Planificador de CPU

Marta Molina Aguilera – 06/06/2021

Índice

Para ejecutar el código será necesario clicar en la carpeta de "Planificador" en el archivo index.html que se abrirá en el navegador.

Introducción.

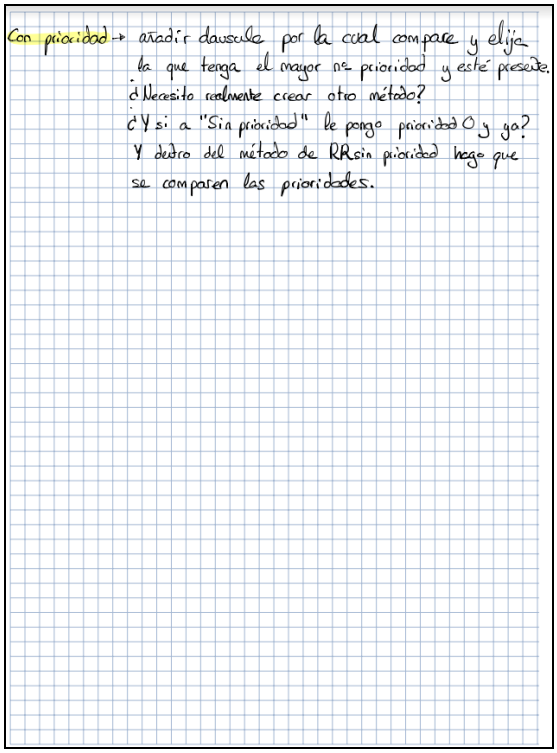
Para la realización de esta actividad me decanté por hacerlo con un formato web. Los recursos elegidos fueron HTML para la estructura,CSS para el estilo JavaScript para la funcionalidad, ya que son lenguajes que hemos usado durante el curso.

Durante el proyecto he experimentado un poco con la programación “orientada a objetos” de JavaScript.

Desarrollo del Proyecto.

Mi primer paso fue pensar cómo daría forma a este reto. Es mejor parar el tiempo que sea necesario y ver las cosas con perspectiva. Releí los apuntes y tomé nota de mis primeras ideas antes de empezar a escribir código. Aunque empezara más tarde, estos pasos me estaban ahorrando tiempo.

Comencé con el proyecto el 22 de mayo.



Mis primeras ideas fueron las que marcaron el rumbo del código principalmente.

Comencé por hacer una estructura básica de HTML y CSS de lo que sería la estructura que acogería el simulador de CPU. En esta primera estructura aparecen 4 botones con los nombres de los algoritmos y una breve descripción de los mismos.

Al pulsar sobre estos (gracias a la función “mostrarForm”)aparece la siguiente pantalla donde se podrán introducir la información de los procesos dependiendo del algoritmo seleccionado(cuyo nombre es pasado por parámetro). La pantalla mostrará según el parámetro: solamente dos (en FIFO/FCFS y SJF: llegada y duración), tres (Round Robin sin prioridad: llegada, duración y quantum) o cuatro en el caso de Round Robin con prioridad(llegada, duración, prioridad y quantum).

POO

Decidí que cada proceso introducido sería un objeto con sus correspondientes atributos:

- **Id (String):** Da una identificación al proceso para dar id a su línea en la tabla y poder hacer referencia a esta para ir incluyendo los tics a su fila.
- **Llegada:** Almacena el momento en el que el proceso se añade a la cola de ejecución.
- **Duracion:** Almacena la duración del tiempo de ejecución del proceso,
- **restaDuracion:** Es un atributo creado para duplicar el de duración, pero poder modificarlo e ir restando el tiempo en que se ejecuta un proceso en los algoritmos de Round Robin. (Traté de almacenar esta información en un array fuera de los objetos, pero el primer dato añadido siempre me daba problemas de NaN (Not a Number). Por ello decidí hace un nuevo atributo que pudiera modificar en el mismo objeto de procesos.)
- **prioridad:** Guarda la prioridad de ejecución en Round Robin con prioridad, en los demás algoritmos se guarda como 0 por defecto.
- **Espera:** Es un atributo que se va autoincrementando por el tiempo que permanezcan en espera los procesos. Inicialmente es 0.
- **Inicio:** Guarda el momento en el que el proceso comience su ejecución. Inicialmente es 0
- **Fin:** Guarda el momento en el que el proceso termine su ejecución. Inicialmente es 0
- **Presente:** es un boolean que determina en la función “pintarColumna()” que se atribuya a la celda la clase “td_enEspera” la cual pone el color de fondo de la celda verde.
- **enEjecucion:** es un boolean que determina en la función “pintarColumna()” que se atribuya a la celda la clase “td_enEjecucion” la cual pone el color de fondo de la celda gris”.
- **Terminado:** es un boolean que determina en la función “pintarColumna()” que se atribuya a la celda la clase “td_noPresente” la cual pone el color de fondo de la celda en blanco y excluye a este proceso de la búsqueda de posibles procesos para ejecutar en la función “buscarPresentes()”.

