Curso 2003/2004. Estructura de Datos y de la Información
I. Informática, I. T. Informática de Gestión y de Sistemas

PRÁCTICA 2

1. Enunciado

Se desea gestionar la ejecución de procesos en un supercomputador para los que se seguirá el orden establecido por una **cola de prioridades**. Los procesos que llegan a esta cola tienen una prioridad asignada y un tiempo de entrada en el sistema.

En este sistema tendrán prioridad los procesos de menor tamaño. De este modo, la prioridad se determina como el resultado de la división entera entre el tamaño del ejecutable y 1000, y los procesos se mantienen ordenados de modo creciente según este cociente. Así, siempre se ejecuta el proceso menos reciente de mayor prioridad (i.e., menor tamaño). En circunstancias normales, los procesos se ejecutan cada **5 milisegundos**. Sin embargo, es posible que la ejecución de un proceso se retrase más de 20 milisegundos debido a que se están ejecutando otros procesos. En este caso su nivel de prioridad aumenta.

2. Implementación

Para resolver el problema será necesario implementar una **cola de prioridad** como una lista de colas, haciendo uso de variables dinámicas para mantener tanto la lista como las colas:

1. Una **lista ordenada** para mantener las prioridades de los procesos que están a la espera.

2. Varias **colas**, una por prioridad, para mantener las procesos en espera.

La figura 1 muestra un ejemplo de estas estructuras y sus interrelaciones para formar, una vez unidas, una cola de prioridad. Como se puede observar, la cola de prioridades mantendrá tantas colas como prioridades vayan apareciendo.

Así pues se implementarán:

- **Unit** `cola.pas`: implementación del `TAD Cola` utilizando variables dinámicas. Se consideran los tipos siguientes:

