



Benemérita Universidad Autónoma de Puebla

- Reporte del proyecto: administrador de tareas -

Christian Amauri Amador Ortega - 201927821

Docente: Josue Pérez Lucero

Contenido

1. Introducción	2
1.1 Planteamiento.....	2
1.2 Conocimientos requeridos	2
2. Desarrollo	3
2.1 Lenguaje de programación	3
2.2 Parámetros técnicos	3
2.3 Código	4
3. Ejecuciones (evidencias de funcionamiento)	8
3.1 definición de los casos	8
3.2 Ejecuciones para los casos de: Monotarea	9
3.3 Ejecuciones para los casos de: Multitarea	11
4. Conclusiones	14
4.1 Qué se aprendió?	14
4.2 Comentarios finales	14

1. Introducción

1.1 Planteamiento.

El presente reporte explica el desarrollo y funcionamiento del proyecto de **Administrador de tareas** el cual tiene como objetivo crear una tabla de resultados que represente una lista de procesos ordenada con una configuración específica, dado un conjunto de datos guardados en un archivo.

Dicha configuración será obtenida a partir de diversas operaciones con valores numéricos que representen tiempo y prioridad para los procesos, y también con una variedad de algoritmos para escoger.

Finalmente, el proyecto debe obtener y mostrar un tiempo total y un tiempo promedio de ejecución, dada la configuración escogida.

1.2 Conocimientos requeridos.

Ahora, para entender cómo funciona este código, se necesitan conocer algunos conceptos que todo alumno de una carrera del área de computación hasta quinto semestre ha trabajado (en algunos más que en otros), estos conceptos son:

- Métodos de ordenamiento
- Paso de parámetros
- Programación orientada a objetos (en este caso en el lenguaje C++)
- Uso de listas ligadas y sus posibles operaciones
- Lectura de archivos (en este caso, de texto)

Quien no conozca de estos temas, no terminará de entender el funcionamiento del proyecto, pues para agilizar la escritura y lectura de este documento, se omitieron los detalles de algunas partes que se explican por sí solas si se conoce lo suficiente.

2. Desarrollo

2.1 Lenguaje de programación.

Este proyecto en particular fue desarrollado en el lenguaje de programación C++. Escogimos este lenguaje por varias razones. Para empezar, este es el lenguaje básico de la programación y es el lenguaje en el que todos los miembros del equipo tenemos más práctica, así que trabajar en este lenguaje haría todo el desarrollo del proyecto menos laborioso y mucho más ágil. Es un lenguaje muy eficaz y versátil en términos de manejo y administración de memoria, además de su gran velocidad y accesibilidad en comparación con otros lenguajes como por ejemplo Java, que es un lenguaje muy potente y equipado, pero al serlo, se vuelve más pesado, lento y exige más requerimientos de entrada, lo que podría traducirse en impedimentos o directamente obstáculos. Además de que desde un inicio no teníamos la intención de implementar una interfaz gráfica, pensamos que bastaría con un programa ejecutado en líneas de comandos fáciles de entender. Para así, enfocarnos en las operaciones que el proyecto debía realizar (lectura de archivos y todas las configuraciones de ordenamiento posibles).

Finalmente, el paradigma en el que nuestro proyecto fue desarrollado es programación orientada a objetos, porque acordamos que trabajar bajo este paradigma, sería la mejor forma de administrar los datos y las operaciones que se requieren.

2.2 Parámetros técnicos.

- Netamente, el código fuente tiene 2394 líneas (aunque un porcentaje de eso se va en comentarios y espacios vacíos).
- Este proyecto fue desarrollado principalmente en un entorno Linux, pero funciona también a la perfección en Windows (no fue probado en ningún otro entorno).
- La versión final del código pesa 88 KB.
- El ejecutable proveniente de dicho código pesa 1903 KB.
- El tipo de archivo que el ejecutable lee para trabajar los datos es .txt.
- El archivo por leer debe estar en la carpeta en la que se encuentra el ejecutable del programa.

Ahora, los requisitos mínimos que el lenguaje C++ requiere son:

- 8 MB de RAM.
- Procesador compatible Intel a 100 MHz.
- 30 MB de espacio libre en el disco duro.

