Symbian/C++

Aplicaciones Móviles
Curso de Adaptación
Grado en Ingeniería de Sistemas Audiovisuales

Celeste Campo - Carlos García Rubio
celeste, cgr@it.uc3m.es

Índice

- Introducción.
- Entorno de desarrollo (S60 Platform SDK y Carbide.c++).
- Ficheros que se generan.
- Symbian C++.
  - Tipos de datos.
  - Convenciones de nombrado.
  - Manejo de excepciones y gestión de recursos.
  - Descriptores.
  - Objetos activos.
- Open C/C++ y Qt.
- Referencias.

Introducción

- Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil.
  - Nokia, Sony Ericsson, PSION, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc.
  - Sus orígenes provienen de su antepasado EPOC32, utilizado en PDA's.
  - En 2008, Nokia compró Symbian tras un acuerdo con el resto de socios.
  - Estableció la Fundación Symbian para convertir este sistema operativo en una plataforma abierta: http://www.symbian.org/
- Symbian cuenta con cinco interfaces de usuario o plataformas para su sistema operativo:
  - Serie 60, Serie 80, Serie 90. Usadas por la mayoría de los móviles con Symbian y elegidas por la Symbian Fundation como plataformas a continuar.
  - UIQ. Usada principalmente por Sony-Ericsson y Motorola, desarrollada por UIQ Technology.
  - MOAP. Usada por algunos móviles 3G de NTT-Docomo.

Plataforma S60

- Plataforma S60 es el nombre actual del interfaz de usuario S60, y es el más usado.
- Proporciona un conjunto de bibliotecas y aplicaciones al desarrollador, sobre el SO Symbian.
- Las aplicaciones se desarrollan de forma nativa en C++ para S60.
  - Standard C++ es diferente de Symbian OS C++.
    - Algunas características de C++ son demasiado complejas para un móvil.
    - Los tipos de datos no están definidos de forma exacta.
    - No soporta DLLs ni (en el momento de crearse Symbian) excepciones.
    - Diferentes requisitos de manejo de errores, eficiencia…
- Para el desarrollo se usa un IDE: Carbide.c++ (basado en Eclipse).

Instalación del entorno de desarrollo

1. Instalar el IDE Carbide.c++:

   – Descargarlo de http://www.forum.nokia.com/info/sw.nokia.com/id/dbb8841d-832c-43a6-be13-f78119a2b4cb.html
   – Ejecutar para iniciar la instalación.

2. Instalar el S60 Platform SDK for Symbian OS:
   – Descargar la última versión del SDK C++ de http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html
   – Descomprimir el fichero zip.

S60 Platform SDK for Symbian OS

- Permite desarrollar aplicaciones escritas en C++ para dispositivos S60.
- Incluye:
  - Herramientas.
  - APIs.
  - Bibliotecas.
  - Documentación.
  - Ejemplos.
  - Compilador GNU C Compiler Embedded (GCCE).
  - Emulador (`epoc.exe`).
- Hay distintas versiones para las distintas ediciones de la plataforma S60.
  - Se puede ver en http://www.forum.nokia.com/ qué versión de SDK le corresponde a cada terminal: http://www.forum.nokia.com/devices/matrix_all_1.html

Ediciones S60

- Ver http://en.wikipedia.org/wiki/S60_%28software_platform%29

S60 edition	S60 version number	Symbian OS version number	Devices
S60, version 0.9	0.9	6.1	•Nokia 7650
S60 1st Edition	1.2	6.1	•Nokia 3650
S60 2nd Edition	2.0	7.0s	•Nokia 6600
S60 2nd Edition, Feature Pack 1	2.1	7.0s	•Nokia 7610
S60 2nd Edition, Feature Pack 2	2.6	8.0a	•Nokia 6680
S60 2nd Edition, Feature Pack 3	2.8	8.1a	•Nokia N70 •Nokia N72 •Nokia N90

Ediciones S60

- Ver http://en.wikipedia.org/wiki/S60_%28software_platform%29

S60 edition	S60 version number	Symbian OS version number	Devices
S60 3rd Edition	3.0	9.1	•Nokia E61 •Nokia N73 •Nokia N80
S60 3rd Edition, Feature Pack 1	3.1	9.2	•Nokia E71 •Nokia N82 •Nokia N95
S60 3rd Edition, Feature Pack 2	3.2	9.3	•Nokia N96
S60 5th Edition	5.0	9.4	•Nokia 5800 •Nokia N97

¿Qué versión del SDK elegir?

