



**Departamento de Lenguajes y Ciencias de la
Computación Universidad de Málaga**

El Sistema Operativo Windows NT

TEMA 2:

Programación con
Win32



E.T.S.I. Informática

**Antonio Jesús Nebro Urbaneja
Manuel Díaz Rodríguez**

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs
E/S Asíncrona
Bibliografía

Programación con Win32

Índice de contenidos

1. Introducción
2. Sistema de Ficheros y Entrada/Salida
3. Excepciones
4. Gestión de Memoria
5. Seguridad
6. Gestión de Procesos y Hebras
7. DLLs
8. Entrada/Salida Asíncrona
9. Bibliografía

on Win32

Proceso

El Sistema Ope

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad

Procesos y Hebras

Conceptos Básicos
Comunic. Entre Procesos
Hebras y Planificación
Sincronización

DLLs
E/S Asíncrona
Bibliografía

Programación con Win32

Procesos y Hebras

- Un Proceso en Win32
 - Contiene su propio espacio virtual de direcciones
 - Es distinto para cada proceso
 - Los ficheros mapeados en memoria son accedidos también desde direcciones virtuales distintas
 - Contiene una o más hebras (*threads*) de ejecución
 - La hebra es la unidad básica de ejecución
 - Las hebras se planifican siguiendo los criterios clásicos
 - disponibilidad de recursos
 - prioridades
 - justicia
 -

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Introducción

- Los procesos además contienen
 - Uno o varios segmentos de código
 - Uno o varios segmentos de datos para las variables globales
 - Información del entorno de ejecución
 - Cadenas de caracteres con información variable
 - path
 - parámetros del entorno
 -
 - El heap del proceso
 - Recursos
 - Manejadores (handles) abiertos
 - Otros heaps
 -

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Introducción

- Cada hebra de un proceso comparte
 - El código
 - Las variables globales
 - El entorno
 - Los recursos
- Cada hebra es planificada de forma independiente y contiene
 - Su propia pila
 - Zona de almacenamiento locas (TLS)
 - Es un array de punteros que permite a cada hebra crear su propio entorno de datos
 - Un argumento en la pila de la hebra padre
 - Una estructura de contexto, mantenida por el kernel

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

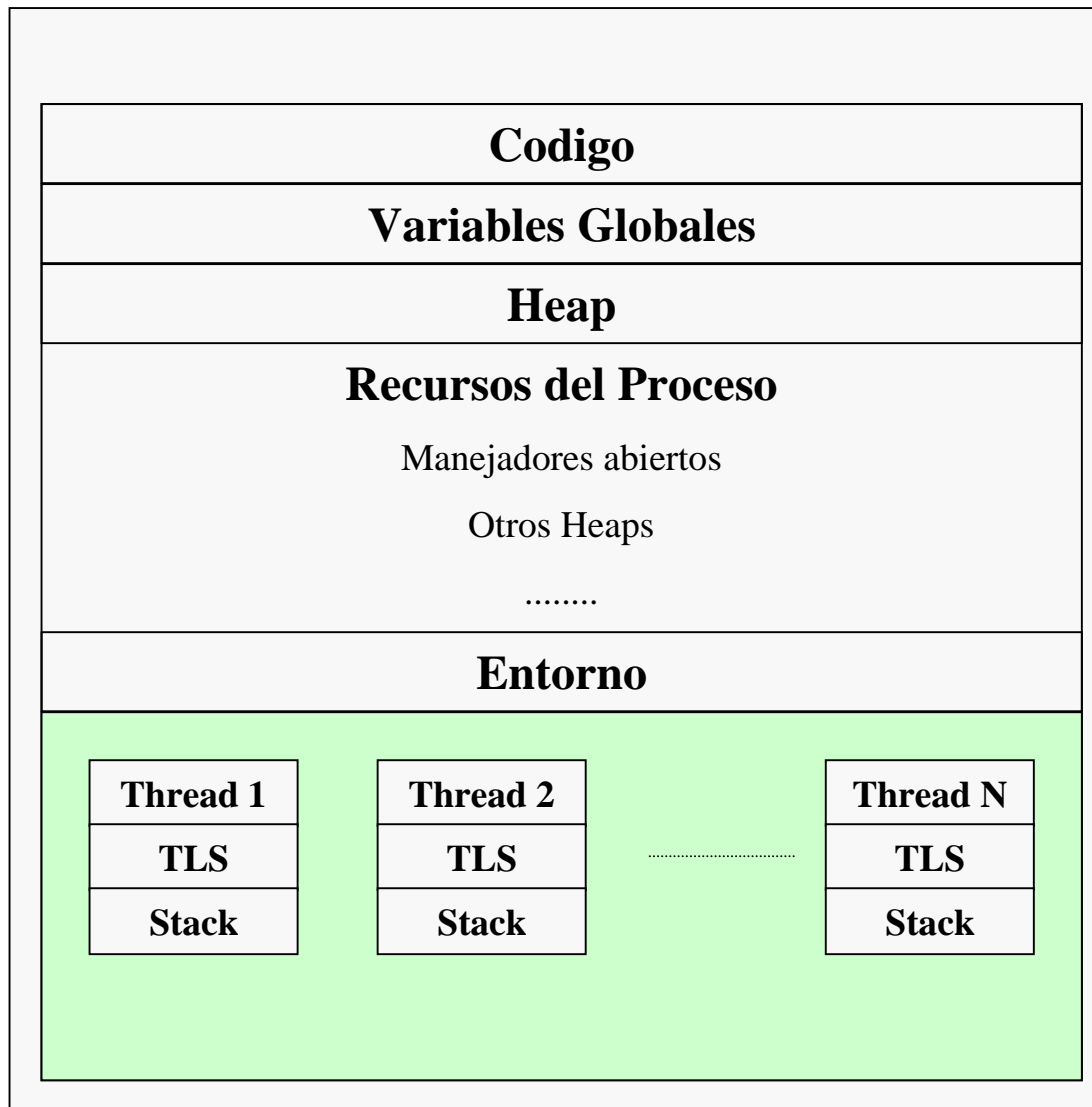
DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Estructura de un Proceso



Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Creación de Procesos

- La función fundamental es la de creación de procesos
 - Crea un proceso con una sola hebra de ejecución
 - Requiere el nombre de un ejecutable
 - No existe la relación padre/hijo entre procesos

```
BOOL CreateProcess (  
    LPCTSTR lpszImageName,           // nombre del ejecutable  
    LPTSTR lpszCommandLine,         // línea de comandos  
    LPSECURITY_ATTRIBUTES lpsaPro,  // atr. Seguridad proceso  
    LPSECURITY_ATTRIBUTES lpsaThr,  // atr. Seguridad Hebra  
    BOOL finherithandle,             // herencia manejadores  
    DWORD fdwCreate,                 // modo de creación  
    LPVOID lpenvironment,           // nuevo entorno  
    LPTSTR lpszCurDir,              // directorio inicial  
    LPSTARTUPINFO lpsiStartInfo,    // ventana principal  
    LPPROCESS_INFORMATION lppiProcInfo // estructura proc.  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Creación de Procesos

- No devuelve un handle, sino un valor booleano indicando si el proceso se ha creado correctamente
- Devuelve dos manejadores
 - Uno para el proceso
 - Otro para la hebra
- Estos manejadores se encuentran en la estructura apuntada por `lppiProcInfo`

```
typedef struct PROCESS_INFORMATION {  
    HANDLE hProcess;           // manejador del proceso  
    HANDLE hThread;           // manejador de la hebra  
    DWORD dwProcessId;        // identificador del proceso  
    DWORD dwThreadId;         // identificador de la hebra  
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Creación de Procesos

- ¿Por qué necesitamos manejadores e identificadores?
 - Los identificadores son únicos para cada objeto durante toda su vida y en todos los procesos
 - Sin embargo,
 - Un proceso o hebra puede tener distintos manejadores con distintos atributos
 - Los manejadores son necesarios para las funciones generales basadas en manejadores (WaitForSingleObject,....)
- Otros parámetros importantes:
 - fdwCreate: Combina distintos flags como:
 - CREATE_SUSPENDED
 - DETACHED_PROCESS y CREATE_NEW_CONSOLE
 - CREATE_NEW_PROCESS_GROUP

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Creación de Procesos

- Otros parámetros importantes (cont.):
 - Imagen ejecutable y línea de comandos
 - ***lpzImageName*** si no es NULL indica el nombre del ejecutable
 - Se puede indicar el path completo o en caso contrario hacer referencia al directorio actual
 - Hay que indicar la extensión (.exe o .bat)
 - Si es NULL entonces el ejecutable será el primer token delimitado por un espacio de ***lpzCommandLine***. Si el nombre no contine el path completo se sigue la siguiente secuencia:
 - El directorio de la imagen de proceso actual
 - El directorio actual
 - El directorio de sistema de Windows (GetSystemDirectory)
 - Los directorios especificados en la variable de entorno ***path***
 - El nuevo proceso puede obtenerla línea de comandos utilizando el mecanismo habitual en C o con la función ***GetCommandLine***.

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos
Hebras y Planificación
Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Creación de Procesos

- Otros parámetros importantes (cont.):
 - Manejadores heredables
 - Frecuentemente el proceso creado requiere los manejadores del proceso padre
 - Por ejemplo, los manejadores de entrada/salida estándar
 - El flag **flnheritHandles** determina si se heredan o no
 - Sólo se heredan los que han sido creados con la opción **binheritHandle** en la estructura de seguridad del manejador
 - El proceso creado necesita conocer estos manejadores. Esto se puede hacer:
 - Con los mecanismos de comunicación normales
 - Utilizando la estructura STARTUPINFO
 - Pasando el valor convertido a texto en una línea de comandos o en una variable de entorno
 - Los manejadores son distintos, por lo que varios procesos pueden estar accediendo al mismo objeto

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Identificación de Procesos

- Un proceso puede obtener el identificador y el manejador de un proceso de distintas formas:
 - Si creas un proceso, a partir de la estructura `PROCESS_INFORMATION`
 - Existen también dos funciones para obtener la identidad del proceso en ejecución

```
HANDLE GetCurrentProcess (VOID)
```

```
DWORD GetCurrentProcessId (VOID)
```

- El manejador es un pseudomanejador y no se puede heredar
- Para obtener un manejador real es necesario usar el identificador de proceso

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos
Hebras y Planificación
Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Identificación y Acceso a Procesos

- Utilizando el manejador de proceso

```
HANDLE OpenProcess (  
    DWORD fdwAccess,           //tipo de acceso al proceso  
    BOOL  finherit,           //heredable o no  
    DWORD IDProcess           //identificador del proceso  
)
```

