] = tiemposRetorno[procesoActual] -
this.info_tejecucion[procesoActual];
                tiemposRespuesta[procesoActual] = tiemposEspera[procesoActual] +
tiemposInicio[procesoActual];
                procesosCompletados[procesoActual] = true;
                procesosCompletadosCount++;
            } else {
                // Si el proceso no ha finalizado, se vuelve a agregar a la cola de
procesos
                colaProcesos.add(procesoActual);
            }

            // Actualizar las this.info_prioridad de los procesos que llegan mientras se
ejecuta el proceso actual
            for (int i = 0; i < n; i++) {
                if (!procesosCompletados[i] && this.info_tllegada[i] <= tiempoTotal && i !=
procesoActual && prioridad_Copia[i] < prioridad_Copia[procesoActual]) {
                    prioridad_Copia[i]++;
                }
            }
        }

        // Calcular los tiempos promedio
        double tiempoEsperaPromedio = calculateAverage(tiemposEspera);
        double tiempoRespuestaPromedio = calculateAverage(tiemposRespuesta);

        // Mostrar los resultados
        System.out.println("Proceso\t\tLlegada\t\tRáfaga\t\tPrioridad\tEspera\t\tRetorno\t\
tFinalización\tRespuesta");
        for (int i = 0; i < n; i++) {
            System.out.printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1,
this.info_tllegada[i], this.info_tejecucion[i],
                    this.info_prioridad[i], tiemposEspera[i], tiemposRetorno[i],
tiemposFinalizacion[i], tiemposRespuesta[i]);
        }

        System.out.println("\nDiagrama de Gantt:");
```

```
        for (int i = 0; i < n; i++) {
            System.out.print("| P" + (i + 1) + " ");
        }
        System.out.println("|");
        for (int i = 0; i < n; i++) {
            System.out.print(tiemposInicio[i] + "\t");
        }
        System.out.println(tiemposFinalizacion[n - 1]);

        System.out.println("\nTiempo promedio de espera: " + tiempoEsperaPromedio);
        System.out.println("Tiempo promedio de respuesta: " + tiempoRespuestaPromedio);
    }
```

Resultados

```
////////////////////////////////////////////////////////////////////////////////////////////////////
6
6.      PlanificaciOn prioritaria con Round-Robin

Proceso         Llegada         RBfaga          Prioridad       Espera          Retorno         FinalizaciOn    Respuesta
P1              1               10              3               3               13              14              9
P2              2               1               1               0               1               3               2
P3              3               2               3               0               2               5               3
P4              4               1               4               1               2               6               6

Diagrama de Gantt:
| P1 | P2 | P3 | P4 |
6       2       3       5       6

Tiempo promedio de espera: 1.0
Tiempo promedio de respuesta: 5.0
////////////////////////////////////////////////////////////////////////////////////////////////////
```

Referencia

https://blogcitochia.wordpress.com/2017/04/05/algoritmo-de-planificacion-round-robin/

https://prezi.com/qu0q30mhph1-/algoritmo-de-planificacion-por-prioridad/

https://claudiaesteffani.wordpress.com/2017/04/05/algoritmo-de-planificacion-round-robin/

2. PARTE 2

Hacer un programa para resolver la atención en una institución financiera usando los conceptos y técnicas estudiadas en el curso, para esto debe

1) **Simular la llegada de clientes usando variable aleatorias con distribución uniforme**

   Los clientes que pueden llegar son:

   a. Clientes con tarjeta
      i. Clientes con cuentas comunes
      ii. Clientes personas naturales VIP
      iii. Clientes personas jurídicas comunes
      iv. Clientes personas jurídicas VIP
   b. Clientes sin tarjeta
   c. Clientes preferenciales
      i. Clientes mayores de 60 años
      ii. Clientes con deficiencia física
      iii. Clientes con necesidades especiales

2) **Simular el funcionamiento de ventanillas usando variables aleatorias con distribución uniforme**
   a. La institución posee N ventanillas, definidas al inicio del programa
   b. El tiempo que cada persona se demora en ventanilla es aleatorio con distribución uniforme
   c. Una ventanilla eventualmente puede dejar de atender, el número de veces y el tiempo que deja de atender también es aleatório
3) **Resolver el problema de asignación de ventanillas a cada cliente que llega a la institución**

Herramientas:

- Utilicen el lenguaje de programación de su elección

Entregable:

- Presentación de los programas en clase por cada grupo, video explicativo(new)
- Informe en PDF que contiene:
  - Explicación del código (estructuras de datos utilizados, algoritmo de planificación, otros)
  - Pruebas de funcionamiento de los programas
  - URL de código fuente en un repositorio de código
  - URL del video explicativo

2.1. **CÓDIGO FUENTE:JAVA**