(Normalmente cualquier equipo de cómputo actual puede cumplir con ellos)

2.3 Código:

Para empezar, las librerías que el código utiliza son:

```
<iostream> *Librería de entrada y salida básica de C++  
<fstream> *Librería utilizada para la lectura del archivo .txt
```

Ahora, como ya se mencionó, la programación del proyecto está orientada a objetos, en este caso, la única clase que tiene el código es la clase:

```
class claseNodo {...}
```

La cual tiene como único atributo un struct privado que da forma a los nodos de las listas ligadas usadas en la ejecución (una principal y dos auxiliares), este atributo tiene los elementos mostrados a continuación:

```
private:  
struct nodo { * cada nodo corresponde a un proceso  
int pid; *Pid del proceso  
int Tllegada; *Tiempo de llegada del proceso  
int Texe; *Tiempo de ejecución del proceso  
int prioridad; *Prioridad que tiene el proceso  
int Tespera; *Tiempo de espera del proceso  
bool fin; *Bandera que indica si el proceso ha llegado al su fin  
int Tfin; *Tiempo de finalización del proceso  
nodo *sig; *apuntador al proceso siguiente (listas ligadas)  
};
```

Finalmente, la clase tiene un total de 32 métodos, unos más sencillos que otros. como ya se mencionó anteriormente, se omitirán algunos detalles en función de los conocimientos requeridos. (aquí viene la parte complicada).

```
nodo *inicializarNodo();
void insertarListaFinal(nodo *&);
void autoListaFinal(nodo *&, int, int, int, int, bool, int);
void ordenarLista(nodo *&, int);
void algoritmo2(nodo *&, nodo *&, nodo *&, int);
void actualizarProceso(nodo *&, nodo *&, int, int);
void repetirCola(nodo *&, nodo *&, int, int);
int checarLista(nodo *);
int sumaEspera(nodo *);
int procesoBusqueda1(nodo *, int, int);
int ingresarMatriz2(int [], int);
int ingresarMatriz(int [], int, int);
int ingresarMatrizMulti2(int [][][2], int);
int menu3();
void menu();
void Mensaje1();

void insertarListaInicio(nodo *&);
void autoListaInicio(nodo *&, int, int, int, int, bool, int);
void mostrarLista(nodo *);
void algoritmo1(nodo *&, nodo *&, nodo *&, int);
void eliminar(nodo *&, int);
void actualizarProcesoMulti(nodo *&, int, int);
void actualizarCola(nodo *&, nodo *&, int);
int checarRepeticiones(int [], int);
int numTexeBusqueda(nodo *, int);
int procesoBusqueda2(nodo *, int, int);
void ingresarMatriz3(int [], int);
int ingresarMatrizMulti(int [][][2], int, int);
int menu4();
int menu2();
void Espacio();
bool listaVacia(nodo *);
```

En el código fuente, estos métodos se encuentran en otro orden, aquí los presentaremos en el orden que nos permita explicar primero los más sencillos y al último los más complicados.

Métodos:

Muy sencillos...

nodo *inicializarNodo(); Método constructor, devuelve el objeto de tipo “ claseNodo” (la lista).
void Mensaje1(); Simplemente escribe en pantalla el mensaje: “sistema a planificar\n”.
void Espacio(); simplemente escribe un espacio “=====\\n”.
bool listaVacia(nodo *); simplemente con un <P = lista; if(p == NULL)> verifica si la lista existe o no
void mostrarLista(nodo *); este método crea un nodo auxiliar p, lo iguala a lista y mediante un while(p !=NULL), imprime los datos de la lista original.

Sencillos...

void menu(); este método es muy muy importante, es el primer método que debe ser llamado una vez creado el objeto, en primer lugar crea las 3 listas, la principal y las dos auxiliares, luego lee los datos del archivo .txt y los ingresa a la lista principal, después de eso, manda a llamar al menu2(), quien es el encargado de obtener la respuesta a la primer pregunta <monotarea o multitarea?>. Luego de eso, mediante un switch define si el siguiente paso es llamar al algoritmo1() o al algoritmo2() (en cualquiera de los dos casos se manda a llamar el menu3(), quien define el algoritmo de tiempo y prioridad que se va a utilizar en el algoritmo1 o el algoritmo2) en ese mismo switch se encuentra también el método para mostrar la lista y la opción para salir.

int menu2(); Define si la configuración será <monotarea o multitarea>.

int menu3(); Define si la configuración será <monoproceso o multiproceso>.

int menu4(); Define la configuración respecto al tiempo, la prioridad, y si es <ascendente o descendente>.

Normales...