- El tipo de acceso puede ser:
 - **SYNCHRONIZE** permite esperar a la terminación del proceso
 - **PROCESS_ALL_ACCESS** permite todas las operaciones
 - **PROCESS_TERMINATE** permite terminar el proceso (**TerminateProcess**)
 - **PROCESS_QUERY_INFORMATION** sólo permite consultar información (**GetPriorityClass** y **GetExitCodeProcess**)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Duplicación de Manejadores

- Se utiliza cuando se necesita un manejador real (por ejemplo para un proceso hijo).

```
BOOL DuplicateHandle (  
    HANDLE hSourceProcess, //manejador proceso origen  
    HANDLE hSource,        //manejador origen  
    HANDLE hTargetProcess, //manejador proceso destino  
    LPHANDLE lphTarget,    //nuevo manejador  
    BOOL finherit,         //heredable o no  
    DWORD fdwOptions       //modo  
)
```

- **fdwAccess** puede tener dos valores distintos
 - **DUPLICATE_CLOSE_SOURCE** hace que se cierre el manejador origen
 - **DUPLICATE_SAME_ACCESS** hace que **fdwAccess** sea ignorado

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos
Hebras y Planificación
Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Salida y Terminación de Procesos

- Un proceso finaliza su ejecución correctamente llamando a

```
VOID ExitProcess (  
    UINT nExitProces //código de salida  
)
```

- Hace terminar al proceso y todas sus hebras
- Otro proceso puede obtener el código de salida con

```
BOOL GetExitCodeProcess (  
    HANDLE hProcess, // manejador proceso  
    LPDWORD lpdwExitCode // código de salida  
)
```

- Un posible valor puede ser **STILL_ACTIVE**
- Un proceso puede cancelar la ejecución de otro

```
BOOL TerminateProcess (  
    HANDLE hProcess, // manejador proceso  
    UINT uExitCode // código de salida  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Espera a la Terminación de un Proceso

- Las funciones de espera son generales, no sólo sirven para procesos
 - Se puede esperar por
 - Un solo proceso
 - El primero de varios especificados
 - Un conjunto de procesos
 - Existe un time-out opcional

```
DWORD WaitForSingleObject (  
    HANDLE hObject,           // manejador proceso  
    DWORD dwTimeOut          // timeout  
)
```

```
BOOL WaitForMultipleObjects (  
    DWORD cObjects,           // numero de objetos  
    LPHANDLE lphObject,      // array de procesos  
    BOOL fWaitAll,           // esperar a todos?  
    DWORD dwTimeOut          // timeout  
)
```

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos
Hebras y Planificación
Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Espera a la Terminación de un Proceso

- Argumentos
 - El número máximo de objetos por los que se puede esperar es **MAXIMUM_WAIT_OBJECTS**
 - dwTimeOut puede ser
 - cero (retorna inmediatamente, es equivalente a un polling)
 - **INFINITE** espera a que termine
 - otro valor en milisegundos
 - El valor de retorno en el caso múltiple puede ser:
 - **WAIT_OBJECT_0** acaba un solo proceso (fwaitAll TRUE)
 - **WAIT_OBJECT_0+n**, con $0 \leq n \leq cObjects$, indica el proceso que ha terminado o el menor si han acabado varios
 - **WAIT_TIMEOUT** ha pasado el tiempo indicado

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Tiempo de Ejecución de un Proceso

```
// Crea un proceso y calcula su tiempo de ejecución
// timep ejecutable

#include <windows.h>
#include <stdio.h>

LPTSTR SkipArg (LPCTSTR);

int main (int argc, LPTSTR argv [])
{
    STARTUPINFO StartUp;
    PROCESS_INFORMATION ProcInfo;
    union { /* estructura para aritmetica con tiempo. */
        LONGLONG li;
        FILETIME ft;
    } CreateTime, ExitTime, ElapsedTime;

    FILETIME KernelTime, UserTime;
    SYSTEMTIME ElTiSys, KeTiSys, UsTiSys, StartTimeSys, ExitTimeSys;
    LPTSTR targv = GetCommandLine ();

    GetStartupInfo (&StartUp);
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Tiempo de Ejecución de un Proceso

```
if (targv == NULL)
    printf ("\nNo puedo leer la línea de comandos.");
/* saltar el nombre del programa*/
targv = SkipArg (targv);
/* ejecutar y esperar a que acabe. */
if (!CreateProcess (NULL, targv, NULL, NULL, TRUE,
                  NORMAL_PRIORITY_CLASS, NULL, NULL,
                  &Startup, &ProcInfo))
    printf ("\nError en la creación de proceso.");
WaitForSingleObject (ProcInfo.hProcess, INFINITE);
GetProcessTimes (ProcInfo.hProcess, &CreateTime.ft,
                &ExitTime.ft, &KernelTime, &UserTime);
ElapsedTime.li = ExitTime.li - CreateTime.li;

FileTimeToSystemTime (&ElapsedTime.ft, &ElTiSys);
FileTimeToSystemTime (&KernelTime, &KeTiSys);
FileTimeToSystemTime (&UserTime, &UsTiSys);
printf ("Real Time: %02d:%02d:%02d:%03d\n",
        ElTiSys.wHour, ElTiSys.wMinute, ElTiSys.wSecond,
        ElTiSys.wMilliseconds);
printf ("User Time: %02d:%02d:%02d:%03d\n",
        UsTiSys.wHour, UsTiSys.wMinute, UsTiSys.wSecond,
        UsTiSys.wMilliseconds);
printf ("Sys Time: %02d:%02d:%02d:%03d\n",
        KeTiSys.wHour, KeTiSys.wMinute, KeTiSys.wSecond,
        KeTiSys.wMilliseconds);
CloseHandle (ProcInfo.hThread);
CloseHandle (ProcInfo.hProcess);
return 0;
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejercicio: Creación y Cancelación de Procesos

- Se trata de crear N procesos, cada uno con una consola
- Cada proceso escribirá en la consola un mensaje indicando que proceso es indefinidamente
- El programa principal debe:
 - **1ª Versión: Esperar a que todos acaben**
 - Los procesos habrá que "matarlos a mano", con CTRL-C
 - **2ª Versión: Leer de consola que proceso debe acabar y hacerlo terminae**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejercicio: Creación y Cancelación de Procesos

```
#include <windows.h>
#include <stdio.h>

int main (int argc, LPTSTR argv [])
{

    int i, cuantos, cual;
    STARTUPINFO StartUp;
    PROCESS_INFORMATION ProcInfo[10];
    HANDLE proceso[10];
    UINT como_acaba=1;
    char nombre[20];

    GetStartupInfo (&StartUp);
    printf("Cuantos? (maximo 10) "); scanf("%d",&cuantos);
    for (i=0;i<cuantos;i++)
    {
        sprintf(nombre,"hola %d",i);
        if (!CreateProcess (NULL, nombre, NULL, NULL, FALSE,
            CREATE_NEW_CONSOLE, NULL, NULL, &StartUp, &ProcInfo[i]))
            printf("\nError en la creación de proceso.");
        else proceso[i]=ProcInfo[i].hProcess;
    }
    for (i=0;i<cuantos;i++)
    {
        printf("\nAcabar proceso: "); scanf("%d",&cual);
        TerminateProcess(ProcInfo[i].hProcess,como_acaba);
        CloseHandle (ProcInfo[i].hThread);CloseHandle (ProcInfo[i].hProcess)
    }
    return 0;
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Compartición de ficheros

- La compartición de ficheros por procesos requiere coordinar y sincronizar el acceso
- Se pueden bloquear ficheros de forma total o parcial
- Los bloqueos (File-locks) pueden ser de:
 - Sólo lectura
 - Permite que otros procesos lean, pero no que modifiquen
 - De lectura-escritura
 - Acceso exclusivo
- Cualquier intento de leer, escribir o bloquear un fichero ya bloqueado fallará inmediatamente

Contenido

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- Conceptos Básicos
- Comunic. Entre Procesos
Hebras y Planificación
Sincronización
- DLLs
- E/S Asíncrona
- Bibliografía

Programación con Win32

Compartición de ficheros

```
BOOL LockFileEx (  
    HANDLE    hFile,           // manejador fichero a bloquear  
    DWORD    dwFlags,         // modo de bloqueo  
    DWORD    dwReserved,     // reservada, siempre a 0  
    DWORD    nNumberOfBytesLow, // Número de bytes  
    DWORD    nNumberOfBytesHigh, // a bloquear  
    LPOVERLAPPED lpOverlapped // comienzo de la  
                                // región a bloquear  
)
```

- El modo de bloqueo puede ser:
 - **LOCKFILE_EXCLUSIVE_LOCK** indica que el bloqueo es lectura y escritura. Si no está activado se considera un bloqueo sólo de lectura
 - **LOCKFILE_FAIL_IMMEDIATELY** indica que se retorne inmediatamente si el registro indicado está bloqueado. En caso contrario la llamada bloquea hasta que se libere el fichero

Contenido

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- Conceptos Básicos
- Comunic. Entre Procesos
Hebras y Planificación
Sincronización
- DLLs
- E/S Asíncrona
- Bibliografía

Programación con Win32

Compartición de ficheros

```
BOOL UnlockFileEx (  
    HANDLE    hFile,           // manejador fichero a bloquear  
    DWORD     dwReserved,     // reservada, siempre a 0  
    DWORD     nNumberOfBytesLow, // Número de bytes  
    DWORD     nNumberOfBytesHigh, // a desbloquear  
    LPOVERLAPPED lpOverlapped // para e/s asíncrona  
)
```

- Se debe desbloquear siempre exactamente el mismo rango que se bloqueó
- Se puede bloquear más allá del final del fichero. Se utiliza para extender un fichero
- Los bloqueos no son heredados por los procesos hijos
- **lpoVerlapped** contiene dos campos que indican el offset, los demás deben ser ignorados
 - **DWORD Offset**
 - **DWORD OffsetHigh**

Contenido

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- Conceptos Básicos
- Comunic. Entre Procesos
- Hebras y Planificación
- Sincronización
- DLLs
- E/S Asíncrona
- Bibliografía

Programación con Win32

Timers

- Win32 soporta múltiples timers con diferentes periodos
- Vamos a ver los timers “normales”, existen otros de mayor resolución y que se manejan de forma similar (timers multimedia)
 - La resolución de los timers normales es del orden de 50 milisegundos
- Los timers no tienen manejadores y se identifican con enteros

```
UINT SetTimer (  
    HWND      hWnd,          // ventana asociada  
    UINT      nIdEvent,     // ident., se ignora sin ventanas  
    UINT      uElapse,      // tiempo en milisegundos  
    TIMERPROC lpTimerFun    // manejador del timer  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Timers

- Cada vez que expira el timer se llama a la función indicada

```
VOID CALLBACK TimerFunc (  
    HWND      hWnd,          // ventana asociada  
    UINT      uMsg,          // mensaje, se ignora sin ventanas  
    UINT      idEvent,       // el mismo con el que se creó  
    DWORD     dwTime         // tiempo del sistema  
)
```

- TimerFunc es el nombre de la función manejador (puede ser cualquiera)
- El tiempo está expresado en formato UTC

