PRÁCTICAS DE SISTEMAS OPERATIVOS II

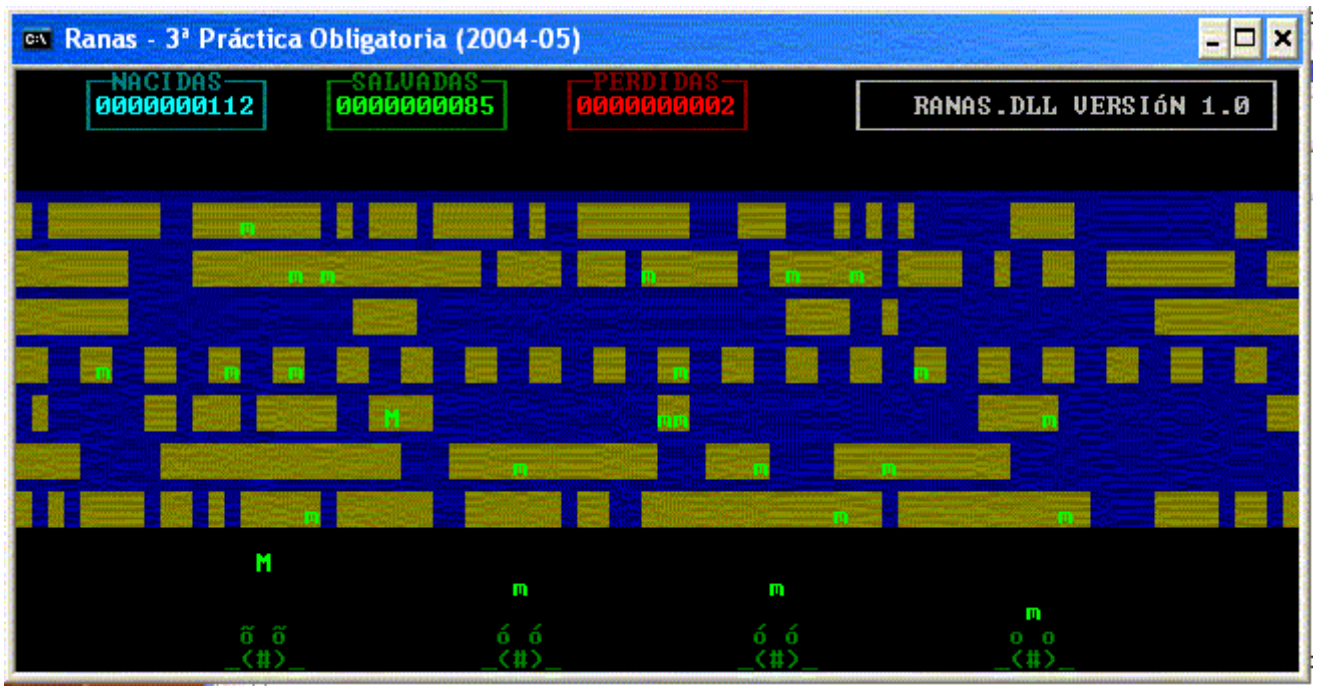
SEGUNDA PRÁCTICA EVALUABLE

Ranas

---

1. Enunciado.

El programa propuesto constará de un único fichero fuente, `ranas.cpp`, cuya adecuada compilación producirá el ejecutable `ranas.exe`. Se trata de simular mediante un programa que realice llamadas a la API de WIN32 la vida de unas ranas, inspirada en el famoso juego de consola clásico "Frogger". Según se va ejecutando el programa, se ha de ver una imagen parecida a la siguiente:



En la imagen pueden observarse las ranas, representadas por una "m" de color verde. Las ranas nacen de una de cuatro ranas madre de color verde oscuro y situadas en la parte inferior de la pantalla. El objetivo de sus vidas es atravesar un río. Para ello, pueden moverse hacia adelante, hacia la derecha o hacia la izquierda según su criterio, pero nunca hacia atrás. Deberán mantenerse dentro de la pantalla. Dos ranas no pueden ocupar la misma posición.

Cuando llegan las ranas a la orilla inferior del río, deben tener cuidado. Solo pueden saltar a posiciones donde se encuentran unos troncos flotando sobre su superficie. Si saltan sobre el agua, se produce un error. Para aumentar la dificultad, los troncos están a la deriva moviéndose continuamente. Una rana puede desaparecer debido a que el tronco sobre el que se encuentra sale de la pantalla. Eso no constituye un error. Esa rana se ha "perdido".