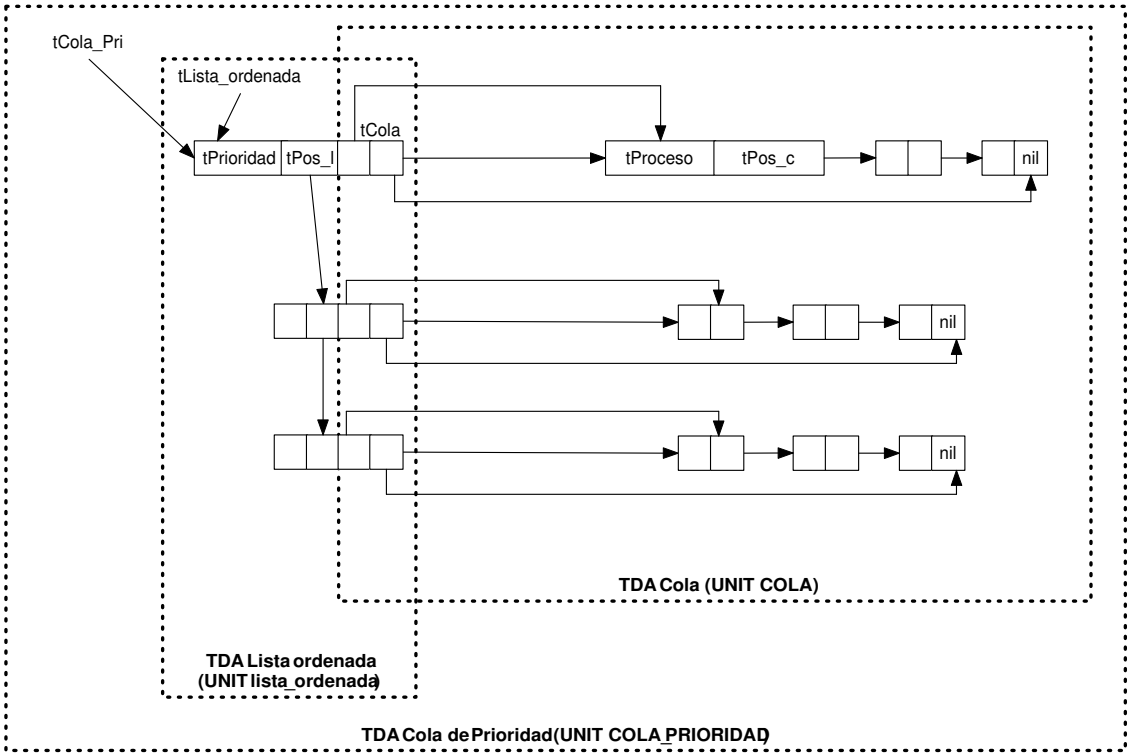

Figura 1: Cola de prioridades

- `tProceso` → tipo de los datos almacenados en la cola. Estará compuesto por:
  - *tTiempo*: tiempo de entrada en el sistema (entero)
  - *tIdProceso*: identificador del proceso (string)
  - *tTamano*: tamaño del ejecutable correspondiente al proceso (entero)
- `tPos_c` → tipo de los punteros a los nodos de la cola
- `tNodo_c` → tipo de los nodos de la cola
- `tCola` → tipo de la cola

Implementar las siguientes operaciones:

`Cola_vacia (tCola)` → `tCola`
{Crea una Cola vacía}

`Es_cola_vacia (tCola)` → `Boolean`
{Determina si Cola está vacía
Precondición: La Cola debe de estar inicializada}

`Insertar_c (tCola, tTiempo, tIdProceso, tTamano)` → `tCola`
{Inserta un elemento con unos contenidos tTiempo, tIdProceso y tTamano en la Cola.

Precondición: La Cola debe de estar inicializada y se supone memoria suficiente para realizar la inserción.}

`Frente_c (tCola)` → `tTiempo, tIdProceso, tTamano,Boolean`
{Devuelve el contenido del elemento que se encuentra al frente de la Cola. Devuelve *verdadero* si se ha podido realizar la operación, *falso* en caso contrario
Precondición: La Cola debe de estar inicializada
Poscondición: La Cola no se modifica}

`Borrar_c (tCola)` → `tCola, Boolean`
{Elimina el elemento que está al frente de la Cola. Devuelve *verdadero* si se ha podido realizar la operación, *falso* en caso contrario
Precondición: La Cola debe de estar inicializada}

- **Unit** `listaOrdenada.pas`: implementación del `TAD Lista Ordenada` mediante variables dinámicas DOBLEMENTE enlazadas. Se consideran los tipos siguientes:

  - `tPrioridad` → tipo de las prioridades de las colas (entero)
  - `tPos_l` → tipo de las posiciones de la lista que mantiene las colas
  - `tNodo_l` → tipo de los nodos de la lista que mantiene cada una de las colas
  - `tLista_ordenada` → tipo de la lista

  Esta unit puede implementarse adaptando la unit `listaDinamica.pas` de la Práctica 1, en los siguientes términos:

  - Incluir en los nodos un enlace al nodo anterior
  - Las operaciones `ListaVacia, EsListaVacia, Primero, Siguiente` y `Ultimo` necesitan ser repasadas para comprobar que el código sigue siendo válido.
  - La operaciones `Anterior`, `Borrar` se reimplementarán teniendo en cuenta los dobles enlaces.
  - La operaciones `ObtenerDato` y `Actualiza` se reimplementarán de la siguiente manera:

    `ObtenerDato (tLista_ordenada, tPos_l)` → `tCola, tPrioridad`
    `Actualiza (tLista_ordenada, tPos_l, tPrioridad)` → `tLista_ordenada`
  - La operación Insertar se reimplementará teniendo en cuenta que debe hacerse una inserción en un lista ordenada y doblemente enlazada. La especificación de esta nueva función será:

`InsertarOrdenado (tLista_ordenada, tPrioridad)` → `tLista_ordenada`
{Inserta un elemento en la lista por orden creciente de una prioridad que quedará almacenada como parte de la información del nuevo nodo insertado.
Precondición: No puede existir en la lista un elemento con la misma prioridad y se supone memoria suficiente para realizar la inserción.
Poscondición: La cola correspondiente quedará inicializada.}

- Se deberá implementar una nueva función `BuscarDato` con la siguiente especificación:

  `BuscarDato (tLista_ordenada, tPrioridad)` → `tPos_l`
  {Determina la posición en la lista de un elemento que tiene una determinada prioridad. El valor de retorno será NULO si no existe en la lista tal elemento. Precondición: La lista debe estar inicializada.}