Para el método que usé para el algoritmo Round Robin decidí incluir un nuevo objeto “QtumYPridad”, que guardaría el quantum vinculado a cada prioridad. Con los atributos:

- Prioridad: Guarda la prioridad.
- Qtum: guarda el quantum asociado a la prioridad.

Todos estos objetos van siendo introducidos en una Array a medida que el usuario los va incluyendo.

annadirProceso(param) y establecerQtum()

Esta función recoge los datos introducidos por el usuario para cada proceso, si la duración introducida es o es dejada en blanco, este muestra un mensaje de error. Va haciendo una conversión de los datos; que inicialmente se recogen como cadenas de caracteres, a números enteros. Una vez ha sido realizada la conversión, el proceso es añadido al array de procesos(arrProcesos).

Si el usuario ha elegido Round Robin con o sin prioridad, se llama desde esta función a "establecerQtum()", la cual hace lo mismo pero registrando los quantums y prioridades. Si el usuario introdujera una prioridad dos veces con distinto quantum, **estos datos vinculados se reescribirían con el nuevo quamtum introducido.** Al terminar esta función el nuevo par de datos es añadido al array de Prioridades.

Funciones de ordenación y de primera ejecución.

Cree varias funciones que son llamadas desde las funciones que se ejecutan los algoritmos. Siempre son el primer paso dentro de los distintos algoritmos.

- **compararLlegada(a, b):** Ordena los procesos de menor llegada a mayor llegada.
- **compararPrioridades(a, b):** Ordena los procesos de mayor prioridad a menor prioridad.
- **function compararDuracion(a,b):** Ordena los procesos de mayor a menor duración. Esta la uso en Shortest Job First. Primeramente antes de usar la de ordenar por llegada los procesos, para que si hay procesos que llegan al mismo tiempo, pero tienen distinta duración, se establezca en el primer puesto el más corto.
- **asignarIdProcYtrs():** Da nombre e id a las filas de la gráfica para poder ir pintando cada fila en su lugar. Concatena "Proceso" más el número que está nombrando: "Proceso 1". Se ejecuta una vez el array de procesos ha sido ordenado según se adecue a cada algoritmo.
- **prepararPantallaTablas():** vacía la pantalla y prepara los contenedores para poner la tabla de datos y la gráfica.

Funciones con llamadas recurrentes dentro de los procesos

Estas funciones son trozos de código que externalicé porque se repetían en todos los procesos. De esta manera simplificaba el código y mi trabajo, siendo código reutilizable en las distintas funciones de cada algoritmo:

- **buscarPresentes( ):** Ordena los procesos de mayor a menor duración. Esta la uso en Shortest Job First. Primeramente antes de usar la de ordenar por llegada los procesos, para que si hay procesos que llegan al mismo tiempo, pero tienen distinta duración, se establezca en el primer puesto el más corto.
- **establecerEnejecucion( ):** Se le pasa el índice por parámeto de un objeto Proceso y cambia su booleana de "enEjecucion" a cierta.
- **function establecerPausa( ):** Retira un proceso en ejecución a presente de nuevo=true, pero sin darlo por finalizado.
- **nombrarProcesosEnGrafica( ):** Escribe el id de cada proceso y lo vincula a la fila de la gráfica para que a las siguientes funciones puedan imprimir las celdas de colores correspondientes a cada proceso.

- **pintarColumna( ):** Teniendo en cuenta si los procesos están en estado de "enEjecucion", "enEjecucion==true", "presente==true" (o no ==false) o "terminado==true". Pinta la celda correspondiente de verde, gris oscuro o claro.
  El bucle se ejecuta por una columna, cómo tantos procesos haya.
  Si el proceso está en estado de presente, aparte de pintar de gris oscuro su celda, va incrementeando con cada ciclo del bucle el tiempo de espera en +1.

- **pintarColumnaAusentes( ):** pinta toda una columna de celdas de gris claro si ningún proceso está presente o en ejecución.