En la parte superior de la pantalla aparecen tres contadores. El primero cuenta las ranas que han nacido. El segundo, las ranas que han alcanzado la orilla superior del río y se han puesto a salvo y el tercero, lleva cuenta de aquéllas que se han perdido porque su tronco desapareció de la pantalla. Al final de la práctica, el número de ranas nacidas debe coincidir con el de ranas salvadas más el de ranas perdidas más el de ranas que permanecen en la pantalla.

El tiempo de la práctica se mide en "tics" de reloj. La equivalencia entre un tic y el tiempo real es configurable. Puede ser incluso cero. En ese caso la práctica irá a la máxima velocidad que le permita el ordenador donde se esté ejecutando.

`ranas.exe` acepta exactamente dos argumentos en la la línea de órdenes. Si no se introducen argumentos, se imprimirá un mensaje con la forma de uso del programa por el canal de error estándar. El primer argumento será un número entero comprendido entre 0 y 1000 que indicará la equivalencia en ms de tiempo real de un tic de reloj. O dicho de otro modo, la "lentitud" con que funcionará la práctica. El segundo argumento es la media de tics de reloj que necesita una rana madre descansar entre dos partos. Es un número entero estrictamente mayor que 0.

Para facilitar la tarea, tenéis a vuestra disposición una biblioteca de funciones de enlazado dinámico `ranas.dll` y un fichero de cabeceras, `ranas.h`. Gracias a la biblioteca, muchas de las funciones no las tendréis que programar sino que invocarlas a la DLL, tal como se explica en la sesión novena. La biblioteca creará cuatro hilos adicionales a los vuestros para su funcionamiento interno, uno para cada rana madre, de los cuales no tendréis que ocuparos. Una descripción detallada de las funciones de la biblioteca aparece más abajo en esta misma página.

La práctica se ejecutará durante 30 segundos. El programa, desde vuestro punto de vista, se simplifica bastante. Se ha de llamar a la función `InicioRanas` de la DLL. El hilo creará los hilos que necesite para otras tareas (como mover los troncos del río, por ejemplo) y dormirá 30 segundos. Invocará a la función `FinRanas`, para hacer que las ranas dejen de parir. Cuando la función `FinRanas` retorne, el programa se debe encargar de que el resto de hilos que estén en ejecución moviendo ranas acaben. Se llamará a continuación a la función `ComprobarEstadIsticas` (nótese la I mayúscula) para que se contrasten las estadísticas que habéis tomado con las calculadas por la DLL. Ahí acabará el programa. Para esperar la muerte de los hilos, podéis hacer una pausa sin consumo de CPU de unos cuantos segundos, aunque se puntuará favorablemente un esquema mejor. Además de todo esto, se programará una función que será llamada por la DLL cada vez que nazca una nueva rana. Dicha *función de rellamada* es registrada en la función `InicioRana` de la DLL.

El prototipo de la función de rellamada se describe aquí:

- **`void f_Criar(int pos)`**
  La DLL llamará a esta función cuando nazca una nueva rana. El código que programéis para esta función, aparte de las necesarias sincronizaciones, consistirá en llamar a la función `PartoRanas` de la DLL, actualizar las estadísticas y crear un nuevo hilo de ejecución que se encargue de mover a la recién nacida. El parámetro `pos` de la función indica la posición 0, 1, 2 ó 3 donde va a nacer la ranita, siendo 0 la madre de más a la izquierda y, numeradas sucesivamente, hacia la derecha.

La posición de la rana en la pantalla se determina mediante dos coordenadas x e y. La x, de izquierda a derecha toma valores desde 0 a 79. La y crece desde la parte inferior de la pantalla a la superior y de 0 a 11. Cuando y está entre 0 y 3, se encuentra antes de cruzar el río. El valor 11 corresponde con la orilla opuesta y los siete valores restantes dan cuenta de las siete filas de troncos flotantes.

Características adicionales que programar

- El programa no debe consumir CPU apreciablemente en los modos de retardo mayor o igual que 1. Para comprobar el consumo de CPU, podéis arrancar el administrador de tareas de Windows, mediante la pulsación de las teclas CTRL+ALT+SUPR. Observad, no obstante, que en las aulas de informática puede que esta opción esté deshabilitada.
- **IMPORTANTE**: Aunque no se indique explícitamente en el enunciado, parece obvio que se necesitarán objetos de sincronización en diferentes partes del programa.