- **Unit** `colapri.pas`: Utiliza las units `cola.pas` y `listaOrdenada.pas` para implementar un TAD Cola de Prioridad. Se considerará, por lo tanto, el nuevo tipo siguiente:

  `tCola_pri` → tipo de la cola de prioridades

  Implementar las siguientes operaciones:

  `Cola_prioridad_vacia (tCola_Pri)` → `tCola_Pri`
  {Crea una Cola de Prioridad vacía}

  `Es_cola_pri_vacia (tCola_Pri)` → `Boolean`
  {Determina si la Cola de Prioridad está vacía
  Precondición: La Cola de Prioridad debe estar inicializada}

  `Insertar_cp (tCola_Pri, tTamano, tTiempo, tIdProceso, tPrioridad)` → `tCola_Pri`
  {Inserta un proceso en la Cola de Prioridad según la prioridad almacenada en tPrioridad
  Precondición: La Cola de Prioridad debe estar inicializada y se supone memoria suficiente para realizar la inserción.}

  `Borrar_cp (tCola_Pri)` → `tCola_Pri, Boolean`
  {Elimina el primer proceso de la Cola de Prioridad. Devuelve *verdadero* si se ha podido realizar la operación, *falso* en caso contrario
  Precondición: La Cola de Prioridad debe estar inicializada}

  `Frente_cp (tCola_Pri)` → `tTiempo, tIdProceso, tTamano, tPrioridad, Boolean`

{Devuelve la información del proceso de máxima prioridad (primer elemento de la Cola de Prioridad), *verdadero* si se ha podido realizar la operación o *falso* en caso contrario.
Precondición: La Cola de Prioridad debe estar inicializada
Poscondición: La Cola de Prioridad no se modifica}

`ReasignaPrioridad_cp (tCola_Pri, tTiempo)` → `tCola_Pri`
{Mueve a la cola de siguiente prioridad (de entre las existentes en ese momento) los procesos con tiempo de espera de más de 20 milisegundos con respecto a *tTiempo*, que es el tiempo actual. Además, al mover un proceso se muestra el mensaje siguiente:

`Aumentando prioridad proceso ...<IDProceso><AntiguaPrioridad><NuevaPrioridad>`
Precondición: La Cola de Prioridad no podrá estar vacía.
Poscondición: Al mover un proceso **se reinicia su *tiempo de espera* al tiempo actual *tTiempo***, lo que equivale a iniciar la espera a cero. }

- **Programa** `principal.pas`. Secuencia de ejecución del programa:

  1. Para simular la entrada de procesos en el sistema, en esta práctica utilizaremos una **cola** que contendrá todos los procesos pendientes. Por lo tanto, como primer paso se lee un fichero "procesos.dat" con la lista de procesos en espera de ejecución y se insertan en una *cola de entrada en el sistema*.

     NOTA IMPORTANTE: Para poder realizar este punto se suministra a los alumnos el fichero "procesos.dat". Este fichero contiene una línea por cada proceso con el identificador de proceso, su tamaño y el tiempo de entrada en el sistema. Los procesos se encuentran ordenados por el campo *tiempo de entrada en el sistema*.

     También se proporciona el código que implementa los procedimientos de lectura de este fichero y el almacenamiento de los datos de los procesos en una cola (lectura.pas). Todo esto puede encontrarse en cualquiera de los directorios:

     `/PRACTICAS/ETIX/EDI/P0` (Alumnos de I.T.I. de Gestión)
     `/PRACTICAS/EIIS/EDI/P0` (Alumnos de I.T.I. de Sistemas)
     `/PRACTICAS/EI/EDI/P0` (Alumnos de I. Informática)
  2. Inicializar a cero un contador de tiempo.
  3. Repetir hasta que no haya más elementos en la cola de prioridad y no haya más elementos en la cola de entrada.