```
BOOL KillTimer (  
    HWND      hWnd,          // ventana asociada  
    UINT      idEvent,       // el ident. que devuelve SetTimer  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Alarma Periódica

```
// El periodo es en segundos e incluye el tiempo de la alarma

#include <windows.h>
#include <stdio.h>

VOID CALLBACK Beeper (HWND, UINT, UINT, DWORD);
UINT TimerId;

int main (int argc, LPTSTR argv [])
{
    UINT idEvent = 0, Period;

    if (argc >= 2) sscanf(argv[1], "%d", &Period);
    else printf("Usage: beep period");

    TimerId = SetTimer (NULL, idEvent, Period*1000, Beeper);
    if (TimerId == 0) printf("Fallo al inicializar el timer");

    MessageBox (NULL, "BEEPER", "Stop", MB_ICONEXCLAMATION);

    KillTimer (NULL, TimerId);
    return 0;
}

VOID CALLBACK Beeper (HWND hWnd, UINT uMsg, UINT idEvent, DWORD dwTime)
{
    Beep (1000 /* Frecuencia */, 250 /* Duracion */);
    printf("Hola\n");
    return;
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. Entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Otra forma de implementar un retraso

- Una segunda forma de implementar un retraso es utilizar la función sleep

```
BOOL Sleep (
    DWORD    Milliseconds    // retraso en milisegundos
)
```

- El retraso puede, aparte de un entero, puede ser:
 - INFINITE, que haría que el proceso nunca despertase
 - 0, que hace que el proceso abandone la CPU, sin utilizar su time-slice, el planificador pone al proceso directamente en la cola de procesos preparados

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Introducción

- Hasta ahora la únicas formas de comunicación y sincronización son:
 - Paso inicial de parámetros
 - Herencia de manejadores
 - Entorno
 - Terminación
 - Sincronización
 - Código de salida
 - Acceso a ficheros compartidos
- Existen muchos más mecanismos
 - Tuberías (Pipes)
 - Buzones (Mailslots)
 - Sockets, RPC, OLE,.....

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías Anónimas

- Características:
 - Un solo sentido (half duplex)
 - Orientadas al carácter
 - Cada tubería tiene dos manejadores (lectura y escritura)

```
BOOL CreatePipe (  
    PHANDLE    phRead,           // manejador de lectura  
    PHANDLE    phWrite,         // manejador de escritura  
    LPSECURITY_ATTRIBUTES lpsa, // seguridad  
    DWORD      cbPipe           // tamaño del buffer  
)
```

- Para comunicarse es necesario pasar uno de los manejadores a otro proceso
- Normalmente esto se hace con la herencia de manejadores al crear un proceso, aunque también se puede enviar por otros mecanismos

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías Anónimas

- Para leer y escribir se utilizan los manejadores y las funciones normales de ficheros
- Si el buffer está vacío la operación de lectura bloquea
- Si el buffer está lleno la operación de escritura bloquea
- El tamaño del buffer es sólo una sugerencia el sistema puede variarlo según su conveniencia

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Conecta dos programas

```
//uso: mi_pipe prog1 arg1 = prog2 arg2

#include <windows.h>
#include <stdio.h>

int main (int argc, LPTSTR argv [])

{
    DWORD i;
    HANDLE hReadPipe, hWritePipe;
    TCHAR Command1 [MAX_PATH];
    SECURITY_ATTRIBUTES PipeSA = {sizeof (SECURITY_ATTRIBUTES), NULL, TRUE}
        /* Manejadores heredables. */

    PROCESS_INFORMATION ProcInfo1, ProcInfo2;
    STARTUPINFO StartInfoCh1, StartInfoCh2;
    LPTSTR targv = GetCommandLine ();
        /* Información de comienzo de los procesos hijo. */
    GetStartupInfo (&StartInfoCh1);
    GetStartupInfo (&StartInfoCh2);
    if (targv == NULL){
        printf("No puedo leer la línea de comandos"); return(1);}

    targv = SkipArg (targv);
    i = 0;          /* Obtenemos los dos comandos */
    while (*targv != '=' && *targv != '\\0') {
        Command1 [i] = *targv;
        targv++; i++;
    }
    Command1 [i] = '\\0';
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Conecta dos procesos

```
if (*targv == '\\0') {printf("No he encontrado el igual"); return(2);}
targv = SkipArg (targv);

if (!CreatePipe (&hReadPipe, &hWritePipe, &PipeSA, 0))
    { printf("No puedo crear pipe"); return(3);}

/*Asignar los manejadores de e/s estándar. */

StartInfoCh1.hStdInput  = GetStdHandle (STD_INPUT_HANDLE);
StartInfoCh1.hStdError  = GetStdHandle (STD_ERROR_HANDLE);
StartInfoCh1.hStdOutput = hWritePipe;
StartInfoCh1.dwFlags    = STARTF_USESTDHANDLES;

if (!CreateProcess (NULL, (LPTSTR)Command1, NULL, NULL,
                    TRUE, /* Heredar Manejadores */
                    0, NULL, NULL, &StartInfoCh1, &ProcInfo1)) {
    printf("No puedo crear proc 1"); return(4);}
CloseHandle (hWritePipe);

/* Simetricamente para el otro proceso. */
StartInfoCh2.hStdInput  = hReadPipe;
StartInfoCh2.hStdError  = GetStdHandle (STD_ERROR_HANDLE);
StartInfoCh2.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);
StartInfoCh2.dwFlags    = STARTF_USESTDHANDLES;

if (!CreateProcess (NULL, (LPTSTR)targv, NULL, NULL,
                    TRUE,
                    0, NULL, NULL, &StartInfoCh2, &ProcInfo2))
    {printf("No puedo crear proc 2"); return(4);}
CloseHandle (hReadPipe);
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Conecta dos procesos

```
/* Esperamos que acaben los procesos  
y cerramos los manejadores */
```

```
WaitForSingleObject (ProcInfo1.hProcess, INFINITE);  
WaitForSingleObject (ProcInfo2.hProcess, INFINITE);  
CloseHandle (ProcInfo1.hThread);  
CloseHandle (ProcInfo1.hProcess);  
CloseHandle (ProcInfo2.hThread);  
CloseHandle (ProcInfo2.hProcess);  
return 0;  
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- Son mucho más generales que las anónimas y pueden servir incluso para implementar aplicaciones distribuidas.
- Características básicas:
 - Son orientados a mensaje
 - El lector recibe mensajes de tamaño variable tal y como los mandó el escritor
 - Son bidireccionales
 - Puede haber múltiples instancias de la misma tubería
 - Muchos clientes pueden comunicarse con un solo servidor
 - El servidor puede responder utilizando la misma instancia
 - El nombre de la tubería es visible en la red y se accede igual sea local o remota

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- La tubería normalmente es creada por el servidor

```
HANDLE CreateNamedPipe (  
    LPTSTR    lpszPipeName,    // nombre de la tubería  
    DWORD     fdwOpenMode,     // modo de acceso  
    DWORD     fdwPipeMode,     // bloqueo/mensajes/carácter  
    DWORD     nMaxInstances,   // número de clientes  
    DWORD     cbOutBuffer,     // tamaño buffer salida  
    DWORD     cbInBuffer,      // tamaño buffer entrada  
    DWORD     dwTimeout,       // timeout por defecto  
    LPSECURITY_ATTRIBUTES lpsa // seguridad  
)
```

- El nombre tiene que tener el formato:
 `\\.\pipe\[path]nombre_tubería`
- El `.` indica la máquina local, no se puede crear una tubería en una máquina remota

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
 - Conceptos Básicos
 - Comunic. entre Procesos
 - Hebras y Planificación
 - Sincronización
- DLLs
- E/S Asíncrona
- Bibliografía

Tuberías con Nombre

- Otros parámetros:
 - **fdwOpenMode** puede ser:
 - **PIPE_ACCESS_DUPLEX** flujo de datos bidireccional
 - **PIPE_ACCESS_INBOUND** flujo de datos del cliente al servidor
 - **PIPE_ACCESS_OUTBOUND** flujo de datos del servidor al cliente
 - **fdwPipeMode** puede tener tres pares de valores mutuamente excluyentes:
 - **PIPE_TYPE_BYTE/PIPE_READ_MODE_MESSAGE** indican si se lee un número arbitrario de caracteres o un mensaje completo (**PIPE_TYPE_MESSAGE**)
 - **PIPE_WAIT/PIPE_NO_WAIT** indica si la lectura bloquea o no. Normalmente se utiliza bloqueante.

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

Cliente 0

```
h=CreateFile(PipeName);
while(){
    WriteFile(h,&Request);
    ReaFile(h,&Response);
}
CloseHandle(h);
```

Cliente N-1

```
h=CreateFile(PipeName);
while(){
    WriteFile(h,&Request);
    ReaFile(h,&Response);
}
CloseHandle(h);
```

```
/*Crear N Instancias*/
for(i=0;i<N;i++)
    h[i]=CreateNamedPipe(PipeName,N)
/*testear cada instancia, obtener
la petición y responder*/
i=0;
while(){
    if PeekNamedPipe(h[i]) {
        ReaFile(h[i],&Request);
        /*Crear Respuesta*/
        WriteFile(h[i],&Response);
    }
    i=i++%N;
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- Para ver si hay un mensaje se usa

```
BOOL PeekNamedPipe (  
    HANDLE        hPipe,           // manejador de la tubería  
    LPVOID        lpvBuffer,      // buffer  
    DWORD         cbBuffer,       // tamaño  
    LPDWORD       lpcbRead,       // leídos  
    LPDWORD       lpcbAvail,      // disponibles  
    LPDWORD       lpcbMessage     // resto del mensaje  
)
```

- Las funciones de espera normales no funcionan con los manejadores de tuberías
- Hay que hacer una espera activa (polling)
- La función lee de forma no destructiva mensajes o caracteres sin bloquear
- ***lpcAvail** nos indica si hay mensajes disponibles, los demás parámetros pueden ser NULL

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- La utilización de las tuberías con nombre siguen casi siempre un mismo esquema:
 - Abrir una instancia de la tubería
 - Repetidamente enviar peticiones y recibir respuestas
 - Cerrar la instancia
 - La secuencia de enviar peticiones (**WriteFile**) y recibir respuestas (**ReadFile**) puede considerarse una sola transacción.