Biblioteca `ranas.dll`

Con estra práctica se trata de que aprendáis a sincronizar y comunicar hilos en WIN32. Su objetivo no es la programación. Es por ello que se os suministra una biblioteca dinámica de funciones ya programadas para tratar de que no tengáis que preocuparos por la presentación por pantalla, la gestión de estructuras de datos (colas, pilas, ...) , etc. También servirá para que se detecten de un modo automático errores que se produzcan en vuestro código. Para que vuestro programa funcione, necesitáis la biblioteca `ranas.dll` y el fichero de cabeceras `ranas.h`.

Ficheros necesarios:

- `ranas.dll` ver. 1.2: Descárgalo de aquí.
- `ranas.h`: Descárgalo de aquí.

Las funciones que la biblioteca exporta para que las uséis se muestran a continuación. Si una función devuelve un `BOOL`, un valor de TRUE indica que no ha habido errores y FALSE, lo contrario:

- **`BOOL InicioRanas(int delta_t, int lTroncos[],int lAguas[],int dirs[],int t_Criar, TIPO_CRIAR f_Criar)`**
  El hilo principal debe llamar a esta función cuando desee que la simulación comience. Los argumentos son:
  1. `delta_t`: valor del tic de reloj en milisegundos.
  2. `lTroncos`: array de siete enteros que contiene el valor de la longitud media de los troncos para cada fila. El índice cero del array se refiere a la fila superior de troncos.
  3. `lAguas`: igual que el parámetro anterior, pero referido a la longitud media del espacio entre troncos.
  4. `dirs`: lo mismo que los dos parámetros anteriores, pero en esta ocasión cada elemento puede valer `DERECHA`(0) o `IZQUIERDA`(1), indicando la dirección en que se moverán los troncos.
  5. `t_Criar`: tiempo de reposo entre dos partos, expresado en tics.
  6. `f_Criar`: puntero a la función de rellamada que será invocada cada vez que una rana madre deba reproducirse. El prototipo de la función tiene que ser de modo que devuelva `void` y reciba un argumento entero, la posición de la rana madre que debe parir.
- **`void Pausa(void)`**
  Hace una pausa, sin consumo de CPU, equivalente a un tic.
- **`void PrintMsg(char *temp)`**
  Imprime el mensaje `temp` en la línea de errores de la aplicación. Sirve para depurar.
- **`BOOL PartoRanas(int i)`**
  Genera una nueva rana delante de la rana madre número i. La rana aparecerá en las coordenadas (15+16i,0).
- **`BOOL PuedoSaltar(int x, int y, int dir)`**
  La función devuelve si una rana situada en la posición (x,y) cumple todas las condiciones para poder avanzar en la dirección `dir`. Los valores posibles para `dir` son los mismos que para `AvanceRana`.
- **`BOOL AvanceRanaIni(int x, int y)`**
  Inicio del avance de una rana situada en (x,y).
- **`BOOL AvanceRana(int *x, int *y, int dir)`**
  Avance efectivo de la rana situada en (x,y). `dir` puede valer `DERECHA`(0), `IZQUIERDA`(1) o `ARRIBA`(2). Las variables `x` e `y` son modificadas por la función y reflejan la posición de la rana después del avance.
- **`BOOL AvanceRanaFin(int x, int y)`**
  Fin del avance de una rana situada ya en su posición de destino (x,y).
- **`BOOL AvanceTroncos(int i)`**
  Avanza una posición la fila de troncos número `i`. Las ranas que hubiera sobre la fila se mueven con ella.
- **`BOOL FinRanas(void)`**
  Hace que las ranas madres dejen de parir más ranitas.
- **`BOOL ComprobarEstadIsticas(LONG lRNacidas, LONG lRSalvadas, LONG lRPerdidas)`**
  Se debe llamar a esta función después de haber llamado a `FinRanas` y cuando hayáis acabado con todos los hilos que mueven las ranas. Se le ha de pasar el número de ranas nacidas, salvadas y perdidas que vuestro programa ha contado para ver si las cuentas coinciden con las que ha realizado la DLL.

Notas acerca de las funciones de la DLL:

1. Se puede establecer la longitud media de troncos y su separación de todas las filas salvo la del medio. Independientemente del valor espedificado, esta fila presenta una sucesión de troncos de tamaño dos separados también una distancia de dos caracteres.
2. La secuencia para que una rana se mueva consiste en llamar primero a `AvanceRanaIni`, luego a `AvanceRana`, ambas con la posición donde se encuentra la rana. Al retornar AvanceRana, en las variables de posición ya se encontrará la nueva posición de la rana. Hay que llamar a la función `Pausa` y, finalmente, a `AvanceRanaFin`, con la nueva posición de la rana.