- Si vamos a desarrollar una aplicación para un terminal concreto, usar la versión del SDK S60 que corresponda a la edición de la plataforma del terminal.
  - Por ejemplo, si queremos desarrollar una aplicación para el Nokia 5800 XpressMusic, usaremos el S60 5th Edition SDK.
- Con un mismo SDK se puede desarrollar para más de una plataforma.
  - Por ejemplo, el S60 3rd Edition Maintenance Release SDK (versión 1.1) permite desarrollar aplicaciones para móviles S60 3rd Edition FP1 y FP2.
- A partir de la S60 3rd Edition, los binarios son compatibles.
  - Aplicaciones desarrolladas con una versión del SDK (suelen) ejecutar sin problemas en ediciones de plataforma posteriores.
  - Por ejemplo, una aplicación desarrollada con el 3rd Edition FP1 SDK puede instalarse en un móvil S60 3rd Edition FP1, S60 3rd Edition FP2 ó S60 5th Edition.
    - No compatibles con S60 2nd Edition y anteriores.
- Para máxima compatibilidad, es habitual usar el SDK de 3rd Ed.

Carbide.c++

- Entorno de desarrollo (IDE) basado en Eclipse.
- Incluye:
  - Plantillas para generar esqueletos de aplicaciones tipo.
  - *UI Designer*, que permite desarrollar interfaces gráficos con entorno de desarrollo *drag-and-drop*.
  - *CodeScanner*, que analiza el código para detectar problemas potenciales, como de memoria, cumplimiento de las reglas de codificación Symbian, etc.
  - Compilación y empaquetado.
  - Depuración, incluyendo depuración en terminal.
  - *Performance Investigator*, para analizar consumo de memoria, procesador y batería.
- Carbide.c++ usa las herramientas por línea de comando del SDK S60.

Compilador

- Dependiendo de si se construye la aplicación S60 para el emulador o para un terminal real, hay que seleccionar en la configuración de construcción (*build configuration*):
  - WINSCW para usar en el emulador.
    - Usa el compilador Nokia C/C++ que viene incluido en Carbice.c++.
  - GCCE para usar en un móvil real.
    - Usa el compilador gratuito GNU C Compiler (GCC), incluido en el S60 SDK, que contruye binarios de acuerdo al ARM Embedded Application Binary Interface (ARM EABI).
  - ARMV5 para usar en móvil real.
    - Usa un producto comercial, no incluido en el S60 DSK, el RealView Developer Suite, que mejora las prestaciones para procesasores ARM.

Ficheros que se generan

- En el proceso de desarrollo de una aplicación Symbian C++ S60 se crean los siguientes ficheros:

1. Ficheros de configuración de la construcción:
   - bld.inf: usado por herramienta bldmake para generar los ficheros abld.bat y make.
   - Fichero .mmp: información del proyecto.
   - Ficheros .mk: makefiles para otras funciones, como convertir imágenes .svg a ficheros .mif.

Ficheros que se generan

2. Ficheros de configuración de la construcción:
   - bld.inf: usado por herramienta bldmake para generar los ficheros abld.bat y make.
   - Fichero .mmp: información del proyecto.
   - Ficheros .mk: makefiles para otras funciones, como convertir imágenes .svg a ficheros .mif.

