1 Introducción

Esta práctica tiene como objetivo la familiarización, de una parte, con el entorno de trabajo del laboratorio de la asignatura *Sistemas Reactivos*, y de otro con el entorno Lustre. Por tanto, en esta práctica se utilizarán:

- Máquina virtual, utilizando `VirtualBox`.
- Sistema operativo *GNU/Linux* (Ubuntu 24.04 LTS).
- Lenguaje de programación síncrono Lustre.

2 Máquina virtual

Para desarrollar las prácticas de la asignatura se utilizará una máquina virtual. Para ello se utiliza la aplicación de virtualización `VirtualBox`. En `VirtualBox` reside una máquina virtual (Ubuntu 24.04 LTS Desktop 64 bits) con sistema operativo Linux, tal y como se ve en la figura 2. Los alumnos tendrán el control total de esta máquina virtual y actuarán como usuarios administrador.

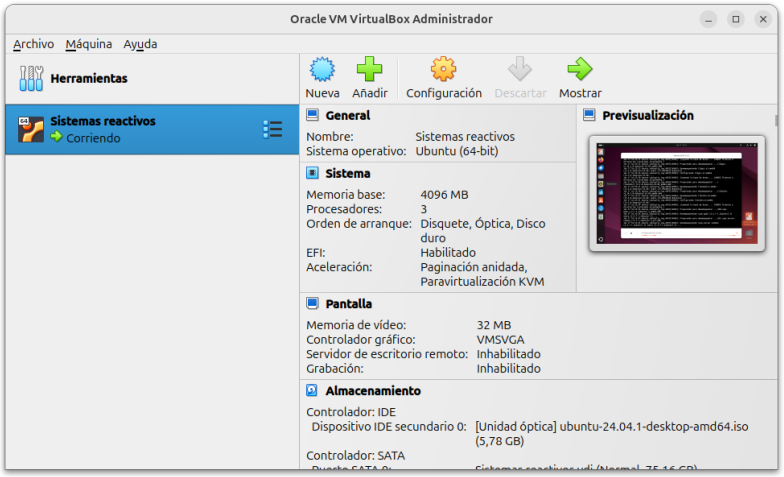


Figure 1: Aplicación VirtualBox con la máquina virtual.

2.1 Arranque de la máquina

Para arrancar la máquina virtual, `VirtualBox` cuenta con un icono (*Iniciar*) que abre automáticamente una nueva ventana con la instancia de la máquina virtual seleccionada.

Una vez arrancada la máquina, cada uno de los usuarios dispondrá de una cuenta de usuario genérica en el sistema (utilizando el sistema operativo Ubuntu Desktop). Ya sea en un entorno gráfico o en modo texto, el sistema nos pedirá un `usuario` (`login`) y una `clave` (`password`):

- `Login`: Nombre de usuario asignado (`alumno`).
- `Password`: Clave de la cuenta (`alumno`, no se muestra nada mientras se introduce en modo texto).

Estas credenciales serán las mismas para el usuario administrador.

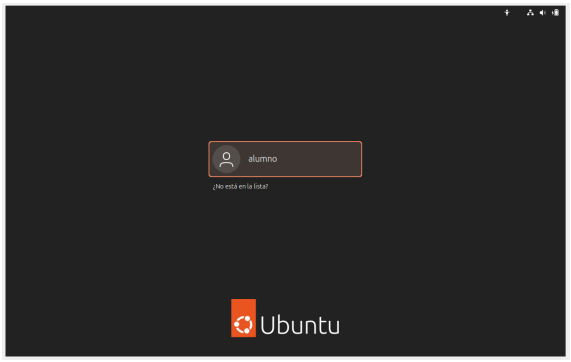


Figure 2: Inicio de sesión.

2.2 Cierre de sesión y apagado de la máquina

Cada usuario debe cerrar su sesión y apagar el equipo al terminar el turno de prácticas para posibilitar la utilización del ordenador por parte de los alumnos de las demás materias.

La sesión en Ubuntu se cerrará en *Sistema* → *Apagar* → *Cerrar la sesión* o *Apagar*.

El usuario también puede enviar una señal de apagado o de reinicio a la máquina virtual desde `VirtualBox`. Las máquinas virtuales sobre `VirtualBox` capturan el teclado y el ratón del sistema huésped, y éstos sólo se liberan presionando en el teclado `Ctrl+Der`.