Sincronización interna de la DLL

La sincronización interna de la DLL está basada en un *mutex*. El esquema de sincronización interna de las funciones que la tienen es el siguiente:

```
Mutex Mu;
Mu=1;
[...]

PrintMsg
========
Wait(Mu);
Imprimir mensaje
ReleaseMutex(Mu);

PartoRanas
```

```
==========
Comprobación de parámetros
Wait(Mu); Animación; ReleaseMutex(Mu); Pausa(); [[ varias veces ]]
Wait(Mu);
si la posición de parto está ocupada
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
Pintar la nueva rana;
Calcular estadísticas;
ReleaseMutex(Mu);
return TRUE;

AvanceRanaIni
=============
Comprobación de parámetros
Wait(Mu);
si no hay rana en esa posición
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
Actualizar el dibujo de la ranita;
ReleaseMutex(Mu);
return TRUE;

AvanceRana
==========
Comprobación de parámetros
Wait(Mu);
si hay rana en el destino
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
si no hay rana en el origen
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
Borrar la rana del punto de origen
si hemos caído al agua
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
Dibujar a la rana en la nueva posición
ReleaseMutex(Mu);
Actualizar las variables de posición de la rana
return TRUE;

AvanceRanaFin
=============
Comprobación de parámetros
Wait(Mu);
si no hay rana en esa posición
    poner error; ReleaseMutex(Mu); return FALSE;
fin_si
Actualizar el dibujo de la ranita;
ReleaseMutex(Mu);
si la posición vertical de la rana es 11
    Wait(Mu);
    Borrar la rana
    Actualizar estadísticas
    ReleaseMutex(Mu);
fin_si
return TRUE;

AvanceTroncos
=============
Comprobación de parámetros
Wait(Mu);
Mover troncos
ReleaseMutex(Mu);
return TRUE;
```

2. Pasos recomendados para la realización de la práctica

En esta práctica, no os indicaremos los pasos que podéis seguir. El proceso de aprendizaje es duro, y ya llega el momento en que debéis andar vuestros propios pasos sin ayuda, aunque exista la posibilidad de caerse al principio.

3. Plazo de presentación.

Consúltese la página de entrada de la asignatura.

4. Normas de presentación.

Acá están. Además de estas normas, en esta práctica se debe entregar un esquema donde aparezcan los mecanismos de sincronización usados, sus valores iniciales y un seudocódigo sencillo para cada hilo con las operaciones realizadas sobre ellos. Por ejemplo, si se tratara de sincronizar con eventos dos hilos C y V para que produjeran alternativamente consonantes y vocales, comenzando por una consonante, deberíais entregar algo parecido a esto:

```
EVENTOS Y VALOR INICIAL: EC* (automático), EV (automático).

SEUDOCÓDIGO:

       C                              V
      ===                            ===
  Por_siempre_jamás              Por _siempre_jamás
     {                              {
      W(EC)                          W(EV)
      escribir_consonante            escribir_vocal
      Set(EV)                        Set(EC)
      }                              }
```

Debéis indicar, asimismo, en el caso de que las hayáis realizado, las optimizaciones de código realizadas.

5. Evaluación de la práctica.

Dada la dificultad para la corrección de programación en paralelo, el criterio que se seguirá para la evaluación de la práctica será: si
a. la práctica cumple las especificaciones de este enunciado y,
b. la práctica no falla en ninguna de las ejecuciones a las que se somete y,
c. no se descubre en la práctica ningún fallo de construcción que pudiera hacerla fallar, por muy remota que sea esa posibilidad...

se aplicará el principio de "presunción de inocencia" y la práctica estará aprobada. La nota, a partir de ahí, dependerá de la simplicidad de las técnicas de sincronización usadas, de las optimizaciones realizadas para producir mejores resultados, etc.
A diferencia de la práctica anterior, para aprobar esta, es necesario que se muevan los troncos, además de lo indicado arriba. En esta práctica no hay limitación en el número de hilos que puede haber en ejecución simultánea.

6. LPEs.

I. No debéis usar la función `TerminateThread` para acabar con los hilos. El problema de esta función es que está diseñada para ser usada sólo en condiciones excepcionales y el hilo muere abruptamente. Puede dejar estructuras colgando e ir llenando la memoria virtual del proceso con basura.

II. Cuando ejecuto la práctica depurando me aparece un error cualquiera en una casilla. Cuando examino la casilla, compruebo que todo es correcto. ¿A qué puede ser debido?

