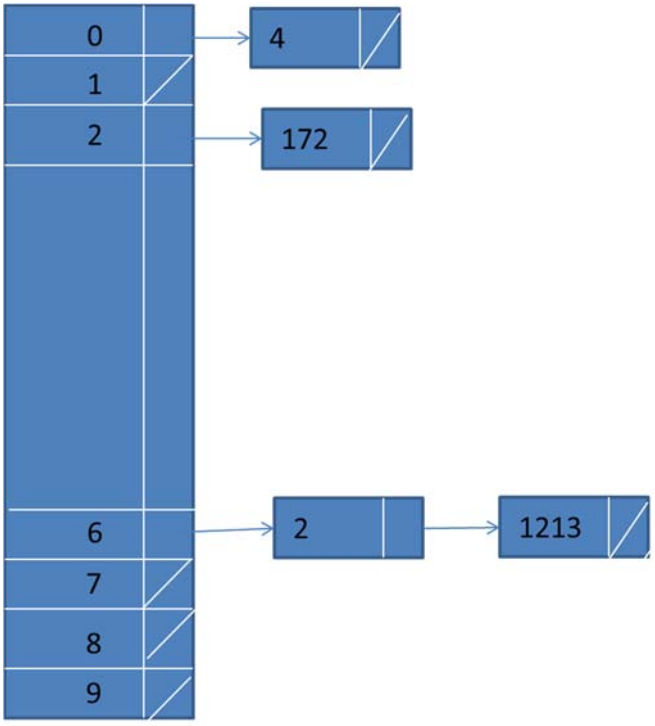
Programación de Sistemas y Concurrencia
Dpto. de Lenguajes y Ciencias de la Computación
Examen 11 de Septiembre 2017

APELLIDOS_____ NOMBRE_____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN_____

Bloque Programación C (3.5 puntos)

Se desea implementar un estructura de procesos en memoria en el que existen 10 niveles de prioridad en los que se clasifican cada proceso por orden de llegada, tal y como se puede ver en la figura. Por ejemplo, el proceso de identificador 4 tiene prioridad 0 y los procesos 2 y 1213 tienen prioridad 6, aunque el proceso 2 está disponible para ejecución antes que el proceso 1213.



Se dispone de un fichero **binario** con el siguiente formato en el que el orden en el que aparecen en el fichero indica el orden de llegada:

`<numprocesos><idproceso><prioridad><idproceso><prioridad><idproceso><prioridad><idproceso><prioridad>`…

Donde:

`<numprocesos>` es el número de procesos de la cola (**int**)

`<idproceso>` es el identificador del proceso (**int**)

`<prioridad>` es la prioridad del proceso (**int**)

Implementar los siguientes procedimientos:

`void Inicializar(TEstProc cp).` Inicializa la estructura haciendo vacías las colas de identificadores de procesos.

`void Insertar(TEstProc cp, int id, int prio).` Inserta en la estructura `cp` el proceso de identificador `id` con prioridad `prio`. El `id` del proceso se inserta al **final** de la cola correspondiente.

`void Mostrar(TEstProc cp).` Muestra el contenido de la estructura de procesos actual.

`void EjecutarUno(TEstProc cp, int id).` Elimina de la estructura y libera la memoria del proceso de identificador `id`, si existe. Si no existe ningún proceso con ese identificador, el procedimiento no hace nada.

`void Ejecutar(TEstProc cp, int prio).` Elimina de la estructura todos los procesos de prioridad `prio`.

`void Destruir(TEstProc cp).` Elimina todos los elementos de la estructura liberando su memoria.

`void Crear(char *nombre, TEstProc cp)`. Inserta en la estructura `cp` la información sobre los procesos almacenada en el fichero con el nombre `nombre`. `cp` es el array de listas que debe ser **inicializado** antes de empezar a insertar. Los identificadores de los procesos se insertan siempre al **final** de las listas.

En el campus virtual, se encuentra

- El fichero `procesos.h` con la definición del tipo `TEstProc` y con la cabecera de los procedimientos que debes implementar.
- Un fichero binario "`procesos.bin`" con la estructura descrita arriba.
- Un fichero `Principal.c` con dos métodos `main` y `main1` que puedes utilizar para probar tu implementación.

Bloque Concurrencia (6.5 puntos)

En alguna parte del Parque Nacional Kruger en Sudáfrica hay un cañón profundo y un cuerda colgante que lo atraviesa. Los monos babuinos pueden cruzar el cañón colgándose de la cuerda utilizando sus manos, pero si dos babuinos cruzan en sentido contrario a la vez y se encuentran en mitad de la cuerda se pelearán, se caerán de la cuerda y morirán. Además, la cuerda es sólo lo suficientemente fuerte como para sostener a 3 babuinos. Si hay más babuinos en la cuerda se romperá.

Por lo tanto, una implementación de este sistema tiene que satisfacer las dos siguientes condiciones de sincronización:

CS1- Un babuino que quiere cruzar el cañón utilizando la cuerda no puede hacerlo si hay babuinos utilizándola en el sentido contrario.

CS2- Un babuino que quiere cruzar el cañón utilizando la cuerda no puede hacerlo si hay 3 babuinos utilizándola en ese momento.

El esqueleto para resolver el ejercicio se encuentra en el campus virtual. Hay dos clases, BabuinoNS y BabuinoSN, que representan a los procesos babuinos que quieren cruzan el cañón en dirección NorteSur/SurNorte, respectivamente. La cuerda está representada por un objeto de la clase Cuerda que proporciona cuatro métodos:

public void entraDireccionNS(**int** id) **throws** InterruptedException

public void entraDireccionSN(**int** id) **throws** InterruptedException

public void saleDireccionNS(**int** id) **throws** InterruptedException

public void saleDireccionSN(**int** id) **throws** InterruptedException

Para simplificar el ejercicio, no hay que tener en cuenta el orden de entrada y de salida de la cuerda, es decir, pueden adelantarse en la cuerda y salir antes de lo que les toca.

Semáforos (3.5 puntos): Implementa la sincronización de los métodos de la clase Cuerda utilizando **semáforos binarios**.

Monitores (3 puntos): Implementa la sincronización de los métodos de la clase Cuerda utilizando **métodos sincronizados o locks**.

Nota: En ambas casos, *se valorará positivamente* que las soluciones sean justas, es decir, no sea posible que una fila continua de babuinos que cruzan en una dirección impida que los de la dirección contraria puedan cruzar.