SIMULACIÓN DEL CONTROL DE PROCESOS EN UN SISTEMA OPERATIVO

Práctica 2: árboles binarios de búsqueda

Juan Pérez Resa DNI: 09147555H
Cristina Martínez Toledo DNI: 09109126E

Índice

0. Introducción:

Para la práctica 2 se quiere implementar un árbol binario de búsqueda que contenga los procesos del “sistema operativo” de la práctica 1 que hayan finalizado su ejecución. Para esto, a la segunda parte de la práctica uno se le han añadido varios TADs nuevos: ”ListaProcesos” con su “NodoListaProcesos” y “Arbol” con su “NodoArbol”. Además, se han modificado ciertas partes de los TADs ya creados anteriormente para facilitar la implementación de las funciones de esta nueva práctica.

1. NodoListaProcesos y ListaProcesos

El TAD NodoListaProcesos es muy similar al “NodoLista” creado en la práctica 1, con un proceso y un puntero al siguiente nodo de la lista dinámica. Sus funciones son solamente el constructor vacío, el constructor al que se le pasa el proceso como parámetro y el destructor.

La implementación de las funciones básicas de ListaProcesos también es bastante similar a “Lista” (añadir por la derecha o la izquierda, borrar el primero o el último elemento, crear una copia de la lista, mostrarla...), sin embargo además de esas funciones se crearon varias nuevas. Estas nuevas funciones añadidas son:

- **1.1 len()**

Calcula y devuelve la longitud de la lista (cantidad de procesos que contiene). Se implementó con un bucle while que recorre una copia de la lista (en cada iteración, mientras no esté vacía, suma 1 a un contador y elimina el primer elemento)

- **1.2 tiempoMedioEjecucionLista()**

Tanto para esta función como para sumaTiemposEstancia() se modificó la clase Proceso. Se le añadieron tres nuevas variables enteras (tiempoLlegada, tiempoFin y tiempoEstancia) inicializadas a -1 y que durante la ejecución del proceso se modificarán. Cuando el tiempo actual sea el tiempo de inicio del proceso y este entre al SO, tiempoLlegada será el tiempo actual. Cuando finalice su ejecución, se cambiará el valor de tiempoFin, y en este momento se llamará a una función del proceso (calcularTiempoEstancia()) que asignará a tiempoEstancia el valor de tiempoFin-tiempoLlegada.
En tiempoMedioEjecuciónLista, si la lista no es vacía (calculado con len()>0), con un bucle while sobre una copia de la lista se irá sumando a una variable el valor devuelto por calcularTiempoEstancia de cada proceso que contenga (de forma muy similar a len()).

Después, se dividirá entre la longitud para dar el tiempo medio de ejecución de los procesos en esa lista, devolviendo este valor como un tipo double.

- **1.3 sumaTiemposEstancia()**

Esta función es una función auxiliar para una en la clase Arbol. Funciona igual que tiempoMedioEjecuciónLista pero en vez de dividir entre la longitud, solamente calcula y devuelve la suma de todos los tiempos medios de estancia de esa lista.

2. Árbol y NodoArbol

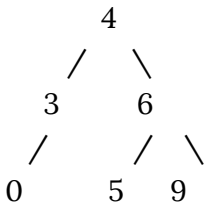
"Arbol" es un TAD que contendrá un "NodoArbol" como raíz y otros dos árboles como sus "ramas" izquierda y derecha. El NodoArbol solamente contendrá un entero que indica la prioridad de ese nivel y la lista de procesos con esa prioridad. Para que el árbol binario siempre quede más o menos equilibrado, se asignó al nodo raíz la prioridad 4, un valor aproximadamente en el centro entre 0 y 9. A continuación, vamos a explicar algunas de las funciones de Árbol más en detalle.