*void insertarListaInicio(nodo *&); inserta datos en la lista ligada, al inicio,(manualmente).*

*void insertarListaFinal(nodo *&); inserta datos en la lista ligada, al final, (manualmente).*

*void autoListaInicio(nodo *&, int, int, int, int, int, bool,int); inserta datos en la lista ligada, al inicio (sin la intervención del usuario) este método se usa a la hora de leer el archivo (solamente la primera lectura).*

*void autoListaFinal(nodo *&, int, int, int, int, int, bool,int); inserta datos en la lista ligada, al final (sin la intervención del usuario) este método se usa a la hora de leer el archivo (todas las veces menos la primera lectura).*

*void ordenarLista(nodo *&, int); Este es el método que se mandará a llamar en el ultimo paso, es quien ordena los datos finalmente antes de ser entregados, mediante una adaptación del método de ordenamiento burbuja adecuado a listas ligadas.*

*void eliminar(nodo *&, int); Elimina un nodo, ya sea el primero, el ultimo, o uno en medio .*

....(operaciones clásicas de listas...)

Complicados...

*void algoritmo1(nodo *&, nodo *&, nodo *&, int); Este método se encarga de anidar Monoproceso-monotarea-multitarea.*

*void algoritmo2(nodo *&, nodo *&, nodo *&, int); Este método se encarga de anidar Multiproceso-monotarea-multitarea.*

*void actualizarProceso(nodo *&, nodo *&, int, int); Este método utiliza 2 listas (una principal y una auxiliar para actualizar sus tiempos de espera y su estado de fin).*

*void actualizarProcesoMulti(nodo *&, int, int); Este método utiliza una unica lista para actualizar sus tiempos de espera y su estado de fin).*

*void repetirCola(nodo *&, nodo *&, int, int); En caso de que Texe sea mayor a 1, se manda a llamar este método para encolar de nuevo los datos .*

*void actualizarCola(nodo *&, nodo *&, int); Este método copia los datos que está en la lista principal a una cola, (origen, destino);*

*int checarLista(nodo *); Este método sirve como contador de elementos en una lista, mediante un while y un contador, emula una iteración.*

int **checarRepeticiones**(*int* [], *int*); Mediante el uso de un for de 0 a 50, cuenta cuantas veces se ha ejecutado el Texe de un proceso

int **sumaEspera**(*nodo* *); Este método suma todos los Tiempos de espera de una lista mediante un while, para después poder sacar el promedio

int **numTexeBusqueda**(*nodo* *, *int*); mediante un for y un while, se verifica cuántos datos ya han sido ejecutados en una lista

int **procesoBusqueda1**(*nodo* *, *int*, *int*); regresa el pid de la siguiente manera T.exe(ligero), Prioridad(Alta), Pid(ASC/DESC)

int **procesoBusqueda2**(*nodo* *, *int*, *int*); regresa el pid de la siguiente manera Prioridad(Alta), T.exe(ligero), Pid(ASC/DESC)

int **ingresarMatriz2**(*int* [], *int*); ingresa los pids en una matriz bidimensional para mostrar al usuario y en unos casos regresa el tiempo finalizacion para que lo use el metodo de actualizarproceso()

void **ingresarMatriz3**(*int* [], *int*); ingresa los pids en una matriz bidimensional para mostrar al usuario y en unos casos regresa el tiempo finalizacion para que lo use el metodo de actualizarproceso()

int **ingresarMatriz**(*int* [], *int*, *int*); ingresa los pids en una matriz bidimensional para mostrar al usuario y en unos casos regresa el tiempo finalizacion para que lo use el metodo de actualizarproceso()

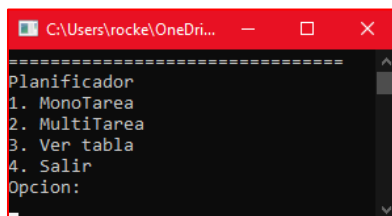
int **ingresarMatrizMulti**(*int* [][][2], *int*, *int*); ingresa los pids en una matriz bidimensional para mostrar al usuario y en unos casos regresa el tiempo finalizacion para que lo use el metodo de actualizarproceso()

int **ingresarMatrizMulti2**(*int* [][][2], *int*); ingresa los pids en una matriz bidimensional para mostrar al usuario y en unos casos regresa el tiempo finalizacion para que lo use el metodo de actualizarproceso()