```
BOOL TransactNamedPipe (  
    HANDLE      hPipe,           // manejador de la tubería  
    LPVOID      lpvWriteBuffer, // buffer de escritura  
    DWORD       cbWriteBuffer,  // tamaño buffer de escritura  
    LPVOID      lpvReadBuffer,  // buffer de lectura  
    DWORD       cbReadBuffer,   // tamaño buffer de lectura  
    LPDWORD     lpcbRead,       // tamaño del mensaje leído  
    LPDOVERLAPPED lpA          // e/s asíncrona  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- El esquema anterior requiere una conexión permanente ya creada
- La siguiente función combina la anterior con la apertura y cierre de la conexión

```
BOOL CallNamedPipe (  
    LCPCTSTR    lpszPipeName,    // nombre de la tubería  
    LPVOID      lpvWriteBuffer,  // buffer de escritura  
    DWORD      cbWriteBuffer,    // tamaño buffer de escritura  
    LPVOID      lpvReadBuffer,   // buffer de lectura  
    DWORD      cbReadBuffer,     // tamaño buffer de lectura  
    LPDWORD    lpcbRead,         // tamaño del mensaje leído  
    DWORD      dwTimeout         // timeout en la conexión  
)
```

- Se aumenta la utilización de la tubería (varios clientes intercalados)
- Se incurre en una sobrecarga (abrir y cerrar)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Tuberías con Nombre

- La utilización de la anterior función, en combinación con la siguiente, permite eliminar la espera activa del servidor

```
BOOL ConnectNamedPipe (  
    HANDLE          hNamedPipe,        // manejador de la tubería  
    LPOVERLAPPED lpo,                // e/s asíncrona  
)
```

- Con **lpo** a **NULL** la función retorna cuando hay una conexión de un cliente
- Una vez que retorna el servidor trata la transacción
- El servidor llama a **DisconnectNamedPipe** una vez que ha terminado la transacción
- Existe otra función (**WaitNamedPipe**) que es solo para sincronización
- Estas funciones están sólo soportadas por NT

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Buzones

- Los buzones (Mailslots) son un mecanismo de difusión de mensajes (broadcast) para comunicación entre procesos.
- Características:
 - Son unidireccionales
 - Puede tener multiple lectores o multiples escritores
 - No existe confirmación de que los mensajes enviados son recibidos
 - Pueden utilizarse para comunicación entre procesos de máquinas remotas
 - Las longitudes de los mensajes están limitadas

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Buzones

- Para usar un buzón:
 - Cada servidor (lector) crea un buzón con **CreateMailslot**
 - El servidor espera a recibir un mensaje con una llamada a **ReadFile**
 - Un cliente abre el buzón con **CreateFile** y escribe mensajes con **WriteFile**.
 - La lectura falla si no hay lectores esperando
 - Un mensaje de un cliente puede ser leído por todos los servidores (todos reciben el mismo mensaje)
 - Un cliente al hacer **CreateFile** puede especificar un nombre como ***\mailslot\nombre**
 - De esta forma localiza a todos los servidores en el dominio de la red

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Buzones

Cliente 0

```
h=CreateMailSlot(  
    "\\.\mailslot\status");  
ReaFile(h,&ServStatus);  
}
```

Cliente N-1

```
h=CreateMailSlot(  
    "\\.\mailslot\status");  
ReaFile(h,&ServStatus);  
}
```

```
while(...){  
    Sleep(..);  
    hms=CreateFile(  
        "\\*\mailslot\status");  
    ....  
    WriteFile(hms,&StatusInform);  
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Buzones

- El cliente (escritor) al abrir el buzón con **CreateFile** puede usar :
 - `\\.\\mailslot\[path]nombre` (buzón local)
 - `\\máquina\\mailslot\[path]nombre` (máquina remota)
 - `\\dominio\\mailslot\[path]nombre` (todos los

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Introducción

- Las hebras son la unidad básica de ejecución en Win32
- Un proceso puede tener múltiples hebras, compartiendo el espacio de direcciones y otros recursos
- Su utilización viene determinada por los siguientes factores:
 - Simplifican el diseño de las aplicaciones
 - Aumentan la eficiencia
 - Permiten implementar fácilmente sistemas reactivos
 - Permiten utilizar múltiple procesadores en el que caso de que la máquina disponga de ellos
 - En entornos distribuidos son esenciales para respuestas a peticiones de servicios remotos

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Estados de un Hebra

- En un momento dado solo una hebra se ejecuta por procesador
- En Windows NT una hebra puede estar en los siguientes estados:
 - Ready
 - Puede ser planificada
 - El planificador elige siempre la hebra con mayor prioridad entre las de este estado
 - Standby
 - La hebra permanece en este estado una vez elegida para su ejecución hasta el próximo cambio de contexto
 - Running
 - Una hebra en standby pasa a este estado cuando queda un procesador libre

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Estados de una Hebra

- Waiting

- Una hebra continua ejecutándose hasta que:

- pasa su time-slice (**ready**)
- es interrumpida por una hebra de mayor prioridad (**ready**)
- se bloquea (**waiting**)
- es terminada (**terminated**)

- Una hebra permanece en el estado de espera hasta que ocurre el evento que está esperando

- Cuando ocurre el evento puede pasar a **ready** o a **transition**

- Transition

- Está lista para ejecutar pero le faltan recursos
- Cuando los recupera pasa a **ready**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

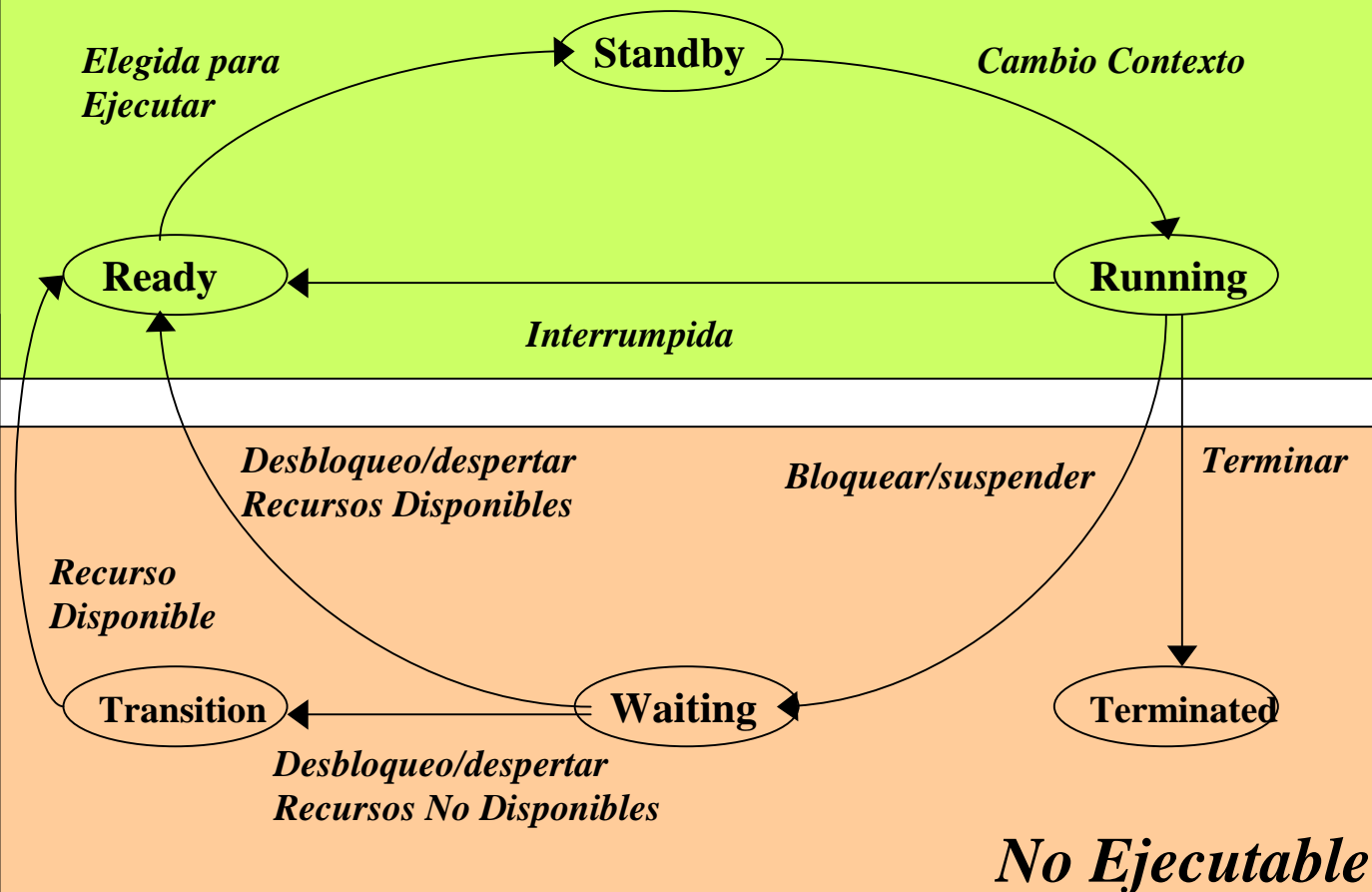
E/S Asíncrona

Bibliografía

Programación con Win32

Estados de una Hebra

Ejecutable



Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Conceptos Básicos

- Las hebras como cualquier otro objeto son accedidas y controladas a través de un manejador

```
HANDLE CreateThread (
    LPSECURITY_ATTRIBUTES lpsa, // seguridad
    DWORD cbStack,           // tamaño de la pila
    LPTHREAD_START_ROUTINE lpStarAddr,
                                // función a ejecutar
    LPVOID lpvThreadParam, // puntero a parámetro
    DWORD fdwCreate,         // comienzo de ejecución?
    LPDWORD lpIDThread       // identificador hebra
)
```

- La función que ejecutará la hebra debe tener el siguiente formato:
DWORD WINAPI funcion_hebra(LPVOID)
- Sólo se le puede pasar un parámetro, que es un puntero a void
 - Los parámetros se suelen empaquetar en una estructura

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Conceptos Básicos

- El modo de creación puede ser:
 - **0** si se empieza a ejecutar inmediatamente
 - **CREATE_SUSPENDED** si se crea suspendida
 - Requiere un ResumeThread para comenzar la ejecución
- Si un proceso termina todas las hebras también lo hacen, pero normalmente las hebras deben finalizar su ejecución con:

```
VOID ExitThread (  
    DWORD      devExitCode,          // código de finalización  
)
```

- Cuando la última hebra de un proceso acaba su ejecución, éste también termina
- Una hebra puede hacer que otra acabe con **Terminate_Thread**, pero sus recursos no son devueltos de forma correcta

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Conceptos Básicos

- Una hebra continuará existiendo hasta que se cierran todos los manejadores que la referencian (aunque haya acabado su ejecución)
- Su código de salida se puede obtener a través del manejador

```
VOID GetExitCodeThread (  
    HANDLE      hThread          // manejador de la hebra  
    LPDWORD     lpdwExitCode     // código de finalización  
)
```

- Si la hebra aún se esta ejecutando devolverá

STILL_ACTIVE

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Conceptos Básicos

- Una hebra puede ser suspendida y reactivada
- Cada hebra tiene asociado un contador de suspensión
- La hebra sólo se ejecuta cuando su contador de suspensión es cero.

