

Programación de Sistemas y Concurrencia

Tema 1: Introducción a la Programación de Sistemas

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores



Índice de contenidos

- Programación de sistemas y sistemas operativos
- Programación de sistemas de caja de negra:
lenguajes de script
- Lenguajes para programación de sistemas
- Gestión de actividades concurrentes: procesos y eventos

Programación de Sistemas

- El concepto de **programación de sistemas** tiene sentido frente al de programación de aplicaciones
- En la **programación de aplicaciones**, el objetivo es producir software para proporcionar **servicios al usuario** (por ej. Un procesador de textos)
- En la **programación de sistemas** el software proporciona servicios al hardware, al sistema operativo o a otros componentes del sistema

Programación de Sistemas

- En la programación de sistemas el grado de abstracción con respecto al hardware y la forma en la que se ejecutan las aplicaciones es menor.
- Un ejemplo de programación de sistemas es la programación de sistemas empotrados/embebidos
- En estos sistemas los computadores forman parte de otros sistemas más complejos, que controlan



Programación de Sistemas

- Algunas características que hacen a la programación de sistemas “especial”:
 - Se interacciona con el mundo físico a través del hardware
 - En esta interacción hay que tener en cuenta factores como:
 - La fiabilidad y la criticidad de los sistemas. Los sistemas deben ser **predecibles**
 - Los recursos (memoria, almacenamientos, comunicaciones,..) son limitados y hay que tenerlos en cuenta al programar
 - El tiempo físico
 - Los ordenadores son inherentemente secuenciales y en el tiempo discreto
 - En el mundo real, el tiempo es continuo y existe la simultaneidad de eventos
 - La interacción con el entorno esta sujeta a restricciones de **tiempo real**

Programación de Sistemas

- ¿Cómo interactuamos con el entorno?
 - A través del **sistema operativo**, que a su vez controla el hardware
 - El sistema operativo controla el hardware y nos proporciona abstracciones de más alto nivel a través de Interfaces de Aplicación
 - Gestiona aspectos como el tiempo, la concurrencia, etc., para que sea más fácil
 - Directamente sobre el hardware “desnudo”. Esto es así en algunos sistemas empotrados con recursos muy limitados (el sistema operativo no cabe) o en el caso de que trabajemos en las primeras capas de un sistema operativo.

Programación de Sistemas

- Los sistemas operativos proporcionan servicios, que pueden ser utilizados mediante:
 - **Llamadas al sistema** desde un lenguaje de programación. Estas llamadas abstraen la interacción con el hardware proporcionando funciones de alto nivel que se acceden a través de bibliotecas del lenguaje de programación
 - Mediante programas escritos en **lenguajes de script**, que son interpretados por el **shell** o interprete de comandos del sistema operativo
- Se suele distinguir entre programación de sistemas de **caja negra** (lenguajes de script) o **caja blanca** (llamadas al sistema), dependiendo de la forma de utilizar los servicios del sistema operativo

Lenguajes de Script

- Los lenguajes de script, son lenguajes de programación **interpretados** que tienen su origen en los lenguajes de control de tareas (**batch languages**) de los primeros sistemas operativos
- Aunque en un principio estos lenguajes era muy simples (básicamente variables y sentencias de control), se han ido sofisticando. Ej. Un script Linux para convertir .jpg a .png

```
for jpg in "$@" ; do          # "$@" indica los parámetros de la línea de control
png="{jpg%.jpg}.png"         # cambia la extensión del nombre de fichero
echo convirtiendo "$jpg" ...  # estado en pantalla
if convert "$jpg" jpg.to.png ; then # convert es el programa de conversión
    mv jpg.to.png "$png"         # Si funciona renombra el fichero de salida
else                             # ...informar acerca del error, guardar el resultado y salir con error
    echo 'error: failed output saved in "jpg.to.png".' 1>&2 exit 1
fi                               # fin del if
Done                            # fin del for
echo conversion realizada con éxito # tell the user the good news
exit 0
```