3. Ejecuciones

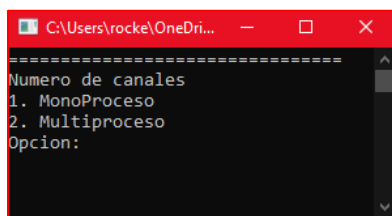
3.1 Definición de los casos:

Como ya se mencionó, existen muchas configuraciones de uso posibles, en particular, 16. Estas configuraciones vienen de el siguiente planteamiento: Como ya sabemos, primero se nos pide definir el tipo de planificador (**Monotarea o Multitarea**):



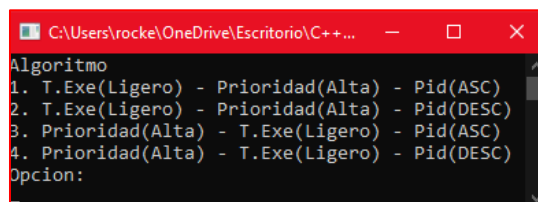
```
C:\Users\rocke\OneDri... - _ X
=====
Planificador
1. MonoTarea
2. MultiTarea
3. Ver tabla
4. Salir
Opcion:
```

Con lo cual, las posibilidades son **2**. En cualquiera de los casos, se nos pide ahora definir el número de canales (**Monoproceso o Multiproceso**):



```
C:\Users\rocke\OneDri... - _ X
=====
Numero de canales
1. MonoProceso
2. Multiproceso
Opcion:
```

Con lo cual ahora las posibilidades se convierten en $2^2 = 4$. Y finalmente se nos pide escoger entre los **4 algoritmos** siguientes:



```
C:\Users\rocke\OneDrive\Escritorio\C++... - _ X
Algoritmo
1. T.Exe(Ligero) - Prioridad(Alta) - Pid(ASC)
2. T.Exe(Ligero) - Prioridad(Alta) - Pid(DESC)
3. Prioridad(Alta) - T.Exe(Ligero) - Pid(ASC)
4. Prioridad(Alta) - T.Exe(Ligero) - Pid(DESC)
Opcion:
```

Tenemos entonces que el número de posibilidades total es de:

$$4 \times 4 = 4^2 = 16$$

En este documento hemos cubierto las 16 posibilidades, con un archivo de muestra que será incluido en el entregable final del proyecto, a continuación mostramos las ejecuciones divididas en: ([**Monotarea**] o [**Multitarea**]).

Con **T = Tiempo de ejecución (ligero)** y **P = prioridad (alta)**.

3.2 Ejecuciones para los casos de: Monotarea.

1-1-1 ([Monotarea] > [Monoproceso] > [T°P (ascendente)]):

```
C:\Users\rocke\OneDri... - □ ×
CPU1 [0] : 1
CPU1 [1] : 2
CPU1 [2] : 12
CPU1 [3] : 9
CPU1 [4] : 11
CPU1 [5] : 11
CPU1 [6] : 3
CPU1 [7] : 3
CPU1 [8] : 7
CPU1 [9] : 7
CPU1 [10] : 8
CPU1 [11] : 8
CPU1 [12] : 4
CPU1 [13] : 4
CPU1 [14] : 4
CPU1 [15] : 6
CPU1 [16] : 6
CPU1 [17] : 6
CPU1 [18] : 10
CPU1 [19] : 10
CPU1 [20] : 10
CPU1 [21] : 5
CPU1 [22] : 5
CPU1 [23] : 5
=====
Total: 70
Promedio: 5.83333
-----
Process exited after 2.946 seconds
with return value 0
Presione una tecla para continuar .
. .
```

1-1-2 ([Monotarea] > [Monoproceso] > [T°P (descendente)]):

```
C:\Users\rocke\OneDri... - □ ×
CPU1 [0] : 2
CPU1 [1] : 1
CPU1 [2] : 12
CPU1 [3] : 9
CPU1 [4] : 11
CPU1 [5] : 11
CPU1 [6] : 3
CPU1 [7] : 3
CPU1 [8] : 7
CPU1 [9] : 7
CPU1 [10] : 8
CPU1 [11] : 8
CPU1 [12] : 4
CPU1 [13] : 4
CPU1 [14] : 4
CPU1 [15] : 10
CPU1 [16] : 10
CPU1 [17] : 10
CPU1 [18] : 6
CPU1 [19] : 6
CPU1 [20] : 6
CPU1 [21] : 5
CPU1 [22] : 5
CPU1 [23] : 5
=====
Total: 70
Promedio: 5.83333
-----
Process exited after 2.771 seconds
with return value 0
Presione una tecla para continuar .
. .
```