Ficheros que se generan

3. Ficheros fuente:
   - Código fuente:
     - .h: cabeceras, .cpp: clases y .pan: enumeración de códigos de pánico.
   - Ficheros de recursos para definir componentes del GUI:
     - .rls: cadenas de caracteres para localización
     - .rss: ficheros de recurso usados por los componentes gráficos de la aplicación
     - .hrh: de cabeceras de recurso.
   - Ficheros gráficos:
     - .bmp (bitmap) y .svg (SVG-tiny, vectorial).
   - Ficheros de registro: información para registrar la aplicación en el teléfono y hacerla visible al sistema operativo.
     - .xml y _reg.rss

Ficheros que se generan

4. Ficheros de paquete:
   - .pkg: indica a la herramienta de creación de ficheros .sis qué ficheros tiene que usar, dónde están en el PC y dónde van en el móvil.
5. Ficheros de salida:
   - Ejecutables:
     - .exe para aplicaciones GUI.
     - .dll (dynamic link library) para bibliotecas compartidas.
   - Ficheros de recursos compilados:
     - .rsc y .rsg a partir del .rss
   - Ficheros de gráficos:
     - .mif a partir de los .svg.
     - .mbm a partir de los .bmp.
     - .mbg de cabeceras de ambos.
   - Ficheros de registro compilados:
     - _reg.rsc

Ficheros que se generan

6. Ficheros de instalación:
   - .sis
   - .sisx (firmado).
     - Todas las aplicaciones deben firmarse antes de instalarse.
     - El firmado indica a qué capacidades (*capabilities*) puede acceder la aplicación.
       - *user capabilities*: LocalServices, NetworkServices, ReadUserData, WriteUserData, UserEnvironment and, Location
       - *system capabilities*: PowerMgmt, ProtServ, ReadDeviceData, SurroundingsDD, SwEvent, TrustedUI, WriteDeviceData
       - *restricted capabilities*: CommDD, DiskAdmin, MultimediaDD and NetworkControl
       - *device manufacturer capabilities*: AllFiles, DRM and TCB
     - Dos opciones:
       - Auto firmado por el usuario que lo instala.
         - Sólo válido para *user capabilities*.
       - Firmado con un certificado por el vendedor.
       - Ambos procedimientos se pueden ver en https://www.symbiansigned.com

Symbian C++

- Como ya hemos comentado, Symbian OS C++ es diferente de Standard C++ .
- Vamos a ver:
  - Tipos de datos.
  - Convenciones de nombrado.
  - Manejo de excepciones y gestión de recursos.
  - Descriptores.
  - Objetos activos.

Symbian C++: Tipos de datos

- Define tipos de datos propios, para ser 100% independiente del compilador.

Standard ANSI	Symbian OS	Description
int	TInt	Signed (32-bit) Integer
unsigend int	TUint	Unsigned (32-bit) Integer
float	TReal32	Single-precision IEEE 754 floating-point
double	TReal, TReal64	Double-precision IEEE 754 floating-point
Long	TInt64	Uses native 64-bit support
bool	TBool (ETrue, False)	Equates to int due to early compilers
void*	TAny*	“Pointer to anything”

  - Otros: TText[8|16], TInt[8|16|32], TUint[8|16|32], TUint64.
  - Observar que todos empiezan por T.

Symbian C++: Convenciones de nombrado

- Objetivo:
  - Hacer el código más autoexplicativo.
  - Facilita la gestión de recursos y el manejo de excepciones.
- ¿Cómo?:
  - Ciertas características y comportamiento de una clase o función se refleja en su nombre (p.ej. la letra inicial del nombre).

Symbian C++: Convenciones de nombrado

- Clases T:
  - Una clase cuya primera letra es una T es una clase simple.
    - No usa memoria *heap*.
    - No necesita destructor.
  - Ejemplo:

```
class TCoordinate
    {
    public
        TInt iX;
        TInt iY;
    };
```

  - Hay que ser prudente usando clases T, porque pueden consumir mucha memoria.
  - Los tipos básicos también llevan el prefijo T (ver transparencia anterior).