3 Lenguaje de programación síncrono Lustre

3.1 Conceptos básicos de Lustre

Un programa o subprograma en **Lustre** se denomina **node** (nodo). Lustre es un lenguaje *funcional* y *declarativo* que opera sobre flujos de datos. Un flujo puede entenderse como una secuencia, ya sea finita o infinita, de valores. Todos los valores en un flujo son del mismo tipo, denominado el tipo del flujo. El comportamiento de un programa en Lustre es cíclico: en cada ciclo de ejecución del programa, todos los flujos involucrados toman el valor correspondiente al ciclo actual. Un nodo establece uno o varios parámetros de salida en función de uno o varios parámetros de entrada, y todos estos parámetros son flujos.

- Funcional: Lustre se basa en la evaluación de funciones, donde los programas se describen mediante la transformación de flujos de datos (streams) a través de funciones y operadores. No tiene efectos secundarios, lo que es característico de los lenguajes funcionales.
- Declarativo: En Lustre, se describe lo que se quiere lograr (por ejemplo, cómo deben ser los flujos de datos o las relaciones entre las variables) sin especificar de manera detallada cómo debe ejecutarse paso a paso. En lugar de decir cómo hacerlo, se declara qué debe ocurrir en función de las entradas.

3.2 El lenguaje de programación Lustre

En Lustre, la unidad fundamental de abstracción es el nodo (node), que define una función o un componente del sistema reactivo. Un nodo se comporta como una caja funcional que recibe entradas, realiza cálculos y devuelve salidas, permitiendo la modularidad y la reutilización del código.

3.2.1 Estructura de un node en Lustre

Un nodo en Lustre tiene la siguiente estructura general:

```
node nombre_nodo (entradas) returns (salidas);
var
    variables locales;
let
    ecuaciones;
tel
```

- **node nombre_nodo (entradas) returns (salidas);**

  Define el nombre del nodo, sus entradas y sus salidas. Las entradas y salidas se declaran con sus respectivos tipos de datos.

- **var variables locales;** (Opcional)

  Se pueden declarar variables internas que se utilicen dentro del nodo para almacenar valores intermedios.

- **let ecuaciones;**

  Contiene el cuerpo del nodo, donde se especifican las ecuaciones que relacionan entradas, variables locales y salidas.

- **tel**

  Marca el final del nodo.

3.3 Entorno de compilación

Si uno pretende saber si el código es sintácticamente correcto, solo necesita invocar:

```
$lv6 nombre_fichero.lus -n nombre_nodo
```

Si todo es correcto, el resultado será el siguiente:

```
This program is syntactically correct. If you want to do something with it, you
could try one the options: -2c, -2c-exec, -exec, -ec, -lic, or -h for more options.
```

Este mensaje indica que el programa no contiene errores de sintaxis. Sin embargo, el mensaje también señala que Lustre no hará nada adicional a menos que se utilice una de las opciones proporcionadas para realizar acciones específicas.

Por tanto, una vez comprobado que no hay errores sintácticos, es necesario generar el código C, para lo que se puede usar la opción `-2c`.

```
$lv6 nombre_fichero.lus -n nombre_nodo -2c
```

Los ficheros generados a partir de `nombre_fichero.lus` son una serie de archivos todos ellos necesarios para ejecutar o integrar el modelo en un sistema de programación en C. Cada archivo tiene un propósito específico para que el sistema resultante funcione correctamente y permita la simulación, la compilación y la ejecución del modelo de control.

Finalmente, para generar el código binario ejecutable, solo será necesario ejecutar el script `.sh` o por el contrario, generar el código C junto con su ejecutable con la opción `-2c-exec`.

```
$sh nombre_nodo.sh
```

o

```
$lv6 nombre_fichero.lus -n nombre_nodo -2c-exec
```

3.3.1 Ejemplo de un node en Lustre

Supongamos que necesitamos un programa en Lustre que realice la media de dos valores de tipo real. Este programa genera un flujo real.