1-1-3 ([Monotarea] > [Monoproceso] > [P°T (ascendente)]):

```
C:\Users\rocke\OneDri... - □ ×
CPU1 [0] : 1
CPU1 [1] : 2
CPU1 [2] : 11
CPU1 [3] : 11
CPU1 [4] : 12
CPU1 [5] : 3
CPU1 [6] : 3
CPU1 [7] : 9
CPU1 [8] : 7
CPU1 [9] : 7
CPU1 [10] : 8
CPU1 [11] : 8
CPU1 [12] : 4
CPU1 [13] : 4
CPU1 [14] : 4
CPU1 [15] : 6
CPU1 [16] : 6
CPU1 [17] : 6
CPU1 [18] : 10
CPU1 [19] : 10
CPU1 [20] : 10
CPU1 [21] : 5
CPU1 [22] : 5
CPU1 [23] : 5
=====
Total: 73
Promedio: 6.08333
-----
Process exited after 3.374 seconds
with return value 0
Presione una tecla para continuar .
. .
```

1-1-4 ([Monotarea] > [Monoproceso] > [P°T (descendente)]):

```
C:\Users\rocke\OneDri... - □ ×
CPU1 [0] : 2
CPU1 [1] : 1
CPU1 [2] : 11
CPU1 [3] : 11
CPU1 [4] : 12
CPU1 [5] : 3
CPU1 [6] : 3
CPU1 [7] : 9
CPU1 [8] : 7
CPU1 [9] : 7
CPU1 [10] : 8
CPU1 [11] : 8
CPU1 [12] : 4
CPU1 [13] : 4
CPU1 [14] : 4
CPU1 [15] : 10
CPU1 [16] : 10
CPU1 [17] : 10
CPU1 [18] : 6
CPU1 [19] : 6
CPU1 [20] : 6
CPU1 [21] : 5
CPU1 [22] : 5
CPU1 [23] : 5
=====
Total: 73
Promedio: 6.08333
-----
Process exited after 3.651 seconds
with return value 0
Presione una tecla para continuar .
. .
```

1-2-1 ([Monotarea] > [Multiproceso] > [T°P (ascendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 1
CPU2 [0] : 2
CPU1 [1] : 12
CPU2 [1] : 9
CPU1 [2] : 11
CPU2 [2] : 11
CPU1 [3] : 3
CPU2 [3] : 3
CPU1 [4] : 7
CPU2 [4] : 7
CPU1 [5] : 8
CPU2 [5] : 8
CPU1 [6] : 4
CPU2 [6] : 4
CPU1 [7] : 4
CPU2 [7] : 6
CPU1 [8] : 6
CPU2 [8] : 6
CPU1 [9] : 10
CPU2 [9] : 10
CPU1 [10] : 10
CPU2 [10] : 5
CPU1 [11] : 5
CPU2 [11] : 5
=====
Total: 10
Promedio: 0.833333
-----
Process exited after 2.521 seconds
with return value 0
Presione una tecla para continuar .

```

1-2-2 ([Monotarea] > [Multiproceso] > [T°P (descendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 2
CPU2 [0] : 1
CPU1 [1] : 12
CPU2 [1] : 9
CPU1 [2] : 11
CPU2 [2] : 11
CPU1 [3] : 3
CPU2 [3] : 3
CPU1 [4] : 7
CPU2 [4] : 7
CPU1 [5] : 8
CPU2 [5] : 8
CPU1 [6] : 4
CPU2 [6] : 4
CPU1 [7] : 4
CPU2 [7] : 10
CPU1 [8] : 10
CPU2 [8] : 10
CPU1 [9] : 6
CPU2 [9] : 6
CPU1 [10] : 6
CPU2 [10] : 5
CPU1 [11] : 5
CPU2 [11] : 5
=====
Total: 10
Promedio: 0.833333
-----
Process exited after 2.704 seconds
with return value 0
Presione una tecla para continuar .