Symbian C++: Convenciones de nombrado

- Clases C:
  - El prefijo C viene de "*cleanup*" (limpiar), e indica que deben usarse con cuidado para evitar pérdidas de memoria.
  - Derivan directamente de la clase `CBase` o de otra clase C.
    - No se puede derivar de más de una clase C.
  - Ejemplo:

```
CMyExample : public CBase
    {
    TInt iNumber;
    //...
    public:
        ~CMyExample();
    };
```

Symbian C++: Convenciones de nombrado

- Clases R:
  - Clases especiales que se usan para manejar recursos.
    - Habitualmente se instancian en la pila (*stack*) o dentro de una clase C en el *heap*.
  - Típicamente debe abrirse y cerrase de alguna manera antes de poder usarlas, por ejemplo con unas llamadas `Connect()` y `Close()`.
    - Es importante recordar cerrar una clase R, si no se producirá pérdida de memoria.
  - Ejemplo:

```
CFileAccess : public CBase
    {
    // R-class, through which we connect to the file server
    RFs iFileServer;
    ...
    public:
        ~CFileAccess()
            {
            iFileServer.Close();
            }
    };
```

Symbian C++: Convenciones de nombrado

- Clases M:
  - Definen un conjunto de interfaces abstractas que consisten únicamente en funciones virtuales.
  - Son las únicas clases que se pueden usar en Symbian junto con herencia múltiple.
  - Ejemplo:

```
class MNotify
{
    virtual void HandleEvent(TInt aEvent) = 0;
}
```

Symbian C++: Convenciones de nombrado

- Estructuras (S):
  - Las estructuras (*struct*) típicas de C ó C++, sin funciones miembro, deben ir precedidas de la letra S.
  - No se usan mucho en Symbian.
  - Ejemplos:

```
struct SEikControlInfo;
struct SEikRange;
struct SEikDegreesMinutesDirection;
```

Symbian C++: Convenciones de nombrado

- Variables miembro (i):
  - Las variables miembro de una clase deben empezar con la letra i (minúscula).
    - Viene de “instancia”.
  - Ejemplo:

```
CMyExample : public CBase
    {
    TInt iNumber;
    TChar iLetter;
    //...
    };
```

- Variables automáticas:
  - Deben comenzar por una letra minúscula.

Symbian C++: Convenciones de nombrado

- Nombres de funciones:
  - Deben empezar por una letra mayúscula, por ejemplo `ShowPath(TFileName &aPath)`.
  - Aquellas funciones que pueden hacer que se salga deben finalizar con una L (de "*leave*"), por ejemplo `NewL()`.
    - Posteriormente comentaremos los "*leave*".
  - Aquellas funciones que meten punteros en el stack de limpieza ("*cleanup*"), deben finalizar con una letra mayúscula C, por ejemplo `NewLC()`.
    - Posteriormente comentaremos los stacks "*cleanup*".
  - Aquellas que terminan en D indican que borran el objeto que las llama, por ejemplo `ExecuteLD()`.

Symbian C++: Convenciones de nombrado

- Argumentos de las funciones (a):
  - Deben empezar por una a minúscula.
  - Ejemplo:

```
TInt CFileConnect::ShowPath(TFileName &aPath)
    {
    return iFs.DefaultPath(aPath);
    }
```

- Constante (K):
  - Las constantes deben empezar con una K mayúscula.

```
const TInt KNumber = 1982;
```

Symbian C++: Convenciones de nombrado

- Namespaces (N):
  - Se usan para agrupar funciones que no pertenecen a ninguna clase.
  - Comienzan por N.
  - Ejemplo:

```
namespace NMyFunctions
{
    ...
}
```

- Enumerados:
  - Los tipos enumerados son tipos, y como tales deben comenzar por T.
  - Sus miembros deben comenzar por E.
  - Ejemplo:

```
enum TGames = {EMonopoly, ETetris, EChess = 0x05};
```