La implementación del programa descrito se encuentra en el fichero `average.lus` que se encuentra disponible en Moovi, cuyo contenido es el siguiente:

```
node average (x , y : real ) returns ( out : real ) ;
let
   out = ( x + y ) / 2.0;
tel
```

Este nodo **average** recibe dos entradas **x** e **y**, y devuelve la media **out** como resultado. La evaluación ciclo a ciclo es la siguiente:

Ciclo	Entrada x	Entrada y	Salida out = (x + y) / 2.0
1	4.0	6.0	(4.0 + 6.0) / 2.0 = 5.0
2	8.0	2.0	(8.0 + 2.0) / 2.0 = 5.0
3	10.0	14.0	(10.0 + 14.0) / 2.0 = 12.0
4	0.0	20.0	(0.0 + 20.0) / 2.0 = 10.0
5	-6.0	6.0	(-6.0 + 6.0) / 2.0 = 0.0

Figure 3: Evaluación paso a paso de average

3.3.2 Ejemplo con condiciones (if-then-else)

Un nodo que determina si un número es positivo o negativo podría escribirse así:

```
node Signo (x: int) returns (s: int);
let
    s = if x > 0 then 1 else if x < 0 then -1 else 0;
tel
```

Aquí, **s** tomará el valor **1** si **x** es positivo, **-1** si es negativo y **0** si es cero.

3.3.3 Ejemplo con memoria (pre)

Supongamos ahora que queremos un nodo que cuente el número de activaciones (ticks) desde el inicio de la ejecución:

```
node Contador () returns (count: int);
var
    prev_count: int;
let
    prev_count = 0 -> pre(count);
    count = prev_count + 1;
tel
```

Aquí se usa el operador *pre*, que almacena el valor anterior de count, permitiendo que el nodo recuerde su estado entre ejecuciones. La expresión *0 -> pre(count)* indica que el primer valor de count es 0, y a partir de ahí se incrementa en cada instante.
La evaluación ciclo a ciclo es la siguiente:

Ciclo (tick)	prev_count	count
1	0	1
2	1	2
3	2	3
4	3	4
5	4	5

Figure 4: Evaluación paso a paso de Contador

Los nodos en Lustre permiten estructurar programas de manera modular, definiendo bloques funcionales que pueden reutilizarse y anidarse dentro de otros nodos. Gracias a su enfoque declarativo, las relaciones entre las señales quedan claramente especificadas, y con el uso de operadores como pre o ->, es posible modelar sistemas con memoria y comportamiento dinámico.

ACTIVIDADES:

- Tarea 1: Compilar el nodo average y ejecutarlo.
- Tarea 2: Compilar el nodo Signo y ejecutarlo.
- Tarea 3: Compilar el nodo Contador y ejecutarlo.
- Tarea 4: Crear un nuevo nodo llamado grados que reciba de entrada una variable entera entre 0 y 360 (indica los grados). La salida será un entero que indique el cuadrante en el que está (0 cuadrante superior derecho, 1 cuadrante superior izquierdo, 2 cuadrante inferior izquierdo, 4 cuadrante inferior derecho). En caso de no estar en ningún cuadrante sino en el eje, devolverá el valor 5.

SOLUCIONES:

- Tarea 1: Compilar el nodo average y ejecutarlo.
- Tarea 2: Compilar el nodo Signo y ejecutarlo.
- Tarea 3: Compilar el nodo Contador y ejecutarlo.
- Tarea 4: Crear un nuevo nodo llamado grados que reciba de entrada una variable entera entre 0 y 360 (indica los grados). La salida será un entero que indique el cuadrante en el que está (0 cuadrante superior derecho, 1 cuadrante superior izquierdo, 2 cuadrante inferior izquierdo, 4 cuadrante inferior derecho). En caso de no estar en ningún cuadrante sino en el eje, devolverá el valor 5.

```
node grados (angulo: int) returns (cuadrante: int);
let
    cuadrante =
        -- Eje horizontal
        if angulo = 0 or angulo = 180 or angulo = 360 then 5
        -- Eje vertical
        else if angulo = 90 or angulo = 270 then 5
        -- Primer cuadrante
        else if angulo > 0 and angulo < 90 then 0
        -- Segundo cuadrante
        else if angulo > 90 and angulo < 180 then 1
        -- Tercer cuadrante
        else if angulo > 180 and angulo < 270 then 2
        -- Cuarto cuadrante
        else if angulo > 270 and angulo < 360 then 3
        else 6;  -- En caso de error (fuera del rango)
tel
```

La evaluación ciclo a ciclo es la siguiente:

Entrada (ángulo)	Salida (cuadrante)	Explicación
0	5	Eje horizontal (positivo)
45	0	Primer cuadrante
90	5	Eje vertical (positivo)
120	1	Segundo cuadrante
180	5	Eje horizontal (negativo)
200	2	Tercer cuadrante
270	5	Eje vertical (negativo)
315	3	Cuarto cuadrante
360	5	Eje horizontal (positivo, mismo que 0)

Figure 5: Evaluación paso a paso de grados

3.3.4 Ciclicidad en Lustre

En Lustre, un código se considera cíclico cuando hay una dependencia circular en las ecuaciones, lo que impide que el compilador determine un orden de evaluación correcto. Esto genera un problema de causalidad, donde una variable depende de sí misma en el mismo instante de ejecución.