```

1-2-3 ([Monotarea] > [Multiproceso] > [P°T (ascendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 1
CPU2 [0] : 2
CPU1 [1] : 11
CPU2 [1] : 11
CPU1 [2] : 12
CPU2 [2] : 3
CPU1 [3] : 3
CPU2 [3] : 9
CPU1 [4] : 7
CPU2 [4] : 7
CPU1 [5] : 8
CPU2 [5] : 8
CPU1 [6] : 4
CPU2 [6] : 4
CPU1 [7] : 4
CPU2 [7] : 6
CPU1 [8] : 6
CPU2 [8] : 6
CPU1 [9] : 10
CPU2 [9] : 10
CPU1 [10] : 10
CPU2 [10] : 5
CPU1 [11] : 5
CPU2 [11] : 5
=====
Total: 12
Promedio: 1
-----
Process exited after 2.347 seconds
with return value 0
Presione una tecla para continuar .

```

1-2-4 ([Monotarea] > [Multiproceso] > [P°T (descendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 2
CPU2 [0] : 1
CPU1 [1] : 11
CPU2 [1] : 11
CPU1 [2] : 12
CPU2 [2] : 3
CPU1 [3] : 3
CPU2 [3] : 9
CPU1 [4] : 7
CPU2 [4] : 7
CPU1 [5] : 8
CPU2 [5] : 8
CPU1 [6] : 4
CPU2 [6] : 4
CPU1 [7] : 4
CPU2 [7] : 10
CPU1 [8] : 10
CPU2 [8] : 10
CPU1 [9] : 6
CPU2 [9] : 6
CPU1 [10] : 6
CPU2 [10] : 5
CPU1 [11] : 5
CPU2 [11] : 5
=====
Total: 12
Promedio: 1
-----
Process exited after 6.729 seconds
with return value 0
Presione una tecla para continuar .

```

3.3 Ejecuciones para los casos de: Multitarea.

2-1-1 ([Multitarea] > [Monoproceso] > [T°P (ascendente)]):

```
C:\Users\rocke\OneDrive\...
CPU1 [0] : 1
CPU1 [1] : 2
CPU1 [2] : 12
CPU1 [3] : 9
CPU1 [4] : 11
CPU1 [5] : 7
CPU1 [6] : 3
CPU1 [7] : 8
CPU1 [8] : 4
CPU1 [9] : 6
CPU1 [10] : 10
CPU1 [11] : 5
CPU1 [12] : 7
CPU1 [13] : 11
CPU1 [14] : 3
CPU1 [15] : 8
CPU1 [16] : 4
CPU1 [17] : 6
CPU1 [18] : 10
CPU1 [19] : 5
CPU1 [20] : 4
CPU1 [21] : 6
CPU1 [22] : 10
CPU1 [23] : 5
=====
Total: 104
Promedio: 8.66667
-----
Process exited after 3.438 seconds
with return value 0
Presione una tecla para continuar .
. .
```

2-1-2 ([Multitarea] > [Monoproceso] > [T°P (descendente)]):

```
C:\Users\rocke\OneDrive\...
CPU1 [0] : 2
CPU1 [1] : 1
CPU1 [2] : 12
CPU1 [3] : 9
CPU1 [4] : 11
CPU1 [5] : 7
CPU1 [6] : 8
CPU1 [7] : 3
CPU1 [8] : 4
CPU1 [9] : 10
CPU1 [10] : 6
CPU1 [11] : 5
CPU1 [12] : 11
CPU1 [13] : 7
CPU1 [14] : 8
CPU1 [15] : 3
CPU1 [16] : 4
CPU1 [17] : 10
CPU1 [18] : 6
CPU1 [19] : 5
CPU1 [20] : 4
CPU1 [21] : 10
CPU1 [22] : 6
CPU1 [23] : 5
=====
Total: 104
Promedio: 8.66667
-----
Process exited after 3.717 seconds
with return value 0
Presione una tecla para continuar .
. .
```

2-1-3 ([Multitarea] > [Monoproceso] > [P°T (ascendente)]):

```
C:\Users\rocke\OneDrive\...
CPU1 [0] : 1
CPU1 [1] : 2
CPU1 [2] : 11
CPU1 [3] : 12
CPU1 [4] : 3
CPU1 [5] : 7
CPU1 [6] : 8
CPU1 [7] : 9
CPU1 [8] : 4
CPU1 [9] : 6
CPU1 [10] : 10
CPU1 [11] : 5
CPU1 [12] : 7
CPU1 [13] : 11
CPU1 [14] : 3
CPU1 [15] : 8
CPU1 [16] : 4
CPU1 [17] : 6
CPU1 [18] : 10
CPU1 [19] : 5
CPU1 [20] : 4
CPU1 [21] : 6
CPU1 [22] : 10
CPU1 [23] : 5
=====
Total: 109
Promedio: 9.08333
-----
Process exited after 3.305 seconds
with return value 0
Presione una tecla para continuar .
. .
```