- **baseTablaDatos( )**: Esta función imprime los datos resultantes de la ejecución. Primero imprime la cabecera de la tabla para a continuación imprimir de cada proceso:

    - Nombre del proceso
    - Llegada: Momento en el que el proceso es añadido a la cola de espera para su ejecución.
    - t(T.Ejecución): Duración del proceso
    - Inicio: Momento en el que comienza su ejecución.
    - Fin: Momento en el que termina la ejecución del proceso
    - T(T.Respuesta): Es la resta del momento en el que el proceso es añadido a la cola de espera (llegada) hasta que comienza su ejecución (inicio).
    - T.Espera: El tiempo que un proceso permanece inactivo después de su llegada a la cola.
    - Penalización: Es la división entre el tiempo de respuesta y el de ejecucción.
    - Promedios, se van realizando el sumatorio de para cada promedio durante el cálculo de todos los datos anteriores:
        - Promedio TE= Promedio del tiempo de ejecución. La suma de todos los tiempos de ejecución es dividida entre el número de procesos.
        - Promedio TR= Promedio del tiempo de respuesta. La suma de todos los tiempos de respuesta y se divide entre el número de procesos.
        - Promedio E= Promedio del tiempo de espera. La suma de todas las esperas y se divide entre el número de procesos.
        - Promedio P= Promedio de penalización. La suma de todas las penalizaciones y se divide entre el número de procesos.

FIFO (First In First Out) /FCFS (First Come First Served)

En esta función lo primero que hago es llamar a la función "prepararPantallaTablas()" para a continuación llamar a "compararLlegada( )" para que ordene los procesos según su llegada.

Llamo a la función “asignarIdProcYtrs( )” para darles id y luego a ‘’nombrarProcesosEnGrafica( )’’ para que cada id y nombre aparezca y quede vinculado a la fila que le corresponde.

Lo siguiente es un bucle en el cual se va a ejecutar hasta que el ultimo proceso del array tenga su boolean “terminado” confirmado cómo cierto. Cómo doble seguridad añadí una variable auxiliar (i) la cual se autoincrementa cada vez que se cambia de proceso, comprándola con el largo del array de procesos; en el momento que coincidan, se rompe el bucle.

Durante la ejecución de este bucle se van cambiando los atributos de espera, inicio y fin de cada objeto para más tarde calcular los datos de la ejecución.

Una vez roto el bucle llamo a la función baseTablaDatos() para que calcule e imprima los datos de todos los procesos.

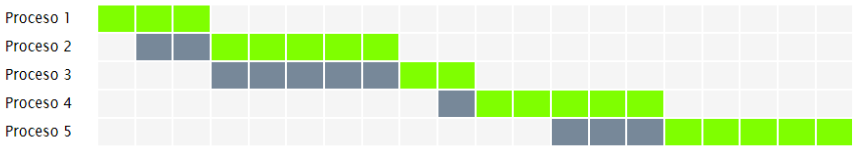
FIFO

El primer proceso en llegar, será el primero en ser atendido.

Proceso	Llegada	t(T.Ejecución)	Inicio	FIN	T(T.Respuesta)	T.Espera	Penalización
Proceso 1	0	3	0	3	3	0	1.00
Proceso 2	1	5	3	8	7	2	1.40
Proceso 3	3	2	8	10	7	5	3.50
Proceso 4	9	5	10	15	6	1	1.20
Proceso 5	12	5	15	20	8	3	1.60

Promedios:

- Promedio t (T.Ejecucción) = 4.00
- Promedio T(Tiempo respuesta) = 6.20
- Promedio espera = 2.20
- Promedio penalización = 1.74



Shortest Job First (SJF)

En esta función lo primero que hago es llamar a la función “prepararPantallaTablas()”.

A continuación ordeno el array de procesos con la función sort, desde la que se invoca a “compararDuracion” para **ordenar el array de procesos por duración**. **A posteriori, hago lo mismo , pero ordenándolos por llegada. De esta manera si dos procesos llegan en el mismo momento, quedará siempre antes el del menor duración**, porque previamente los ordenamos de esta forma.

Llamo a la función “asignarIdProcYtrs( )” para darles id y luego a ‘’nombrarProcesosEnGrafica( )’’.

En esta función tengo una variable que es “elegido”que guarda el índice del proceso elegido para ejecutarse. Y una booleana “terminados” que será true una vez todos los procesos hayan sido marcados como terminados.

El imprimir la gráfica comienda con un bucle de tipo while; cuya duración será hasta que “terminados” sea cierta.

Primero compruebo que el proceso elegido tenga un momento de llegada igual al momento en el que nos encontramos. Si es así, llamará a la función “establecerEnejecucion(elegido)” y buscará el resto de procesos presentes. Imprimirá una columna por cada momento de duración del proceso actual e incrementará el momento en +1 por cada vuelta a este.

Una vez el proceso es finalizado, lo marca como terminado con “establecerTerminado(elegido);” y lo añade al array de excluidos para no volverlo a seleccionar si la condición de terminado fallara. Busca a los presentes.