    Este mensaje u otros similares que parecen estar equivocados tienen su razón en que estáis "manchando" la pantalla con mensajes propios o de depuración. Para evitar este tipo de mensajes, mejor depurad la práctica enviando la información de trazado a un fichero añadiendo al final de la línea de órdenes `2>salida`. De este modo, toda la información aparecerá en el fichero `salida` y no confundiréis a la DLL.

III. Tengo muchos problemas a la hora de llamar a la función `XXXX` de la biblioteca. No consigo de ningún modo acceder a ella.

    El proceso detallado viene en la última sesión. De todos modos, soléis tener problemas en una conversión de tipos, aunque no os deis cuenta de ello. No voy a deciros qué es lo que tenéis que poner para que funcione, pues lo pondríais y no aprenderíais nada. Sin embargo y dada la cantidad de personas con problemas, aquí viene una pequeña guía:
    1. Primero debéis definir una variable puntero a función. El nombre de la variable es irrelevante, pero podemos llamarle `XXXX` por lo que veremos más abajo. Para definir el tipo de esta variable correctamente, debéis conocer cómo son los punteros a función. En la última sesión de Sistemas Operativos I, se describe una función, `atexit`. Dicha función en sí no es importante para lo que nos traemos entre manos, pero sí el argumento que tiene. Ese argumento es un puntero a función. Fijándoos en ese argumento, no os resultará difícil generalizarlo para poner un puntero a funciones que admiten otro tipo de parámetros y devuelve otra cosa. Notad, además, que, al contrario que ocurre con las variables "normales", la definición de una variable puntero a función es especial por cuanto su definición no va sólo antes del nombre de la variable, sino que lo rodea. Tenéis que poner algo similar a: `#$%&%$ XXXX $%&$@;`, es decir, algo por delante y algo por detrás.
    2. Después de cargar la biblioteca como dice en la última sesión, debéis dar valor al puntero de función. Dicho valor lo va a proporcionar `GetProcAddress`. Pero, ¡cuidado!, `GetProcAddress` devuelve un `FARPROC`, que sólo funciona con punteros a funciones que devuelven `int` y no se les pasa nada `(void)`. Debéis hacer el correspondiente *casting*. Para ello, de la definición de vuestro puntero, quitáis el nombre, lo ponéis todo entre paréntesis y lo añadís delante de `GetProcAddress`, como siempre.
    3. Ya podéis llamar a la función como si de una función normal se tratara. Ponéis el nombre del puntero y los argumentos entre paréntesis. Como os advertí más arriba, si habéis puesto `XXXX` como nombre al puntero, ahora no se diferenciarán en nada vuestras llamadas a la función respecto a si dicha función no perteneciera a una DLL y la hubierais programado vosotros.

IV. Os puede dar errores en el fichero de cabecera `.h` si llamáis a vuestro fichero fuente con extensión `.c`. Llamadlo siempre con extensión `.cpp`.

V. Tened mucho cuidado si usáis funciones de memoria dinámicas de `libc` (`malloc` y `free`). Son funciones que no *están sincronizadas*, es decir, no se comportan bien en entornos multihilo. O bien las metéis en una sección crítica o, mejor aún, tratad de evitarlas.

VI. Para simplificar la realización de la práctica aún más, podéis hacer "semiespera ocupada" en el movimiento de las ranas. Si una rana piensa moverse a una posición y no puede, puede llamar a la función `Pausa` y reintentarlo después. Evidentemente, es mejor un esquema puro de sincronización sin consumo de CPU, pero se permite éste. Donde no se permite semiespera es en el parto de las ranas. Si la posición de parto está ocupada, hay que bloquearse, sin consumo de CPU, hasta que esta posición se libere.

VII. Pasos para que no dé problemas en Visual Studio 2008:
    1. Es importante que seleccionéis, al crear un proyecto nuevo, "Aplicación de consola" de Win32
    2. Dejad la función main tal cual la pone él: `int _tmain(int argc, _TCHAR* argv[])`
    3. En las propiedades del Proyecto, propiedades de configuración, general, a la derecha, poned "Juego de caracteres", sin establecer.

    Probad un "Hola, mundo" y, si no os da errores, los errores que aparezcan a continuación podéis confiar que son de que no estáis haciendo bien el acceso a la DLL. En general, o que no declaráis bien los punteros para acceder a las funciones o que no hacéis el correspondiente *casting* cuando llamáis a `GetProAddress`.

VIII. Tenéis que incluir el fichero de cabeceras `windows.h` antes que el fichero de cabeceras de la DLL