```
DWORD ResumeThread (  
    HANDLE        hThread        // manejador de la hebra  
)  
DWORD SuspendThread (  
    HANDLE        hThread        // manejador de la hebra  
)
```

- Estas funciones incrementan y decrementan el contador de suspensión
- Ambas funciones devuelven el valor anterior del contador

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Conceptos Básicos

- Una hebra también puede ser creada en un proceso remoto, siempre que:
 - tengamos acceso a su manejador
 - el proceso fuese creado con el privilegio **PROCESS_CREATE_THREAD**
 - es peligroso utilizarla, aunque facilita el desarrollo de determinado tipo de software (depuradores,...)
 - en Windows 95 no está implementada

```
HANDLE CreateRemoteThread (  
    HANDLE    hProcess          // manejador proceso remoto  
    LPSECURITY_ATTRIBUTES lpsa, // seguridad  
    DWORD     cbStack,         // tamaño de la pila  
    LPTHREAD_START_ROUTINE lpStarAddr,  
                                // función a ejecutar  
    LPVOID    lpvThreadParam,  // puntero a parámetro  
    DWORD     fdwCreate,       // comienzo de ejecución?  
    LPDWORD   lpIDThread       // identificador hebra  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Creación de Hebras

```
#include <windows.h>
#include <stdio.h>

DWORD WINAPI hebra(LPVOID argumento);
DWORD compartida=1;

int main (int argc, LPTSTR argv[])
{
    HANDLE hebra1, hebra2;
    DWORD hid1,hid2;
    int id1=1, id2=2;

    hebra1=CreateThread(NULL,0,hebra,(LPVOID) &id1,0,&hid1);

    if (!hebra1) { printf("Error Hebra1\n"); return (1);}

    hebra2=CreateThread(NULL,0,proceso,(LPVOID) &id2,0,&hid2);

    if (!hebra2) { printf("Error Hebra2\n"); return (2);}

    WaitForSingleObject(hebra1,INFINITE);
    WaitForSingleObject(hebra2,INFINITE);
    CloseHandle(hebra1);
    CloseHandle(hebra2);
    printf("El valor de compartida es: %d\n",compartida);
    return(0);
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Creación de Hebras

```
DWORD WINAPI hebra(LPVOID argumento)
{
    int i,*id;
    char cadena[25];

    id=(int*) argumento;
    sprintf(cadena,"Hola soy la hebra %d\n",*id);
    while (1) {
        for (i=0;i<strlen(cadena);i++) putchar(cadena[i]);
    }

    return(0);
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejercicio: Suspende y Reanuda

- A partir del problema anterior:
 - Crear N hebras
 - Leer de teclado:
 - **s-indent:**
 - suspender a la hebra ident
 - **r-indent**
 - reanudarla
 - **fin**
 - acabar todas la hebras
- Hay que configurar el proyecto de visual C++ para multihebra.
 - Incluir **/MT** o **/MTd** como parámetro de compilación
 - En algunos libros “antiguos” se aconseja utilizar
 - **_beginthreadex**
 - **_endthreadex**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Almacenamiento Local en Hebras

- En algunas ocasiones es conveniente que las hebras utilicen un almacenamiento global para cada una.
 - Se puede hacer "a mano", pasando una estructura distinta para cada hebra cuando se crean
 - Win32 proporciona un almacenamiento local (TLS- Thread Local Storage) que da a cada hebra un array de punteros propio



Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Almacenamiento Local en Hebras

- **TLS:**
 - Inicialmente ningún proceso tiene asignado ningún puntero:
 - **Para solicitar un índice**
DWORD TlsAlloc(VOID)
 - **Para devolverlo**
BOOL TlsFree(DWORD dwIndex)
 - Cada hebra puede modificar los valores de estos punteros
LPVOID TlsGetValue(dwTlsIndex)

BOOL TlsSetValue(DWORD dwTlsIndex,
LPVOID lpvTlsValue)
 - El almacenamiento es global, pero distinto para cada hebra (NO PUEDE SER ACCEDIDO POR OTRAS HEBRAS)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Planificación

- En Windows NT siempre se ejecuta la hebra de mayor prioridad lista para ejecución
- Las hebras reciben su prioridad de forma relativa a la clase de prioridad de su proceso
- Existen cuatro clases de prioridad, fijadas inicialmente en CreateThread
- Cada clase tiene una **prioridad base**:
 - **IDLE_PRIORITY_CLASS**, prioridad base 4
 - **NORMAL_PRIORITY_CLASS**, prioridad base 9 o 7
 - **HIGH_PRIORITY_CLASS**, prioridad base 13
 - **REAL_TIME_PRIORITY_CLASS**, prioridad base 24

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Planificación

- La prioridad base de la clase normal es 9 si la ventana utiliza el teclado y 7 en cualquier otro caso
- Un proceso puede cambiar su prioridad o la de otro proceso si lo permiten sus atributos de seguridad

```
BOOL SetPriorityClass (  
    HANDLE    hProcess,           // manejador proceso  
    DWORD     fdwPriority         // prioridad  
)
```

```
DWORD GetPriorityClass (  
    HANDLE    hProcess,           // manejador proceso  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Planificación

- Prioridad de una hebra
 - Es la misma que la del proceso cuando se crea
 - Puede variar en +/-2 de la prioridad base
 - **THREAD_PRIORITY_LOWEST**
 - **THREAD_PRIORITY_BELOW_NORMAL**
 - **THREAD_PRIORITY_NORMAL**
 - **THREAD_PRIORITY_ABOVE_NORMAL**
 - **THREAD_PRIORITY_HIGHEST**

```
BOOL SetThreadPriority (  
    HANDLE    hThread,        // manejador hebra  
    int       nPriority       // prioridad  
)
```

```
int GetThreadPriority (  
    HANDLE    hThread,        // manejador hebra  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Planificación

- Existen otros dos niveles de prioridad, que son absolutos y que se utilizan solo en casos especiales
 - **THREAD_PRIORITY_IDLE (1 o 16 en tiempo real)**
 - **THREAD_PRIORITY_TIME_CRITICAL (15 o 31 en tiempo real)**
- Las prioridades de las hebras cambian de manera automática cuando cambian la de su proceso
- El sistema operativo puede ajustar la prioridad de la hebra dinámicamente en función del comportamiento de ésta
- **Windows NT no es adecuado para aplicaciones de tiempo real duro**
 - No es totalmente interrumpible
 - Inversión de prioridades,.....

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Código seguro para hebras

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Operaciones atómicas predefinidas

- Operaciones tan simples como incrementar una variable global o intercambiar dos variables pueden producir malfuncionamientos
- Para estas operaciones tan simples se pueden llamar a funciones especiales:
`InterlockedIncrement (&N) ;`
`InterlockedDecrement (&N) ;`
`InterlockedExchange (&A, &B) ;`
- Sin embargo, tienen una utilidad muy limitada y son poco portables

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Sincronización

- Ya hemos visto tres mecanismos de sincronización:
 - WaitForSingleObject y WaitForMultipleObjects
 - Tuberías
 - Bloqueos de fichero
- No son suficientes
- Necesitamos
 - Mecanismos para exclusión mutua
 - Mecanismos para implementar condiciones de sincronización
- Algunos lenguajes de programación ya los tienen integrados (Java o Ada), pero de todas formas se basan en los mecanismos del sistema operativo.

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Secciones Críticas

- Una sección crítica es un concepto general en Programación Concurrente que se define como

Sección de código que desde el punto de vista de otro proceso debe ser ejecutada como una instrucción atómica

- No debe confundirse el concepto general con el mecanismo concreto de sincronización en Windows NT
- Tampoco debe confundirse con una sección de código no interrumpible
 - Una sección de código no interrumpible sería una sección crítica para **todos los procesos del sistema**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Secciones Críticas

- Son el mecanismo más eficiente y más simple, aunque tiene problemas de estructuración
- Una sección crítica es un objeto que puede ser inicializado y borrado
 - No tiene manejadores
 - No puede ser compartido por otros procesos (ésta es la principal causa de su eficiencia)
- Existe un tipo predefinido (**CRITICAL_SECTION**) y cuatro funciones para su manejo

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Secciones Críticas

```
VOID InitializeCriticalSection (  
    LPCRITICALSECTION lpCriticalSection // Sec. Crítica  
)
```

```
VOID DeleteCriticalSection (  
    LPCRITICALSECTION lpCriticalSection // Sec. Crítica  
)
```

```
VOID EnterCriticalSection (  
    LPCRITICALSECTION lpCriticalSection // Sec. Crítica  
)
```

```
VOID LeaveCriticalSection (  
    LPCRITICALSECTION lpCriticalSection // Sec. Crítica  
)
```

- Al entrar en una sección crítica se comprueba si ésta está ya ocupada y si es así se suspende
- Al salir se despierta a uno de los procesos bloqueados

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Secciones Críticas

- Si la hebra ya posee la sección crítica, no bloquea cuando llama de nuevo a EnterCriticalSection
- Se mantiene una cuenta de cuantas veces se ha entrado
- El proceso deberá abandonar la sección crítica tantas veces como haya entrado
- Si un proceso intenta abandonar una sección crítica que no posee se pueden producir resultados no predecibles (incluido su bloqueo)
- No se puede testear si otro proceso está en una sección crítica o no
- Tampoco se pueden especificar time-outs
- Las secciones críticas no son objetos del kernel y se mantienen en el espacio de direcciones del usuario, por eso son más eficientes

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Acceso a la Pantalla

```
#include <windows.h>
#include <stdio.h>

DWORD WINAPI hebra(LPVOID argumeto);
DWORD compartida=1;
CRITICAL_SECTION pantalla;

int main (int argc, LPTSTR argv[])
{
    HANDLE hebra1, hebra2;
    DWORD hid1,hid2;
    int id1=1, id2=2;

    InitializeCriticalSection(&pantalla);

    hebra1=CreateThread(NULL,0,hebra,(LPVOID) &id1,0,&hid1);

    if (!hebra1) { printf("Error Hebra1\n"); return (1);}

    hebra2=CreateThread(NULL,0,hebra,(LPVOID) &id2,0,&hid2);

    if (!hebra2) { printf("Error Hebra2\n"); return (2);}
    WaitForSingleObject(hebra1,INFINITE);
    WaitForSingleObject(hebra2,INFINITE);

    DeleteCriticalSection(&pantalla);
    CloseHandle(hebra1);
    CloseHandle(hebra2);
    printf("El valor de compartida es: %d\n",compartida);
    return(0);
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Acceso a la Pantalla