Programación de Sistemas

- Los lenguajes de Script se han ido sofisticando y en la actualidad no se utilizan solo en el contexto de los sistemas operativos. Algunos ejemplos:
 - **Perl** se diseñó originalmente como un lenguaje para transformación tratamiento de ficheros de texto, pero evolucionó a un lenguaje de prototipado rápido bastante sofisticado:
 - Soporte de expresiones regulares y pattern matching
 - Gestión de ficheros
 - Definición de procedimientos y funciones
 - Modularidad y programación orientada a objeto (**versiones 5.5 y +**)
 - **Web browsers**. Los scripts se ejecutan en el contexto del navegador. Han sido lenguajes con mucho éxito en los últimos años (JavaScript, VBScript))

Programación de Sistemas

- Un ejemplo de lenguajes de Script en Web, **JavaScript**:
 - Lenguaje de prototipado (al menos en sus inicios!), pensado para su ejecución en Web browsers
 - No tiene nada que ver con Java, sistema de tipos débil (soporta polimorfismo y conversión de tipos)
 - Soporta construcciones propias de los lenguajes funcionales y orientados a objetos

```
function sum()
{
var i, x = 0;
for (i = 0; i < arguments.length; ++i) { x += arguments[i]; } return x;
}
sum(1, 2, 3); // devuelve 6
```

Programación de Sistemas

- ¿Cómo se utiliza desde un navegador web?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
  <head> <title>pagina simple</title></head>
  <body>
    <h1 id="header">This is JavaScript</h1>
    <script type="text/javascript">
      document.write('Hola programación de sistemas!');
      var h1 = document.getElementById("header"); // referencia al tag <h1>
      h1 = document.getElementsByTagName("h1")[0]; // accediendo al elemento <h1>
    </script>
  <noscript>
    Tu navegagadopr. </noscript>
</body>
</html>
```

El MCM en JavaScript

```
/* Finds the lowest common multiple of two numbers */
function LCMCalculator(x, y) { // constructor function
var checkInt = function (x) { // inner function
  if (x % 1 !== 0) { throw new TypeError(x + " is not an integer"); // throw an exception }
  return x; };
  this.a = checkInt(x);
  this.b = checkInt(y);

  LCMCalculator.prototype = { // object literal
    constructor: LCMCalculator, // when reassigning a prototype, set the constructor property
    gcd: function () { // method that calculates the greatest common divisor
      var a = Math.abs(this.a), b = Math.abs(this.b), t;
      if (a < b) { // swap variables t = b; b = a; a = t; }
      while (b !== 0)
      {
        t = b; b = a % b; a = t;
      }
      this['gcd'] = function () { return a; }; return a; },

    function output(x) { document.write(x + "<br>"); }

    [[25, 55], [21, 56], [22, 58], [28, 56]].map(function (pair) { // array literal + mapping function return new
      LCMCalculator(pair[0], pair[1]); }).sort(function (a, b) { // sort with this comparative function return a.lcm()
        - b.lcm(); }).forEach(function (obj) { output(obj + ", gcd = " + obj.gcd() + ", lcm = " + obj.lcm()); });
```

Lenguajes para Programación de Sistemas

- Aunque los lenguajes de script tienen una amplia aplicación en algunos contextos (GUIs, navegadores,...), la mayor parte de los programas para programación de sistemas son lenguajes de bajo nivel.
- Algunas características:
 - Acceso a memoria física, sin recolección de basura, direcciones absolutas, bits sin signo, tipos de memoria (ROM, RAM, flash,...)
 - Posibilidad de acceso al hardware bien directamente (Ada), bien a través de interfaces de acceso al sistema operativo en bibliotecas (C, C++,..)
 - Soporte de eventos y/o concurrencia (del mismo modo)
 - Eficientes y con capacidad de análisis de consumo de recursos (predecibles)

Lenguajes para Programación de Sistemas

- Con frecuencia los sistemas de desarrollo son distintos de los sistemas en los que se va a ejecutar la aplicación final
- Los compiladores que se utilizan se denominan **compiladores cruzados** y generan código para plataformas distintas en las que se ejecutan.
 - La máquina en la que se desarrolla se denomina **host** y en la que se ejecuta **target**
 - La depuración se realiza parcialmente en el host utilizando **emuladores** y simuladores del hardware
 - La depuración final es compleja, porque en ocasiones el **target** no tiene interfaces de entrada/salida

Lenguajes para Programación de Sistemas

- El tiempo:
 - Suelen existir primitivas para acceso al reloj del sistema y para expresar retrasos en la ejecución del programa
 - En Java:
 - `System.currentTimeMillis()`, reloj del sistema, se afecta por los cambios de otros comandos (por ejemplo al cambiar la fecha)
 - `System.nanoTime()`, reloj absoluto desde el comienzo de la ejecución de la máquina virtual
 - Ejercicio: ¿Cómo medir el tiempo que tarda en ejecutarse un trozo de código?

Lenguajes para Programación de Sistemas

...