2-1-4 ([Multitarea] > [Monoproceso] > [P°T (descendente)]):

```
C:\Users\rocke\OneDrive\...
CPU1 [0] : 2
CPU1 [1] : 1
CPU1 [2] : 11
CPU1 [3] : 12
CPU1 [4] : 3
CPU1 [5] : 7
CPU1 [6] : 8
CPU1 [7] : 9
CPU1 [8] : 4
CPU1 [9] : 10
CPU1 [10] : 6
CPU1 [11] : 5
CPU1 [12] : 11
CPU1 [13] : 7
CPU1 [14] : 8
CPU1 [15] : 3
CPU1 [16] : 4
CPU1 [17] : 10
CPU1 [18] : 6
CPU1 [19] : 5
CPU1 [20] : 4
CPU1 [21] : 10
CPU1 [22] : 6
CPU1 [23] : 5
=====
Total: 109
Promedio: 9.08333
-----
Process exited after 2.928 seconds
with return value 0
Presione una tecla para continuar .
. .
```

2-2-1 ([Multitarea] > [Multiproceso] > [T·P (ascendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 1
CPU2 [0] : 2
CPU1 [1] : 12
CPU2 [1] : 9
CPU1 [2] : 11
CPU2 [2] : 7
CPU1 [3] : 3
CPU2 [3] : 8
CPU1 [4] : 4
CPU2 [4] : 6
CPU1 [5] : 10
CPU2 [5] : 5
CPU1 [6] : 7
CPU2 [6] : 11
CPU1 [7] : 3
CPU2 [7] : 8
CPU1 [8] : 4
CPU2 [8] : 6
CPU1 [9] : 10
CPU2 [9] : 5
CPU1 [10] : 4
CPU2 [10] : 6
CPU1 [11] : 10
CPU2 [11] : 5
=====
Total: 28
Promedio: 2.33333
-----
Process exited after 5.659 seconds
with return value 0
Presione una tecla para continuar .
. .

```

2-2-2 ([Multitarea] > [Multiproceso] > [T·P (descendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 2
CPU2 [0] : 1
CPU1 [1] : 12
CPU2 [1] : 9
CPU1 [2] : 11
CPU2 [2] : 7
CPU1 [3] : 8
CPU2 [3] : 3
CPU1 [4] : 4
CPU2 [4] : 10
CPU1 [5] : 6
CPU2 [5] : 5
CPU1 [6] : 11
CPU2 [6] : 7
CPU1 [7] : 8
CPU2 [7] : 3
CPU1 [8] : 4
CPU2 [8] : 10
CPU1 [9] : 6
CPU2 [9] : 5
CPU1 [10] : 4
CPU2 [10] : 10
CPU1 [11] : 6
CPU2 [11] : 5
=====
Total: 28
Promedio: 2.33333
-----
Process exited after 2.121 seconds
with return value 0
Presione una tecla para continuar .
. .

```

2-2-3 ([Multitarea] > [Multiproceso] > [T·P (ascendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 1
CPU2 [0] : 2
CPU1 [1] : 11
CPU2 [1] : 12
CPU1 [2] : 3
CPU2 [2] : 7
CPU1 [3] : 8
CPU2 [3] : 9
CPU1 [4] : 4
CPU2 [4] : 6
CPU1 [5] : 10
CPU2 [5] : 5
CPU1 [6] : 7
CPU2 [6] : 11
CPU1 [7] : 3
CPU2 [7] : 8
CPU1 [8] : 4
CPU2 [8] : 6
CPU1 [9] : 10
CPU2 [9] : 5
CPU1 [10] : 4
CPU2 [10] : 6
CPU1 [11] : 10
CPU2 [11] : 5
=====
Total: 30
Promedio: 2.5
-----
Process exited after 2.49 seconds
with return value 0
Presione una tecla para continuar
. . .

```

2-2-4 ([Multitarea] > [Multiproceso] > [T·P (descendente)]):