```
DWORD WINAPI hebra(LPVOID argumento)
{
    int i,j,*id;
    char cadena[25];

    id=(int*) argumento;
    sprintf(cadena,"Hola soy la hebra %d\n",*id);
    for (j=0;j<100;j++) {
        EnterCriticalSection(&pantalla);
        for (i=0;i<strlen(cadena);i++) putchar(cadena[i]);
        compartida++;
        LeaveCriticalSection(&pantalla);
    }

    return(0);
}
```

• Ejercicio:

- Utilizar dos secciones críticas y adquirirlas en cada hebra en un orden distinto
- Estudiar la ejecución:
 - Interbloqueo
 - Depuración con hebras

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Mutexes

- Un mutex proporciona más funcionalidad que las secciones críticas:
 - Pueden tener nombres y manejadores
 - Se pueden usar por hebras en diferentes procesos
 - Sincronización en el acceso a ficheros mapeados en memoria
 - Permiten timeouts
 - Son abandonados de forma automática cuando termina el proceso
 - este es un problema de las secciones críticas: si una hebra acaba con un error dentro de una sección crítica, ésta permanece ocupada

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Mutexes

- Para conseguir bloquear un mutex se utilizan las funciones estándar de espera en procesos con el manejador del mutex como parámetro

```
WaitForSingleObject(mutex_handle,timeout)
```

```
WaitForMultipleObjects(n,mutex_array,timeout)
```

- Un mutex puede ser adquirido varias veces, sin esperar cuando ya se tiene
- Siempre debe devolverse el mismo número de veces
- Para abandonar un mutex se utiliza

```
BOOL ReleaseMutex (  
    HANDLE hmutex // manejador del mutex  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Mutexes

- Para crear un mutex

```
HANDLE CreateMutex (  
    LPSECURITY_ATTRIBUTES lpsa, // seguridad  
    BOOL fInitialOwner // coge el mutex al crearlo  
    LPCSTR lpzMutexName // nombre  
)
```

- El nombre del mutex se utiliza para que otros procesos puedan acceder a él
- El espacio de nombre es único para los mutexes, semáforos y eventos
- El nombre puede ser NULL (mutex anónimo)

- Para usar un mutex en otro proceso

```
HANDLE OpenMutex(  
    DWORD dwDesiredAccess, // access flag  
    BOOL bInheritHandle, // herencia de manejador  
    LPCTSTR lpName // nombre mutex  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Mutexes

```
#include <windows.h>
#include <stdio.h>

DWORD WINAPI hebra(LPVOID argumeto);
DWORD compartida=1;
HANDLE mutex;

int main (int argc, LPTSTR argv[])
{
    HANDLE hebra1, hebra2;
    DWORD hid1,hid2;
    int id1=1, id2=2;

    mutex=CreateMutex(NULL,FALSE,NULL);

    hebra1=CreateThread(NULL,0,hebra,(LPVOID) &id1,0,&hid1);
    if (!hebra1) { printf("Error Hebra1\n"); return (1);}

    hebra2=CreateThread(NULL,0,hebra,(LPVOID) &id2,0,&hid2);
    if (!hebra2) { printf("Error Hebra2\n"); return (2);}

    WaitForSingleObject(hebra1,INFINITE);
    WaitForSingleObject(hebra2,INFINITE);
    CloseHandle(mutex);
    CloseHandle(hebra1);
    CloseHandle(hebra2);
    printf("El valor de compartida es: %d\n",compartida);
    return(0);
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Mutexes

```
DWORD WINAPI hebra(LPVOID argumento)
{
    int i,j,*id;
    char cadena[25];

    id=(int*) argumento;
    sprintf(cadena,"Hola soy la hebra %d\n",*id);
    for (j=0;j<100;j++) {
        WaitForSingleObject(mutex,INFINITE);
        for (i=0;i<strlen(cadena);i++) putchar(cadena[i]);
        compartida++;
        ReleaseMutex(mutex);
    }

    return(0);
}
```

- Se puede ver si un mutex está ocupado o no utilizando un timeout de 0
- Las secciones críticas son aproximadamente 10 veces más rápidas que los mutexes
- Si se hace un WaitForSingleObject en un mutex abandonado devuelve **WAIT_ABANDONED_0**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Mutexes

```
DWORD WINAPI hebra(LPVOID argumento)
{
    int i,j,*id;
    char cadena[25];

    id=(int*) argumento;
    sprintf(cadena,"Hola soy la hebra %d\n",*id);
    for (j=0;j<100;j++) {
        WaitForSingleObject(mutex,INFINITE);
        for (i=0;i<strlen(cadena);i++) putchar(cadena[i]);
        compartida++;
        ReleaseMutex(mutex);
    }

    return(0);
}
```

- Se puede ver si un mutex está ocupado o no utilizando un timeout de 0
- Las secciones críticas son aproximadamente 10 veces más rápidas que los mutexes
- Si se hace un WaitForSingleObject en un mutex abandonado devuelve **WAIT_ABANDONED_0**

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Semáforos

- Objeto de sincronización con un valor entero asociado mayor o igual que 0
- Aparte de la creación soporta dos operaciones atómicas:
 - wait
 - En el caso de Win32 se utiliza la función común de espera en un objeto
 - Si el valor es mayor que 0 decrementa en 1, de forma atómica, este valor
 - Si vale 0 suspende a la hebra
 - signal
 - En el caso de Win32 la función se denomina ReleaseSemaphore
 - Despierta un proceso si existen procesos suspendidos

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Semáforos

- Otras características
 - A diferencia de las secciones críticas y los mutexes, sucesivos waits en un mismo semáforo bloquean a la hebra
 - En el caso de Win32 una operación **ReleaseSemaphore** incrementa el valor del semáforo con un valor entero positivo arbitrario
 - Los semáforos tienen limitado su valor máximo de forma que si se intenta incrementar por encima de este valor se produce un error
 - Se utiliza para permitir el acceso simultáneo de hasta N procesos a una sección crítica (semáforo de valor N)
 - Al igual que los mutexes pueden ser compartidos entre procesos, con un nombre y utilizando la función **OpenSemaphore** (similar a **OpenMutex**)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Semáforos

- Creación

```
HANDLE CreateSemaphore (  
    LPSECURITY_ATTRIBUTES lpsa, // seguridad  
    LONG cSemInitial, // valor inicial  
    LONG cSemMax, // valor máximo  
    LPCSTR lpzSemName // nombre  
)
```

- Operación signal

```
HANDLE ReleaseSemaphore (  
    HANDLE hSemaphore, // manejador  
    LONG cReleaseCount, // incremento, normalmente 1  
    LPLONG lpPreviousCount, // valor anterior  
)
```

- Apertura por otro proceso

```
HANDLE OpenSemaphore(  
    DWORD dwDesiredAccess, // access flag  
    BOOL bInheritHandle, // herencia de manejador  
    LPCTSTR lpName // nombre semáforo  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Semáforos

```
hSem=CreateSemaphore(NULL,0,3,NULL); //crear un semáforo sin nombre

if (hSem==NULL) { // manejar error }

...

ReleaseSemaphore(hSem,3,NULL); //permitir el acceso a 3 hebras

...

Status = WaitForSingleObject(hSem, INFINITE);

if (Status==-1) { // error al intentar bloquear el semáforo }

// Acceder al recurso compartido

ReleaseSemaphore(hSem,1,NULL); //Abandonar recurso
```

- Ejercicio:

- ¿Cómo se podría decrementar un semáforo en dos?
- Analizar la posibilidad de interbloqueo

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Productor/Consumidor

```
#include <stdio.h>
#include <windows.h>

#define BUFSIZE 5
int SharedBuffer[BUFSIZE];
int head,tail;
int count;

HANDLE hMutex, Elementos, Huecos;

void WINAPI Productor()
{
    int i;

    for (i=0;i<20;i++)
    {
        WaitForSingleObject(Huecos,INFINITE);
        // producir
        if (WaitForSingleObject(hMutex,INFINITE) == WAIT_FAILED){
            fprintf(stderr,"ERROR: Productor()\n");
            ExitThread(0);}

        else {
            SharedBuffer[tail] = i;
            tail = (tail+1) % BUFSIZE;
            printf("Produce: %d\n", i);
            ReleaseMutex(hMutex);
        }

        ReleaseSemaphore(Elementos,1,NULL);
    }
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Productor/Consumidor

```
void WINAPI Consumidor()
{
    int i;

    for (i=0;i<20;i++)
    {
        WaitForSingleObject(Elementos,INFINITE);
        // Consumir
        if (WaitForSingleObject(hMutex,INFINITE) == WAIT_FAILED){
            fprintf(stderr,"ERROR: Consumidor()\n");
            ExitThread(0);
        }

        else {
            SharedBuffer[head] = i;
            head = (head+1) % BUFSIZE;
            printf("Consume: %d\n", i);
            ReleaseMutex(hMutex);
        }

        ReleaseSemaphore(Huecos,1,NULL);
    }
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejemplo: Productor/Consumidor

```
int main(int argc, LPTSTR arg[])
{
    HANDLE hThreadVector[2];
    DWORD ThreadID;

    count = 0;
    head = 0;
    tail = 0;

    hMutex = CreateMutex(NULL, FALSE, NULL);
    Elementos = CreateSemaphore(NULL, 0, BUFSIZE, NULL);
    // NO HAY ELEMENTOS
    Huecos = CreateSemaphore(NULL, BUFSIZE, BUFSIZE, NULL);
    // HAY BUFSIZE HUECOS

    hThreadVector[0] = CreateThread (NULL, 0,
                                    Productor,
                                    NULL, 0, &ThreadID);
    hThreadVector[1] = CreateThread (NULL, 0,
                                    Consumidor,
                                    NULL, 0, &ThreadID);

    WaitForMultipleObjects(2, hThreadVector, TRUE, INFINITE);
    return (0);
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Eventos

- Se utilizan para modelar condiciones de sincronización
- Características:
 - No llevan una cuenta asociada (semáforos)
 - Permiten despertar a varios procesos que esperan en un evento al mismo tiempo
 - Al igual que los otros objetos de sincronización tienen dos estados (señalados y no señalados)
 - La espera se realiza de la misma forma que en los otros objetos (funciones genéricas)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Eventos

- Existen dos tipos de eventos
 - Reset manual
 - se despiertan varios procesos de una vez
 - hay que resetear el evento
 - Autoreset
 - solo se despierta a un proceso
 - no hay que resetearlo
- Creación

```
HANDLE CreateEvent (  
    LPSECURITY_ATTRIBUTES lpsa, // seguridad  
    BOOL fManualReset, // tipo de evento  
    BOOL fInitialState, // valor inicial  
    LPCSTR lpszSemName // nombre global  
)
```

- Como en los otros mecanismo puede ser accedido desde otro proceso a través del nombre (**OpenEvent**)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Eventos

- Hay tres funciones para el control de eventos

```
HANDLE SetEvent (  
    HANDLE hEvent // manejador de evento  
)
```