Aquí tienes algunos ejemplos de código cíclico en Lustre:

- Ejemplo 1: Dependencia circular directa

```
node Ciclico1 (x: int) returns (y: int);
let
    y = y + x;  -- Error: y depende de sí misma en el mismo instante
tel
```

✗ Error: **y** depende de su propio valor dentro del mismo ciclo, lo que impide calcularlo de manera determinista.

- Ejemplo 2: Dependencia circular indirecta

```
node Ciclico2 (a: int) returns (b: int; c: int);
let
    b = c + a;  -- c depende de b, pero b depende de c
    c = b * 2;
tel
```

✗ Error: **b** y **c** se refieren mutuamente en el mismo instante de ejecución, creando un bucle de dependencias.

- Ejemplo 3: Uso Incorrecto de pre

```
node Ciclico3 () returns (z: int);
let
    z = pre(z);  -- Error: pre necesita un valor inicial explícito
tel
```

✗ Error: **z** usa **pre(z)**, pero sin una inicialización adecuada (como **0** -> **pre(z)**), lo que genera un problema de inicialización.

ACTIVIDADES:

- Tarea 1: Compilar el nodo Ciclico1 y ejecutarlo.
- Tarea 2: Compilar el nodo Ciclico2 y ejecutarlo.
- Tarea 2: Compilar el nodo Ciclico3 y ejecutarlo.

¿Qué ocurre?

3.3.5 Cómo Corregir los Ciclos

Para evitar dependencias cíclicas, se pueden usar:

- Inicialización explícita: **0 -> pre(x)** para romper la dependencia en el primer ciclo.
- División de cálculos en varios ciclos: Utilizando **pre()** para hacer que un valor se actualice en el siguiente instante.

ACTIVIDAD:

- Corregir el ejemplo 1.
- Corregir el ejemplo 2.
- Corregir el ejemplo 3.

SOLUCIONES:

- Ejemplo corregido del Ejemplo 1:

```
node NoCiclico1 (x: int) returns (y: int);
var prev_y: int;
let
    prev_y = 0 -> pre(y);
    y = prev_y + x;  -- Ahora depende del valor del ciclo anterior
tel
```

- Ejemplo corregido del Ejemplo 2:

```
node NoCiclico2 (a: int) returns (b: int; c: int);
var prev_b: int; prev_c: int;
let
    prev_b = 0 -> pre(b);
    prev_c = 0 -> pre(c);

    b = prev_c + a;  -- b usa el valor anterior de c
    c = prev_b * 2;  -- c usa el valor anterior de b
tel
```

- Ejemplo corregido del Ejemplo 3:

```
node NoCiclico3 () returns (z: int);
let
    z = 0 -> pre(z);  -- Inicializa z en 0 en el primer ciclo
tel
```

3.4 Ejercicios de Lustre

En esta práctica se propone la implementación de la secuencia de **Fibonacci** y la implementación del **factorial** de un número.

SOLUCIONES:

- Solución serie de Fibonacci:

```
node Fibonacci() returns (f: int);
var
    f1, f2: int;
let
    f1 = 0 -> pre(f2);  -- F(n-2)
    f2 = 1 -> pre(f);   -- F(n-1)
    f  = f1 + f2;       -- F(n) = F(n-1) + F(n-2)
tel
```

- Solución factorial de un número:

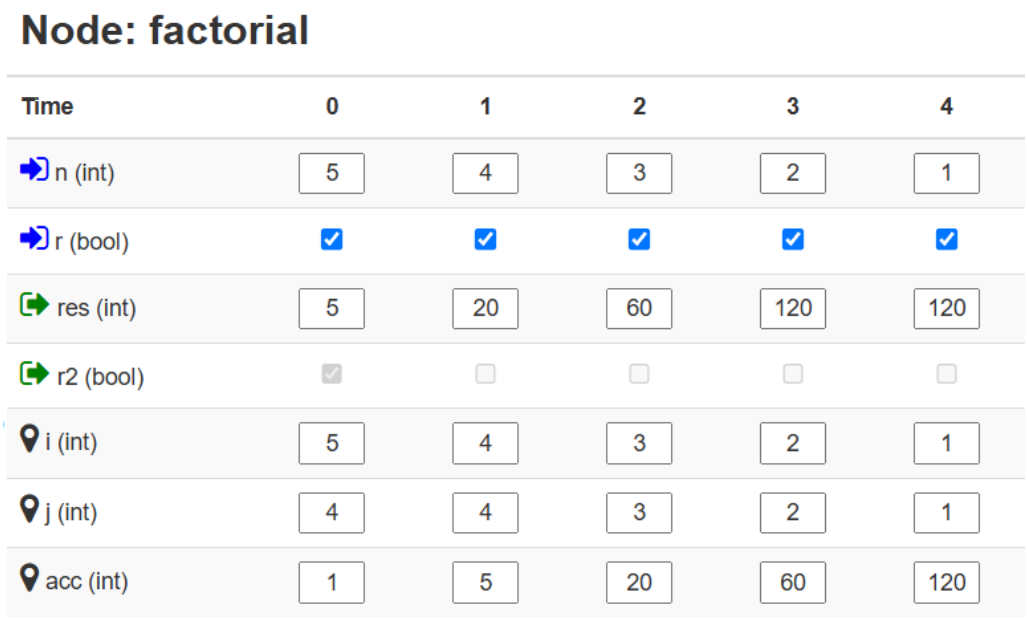
```
node factorial(n: int;r:bool) returns (res: int; r2:bool);
var i,j: int; acc: int;
let
  i = if r=true then n else 1;
  j= n-1 ->pre(n)-1;
  r2 = true-> if n=j then false else true;
  acc = 1 -> if n<>j then 0 else pre(res);
  res = i * acc;
tel;
```

La evaluación ciclo a ciclo de Fibonacci es la siguiente:

Ciclo	f1	f2	f
0	0	1	1
1	1	1	2
2	1	2	3
3	2	3	5
4	3	5	8
5	5	8	13

Figure 6: Evaluación paso a paso de Fibonacci

La evaluación ciclo a ciclo de Factorial es la siguiente:



Node: factorial

Time	0	1	2	3	4
n (int)	5	4	3	2	1
r (bool)	☑	☑	☑	☑	☑
res (int)	5	20	60	120	120
r2 (bool)	☑	☐	☐	☐	☐
i (int)	5	4	3	2	1
j (int)	4	4	3	2	1
acc (int)	1	5	20	60	120

Figure 7: Evaluación paso a paso de Factorial

¿Se puede hacer en un solo ciclo?

No directamente en Lustre puro, ya que no tiene estructuras iterativas tradicionales (for, while). Sin embargo si se usa Lustre dentro de un sistema más amplio, se puede calcular el factorial en otro lenguaje y enviarlo a Lustre en un solo instante.

La explicación del código es la siguiente:

1. Explicación línea por línea
   - Entradas:
     - n: int: Número del que queremos calcular el factorial.
     - r: bool: Señal de reinicio.
   - Salidas:
     - res: int: Resultado del factorial.
     - r2: bool: Señal de control que indica si el cálculo sigue activo o no.
   - Variables internas:
     - i: Controla el número actual a multiplicar.
     - j: Controla la iteración para reducir n hasta 1.
     - acc: Almacena el valor acumulado del factorial.
2. Lógica del Código:
   - i = if r = true then n else 1;
     - Si r es true, el valor inicial de i es n.
     - Si r es false, i toma el valor 1.
   - j = n - 1 -¿ pre(n) - 1;
     - j comienza con n - 1 (valor inicial).
     - En los siguientes ticks, toma pre(n) - 1, es decir, el valor de n del tick anterior menos 1.

- r2 = true -¿ if n = j then false else true;
  - r2 empieza en true.
  - Luego, si n es igual a j, se pone en false (indica que debe detenerse el cálculo).
  - De lo contrario, sigue en true.
- acc = 1 -¿ if n ¡¿ j then 0 else pre(res);
  - acc comienza en 1.
  - Si n es diferente de j, lo resetea a 0.
  - Si n = j, toma el valor del resultado anterior (pre(res)).
- res = i * acc;
  - Calcula res multiplicando i por acc.