Introducción

- Programación concurrente
- Comunicación y sincronización
- Fiabilidad y tolerancia a fallos
- Manejo de excepciones
- Uso del tiempo

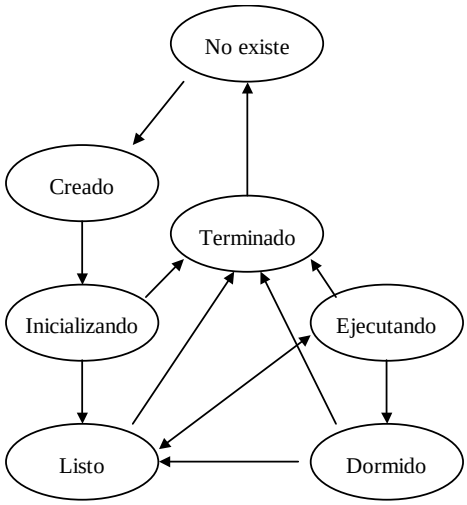
Programación concurrente

Definición:

Un lenguaje de programación concurrente es aquel que permite expresar una colección de procesos secuenciales autónomos, ejecutándose (lógicamente) en paralelo.

- Implementación:
  - Ejecución multiplexada en un único procesador
  - Ejecución multiplexada en un sistema multiprocesador con acceso a memoria compartida
  - Ejecución multiplexada en varios procesadores sin memoria compartida (sistemas distribuidos)
- Los lenguajes concurrentes permiten expresar de forma lógica las distintas actividades sin considerar la implementación

Estados en la vida de un proceso



- Se usa la CPU solo en el estado Ejecutando
- Antes de ejecutarse siempre se pasa por el estado Listo
- El estado más habitual es Dormido

Run Time System Support

- Se encarga de la ejecución de una aplicación concurrente.
- Puede tener varias formas:
  - Un programa incorporado como un componente más de la aplicación
  - Un sistema software estándar generado con el código del programa por el compilador (Ada)
  - Un microcódigo incorporado en el microprocesador (transputers programados con occam2)
- Una alternativa es apoyarse en los servicios de un Sistema Operativo

Sistemas Operativos

- Permite crear y administrar procesos
- Cada proceso se ejecuta en una máquina virtual independiente, aprovechando los mecanismos de protección de memoria y paginación
- Se consigue mayor eficiencia utilizando procesos ligeros (threads)
- Los procesos ligeros soportados por el Sistema Operativo consiguen un mejor comportamiento que soportados por librería

Construcciones para la programación concurrente

- Facilidades fundamentales que se deben ofrecer
  - Noción de proceso
  - Sincronización
  - Comunicación
- Tipos de comportamiento en los procesos
  - Independiente
  - Cooperativo
  - Competitivo

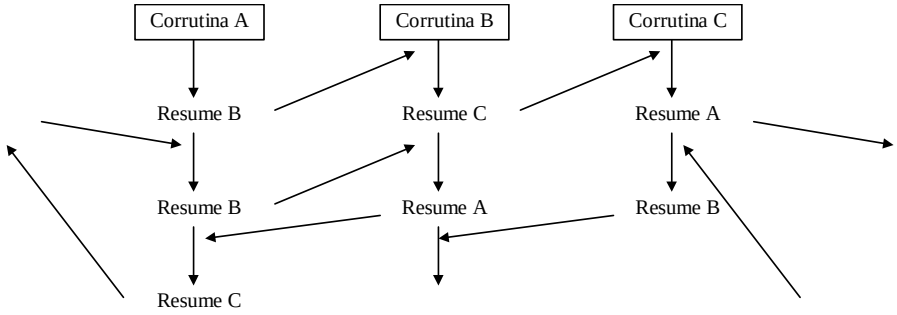
Procesos y objetos

- OOP: Sistema como una colección de objetos que colaboran entre ellos.
- Tipos de objetos
  - Activos (agentes activos)
  - Pasivos (sin restricciones en el acceso)
  - Reactivos (agentes pasivos, acceso único)
- Servidores: Agentes activos que controlan el uso de un recurso

Representación de los procesos

- Corrutinas
  - No se trata de procesos (no son independientes)
  - Una rutina pasa el control a otra con la sentencia **resume**
- Fork y Join
  - **Fork** crea un proceso
  - **Join** espera a que finalice
- Cobegin y Coend
  - Define un bloque con control donde todas las sentencias se ejecutan en paralelo
- Declaración explícita
  - Un procedimiento se declara como concurrente
  - La ejecución comienza donde es declarado

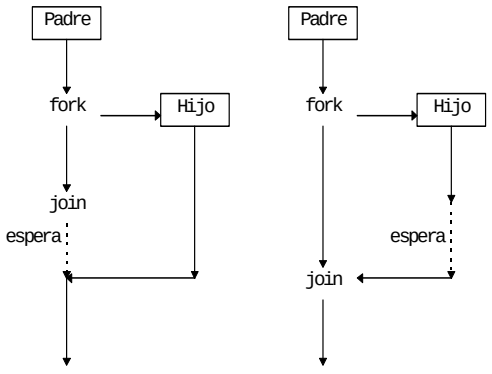
Corrutinas



Fork y Join

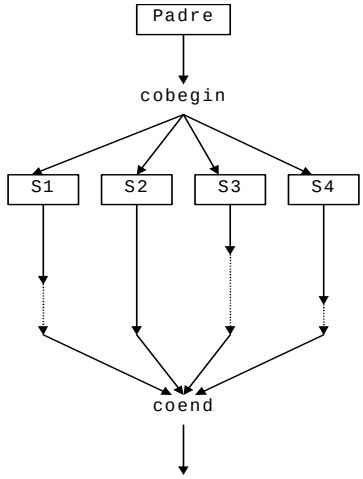
```
function F return ...
.
.
end F;

procedure P
  ...
  C := fork F;
  .
  .
  J := join C;
end P
```



Cobegin y Coend

```
cobegin
  S1;
  S2;
  S2;
  .
  .
  Sn;
coend
```



Declaración explícita

```
procedure Tareas_concurrentes
    task A;
    task B;

    task body A is
            -- declaraciones locales de la tarea A
    being
            -- secuencia de instrucciones de la tarea A
    end;

    task body B is
            -- declaraciones locales de la tarea B
    being
            -- secuencia de instrucciones de la tarea B
    end;

begin
    -- Las tareas A y B comienzan a ejecutarse
    -- antes de la primera sentencia del programa
    .
    .
end Tareas_concurrentes;
    -- el programa no termina hasta que no terminen
    -- las tareas A y B
```