- Incrementar contador de tiempo y mostrarlo
- Repetir mientras haya procesos en la cola de entrada con *tiempo de entrada en sistema* igual al tiempo actual
  - Recuperar el primer proceso de la cola de entrada e insertarlo en la cola de prioridad de acuerdo a la prioridad del proceso.
  - Imprimir un mensaje de acuerdo al formato siguiente:

    `Inicializando ejecución de proceso ...<IDProceso><Prioridad>`

    `<TEntradaSistema>`
- Si han transcurrido 5 milisegundos desde que se inició la última ejecución de un proceso, ordenar la ejecución del elemento de máxima prioridad eliminándolo de la cola, y mostrar un mensaje con el formato siguiente:

  `Ejecutando proceso...<IDProceso><Prioridad><TEntradaSistema>`
- Si existe algún elemento de alguna de las colas de la cola de prioridad que tenga un retraso de más de 20 milisegundos, eliminarlo de la cola, aumentar su prioridad, cambiar su tiempo de entrada en el sistema al tiempo actual e insertarlo de nuevo en la cola de prioridad.

4. Fin

NOTA IMPORTANTE

Se recuerda que:

1. Desde fuera de las units que implementan los TAD cola, Lista Ordenada y Cola de Prioridad no se podrá acceder directamente a las estructuras definidas más que a través de las operaciones proporcionadas en las secciones de INTERFACE de las correspondientes UNITs.

2. El desarrollo de una buena interfaz, aunque importante, **NO ES EL OBJETIVO** de esta asignatura. Por lo tanto, no se permitirá el uso de la UNIT CRT en la implementación de la práctica.

3. Normas de realización y entrega

- Las prácticas serán INDIVIDUALES y OBLIGATORIAS.
- La entrega de TODAS las prácticas en las FECHAS indicadas es requisito IMPRESCINDIBLE para aprobar la asignatura en la convocatoria ordinaria de JUNIO. Paras las convocatorias de SEPTIEMBRE y DICIEMBRE las prácticas serán las mismas pero se fijarán otras fechas de entrega.
- Las prácticas tendrán que ser ejecutables en FreePascal en la máquina LIMIA.

- **Fecha límite de entrega: 4 de Junio de 2004**.
- Forma de entrega: Las prácticas quedarán depositadas en la red. No se admitirán discos ni papel. El proceso para depositarlas será el siguiente:

  1. Conectarse a la máquina ALBA
  2. Situarse en los directorios:
     `/PRACTICAS/ETIX/EDI/P2` (Alumnos de I.T.I. de Gestión)
     `/PRACTICAS/ETIS/EDI/P2` (Alumnos de I.T.I. de Sistemas)
     `/PRACTICAS/EI/EDI/P2` (Alumnos de I. Informática)
  3. Situarse en el directorio que coincida con el login del usuario y copiar allí las **units** y el **programa principal** (sólo los ficheros **fuentes**, no los ejecutables).
     IMPORTANTE: el nombre del fichero fuente que contenga el programa principal deberá ser `principal.pas`.
  4. Pasada la fecha de entrega no se permitirá el acceso a estos directorios.

- Estructura que deberá tener cada programa desarrollado en la práctica:

  - **Encabezamiento del programa**. Constará de la siguiente información entre comentarios:

    ```
    TITULO:    Practicas de EDI
    SUBTITULO: Bloque 2
    AUTOR: ______________________________________
    LOGIN: ________
    GRUPO: E.I. / E.T.I.X / E.T.I.S
    FECHA: __/__/____
    ```

  - **Cuerpo del programa**.
    - El código irá comentado. Los comentarios han de ser **concisos** pero **explicativos**.
    - Después de la cabecera de cada procedimiento o función se incluirá lo siguiente:
      - Objetivo
      - Entradas
      - Salidas
      - Precondiciones (Condiciones que han de cumplir las entradas para el correcto funcionamiento de la subrutina).
      - Poscondiciones (Otras consecuencias de la ejecución de la subrutina que no queden reflejadas en la descripción del Objetivo o de las Salidas).

- **Criterios de valoración de la práctica**:

  - *Eficacia*: que se cumplan las especificaciones.
  - *Control de errores*: control intensivo de todos los errores de ejecución que sea posible detectar.
  - *Claridad*: que el programa se pueda entender con facilidad, que contenga los comentarios oportunos, la indentación adecuada, nombres de variables significativos, etc.
  - *Modularidad*: que esté construido con módulos intercambiables, reutilizables y portables.