```
package sou1.so_u1_r2_atencionclientes;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

class Cliente {
    private String nombre;
    private int tipoCliente;
    private int tiempoAtencion;

    public Cliente(String nombre, int tipoCliente) {
        this.nombre = nombre;
        this.tipoCliente = tipoCliente;
        this.tiempoAtencion = generarTiempoAtencion();
    }

    public String getNombre() {
        return nombre;
    }

    public int getTipoCliente() {
        return tipoCliente;
    }

    public int getTiempoAtencion() {
        return tiempoAtencion;
    }

    private int generarTiempoAtencion() {
        Random random = new Random();
        return random.nextInt(10) + 1; // Genera un tiempo de atención aleatorio entre 1 y 10 segundos
    }
}

class Ventanilla {
    private String nombre;
    private Cliente clienteActual;
    private int tiempoRestante;

    public Ventanilla(String nombre) {
        this.nombre = nombre;
        this.clienteActual = null;
        this.tiempoRestante = 0;
    }

    public String getNombre() {
        return nombre;
    }

    public Cliente getClienteActual() {
        return clienteActual;
    }

    public int getTiempoRestante() {
        return tiempoRestante;
    }

    public void asignarCliente(Cliente cliente) {
        this.clienteActual = cliente;
        this.tiempoRestante = cliente.getTiempoAtencion();
        System.out.println("[" + this.nombre + "] Atendiendo al cliente: " + cliente.getNombre()+"
Prioridad: "+cliente.getTipoCliente());
    }

    public void atenderCliente() {
        if (clienteActual != null) {
            tiempoRestante--;

            if (tiempoRestante <= 0) {
                System.out.println("[" + this.nombre + "] Cliente " + clienteActual.getNombre() + "
atendido."+" Prioridad: "+clienteActual.getTipoCliente());
                clienteActual = null;
```

```
                tiempoRestante = 0;
            }
        }
    }
}

public class AtencionClientes {
    public void ejecutar() {
        List<Cliente> clientes = new ArrayList<>();
        clientes.add(new Cliente("Cliente 1", 1));  // Cliente preferencial mayor de 60 años
        clientes.add(new Cliente("Cliente 2", 3));  // Cliente preferencial con necesidades especiales
        clientes.add(new Cliente("Cliente 3", 4));  // Cliente sin tarjeta
        clientes.add(new Cliente("Cliente 4", 2));  // Cliente preferencial con deficiencia física
        clientes.add(new Cliente("Cliente 5", 5));  // Cliente con tarjeta, cuenta común
        clientes.add(new Cliente("Cliente 6", 7));  // Cliente con tarjeta, persona jurídica común
        clientes.add(new Cliente("Cliente 7", 6));  // Cliente con tarjeta, persona natural VIP
        clientes.add(new Cliente("Cliente 8", 8));  // Cliente con tarjeta, persona jurídica VIP
        clientes.add(new Cliente("Cliente 9", 5));  // Cliente con tarjeta, persona jurídica VIP

        List<Ventanilla> ventanillas = new ArrayList<>();
        ventanillas.add(new Ventanilla("Ventanilla 1"));
        ventanillas.add(new Ventanilla("Ventanilla 2"));
        ventanillas.add(new Ventanilla("Ventanilla 3"));

        /*Priority Scheduling - Planificación por prioridad*/
        while (!clientes.isEmpty()) {
            int maxPriority = Integer.MIN_VALUE;
            Cliente nextCliente = null;

            for (Cliente cliente : clientes) {
                if (cliente.getTipoCliente() > maxPriority) {
                    maxPriority = cliente.getTipoCliente();
                    nextCliente = cliente;
                }
            }

            if (nextCliente != null) {
                Ventanilla ventanillaDisponible = null;

                for (Ventanilla ventanilla : ventanillas) {
                    if (ventanilla.getClienteActual() == null) {
                        ventanillaDisponible = ventanilla;
                        break;
                    }
                }

                if (ventanillaDisponible != null) {
                    ventanillaDisponible.asignarCliente(nextCliente);
                    clientes.remove(nextCliente);
                }
            }

            System.out.println("------------");
            System.out.println("Estado actual:");
            for (Ventanilla ventanilla : ventanillas) {
                System.out.println("[" + ventanilla.getNombre()+"] Cliente actual: " +
(ventanilla.getClienteActual() != null ? ventanilla.getClienteActual().getNombre() : "Ninguno"));
            }
            System.out.println("------------");

            for (Ventanilla ventanilla : ventanillas) {
                ventanilla.atenderCliente();
            }

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

3. LINK DRIVE (VIDEO)

https://drive.google.com/drive/folders/1keV1CLtmdWgyNJjmftDyPf0DbCHCKn55

4. LINK GITHUB

https://github.com/mrrows45/epis_so_u1_tf