- **2.1 insertarProcesoEnArbol() (función auxiliar) e insertarProceso()**

A ambas funciones se les pasará el proceso como parámetro. insertarProceso() comprobará si en el árbol ya se encuentra un nivel con la prioridad del proceso dado (con estaPrioridad). Si está, recursivamente buscará el nivel correcto y añadirá el proceso por la derecha a la lista. Si no está, llamará a insertarProcesoEnArbol. Esta otra función buscará recursivamente el "lugar" correcto donde añadir el nuevo nivel y lo creará; si la prioridad es menor (o igual, aunque nunca vaya a serlo) a la prioridad de la raíz actual, buscará en la izquierda, y si no en la derecha. Si el hijo izquierdo/derecho (según corresponda) de la raíz actual es nulo, se creará un nuevo árbol con el constructor (el no vacío) al que se le introduce como parámetros la prioridad y la lista de procesos (que contendrá solamente el nuevo proceso). Si no es nulo, se buscará recursivamente en el hijo izquierdo o derecho.

- **2.2 mostrarArbol()**

Esta función recursiva mostrará el árbol en preorden, enseñando primero la raíz, después el hijo izquierdo y finalmente el derecho de cada nivel. Por ejemplo, si nuestro árbol tuviese la forma:



La función mostraría lo siguiente:

NODO PRIORIDAD 4
Lista de procesos: .....
----------Nodos izquierdos----------
NODO PRIORIDAD 3
Lista de procesos: .....
----------Nodos izquierdos----------
NODO PRIORIDAD 0
Lista de procesos: .....
----------Nodos derechos----------
NODO PRIORIDAD 6
Lista de procesos: .....
----------Nodos izquierdos----------
NODO PRIORIDAD 5
Lista de procesos: .....
----------Nodos derechos----------
NODO PRIORIDAD 9
Lista de procesos: .....

- **2.3 estáPrioridad()**

Esta función, si detecta que la prioridad pasada como parámetro es la prioridad de la raíz actual, devolverá true. Si no, buscará recursivamente en los subárboles izquierdo y derecho del árbol (y devolverá true si lo encuentra en uno de ellos). Si recorre todo el árbol y no encuentra el nivel, devolverá false.

- **2.4 mostrarPrioridadDada()**

Si la prioridad pasada como parámetro existe en el árbol (si estaPrioridad devuelve true), se buscará el nivel "correcto" de la misma forma que en insertarProceso y mostrará la raíz y lista de procesos por pantalla. Si no se encuentra el nivel en el árbol, se mostrará que no hay ningún nivel de esa prioridad.

- **2.5 mostrarPrioridadesEjecutadas()**

Esta función muestra todos los niveles de prioridad que han tenido al menos un proceso
Ejecutado, con las prioridades ordenadas de forma numérica (de menor a mayor valor de prioridad; primero el 0 y finalmente el 9). La función lo que hará será mostrar el árbol en inorden, comprobando primero para cada nivel el subárbol izquierdo, mostrando la raiz y finalmente el derecho

- **2.6 mayor/menorNumeroProcesosAux() y mayor/menorNumeroProcesos()**

Las funciones mayorNumeroProcesos() y menorNumeroProcesos(), se encargan de mostrar la prioridad con mayor y menor número de procesos ejecutados respectivamente. Para facilitar la programación, se ha empleado una función auxiliar que es la que se encarga de llevar a cabo la sucesivas llamadas recursivas, y la función mayorNumeroProcesos/menorNumeroProcesos que se encarga antes de llamar a la función recursiva, crear una nueva estructura de datos, en concreto un árbol, donde se irán insertando los procesos. El principal motivo para usar un árbol es el hecho de que es necesario mostrar las prioridades con misma cantidad de procesos, es decir, puede haber dos o más prioridades con la misma cantidad de procesos ejecutados, y hay que mostrar todas esas prioridades. Una vez comprendido esto, la lógica de la función es sencilla. En primer lugar, se crea el árbol que se usará (estableciéndose como raíz del árbol, la raíz del árbol original), y se llama a la función auxiliar que llevará a cabo las llamadas recursivas. Si nos encontramos con que la prioridad de la raíz tiene la misma cantidad de procesos que la prioridad en la que nos encontramos, entonces hay que añadir esos procesos al árbol. En el caso de que nos encontrásemos con una prioridad con mayor número de procesos (menor número de procesos en el caso de la función menorNumeroProcesos()) que la cantidad de procesos de las prioridades del árbol, entonces, hay que destruir el árbol, ya que ya no nos sirve el árbol con el que contábamos. Este proceso se repetirá mientras sigan habiendo hijos izquierdos o hijos derechos.