Symbian C++: Gestión de recursos

- En las aplicaciones móviles necesitamos gran robustez y fiabilidad.
- Por tanto es vital hacer una gestión efectiva de la memoria.
  - Los móviles pueden estar encendidos durante días o semanas.
  - Memoria limitada.
- ¡Podemos tener pérdidas de memoria!
- El manejo de excepciones y la gestión de recursos en Symbian están relacionados con esto.
- Las excepciones se usan para avisar en tiempo de ejecución que un código no está realizando la tarea que tiene asignada, por ejemplo por escasez de recursos.
  - Mecanismo más ligero que en Standard C++, basado en mecanismos de *leave* y *trap*.
    - Un *leave* es parecido a un *throw* de C++, y un *trap* a un *catch*.

Symbian C++: Gestión de recursos

- Funciones *leave* (salir):
  - Un *leave* pude ocurrir porque:
    - Se llama a otra función que hace *leave*.
    - Se llama explícitamente a la función de sistema `User::Leave()`.
  - Un *leave* lleva un código de error que indica la causa.
    - Hay muchos definidos en el fichero `\Epoc32\include\e32err.h`.
      - Por ejemplo: `KErrNotFound = -1`, `KErrGeneral = -2`, `KErrCancel = -3`, etc.
  - Las funciones que pueden hacer *leave* se identifican porque terminan por L, lo que nos permite en el código que las llama tomar medidas para tratar adecuadamente ese *leave*.
    - La ejecución del programa para al llamarse al *leave*, y se devuelve al *trap* desde el que se llamó a la función.

Symbian C++: Gestión de recursos

- *Traps* (trampas).
  - TRAPD(var, func).
  - Ejemplo:

```
TRAPD( err , DoMagicTrickL () );
if ( err != KErrNone )
   {
   // Something went wrong
   }
```

  - Cuando ocurre el *leave*, el control se pasa inmediatamente al *trap* desde el que se le hizo la llamada.
    - Se pueden anidar varias llamadas *trap*, pero se debe intentar no hacerlo por consumo de recursos.
    - Además como sólo se devuelve un número de error (TInt) es difícil saber la causa del error (ver siguiente transparencia).

Symbian C++: Gestión de recursos

- Reserva de memoria:
  - Cuando reservamos memoria con el operador `new()` es posible que la operación falle por falta de memoria en el teléfono.
  - Tendríamos que probarlo del siguiente modo:

```
CSomeObject *ptr= new CSomeObject;
if ( ptr== 0)
    {
    User::Leave(KErrNoMemory) ;
    }
```

  - Como esto es tedioso, el operador `new()` toma el parámetro `ELeave` que saldrá con un error `KErrNoMemory`:

```
CSomeObject *ptr= new (ELeave)CSomeObject;
```

Symbian C++: Gestión de recursos

- *Cleanup stack* (la pila de limpieza):
  - Problema:
    - Cuando se produce un *leave*, el control a la primera (si hay varias anidadas) *trap* que lo llamó.
    - Pueden quedarse objetos en memoria sin liberar.
  - Ejemplo:

```
TRAPD(error,DoExampleL());
    ...
void DoExampleL()
{
    // Allocating memory for the first object
    CSomeObject* myObject1=new (ELeave) CSomeObject;
    // Allocating memory for second...What happens if memory
    // is not enough...
    CSomeObject* myObject2=new (ELeave)CSomeObject;
    delete myObject1; // These lines will never be executed
    delete myObject2; // because leave occurred.
}
```

    - No se libera la memoria de `myObject1`.

Symbian C++: Gestión de recursos

- *Cleanup stack* (la pila de limpieza):
  - Solución:
    - El *cleanup stack* es una pila donde podemos almacenar punteros de objetos a los que hemos asignado memoria (como en el ejemplo anterior `myObject1`), de manera que si ocurre un *trap*, se libera la memoria de los objetos que están en esa pila.