Después con un bucle for busco el siguiente presente para establecerlo cómo elegido. Hago la variable “terminados” cierta y entro en un nuevo bucle, el cual compara la duración de todos los presentes, si hay algún proceso más corto presente, cambia a este cómo elegido.

El bucle se vuelve a repetir, si no hay ningún proceso presente aún pinta una columna en blanco ese momento de la ejecución y vuelve a buscar.

Por último llama a la función” baseTablaDatos( )” para imprimir la tabla con los datos de la ejecución y el promedio.

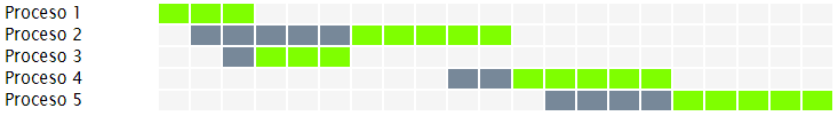
SJF (Shortest Job First)

El proceso más corto será el primero en ser atendido.

Proceso	Llegada	t(T.Ejecución)	Inicio	FIN	T(T.Respuesta)	T.Espera	Penalización
Proceso 1	0	3	0	4	4	0	1.00
Proceso 2	1	5	6	12	11	5	2.00
Proceso 3	2	3	3	7	5	1	1.33
Proceso 4	9	5	11	17	8	2	1.40
Proceso 5	12	5	16	22	10	4	1.80

Promedios:

- Promedio t (T.Ejecucción) = 4.20
- Promedio T(Tiempo respuesta) = 7.60
- Promedio espera = 2.40
- Promedio penalización = 1.51



Round Robin con y sin prioridad.

Lo primero que hago es preparar la pantalla con ”prepararPantallaTablas();” y ordenar los procesos por duración con una llamada a “compararLlegada”, en el de con priorirdad lo ordeno previamente por prioridad. Les asigno un id una vez ordenados con “asignarIdProcYtrs();” y les nombro en la gráfica con “nombrarProcesosEnGrafica();”

En el Round Robin sin prioridad un array guardo la duración de cada proceso para ir restando el tiempo que está en ejecución del que le queda. En el Robin con prioridad este mismo método siempre en el primer número del array me daba un error de NaN sin haber cambiado nada dentro del código así que hago la resta a restaDuracion en el mismo objeto del Proceso.

Tengo un nuevo array en el que guardo los procesos que hayan terminado “excluidos”, una vez la longitud de este sea mayor o equivalente a la del array de procesos, el bucle while finaliza.

Dentro del bucle tengo un if con la condición de comprobar si el momento de llegada del proceso elegido es equivalente al momento de ejecución actual.

Si es así llamará dentro de un bucle for a la función “pintarColumna();” por la duración del quantum vinculado a la prioridad del proceso (que será la misma, 0, en el RR sin prioridad) o por la duración restante del proceso; lo que sea menor de ambos datos.

Una vez finalizado el bucl si el tiempo de ejecución restante del proceso es igual o menor que cero se da por finalizado y lo añadirá al array de excluidos para buscar los procesos presentes.

Si el elegido no estuviera aún en ejecución, se imprimiría una columna en blanco y se buscarían los presente de nuevo.

Una vez finalizado el bucle de código del if; se buscan los procesos presentes para establecer el nuevo elegido comparando su llegada y nivel de prioridad.

Round Robin (con prioridad)

Se establece un nivel de prioridad a cada proceso. Esta prioridad tiene asociado un quantum. Una vez este es agotado el periodo, pasa al siguiente proceso con una prioridad igual o superior, si no lo hay, baja un nivel de prioridad.

Proceso	Llegada	t(T.Ejecución)	Inicio	FIN	T(T.Respuesta)	T.Espera	Penalización	Prioridad
Proceso 1	0	12	16	21	21	8	1.67	5
Proceso 2	2	6	30	33	31	24	5.00	1
Proceso 3	3	11	20	24	21	9	1.82	5
Proceso 4	3	3	29	31	28	24	9.00	1

Promedios:

- Promedio t (T.Ejecucción) = 8.00
- Promedio T(Tiempo respuesta) = 25.25
- Promedio espera = 16.25
- Promedio penalización = 4.37

Prioridades:

- Prioridad: 5 => Qtum: 4
- Prioridad: 1 => Qtum: 2

Conclusión

Ha sido un buen reto para probar mis conocimientos de SSII y lo aprendido en lenguaje de marcas y youtube sobre JS.

Me gustaría haberle hecho alguna mejora gráfica al proyecto y simplificar el código, pero me centré más en que funcionara de manera óptima. Ha sido un buen reto, seguramente en verano mejore los fallos y lo añada a mi portfolio de proyectos.