```

C:\Users\rocke\OneDri...
CPU1 [0] : 2
CPU2 [0] : 1
CPU1 [1] : 11
CPU2 [1] : 12
CPU1 [2] : 3
CPU2 [2] : 7
CPU1 [3] : 8
CPU2 [3] : 9
CPU1 [4] : 4
CPU2 [4] : 10
CPU1 [5] : 6
CPU2 [5] : 5
CPU1 [6] : 11
CPU2 [6] : 7
CPU1 [7] : 8
CPU2 [7] : 3
CPU1 [8] : 4
CPU2 [8] : 10
CPU1 [9] : 6
CPU2 [9] : 5
CPU1 [10] : 4
CPU2 [10] : 10
CPU1 [11] : 6
CPU2 [11] : 5
=====
Total: 30
Promedio: 2.5
-----
Process exited after 15.4 seconds
with return value 0
Presione una tecla para continuar
. . .

```

Y claro, al inicio junto con las opciones de monotarea y multitarea tenemos también las opciones de consultar la tabla de datos ingresada y de salir...

3 (Ver tabla):

```
C:\Users\rocke\OneDri...
=====
Sistema a Planificar
=====
Planificador
1. MonoTarea
2. MultiTarea
3. Ver tabla
4. Salir
Opcion:
3
Lista de procesos:

PID: 1
T.Llegada: 0
T.Exe: 1
Prioridad: 4
Fin: NO
T.Fin: 0
T.Espera: 0

PID: 2
T.Llegada: 0
T.Exe: 1
Prioridad: 4
Fin: NO
T.Fin: 0
T.Espera: 0

PID: 3
T.Llegada: 0
T.Exe: 2
```

4 (salir):

```
C:\Users\rocke\OneDri...
T.Exe: 2
Prioridad: 4
Fin: NO
T.Fin: 0
T.Espera: 0

PID: 12
T.Llegada: 1
T.Exe: 1
Prioridad: 3
Fin: NO
T.Fin: 0
T.Espera: 0

=====
Sistema a Planificar
=====
Planificador
1. MonoTarea
2. MultiTarea
3. Ver tabla
4. Salir
Opcion:
4
Saliendo...

-----
Process exited after 21.16 seconds
with return value 0
Presione una tecla para continuar
. . .
```

4. Conclusiones

4.1 *¿Qué se aprendió?*

Para empezar el origen de esto está en la teoría de los procesadores y sus núcleos (es bueno tomarlo en cuenta). Y aunque en este código fuente no los trabajamos tal cual, entendimos el concepto de “procesos” o “Tareas” y entendimos el papel que estos juegan en los sistemas operativos e incluso en diversos productos tales como podrían ser: juegos, administradores de ventas, servicios de streaming, entre otros.

El desarrollo de este proyecto nos permitió entender la importancia de priorizar unas tareas respecto de otras, y de cuántas formas se pueden priorizar estas en función de variables como el tiempo (en general) que estas requieran, el trabajo (del procesador) que requieren, e incluso el interés neto que usuario tenga en alguna de ellas. Y con todo esto, aprendimos los conceptos de “Monotarea”, “Multitarea”, “Monoproceso” y “Multiproceso”

Finalmente (y probablemente lo más importante) vimos en primera persona cómo se lleva a cabo ese proceso de priorización / selección. Las variables que se requieren (iteradores, contadores, banderas, valores netos de diversa índole), las estructuras de datos que se requieren (arreglos, matrices, listas ligadas; en nuestro caso incluso trabajamos el administrador como un objeto) y las operaciones que se requieren (lectura de datos, inserción de datos, ordenamiento de datos, anidación de llamados a diversos métodos). ¡¡Y además!! de que un proyecto como este, ayuda a un programador a repasar todos los temas mencionados y a terminar de entenderlos permanentemente.

4.2 *Comentarios finales.*

- La redacción de este documento fue hecha con un enfoque técnico, pero también se hizo lo más accesible posible a un público general, de modo que un estudiante de quinto semestre puede leer esto para

guiarse un poco para sus propios trabajos/proyectos correspondientes. Pero también lo podría leer un estudiante de primeros semestres para darse una idea de lo que puede esperar en el futuro. E incluso, podría ser muy interesante para un estudiante de preparatoria que esté pensando en estudiar algo relacionado al mundo de la computación.

- Se podrían optimizar algunos detalles mínimos del código fuente final, incluso agregar una interfaz gráfica. Pero eso no significa que le falte algo al proyecto, ¡el proyecto funciona!
- Los requisitos mínimos mencionados en el punto **3.2** fueron consultados en:
https://www.tel.uva.es/personales/josdie/fprog/Sesiones/manualDevCpp/requisitos_del_sistema.html