Symbian C++: Gestión de recursos

- *Cleanup stack* (la pila de limpieza):
  - Para almacenar un objeto en esta pila de limpieza:
    - `CleanupStack::PushL(myObject1)`
  - Para sacar de la pila el último elemento añadido :
    - `CleanupStack::Pop()`
  - Para sacar el último elemento añadido y liberar la memoria:
    - `CleanupStack::PopAndDestroy()`
  - El ejemplo anterior quedaría (siguiente transparencia):

Symbian C++: Gestión de recursos

- *Cleanup stack* (la pila de limpieza):

```
TRAPD(error,DoExampleL());
    ...
void DoExampleL()
{
    // We create objects and put them to the cleanup stack
    CSomeObject* myObject1=new (ELeave) CSomeObject;
    CleanupStack::PushL( myObject1 );
    // If out of memory occurs next, we have myObject1 in the
    // cleanup stack so it is freed automatically
    CSomeObject* myObject2=new (ELeave)CSomeObject;
    CleanupStack::PushL( myObject2 );
    // At the end we can take objects from the cleanup stack
    // and free the memory.
    CleanupStack::Pop();
    CleanupStack::Pop();
    delete myObject1;
    delete myObject2;
    // The previous four lines could be replaced by
    // CleanupStack::PopAndDestroy(); // Pop and delete
    // CleanupStack::PopAndDestroy(); // Pop and delete
}
```

Symbian C++: Gestión de recursos

- Construcción en dos fases:
  - Una clase C contendrá habitualmente punteros a otras clases C, de las que será responsable de reservar y liberar memoria.
  - Ejemplo:

```
class CMyCompoundClass: public CBase
{
public:
    CMyCompoundClass();
    ~CMyCompoundClass();
    ...
private:
    CMySimpleClass* iSimpleClass1; // owns an instance of
                                   // another class
};
```

Symbian C++: Gestión de recursos

- Construcción en dos fases:
  - No es seguro hacer lo siguiente:

```
CMyCompoundClass::CMyCompoundClass()
{
    iSimpleClass1 = new (ELeave)CMySimpleClass();
}
```

  - Cuando construimos el objeto con:

```
CMyCompundClass *ptr= new (ELeave)CMyCompundClass();
```

    1. Se reserva memoria para `CMyCompoundClass`.
    2. Se llama al constructor de `CMyCompoundClass`.
    3. Se reserva memoria para `CMySimpleClass` y se llama a su constructor.
    4. Se almacena un puntero a `CMySimpleClass` en `iSimpleClass1`.
    5. La construcción está finalizada.

Symbian C++: Gestión de recursos

- Construcción en dos fases:
  - Si al construir el objeto anterior fallase el paso 3, se volvería al *trap* y no se destruiría el objeto `CMyCompoundClass` parcialmente creado.
  - Para evitar este problema, en Symbian se usa un paradigma mejor de construcción de objetos: la construcción en dos fases.
  - La idea es mover la construcción de objetos dentro del nuestro que puedan dar lugar a un *leave* a otra función, que por convención se llama `ConstructL()`.

```
void CMyCompoundClass::ConstructL()
{
    iSimpleClass1 = new (ELeave)CMySimpleClass();
}
```

Symbian C++: Gestión de recursos

- Construcción en dos fases:
  - Ahora haríamos:

```
CMyCompundClass *ptr= new (ELeave)CMyCompundClass();
CleanupStack::PushL(ptr);
// Finish the construction
ptr->ConstructL();
CleanupStack::Pop();
```

  - Por conveniencia se suelen crear dos funciones `NewL()` y `NewLC()`:

Symbian C++: Gestión de recursos

- Construcción en dos fases:

```
CSomeObject* CSomeObject::NewLC()
    {
    CSomeObject *self = new (ELeave) CSomeObject;
    CleanupStack::Push(self);
    self->ConstructL();
    return self;
    }

CSomeObject* CSomeObject::NewL()
    {
    CSomeObject *self = CSomeObject::NewLC();
    CleanupStack::Pop (self);
    return self;
    }
```