- Señaliza un evento
- Si el evento es autoreset sólo una hebra se despertará y volverá al estado de no señalado
- Si no hay procesos el evento permanece señalado para la primera hebra que haga un wait
- Si el evento manual
 - todas las hebras se despiertan
 - el evento permanece señalado hasta que se haga un Reset (todos los procesos que hagan un wait antes seguirían sin esperar)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Eventos

```
HANDLE ResetEvent (  
    HANDLE hEvent    // manejador de evento  
)
```

```
HANDLE PulseEvent (  
    HANDLE hEvent    // manejador de evento  
)
```

- Con **PulseEvent** todas las hebras que esperan en un evento manual se despiertan, pero el evento se resetea automáticamente
- Si no hay hebras esperando no tiene efecto
- Hay que tener cuidado cuando se utiliza **WaitForMultipleObjects** con un varios eventos
 - Solo tiene éxito si simultáneamente todos están en el estado señalado en algún momento

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Ejercicio: Productor/Consumidor con Eventos

- Modificar el problema del productor consumidor para utilizar eventos en lugar de semáforos
- Hay que tener en cuenta:
 - Hay que llevar una cuenta de los elementos que hay en el buffer
 - Habrá que suspender cuando:
 - el número de elementos sea igual al tamaño (productor)
 - el número de elementos sea cero (consumidor)
 - Tener en cuenta la posibilidad de interbloqueo

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Productor/Consumidor con Eventos

```
int SharedBuffer[BUFSIZE];
int head,tail,count;

HANDLE hMutex;
HANDLE Evento_No_Lleno, Evento_No_Vacio;

void WINAPI Productor()
{ int i,puedo_producir;

  for (i=20; i>=0; i--) {
    puedo_producir=FALSE;
    do {
      if (WaitForSingleObject(hMutex,INFINITE) == WAIT_FAILED){
        fprintf(stderr,"ERROR: Productor()\n"); ExitThread(0);}
      if (count == BUFSIZE) { // buffer lleno
        ReleaseMutex(hMutex);
        // esperar a que no este lleno
        WaitForSingleObject(Evento_No_Lleno,INFINITE);
      }
      else puedo_producir=TRUE;

    } while (!puedo_producir);
    printf("Produce: %d\n", i); //producir
    SharedBuffer[tail] = i; tail = (tail+1) % BUFSIZE;
    count++;
    ReleaseMutex(hMutex); // fin sección crítica
    PulseEvent(Evento_No_Vacio); // despertar al consumidor
  }
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Productor/Consumidor con Eventos

```
void WINAPI Consumidor()  
{  
    int result;  
  
    while (TRUE) {  
        if (WaitForSingleObject(hMutex,INFINITE) == WAIT_FAILED){  
            fprintf(stderr,"ERROR: Consumidor()\n");  
            ExitThread(0);  
        }  
  
        if (count == 0) { // buffer vacio  
            ReleaseMutex(hMutex); // abandono el mutex  
            WaitForSingleObject(Evento_No_Vacio,INFINITE); }  
        else if (SharedBuffer[head] == 0) { // último dato  
            printf("Consumido %d: último dato\n",SharedBuffer[head]);  
            ReleaseMutex(hMutex);  
            ExitThread(0);  
        }  
        else { // tengo el mutex  
            result = SharedBuffer[head];  
            printf("Consumido: %d\n", result);  
            head = (head+1) % BUFSIZE;  
            count--;  
            ReleaseMutex(hMutex);  
            PulseEvent(Evento_No_Lleno); // despertar al productor  
        }  
    }  
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

Conceptos Básicos

Comunic. entre Procesos

Hebras y Planificación

Sincronización

DLLs

E/S Asíncrona

Bibliografía

Programación con Win32

Productor/Consumidor con Eventos

```
int main(int argc, LPTSTR arg[])
{
    HANDLE hThreadVector[2];
    DWORD ThreadID;

    count = 0;
    head = 0;
    tail = 0;

    hMutex = CreateMutex(NULL, FALSE, NULL);

    Evento_No_Lleno = CreateEvent(NULL, TRUE, FALSE, NULL);
    // evento manual
    Evento_No_Vacio = CreateEvent(NULL, TRUE, FALSE, NULL);
    // evento manual

    hThreadVector[0] = CreateThread(NULL, 0,
                                    Productor,
                                    NULL, 0, &ThreadID);
    hThreadVector[1] = CreateThread(NULL, 0,
                                    Consumidor,
                                    NULL, 0, &ThreadID);

    WaitForMultipleObjects(2, hThreadVector, TRUE, INFINITE);
    return (0);
}
```

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras

DLLs

Introducción
Linkado Explícito
Linkado Implícito
E/S Asíncrona
Bibliografía

Programación con Win32



DLLS

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs

Introducción

Linkado Explícito

Linkado Implícito

E/S Asíncrona
Bibliografía

Programación con Win32

Introducción

- La forma más común de construir un programa es:
 - compilar todos los fuentes y linkar los objetos, junto con las librerías que éstos utilizan
 - se genera un solo ejecutable, que contiene todo el código y que se carga completamente cada vez que se ejecuta el programa
 - Esta forma mono-módulo de construir un programa tiene varias desventajas
 - imágenes ejecutables muy grandes
 - las actualizaciones del programa requieren construir el programa completo
 - cada programa que utiliza un función común tendrá una copia en su ejecutable
 - no se puede ajustar dinámicamente el programa al entorno

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs

Introducción

Linkado Implícito

Linkado Explícito

E/S Asíncrona
Bibliografía

Programación con Win32

Introducción

- Las DLLs resuelven estos problemas:
 - Las funciones de librería no se linkan a la hora de construir el programa, sino al cargarlo (linkado implícito) o en tiempo de ejecución (linkado explícito)
 - Las DLLs se pueden utilizar para construir librerías compartidas
 - muchos programas comparten la misma librería
 - existe una sola copia en memoria
 - los programas mapean su espacio de direcciones al código de la DLL
 - cada hebra tiene su stack, con su propio entorno para la DLL
 - Con el linkado explícito cada programa puede decidir en tiempo de ejecución que funciones utilizar

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs

Introducción

Linkado Implícito

Linkado Explícito

E/S Asíncrona
Bibliografía

Programación con Win32

Introducción

- Las DLLs de una forma u otra son y han sido utilizadas en casi todos los sistemas operativos
 - librerías compartidas en Unix
 - todas las versiones de Unix para implementar parte del sist. operativo
- Las DLLs deben ser seguras para hebras, ya que corren en el espacio de direcciones del proceso llamante
- Una DLL puede exportar
 - Funciones
 - Variables

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

Introducción

Linkado Implícito

Linkado Explícito

E/S Asíncrona

Bibliografía

Programación con Win32

Linkado Implícito

- Se denomina también linkado en tiempo de carga (load time)
- Los pasos para construir una DLL de este tipo son los siguientes
 - Se abre el proyecto como DLL (en lugar de consola,...)
 - Al construir el programa genera un .LIB, que es un stub para el código real
 - este fichero debe ser copiado al directorio de librerías de proyecto que use la DLL
 - También se genera un fichero DLL que contiene la imagen ejecutable
 - se suele copiar en el mismo directorio de la aplicación
 - sino se encuentra se sigue la búsqueda habitual
 - se carga durante la inicialización

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- DLLs
 - Introducción
 - Linkado Implícito
 - Linkado Explícito
- E/S Asíncrona
- Bibliografía

Linkado Implícito

- Para construir la DLL es necesario importar y exportar los interfaces
- Se utilizan los modificadores
 - `_declspec(dllexport)`
 - `_declspec(dllimport)`
- Una forma habitual de hacerlo es construir un fichero de cabecera de la siguiente forma

```
#ifdef _DLLLIB
#define LIBSPEC _declspec (dllexport)
#else
#define LIBSPEC _declspec (dllimport)
#endif
LIBSPEC DWORD Mi_Función(...);
```

- En el código DLL se define `_DLLLIB`

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- DLLs
 - Introducción
 - Linkado Implícito
 - Linkado Explícito
- E/S Asíncrona
- Bibliografía

Ejemplo: Linkado Implícito

- Fichero hola.h

```
#ifdef _DLLLIB
```

```
#define LIBSPEC _declspec (dllexport)
```

```
#else
```

```
#define LIBSPEC _declspec (dllimport)
```

```
#endif
```

```
LIBSPEC void hola();
```

- Introducción
- Sist. Ficheros y E/S
- Excepciones
- Gestión de Memoria
- Seguridad
- Procesos y Hebras
- DLLs
 - Introducción
 - Linkado Implícito
 - Linkado Explícito
- E/S Asíncrona
- Bibliografía

Ejemplo: Linkado Implícito

- Fichero hola.c

```
#include <stdio.h>
#define _DLLLIB
#include "hola.h"
LIBSPEC void hola()
{
    printf("hola\n");
}
```

- El programa que la utiliza

```
#include "usa_dll.h"
#include <stdio.h>
int main( int argc, char *argv[])
{
    printf("Voy a DLL\n");
    hola();
    printf("Me voy de DLL\n");
}
```

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs

Introducción
Linkado Implícito

Linkado Explícito

E/S Asíncrona
Bibliografía

Programación con Win32

Linkado Explícito

- En este caso la DLL se carga explícitamente desde el programa
 - No se importa el fichero de cabecera de la librería
 - Se declaran punteros a función que posteriormente son instanciados, con lo cual se evita el linkado
- Las funciones principales para el manejo del linkado explícito son

```
HINSTANCE LoadLibrary (  
    LPCSTR  LpLibFileName           // nombre librería  
)  
  
HINSTANCE FreeLibrary (  
    HINSTANCE hLibModule           // manejador librería  
)  
  
FARPROC GetProcAddress (  
    HMODULE  hModule               // manejador librería  
    LPCSTR  lpProcName            // nombre función  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

Introducción

Linkado Implícito

Linkado Explícito

E/S Asíncrona

Bibliografía

Programación con Win32

Linkado Explícito

```
#include <windows.h>
#include <stdio.h>

typedef void (*lpFunc)(void);

int main( int argc, char *argv[])
{
    HINSTANCE hLibrary;
    lpFunc funcion;

    hLibrary = LoadLibrary("midll.dll");

    if (hLibrary == NULL){
        printf("Error en DLL: no encuentro la librería\n");
        return(1);
    }

    funcion = (lpFunc) GetProcAddress(hLibrary,"hola");

    if (funcion == NULL){
        printf("Error en DLL: no encuentro la función\n");
        return(2);
    }

    printf("Voy a DLL\n");
    (funcion)();
    printf("Me voy de DLL\n");
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

Introducción

Linkado Implícito

Linkado Explícito

E/S Asíncrona

Bibliografía

Programación con Win32

Linkado Explícito

- Las DLLs tienen un punto de entrada por defecto, que se utiliza para inicialización y terminación
- Existe una implementación por defecto, por lo que no es necesario incluirla, salvo que sea necesario
- El nombre real del punto de entrada es
 - `_DllMainCRTStartup`
 - Inicializa la librería de ejecución de C/C++ y llama a `_DllMain`
- Se puede redefinir `DllMain` en el código de usuario