```
long start = System.nanoTime(); // necesita java 1.5
```

```
// Código a monitorizar
```

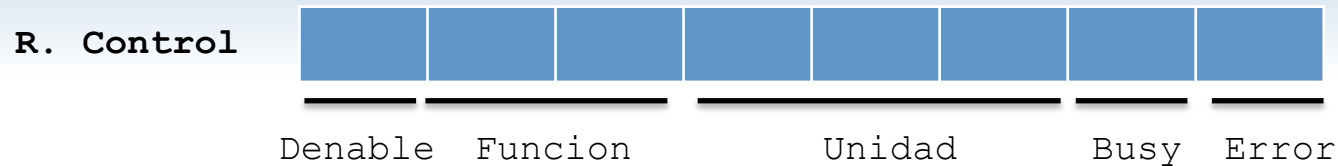
```
double elapsedTimeInSec = (System.nanoTime() - start) * 1.0e-9
```

- Problemas:
 - Interferencia de otros programa, procesos
 - Gestión de la memoria. Las variables pueden cambiar de sitio (memoria principal, cache,...) sin nuestro control
 - Pero necesitamos, conocer el peor tiempo: Lenguajes para **Sistemas de Tiempo Real**

Lenguajes para Programación de Sistemas

- La gestión de memoria:
 - En los lenguajes actuales la gestión de la memoria se ha automatizado mucho
 - No sabemos físicamente dónde están nuestras variables u objetos
 - Pueden cambiar de sitio de forma transparente y son eliminados cuando no van a ser utilizados de nuevo
- En la programación de sistemas necesitamos tener un mayor control
- Por ej. El hardware se accede a través de registros que se encuentran en direcciones físicas de memoria fijas
- Ese registro habrá que mapearlo a una variable de mi programa para poderlo manipular

Un ejemplo en Ada



```
for Reg_Cont use record
```

```
    Denable : at 0 range 0..0 ;
```

```
    Dfun : at 0 range 1..2 ;
```

```
    Unit : at 0 range 3..5;
```

```
    Busy :at 0 range 6..6;
```

```
    Error : at 0 range 7..7 ;
```

```
end record ;
```

```
for Reg_Cont'Size use 8 ; -- 16 bits
```

```
for Reg_Cont'Alignment use 2; -- 1 bytes de para alineación
```

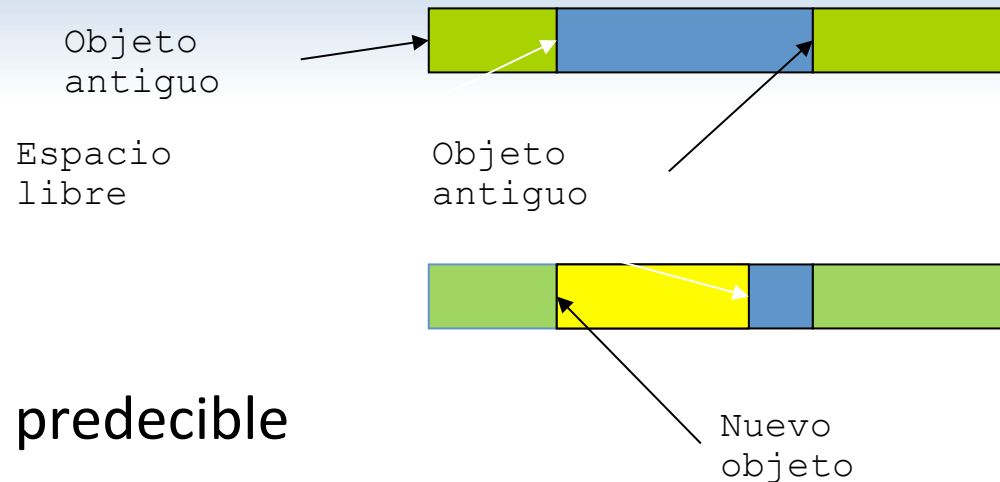
```
for Reg_Cont'Bit_Order use Low_Order_First ;
```

Un ejemplo en Ada