  - de manera que se hace lo siguiente:

```
CSomeObject *ptrSomeObject = CSomeObject::NewL();
```

Symbian C++: Descriptores

- Las clases *descriptor* son como los *strings* (cadenas de caracteres) de Symbian, pero también se usan para guardar datos binarios (arrays de bytes).
  - Hay descriptores de 8 bits y de 16 bits
    - Par datos binarios usaremos los de 8.
    - Para texto unicode usaremos los de 16.
  - Si no se dice explícitamente (p.ej. `TDesC`) se entiende que es el de 16, pero se puede decir explícitamente (p.ej. `TDesC8, TDesC16`).
- Hay tres tipos de descriptores:
  - *Pointer descriptors*.
  - *Buffer descriptors*.
  - *Heap descriptors*.
- Es muy importante comprender cómo se manejan los descriptores, porque se usan mucho en el API de Symbian.
- Jerarquía de descriptores:
  - Todos derivan de la clase base `TDesC`.

Symbian C++: Descriptores

- En `TDesC`, que el nombre del tipo termine por C indica que su contenido no se puede modificar, es constante.
- Si queremos un descriptor que podamos modificar, usaremos `Tdes`.
- *Buffer descriptors*:
  - `TBuf<n>` y `TBufC<n>`.
  - El <n> indica el tamaño máximo, y si no se indica se determina en tiempo de compilación.
  - Almacena los datos y junto con su descripción
  - `TBuf<n>` y `TBufC<n>` sólo se deben usar para datos pequeños.
  - Por ejemplo:
    - Definimos un tipo `TFileName` como `TBufC<256>` para almacenar nombres de fichero (con camino) de hasta 256 caracteres.
      - Ocupa 512 bytes.
    - Si lo pasamos como parámetro a funciones, ocupará 512 bytes en la pila.
    - Esto es demasiado: cuando usamos 8 KBytes de pila el SO da un *panic* y mata nuestro hilo (lo considera un error de programación).
  - Hace falta otro tipo para datos grandes.

Symbian C++: Descriptores

- *Pointer descriptors*:
  - `TPtr` y `TPtrC`.
  - Almacenan la referencia a dónde están los datos.
    - No son propietarios de esos datos, pueden apuntar a cualquier zona de memoria: ROM, stack, heap…
- *Heap descriptors*:
  - `RBuf` y `HBufC`.
  - Almacena los datos en el *heap*.
  - Se usan normalmente para datos muy grandes o cuando a priori no se puede saber su tamaño.
  - Pequeña diferencia entre `RBuf` y `HBufC`:
    - `RBuf` está en el stack, sólo sus datos están en el heap.
      - Es una clase R, es propietaria de los datos que están en el heap.
    - `HBufC` está en el *heap*.
      - De ahí que su nombre empiece por H, aunque es una clase R, es propietaria de los datos en el *heap*.

Symbian C++: Descriptores

- Descriptores como argumento de funciones:
  - Si se quiere pasar una cadena como argumento a una función, se debe usar:
    - `TDes` si se desea que la función pueda modificar su valor.
    - `TDesC` si se desea que no pueda cambiarlo.

```
void modifyThis(TDes &aDes );
void doNotModifyThis(const TDesC &aDes);
```

Symbian C++:
Objetos activos

- Los objetos activos permiten tener multitarea en un sistema operativo como Symbian OS:
  - Con recursos muy limitados.
  - Fuertemente orientado a eventos.
- La mayoría de las aplicaciones son también orientadas a eventos:
  - El usuario pulsa tal tecla.
  - Un servidor me envía un resultado.
  - …
- Cuando una aplicación pide un servicio (p.ej. enviar un SMS), este servicio puede completarse de forma síncrona o asíncrona.
  - Síncrono: similar a una llamada a función (finaliza cuando devuelve el resultado).
  - Asíncrono: primero finaliza la llamada, después llega el resultado.
    - Mientras tanto la aplicación puede hacer otras cosas.
    - Nos notifica cuando llega el resultado.
    - Esto en dispositivos no limitados se hace con hilos, aquí muy costoso.