Contenido

Introducción
Sist. Ficheros y E/S
Excepciones
Gestión de Memoria
Seguridad
Procesos y Hebras
DLLs

E/S Asíncrona

Introducción
E/S superpuesta
E/S extendida
E/S asínc. con hebras
Bibliografía

Entrada/Salida Asíncrona

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

Introducción

- Con el término e/s asíncrona hacemos referencia a las técnicas existentes para evitar la espera a la finalización de las operaciones
- Existen tres técnicas para realizar e/s asíncrona en Win32 (en NT, ya que en Windows 95 no se soporta la e/s sobre disco):
 - Utilización de hebras independientes para e/s
 - E/S superpuesta
 - La hebra continua su ejecución después de realizar la operación de e/s
 - Cuando necesita los datos espera en el manejador o en un evento especificado
 - E/S extendida (rutinas de continuación)

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S superpuesta

- El tipo **OVERLAPPED** ha sido utilizado en varias rutinas de e/s:
 - ReadFile
 - WriteFile
 - TransacNamedPipe
 - ConnectNamedPipe
- Cuando un fichero se abre en modo superpuesto, tiene que ser utilizado solo en ese modo
 - Para especificarlo se indicaba con el flag **FILE_FLAG_OVERLAPPED** en **CreateFile** o **CreateNamedPipe**
- La e/s superpuesta no funciona con las tuberías anónimas

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S superpuesta

- La e/s superpuesta tiene varias consecuencias:
 - Las operaciones de e/s no bloquean
 - El valor que devuelven las funciones no es valido
 - se necesita un mecanismo alternativo para indicar el resultado de la operación
 - El valor del número de bytes transferidos tampoco es válido
 - Se pueden realizar varias operaciones de e/s sobre el mismo descriptor al mismo tiempo, por lo que los punteros del fichero no tienen un significado válido
 - Deben existir mecanismos para sincronizar con la terminación de las operaciones
 - Si hay varias pendientes debemos ser capaces de identificar cual ha terminado

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S superpuesta

- La estructura utilizada para proporcionar información sobre la e/s es la siguiente:

```
typedef struct OVERLAPPED {  
    DWORD   Internal;           // reservada sistema  
    DWORD   InternalHigh;      // reservada sistema  
    DWORD   Offset;            // posición del fichero  
    DWORD   OffsetHigh;        // posición del fichero  
    HANDLE  hEvent;            // manejador de evento  
} OVERLAPPED
```

- El manejador del evento puede tener nombre o no, pero en cualquier caso debe ser un evento de reset manual
- El manejador del evento puede ser NULL. En ese caso hay que esperar sobre el manejador del fichero
- El evento se resetea cuando se efectua una operación y se señala cuando está ha finalizado

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S superpuesta

- El resultado de la operación se puede obtener con:

```
BOOL GetOverlappedResult (  
    HANDLE          hFile,           // manejador del fichero  
    LPOVERLAPPED    lpOverlapped,   // datos e/s superpuesta  
    LPDWORD         lpcbTransfer,    // bytes transferidos  
    BOOL            fWait            // esperar a fin  
)
```

- Si fWait no es TRUE se puede utilizar para testear de forma activa (polling) si la operación ha finalizado

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

Ejemplo: E/S superpuesta

```
int main( int argc, char *argv[])
{
    OVERLAPPED ov= {0,0,0,0,NULL}; // sin evento
    HANDLE hF;
    DWORD nread;
    BYTE Buffer[BUF_SIZE];
    BOOL foverlapped;
    ....
    hF= CreateFile(...,FILE_FLAG_OVERLAPPED,...);

    if (!ReadFile(hF,Buffer,sizeof(Buffer),&nRead,&ov));
        {
            if (GetLastError() != ERROR_IO_PENDING) {
                // otro error
                ExitProcess(0); }

            else
                // operación encolada
                fOverlapped = TRUE; }

    else
        // ya ha terminado
        fOverlapped = FALSE;

    if (fOverlapped) {
        WaitForSingleObject(hF, INFINITE);
        GetOverlappedResult (hF, &ov, &nRead, FALSE);
        // Aquí las variables si son válidas
    }

    ....
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Superpuesta

- Hay que tener cuidado con dos cosas:
 - Las estructuras OVERLAPPED no pueden ser reutilizadas hasta que acaba la operación, en caso contrario pueden producirse efectos imprevisibles
 - Necesitaremos una estructura por cada una de las e/s pendientes
 - Lo mismo ocurre con los buffers
- En ocasiones al realizar una e/s asíncrona nos puede seguir pareciendo síncrona:
 - No hay tiempo para realizar tareas antes de que llegue el resultado
 - Para detectarlo, siempre hay que comprobar el valor de Read.

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Superpuesta

- Existen varias causas que hay que tener en cuenta:
 - **Compresion**
 - Los ficheros NTFS comprimidos no se pueden acceder de forma
 - **Extensión de un fichero**
 - **Cache**
 - La mayoría de los drivers si encuentran los datos en cahe los devuelven
 - **Los datos no están en la cache y el manejador de cache esta saturado**
 - Bloqueará al proceso hasta que tenga hebras para leer la página disponibles
 - Hay tres hebras en un sistema con 16MB

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Superpuesta

- El comportamiento de la cache puede variar también si el fichero se accede secuencialmente o aleatoriamente. Para que funciones correctamente hay que indicarlo con el flag adecuado
 - **FILE_FLAG_SEQUENTIAL_SCAN**
 - **FILE_FLAG_RANDOM_ACCESS**
- El flag **FILE_FLAG_NO_BUFFERING** hace que todas las operaciones asíncronas sean realmente asíncronas, ya que deshabilita la cache.

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- Es otra de las formas de realizar e/s asíncrona sin utilizar hebras adicionales
- En este caso el sistema llama a una rutina una vez que ha terminado la operación
 - La rutina puede lanzar otra nueva operación, así como otras operaciones relacionadas con la finalización de la operación
 - Para especificar las rutinas de terminación es necesario utilizar operaciones de entrada/salida especiales

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- Rutinas de e/s extendida

```
BOOL WriteFileEx (  
    HANDLE hFile // manejador del fichero  
    CONST VOID *lpBuffer // buffer a escribir  
    DWORD nNumberOfBytesToWrite // max. bytes a escribir  
    LPOVERLAPPED lpOverlapped // para E/S asíncrona  
    LPOVERLAPPED_COMPLETION_ROUTINE lpocr // rutina de  
                                                terminación  
)
```

```
BOOL ReadFile (  
    HANDLE hFile // manejador del fichero  
    LPVOID lpBuffer // buffer en el que leer  
    DWORD nNumberOfBytesToRead // max. bytes a leer  
    LPOVERLAPPED lpOverlapped // para E/S asíncrona  
    LPOVERLAPPED_COMPLETION_ROUTINE lpocr // rutina de  
                                                terminación  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- La rutina de terminación debe tener el siguiente formato

```
VOID WINAPI RutinaTerminacion (  
    DWORD fdwError,          // Éxito (0) o ERROR_HANDLE_EOF  
    DWORD cbTransferred,    // bytes transferidos  
    LPOVERLAPPED lpO        // estructura E/S asíncrona  
)
```

- Para que la rutina sea llamada:
 - La operación de e/s debe haber terminado
 - La hebra que ha invocado la operación debe estar en un estado de **espera alertable**
- En este estado la hebra notifica al sistema que puede ejecutar todas las llamadas a rutinas de terminación pendientes

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- Para entrar en este estado hay que utilizar alguna de las siguientes funciones con **fAlertable** a **TRUE**

```
DWORD WaitForSingleObjectEx (  
    HANDLE hObject,           // manejador proceso  
    DWORD dwTimeOut          // timeout  
    BOOL falertable           // estado alertable  
)
```

```
BOOL WaitForMultipleObjectsEx (  
    DWORD cObjects,           // numero de objetos  
    LPHANDLE lphObject,      // array de procesos  
    BOOL fWaitAll,           // esperar a todos?  
    DWORD dwTimeOut          // timeout  
    BOOL falertable           // estado alertable  
)
```

```
DWORD SleepEx (  
    DWORD dwTimeOut          // timeout  
    BOOL falertable           // estado alertable  
)
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- Estas funciones retornarán cuando
 - Pase el timeout o sus manejadores sean señalados
 - Si **fAlertable** está a **TRUE**, después de ejecutar todas las rutinas que terminación pendientes
- Los eventos de las operaciones de espera no tienen nada que ver con las operaciones de e/s
- Cuando las operaciones de e/s acaban, las rutinas son encoladas para su ejecución, con los parámetros adecuados
- Son ejecutadas secuencialmente, pero no en el orden en que acabaron

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

- Para saber si se ha ejecutado alguna rutina de terminación
 - `SleepEx` devuelve `WAIT_IO_Completion`
 - `Get_Last_Error` devuelve el mismo valor si se ejecuta despues de alguna de las funciones wait
- Para obligar a que se ejecuten las rutinas sin esperar se puede especificar un timeout 0 en las llamadas
- El parámetro `hEvent` de la estructura de solapamiento se puede utilizar para mandar información a la rutina de terminación

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S Extendida

```
int main( int argc, char *argv[])
{
    OVERLAPPED ov[4];
    HANDLE hF;
    DWORD nread,ndone;
    BYTE Buffer[4][BUF_SIZE];
    BOOL foverlapped;
    ....
    hF= CreateFile(...,FILE_FLAG_OVERLAPPED,...);

    for (i=0;i<3;i++) {
        ov[i].hevent = (HANDLE) i+1;
        ov[i].Offset = i*BUF_SIZE
        ReadFileEx(hf,Buffer[i],sizeof(Buffer),&nRead,&ov[i],termina));
    }

    ....
    SleepEx(0,TRUE);
}

VOID WINAPI termina(DWORD Code, DWORD Nbytes, LPOVERLAPPED ov)
{
    DWORD ic;
    ....
    ic = (DWORD) ov->hEvent;
    ....
}
```

Contenido

Introducción

Sist. Ficheros y E/S

Excepciones

Gestión de Memoria

Seguridad

Procesos y Hebras

DLLs

E/S Asíncrona

Introducción

E/S superpuesta

E/S extendida

E/S asínc. con hebras

Bibliografía

Programación con Win32

E/S asíncrona con hebras

- Las dos técnicas anteriores realizan e/s asíncrona en la misma hebra
- Otra forma de llevarlo a cabo es crear varias hebras, que lleven a cabo operaciones síncronas
- Hay que utilizar un manejador de fichero para cada hebra (se puede utilizar **DuplicateHandle**)
- Es la forma más simple de programar y la única que se puede utilizar en Windows95
- La eficiencia de todos los métodos es comparable