```
Control: Reg_Cont;  
for Control' Address use 8#177566# ;  
Tmp: Reg_Cont; --  
Tmp :=( Denable=>True, Dfun=>Read, Unit=>4, Errors=>None) ;  
Tcsr :=Tmp ;
```

- Se especifica la dirección en la que se encuentra el registro
- Los registros se manipulan como variables normales del programa

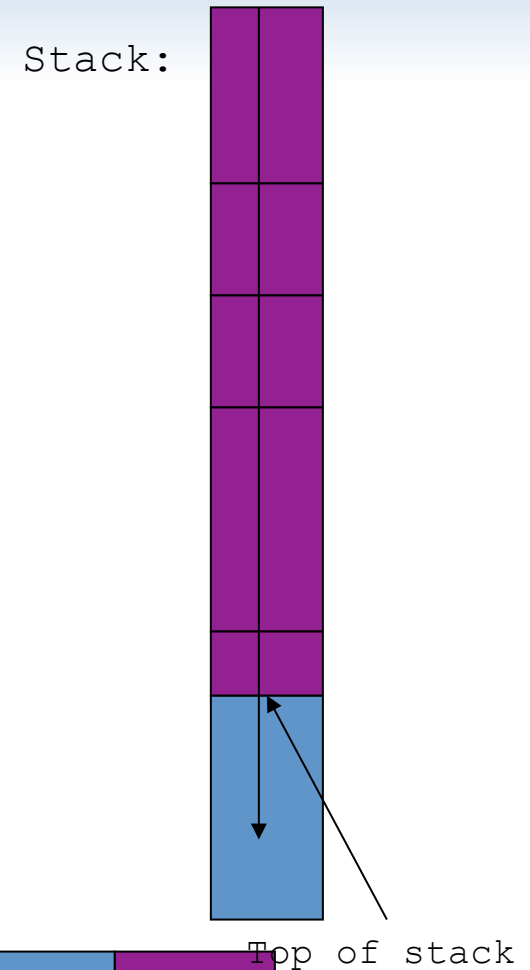
Gestión de memoria



- El problema:
 - Gestión de memoria no predecible
 - El esfuerzo para encontrar un hueco depende del estado previo de la memoria
 - Fragmentación
 - Si tienes un hueco de tamaño N y creas un objeto de tamaño M , con $m < N$ tenemos un fragmento de tamaño $M - N$
 - Después de un tiempo de ejecución, la mayor parte de la memoria son huecos (recolección de basura)
 - Las direcciones correspondientes al hardware no se pueden recolectar

Gestión de memoria

- Solucion: pre-asignar
 - Objetos globales
 - Creados al inicio del programa con un tamaño fijo
 - Usar pilas
 - Crecen y decrecen sólo en la cima
 - Sin fragmentar
 - Operaciones de tiempo constante
 - Pools de objetos de tamaño fijo
 - Los asignamos y devolvemos “mano”
 - No hay fragmentación
 - Operaciones de tiempo constante



Lenguajes de Programación: Gestión de Memoria

- Las anteriores limitaciones hace que no todos los lenguajes puedan ser utilizados en la programación de sistemas
- Los más utilizados con diferencia son C y C++, que permiten acceder a la memoria física y no tienen recolector de basura
 - Son lenguajes extremadamente propensos a errores
 - Están impuestos por la industria
- Lenguajes como Ada, de más alto nivel y específicamente diseñados para programación de sistemas han tenido un impacto limitado y sólo se usan en algunos sectores, como el aeroespacial
- Java de tiempo real, con un sistema de gestión de memoria muy complejo (automática, inmortal,...) ha tenido un impacto también muy limitado.

Gestión de actividades concurrentes: Eventos y Procesos

- Como consecuencia de la interacción con el mundo físico, la programación de sistemas tiene que hacer frente a interacciones con el entorno que pueden ser concurrentes simultáneas o de las que simplemente desconocemos el orden.
- Un ejemplo sencillo: gestión del teclado de un reproductor MP3



Gestión de actividades concurrentes: Eventos y Procesos

- Supongamos un paquete en java, que nos proporciona la siguiente interfaz:

```
public interface MP3Player
{
    boolean PlayKeyPressed();
    boolean FastForwardKeyPressed();
    boolean RewindKeyPressed();
    boolean StopKeyPressed();

    void Play();
    void FastForward();
    void Rewind();
    void Stop();
}
```

- ¿Cómo lo hacemos?

Gestión de actividades concurrentes: Eventos y Procesos

```
public class MP3Example
{
    public void Test()
    {
        MP3Example player = MP3PlayerManager.GetPlayer();
        while(true)
        {
            if(player.PlayKeyPressed())
            {
                player.Play();
            }
            else if(player.FastForwardKeyPressed())
            {
                player.FastForward();
            }
            ...
        }
    }
}
```

Eventos y Procesos

- Si las funciones son no bloqueantes consumimos constantemente CPU
- Si son bloqueantes, no sabemos en que orden se van a pulsar las teclas
- En ambos casos, lo más importante:
 - Las teclas se pueden pulsar en cualquier momento y en cualquier orden
 - ¿Cómo controlamos la gestión de las teclas mientras que se ejecuta `player.Play()`,...?
 - ¿Cuándo/Cómo empiezan y acaba el sistema?
- Se trata de un ejemplo sencillo, situaciones mucho más complejas ocurren continuamente en Programación de Sistemas

Eventos y Procesos

- Los sistemas operativos y (algunos) lenguajes proporcionan abstracciones para tratar con la interacción con el entorno y con otros sistemas software
- Las más comunes son:
 - **Concurrencia**: Distintas entidades (procesos/hebras) se ejecutan (conceptualmente) al mismo tiempo. En el ejemplo anterior cada proceso se encargaría de una de las funciones. El orden de ejecución viene impuesto por el entorno
 - **Eventos**: En este caso, el flujo del programa no es secuencial, sino que viene determinado por los eventos externos. Las aplicaciones están estructuradas en dos bloques separados (detección y selección de eventos y el manejo de estos eventos).
- Las abstracciones no son excluyentes. Se pueden y se pueden combinar en algunos casos (Java soporta ambas)

En resumen...

- La programación de sistemas tiene muchos aspectos que la diferencian de la programación de aplicaciones de usuario
 - Interacción con el sistema operativo y el hardware
 - El tiempo real es importante y hay que saber gestionarlo
 - Lenguajes de scripting
 - No todas las abstracciones de la programación más convencional se pueden utilizar. Gestión de memoria diferente
 - Se necesitan nuevas abstracciones para tratar con la concurrencia
- En el resto de la asignatura, nos centraremos en:
 - La gestión de memoria y programación de bajo nivel con el lenguaje C
 - Estudiar la concurrencia como la abstracción más importante en programación de sistemas

En resumen...

- En el resto de la asignatura, nos centraremos en:
 - La gestión de memoria y programación de bajo nivel con el lenguaje C
 - Estudiar la concurrencia como la abstracción más importante en la programación de sistemas y también en:
 - Sistemas Distribuidos y Paralelos
 - Comunicaciones
 - Sistemas de Tiempo Real
 - Estudiar la programación basada en eventos y como esta se puede combinar con la concurrencia
 - El diseño de Interfaces Gráficas de Usuario complejas combinando ambas técnicas