Symbian C++:
Objetos activos

- Los objetos activos permiten tener multitarea con un solo hilo.
  - Derivan de la clase `CActive`.
- Hay un *active scheduler* (`CActiveScheduler`) que mantiene una lista de todos los objetos activos a la espera de una respuesta de un servicio.
  - Todos los eventos de finalización que se reciben de servidores se entregan a este planificador.
  - Cuando recibe un evento, el planificador lo envía al objeto activo adecuado (el que solicitó el servicio).

Open C/C++

- Open C/C++ es una solución que permite portar fácilmente código C y C++ desarrollado par Linux / UNIX a Symbian.
  - Han portado las librerías C más utilizadas a Symbian (PIPS = PIPS is POXIS on Symbian)
    - POSIX (Especifica las interfaces de usuario y software al sistema operativo en la familia UNIX).
    - OpenSSL (biblioteca relacionadas con la criptografía).
    - zlib (bibliotecas de compresión de datos).
    - GLib (biblioteca de propósito general).
  - No incluye librerías de interfaz de usuario, pero el código se puede llamar desde código Symbian C++.
- A partir del SDK del S60 3rd Edition FP2 forma parte del SDK (para versiones anteriores hay que descargar un plug-in).

Open C/C++

- Librerías:
  - Libc: librerías estándar de C, incluyendo rutinas de entrada salida estándar, operaciones con bit, con strings, DES, funciones de tiempo, sockets, IPC…
  - Libm: funciones aritméticas y matemáticas.
  - Libpthread: pthear API de POSIX.
  - Libz: funciones de compresión y decompresión.
  - Libdl: permite crear DLLs
  - Libcrypto: funciones criptográficas de OpenSSL: S/MIME, SSH, OpenPGP, etc.
  - Libssl: implementación de SSL y TLS de OpenSSL.
  - Libcrypt: funciones de encriptado/desencriptado.
  - Libglib: tipos de datos útiles, funciones de conversión, utilidades de ficheros, de cadenas…

Qt

- Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.
- Inicialmente desarrollada por la empresa noruega Trolltech (en aquel momento «Quasar Technologies») siguiendo un desarrollo basado en el código abierto.
- En 2008 la compró Nokia.
- Sigue siento distribuida bajo los términos de GNU Lesser General Public License (y otras), Qt es software libre y de código abierto.
- Qt se ha usado para desarrollar aplicaciones como KDE, Google Earth, etc.
- Nokia ha desarrollado recientemente Qt for Symbian y Qt for maemo, y pretende que con ello se pueda desarrollar fácilmente aplicaciones para el PC de sobremesa (Apple, Microsoft y Linux) y para móvil (Symbian y maemo).

Referencias

- “Symbian/C++”, de Morten V. Pedersen y Frank H.P. Fitzek. Capítulo 4 del libro: "Mobile Phone Programming and its Application to Wireless Networking". Fitzek, Frank H. P. and Reichert, Frank (Editors). Springer 2007. http://books.google.es/books?id=s_OnKP3VAQ4C&lpg=PA126&ots=N59P-eqDCP&dq=%22requesting%20a%20service%20from%20one%20of%20the%20Symbian%20OS%20system%22&pg=PA95#v=onepage&q=%22requesting%20a%20service%20from%20one%20of%20the%20Symbian%20OS%20system%22&f=false
  (el capítulo 5 de este mismo libro trata de Open C).
- Sección “Getting started”, del Nokia S60 5th Edition C++ Developer's Library v2.1, http://library.forum.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-88FD209C-10EB-43B3-A5AE-97B7F6AE9C3C.html
- “Symbian OS: Python/PyS60”, Andreas Jakl, Marzo 2009, http://symbianresources.com/tutorials/general.php
- Aalborg University, Mobile Device Group http://mobiledevices.kom.aau.dk/