# PRÁCTICAS DE SISTEMAS OPERATIVOS II

## SEGUNDA PRÁCTICA EVALUABLE

## Pistoleros de Wichita

---

1. **Enunciado.**

   El programa propuesto constará de un único fichero fuente, `pist3.cpp`, cuya adecuada compilación producirá el ejecutable `pist3.exe`. Por favor, mantened los nombres tal cual, incluida la extensión. Se trata de realizar una práctica casi idéntica a la práctica anterior de este año, pero mediante un programa que realice llamadas a la API de WIN32.

   Las principales diferencias respecto a la práctica segunda son:

   - Se proporcionará una *Biblioteca de Enlazado Dinámico* (DLL) en lugar de la biblioteca estática `libpist.a`. La biblioteca se llamará `pist3.dll`.
   - Desaparece el buzón de mensajes. Se deja a los autores el modo de comunicar la muerte a los hilos pistoleros. También varían los prototipos de algunas de las otras funciones.
   - Se realizará la práctica mediante hilos, uno para cada pistolero que participe en la partida.
   - En lugar del array de semáforos se usarán los semáforos independientes u otros mecanismos de sincronización de WIN32 que se estimen convenientes.

   Debéis manejar la biblioteca de enlazado dinámico tal como se explica en la sesión novena. Disponéis asimismo del fichero de cabeceras `pist3.h`. Las funciones proporcionadas por la DLL que devuelven un entero, devuelven -1 en caso de error. A continuación se ofrecen los prototipos de todas las funciones incluidas en la DLL:

   - `int PIST_inicio(unsigned int nPistoleros, int ret, int semilla)`
     La misma función que en `libpist.a`, pero con algún parámetro menos.
   - `int PIST_nuevoPistolero(char pist)`
     La llama un hilo cuando nace, indicando su letra identificadora.

- `char PIST_vIctima(void)`
  Devuelve la letra identificadora del pistolero al que se debe matar en la ronda.
- `int PIST_disparar(char pist)`
  El hilo que la invoca mata al pistolero cuya letra identificadora se pasa como parámetro.
- `void PIST_morirme(void)`
  Al igual que en el caso de la práctica de UNIX, un pistolero, después de que todos hayan disparado, invoca a esta función si recibió algún disparo.
- `int PIST_fin(void)`
- `void pon_error(char *mensaje)`

Estad atentos pues pueden ir saliendo versiones nuevas de la biblioteca para corregir errores o dotarla de nuevas funciones. La sincronización interna de la biblioteca sigue los mismos esquemas expuestos en la práctica segunda.

**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 `pist3.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 `pist3.dll` y el fichero de cabeceras `pist3.h`.

Esta biblioteca, aunque es para Windows, ha sido croscompilada desde Linux. Es por ello que puede dar algún problema al inicio y sea

necesario actualizarla con nuevas versiones. Estad atentos por si eso ocurre a este apartado.

**Ficheros necesarios:**

- `pist3.dll` ver. 1.0: Descárgalo de aquí.
- `pist3.h` ver. 1.0: Descárgalo de aquí.

## 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 programado, 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

. la práctica cumple las especificaciones de este enunciado y,
a. la práctica no falla en ninguna de las ejecuciones a las que se somete y,
b. 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.

**6. LPEs.**

. No debéis usar la función `TerminateThread` para acabar con los hilos o `TerminateProcess` para acabar con los procesos. El problema de estas funciones es que están diseñada para ser usada sólo en condiciones excepcionales y los hilos mueren abruptamente. Puede dejar estructuras colgando, ir llenando la memoria virtual del proceso con basura o no invocar adecuadamente las funciones de descarga de la DLL.

I. Al ejecutar la práctica, no puedo ver lo que pasa, porque la ventana se cierra justo al acabar.

Para evitar esto, ejecutad la práctica desde el "Símbolo del sistema", que se encuentra en el menú de "Accesorios". **También es necesario que la ventana que uséis tenga un tamaño de 80x25 caracteres. Si no lo tenéis así, cambiadlo en el menú de propiedades de la ventana.**

II. Al ejecutar la función `LoadLibrary`, en lugar de aparecer la pantalla de presentación, aparece un mensaje que pone "`En DllMain`".

**Es necesario que la ventana que uséis tenga un tamaño de 80x25 caracteres. Si no lo tenéis así, cambiadlo en el menú de propiedades de la ventana.**

III. Cuando ejecuto la práctica depurando la pantalla se emborrona. ¿Cómo lo puedo arreglar?

Mejor depurad la práctica enviando la información de trazado escrita con `fprintf(stderr,...)` a un fichero, añadiendo al final

de la línea de órdenes `2>salida`. De este modo, toda la información aparecerá en el fichero `salida` para su análisis posterior. No os olvidéis de incluir el identificador del hilo que escribe el mensaje.

IV. 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 vamos 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:

0. 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 solo 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.
1. 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.
2. 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.

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

VI. 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.

VII. En algunas versiones de Visual Studio os puede dar un error del tipo: `error XXXXX: 'FuncionW': no se puede convertir de 'const char[X]' a 'LPCWSTR'`. El motivo del error es que, por defecto, esa versión de Visual Studio supone que deseáis usar UNICODE (caracteres de 16 bits) en lugar de los normales (caracteres de 8 bits). La solución pasa por transformar el código fuente para que se ajuste a la programación en UNICODE de Microsoft o decirle a Visual Studio que no, que no queréis trabajar con UNICODE. Unos compañeros vuestros nos escriben diciendo que si en la configuración del proyecto seleccionáis "Juego de Caracteres->Sin establecer", se soluciona.

---