- **2.7 tiempoMedioEjecucionNivel()**

Calcula el tiempo medio de ejecución para un nivel dado (una prioridad específica). Primero, comprueba si ese nivel existe en el árbol. Si lo hace, busca el nivel correcto (igual que en insertarProceso), y cuando lo encuentra calcula y devuelve (como double) el tiempo medio de estancia llamando a tiempoMedioEjecucionLista de su lista de procesos asociada. Si no la encuentra, devolverá -1 y mostrará por pantalla un mensaje de que no existe ese nivel en el árbol.

- **2.8 tiempoMedioEstanciaAux() y tiempoMedioEstancia()**

TiempoMedioEstancia calculará mediante los procesos del árbol el tiempo medio de estancia general (para todos los niveles) en el SO. Estos procesos serán los ya finalizados del SO, o según las especificaciones de la práctica, añadidos manualmente por teclado. Si no se añade ningún nuevo proceso manualmente, este tiempo será igual al calculado en la práctica 1, pero si se añade uno, el tiempo medio total de estancia variará debido a que su tiempo medio de estancia será de 0 y se dividirá por un proceso más (al igual que el tiempo de estancia específico del nivel donde se añada también cambiará).
La función llamará a tiempoMedioEstanciaAux, pasándole dos variables inicializadas a 0 (double sumaTiempos e int cuentaProcesos) por referencia. tiempoMedioAux recorrerá todo el árbol sumando a cuentaProcesos la longitud (len()) y a sumaTiempos el valor devuelto por tiempoMedioEjecucionLista de cada lista. La función auxiliar finalmente devolverá el double sumaTiempos (aunque realmente se podría eliminar el return y convertir la función en un void). Entonces, tiempoMedioEstancia si el número de procesos es mayor que 0, devuelve la división entre sumaTiempos y cuentaProcesos (si recordamos de la práctica anterior, el tiempo medio de estancia total es igual a: $\frac{\sum(tiempo\ de\ finalización - tiempo\ de\ inicio)}{número\ de\ procesos\ total}$)

3. Cambios en SistemaLista()

De la práctica 1, se tuvo que modificar ligeramente SistemaLista para añadir los cambios nuevos. Además de, como se dijo anteriormente, modificar las variables de los tiempos de los procesos según su estado en el sistema, se creó un nuevo árbol llamado “abb” en el mismo. Primero, en la clase Núcleo se modificó la función terminarproceso para que en lugar de ser un void devolviese el proceso que acaba de finalizar su ejecución. Cada vez que finaliza un proceso (en pasarTiempo o en procesoComienzo), se inserta el mismo en el abb.
Además, para poder acceder desde el main a las funciones del árbol del sistema, se crearon varias funciones llamadoras, y para implementar que se pueda añadir directamente un proceso al abb desde el teclado, se creó una función a la que se le pasan todos los datos de Proceso como parámetro, comprueba si son datos válidos (PID>0, inicioProceso>=0, tiempoVida>0, prioridad entre 0 y 9, PPID>=-1) y si lo son crea el proceso y lo inserta en el árbol.

4. Nuevo main()

Para cumplir las exigencias de la nueva práctica, se modificó el main, primero obviamente cambiando el “prompt” del menú para poder también mostrar las nuevas opciones y añadiendo al switch los nuevos “cases”. Los casos del menú nuevos de esta práctica son los

del 8 al 15, y todos ellos son simplemente llamadas a las funciones nuevas de SistemaLista, que a su vez llamarán a las correspondientes funciones de abb. En algunos casos, como el 8 (leer de teclado un proceso e introducirlo al abb), el 10 (Mostrar los procesos del abb con la prioridad introducida) o el 13 (ver el tiempo promedio de ejecución de los procesos con una prioridad específica), antes se leen datos de teclado para poder pasarlos a las funciones llamadas como parámetro.