Técnicas de Programación Concurrente I – Primeros temas a considerar.

1. *Unidad 1: Conceptos básicos de la programación concurrente.*
   1.1. Introducción
   1.2. Programación secuencial vs programación concurrente.
   1.3. Necesidad de la programación concurrente.
   1.4. Problemas en la programación concurrente.
   1.5. Arquitecturas de hardware que soportan la programación concurrente.

Introducción

Se puede describir la programación concurrente mediante su comparación con la programación secuencial.
En la programación secuencial se define un orden total para los eventos que se suceden en un proceso. En la programación concurrente, el orden de los eventos es parcial.
Los programas secuenciales se comportan de manera tal que ante la misma entrada, el flujo de ejecución es siempre el mismo.
Los programas concurrentes no necesariamente ejecutan sus operaciones en el mismo orden en el que corren repetidas veces, aunque la entrada sea la misma. Este no determinismo de la programación concurrente necesita de herramientas que posibiliten salvar las dificultades que puedan surgir como consecuencia de este fenómeno.
La programación secuencial responde al modelo de arquitectura de Von Neumann, esto es, se cuenta con un único procesador, una única unidad central de proceso, la cual está conectada a la memoria central y a los dispositivos de entrada / salida mediante canales o "buses". Además, tanto las instrucciones como los datos se almacenan en la memoria central. También, el procesador ejecuta un ciclo de máquina-instrucción repetidas veces y como solamente hay un procesador, el sistema sólo puede ejecutar una instrucción a la vez.
El modelo de programación concurrente no es como el antes descripto. De acuerdo con Booch (1987), "la programación concurrente libera al programador de la forma de pensar Von Neumann.

Orden total de un programa secuencial

Sean las siguientes, tres instrucciones de un programa secuencial:
I_1; I_2; I_3;
I_1 precede a I_2 y ésta a I_3.
Esto significa que antes de comenzar a ejecutarse I_2, I_1 ha debido terminar de ser ejecutada y lo mismo entre I_2 e I_3.
Todo esto sirve para ilustrar que no debe haber solapamiento o superposición en la ejecución de las instrucciones. Es así que es posible definir una relación de orden total entre las instrucciones de un programa secuencial, la relación "se ejecuta antes que".
De otra manera, se puede decir que dadas I y J, dos instrucciones de un programa secuencial, resulta verdadera alguna de las dos siguientes afirmaciones:
Para toda ejecución válida x -> y o para toda ejecución válida y -> x.

$\forall$ e ● x -> y o $\forall$ e ● y -> x

Se dice que un programa secuencial es determinístico, esto es, dada la misma entrada, siempre se ejecuta de la misma forma, en la misma secuencia de instrucciones y siempre produce la misma salida, los mismos resultados.

Características de la programación secuencial

- El orden textual de las sentencias especifica el orden de ejecución de las mismas,
- Instrucciones sucesivas deben ser ejecutadas sin solapamiento alguno (en tiempo), con alguna otra.

Procesadores múltiples.

En este caso se puede producir la ejecución paralela de instrucciones de un mismo programa. Esto indica que las instrucciones podrían solaparse o superponerse en el tiempo, es decir que sin que haya terminado de ejecutarse una instrucción podría iniciarse la ejecución de otra.
Una forma tradicional para indicar este fenómeno es la construcción cobegin – coend o parbegin- parend:
Sean I_1 hasta I_7 instrucciones de un programa, entonces:

I_1; I_2; cobegin I_3; I_4; I_5; coend I_6; I_7;

se entiende que I_1 e I_2 se ejecutan secuencialmente, así como I_6 e I_7. Sin embargo, I_3; I_4; I_5; se ejecutan paralelamente o concurrentemente. También se puede utilizar la notación $I_3 \parallel I_4 \parallel I_5$
Otra forma de indicar esto es:

$\forall$ e ● I_1 -> I_2 (cobegin I_3; I_4; I_5; coend) -> I_6 -> I_7;

Definición. Dadas I, J instrucciones de un programa concurrente, I || J sii $\exists$ e ● no(I>J) y no(J->I).
En palabras, I y J son concurrentes sii existe una ejecución válida del programa en la cual I y J se solapan o sobreponen.
Para estudiar estos temas, se supone que los procesadores son lógicos, de manera de no comprometerse con procesador físico alguno.

Se tendrán como premisas para estudiar a los programas concurrentes, las siguientes dos propiedades relacionadas con el tiempo de procesamiento:
Primera: no se hacen suposiciones acerca de la rapidez de la operación de un procesador lógico en relación con otro.
Segunda: no existe relación entre rapidez de operación de procesadores lógicos y los relojes normales de medición del tiempo.

Definición. Se dice que existe intercalado (interleaving) de instrucciones cuando algunas instrucciones son ejecutadas desde un proceso y luego la unidad central de proceso o UCP deja de ejecutar ese proceso, dando lugar a la ejecución de otro proceso y así siguiendo (switching).

Operador ||

|| es el operador concurrente. Si dos operaciones en un proceso, son concurrentes entre sí, se cumple que:

- está permitido que las operaciones estén superpuestas en el tiempo,
- el orden textual no define el orden de ejecución.

Definición. Operador || u operador de concurrencia fue definido por Hoare en 1985. Este operador satisface las propiedades de conmutatividad, asociatividad y transitividad. Sean P, Q y R procesos, entonces:
P || Q = Q || P
(P || Q) || R = P || (Q || R)

Orden parcial de la programación concurrente.

Dada la siguiente construcción:
cobegin I; J; K coend;
algunas de las posibles secuencias de ejecución son:
I; J; K;
K; J; I;
Sin embargo, dado que puede haber solapamiento o superposición, el número de posibles ejecuciones puede ser mucho mayor si se considera la descomposición de cada instrucción de alto nivel en varias en lenguaje de máquina.

De manera que las operaciones concurrentes entre sí están parcialmente ordenadas.
Los programas concurrentes pueden tener un comportamiento no determinístico puesto que pueden ofrecer diferentes resultados como fruto de su repetición con la misma entrada.

Ejemplos:
El siguiente fragmento de operaciones es no determinístico, porque según como se ejecute, A finalmente puede valer 1 ó 2:
A = 1;
A = 2;

F1 es el siguiente fragmento de código:
X = 2;
Y = X-1;

F2, a su vez es:
X = 3;
Y = X + 1;

Posibles ejecuciones de F1 y F2 en paralelo son:

G1:
X = 3; Y = 4;
G2:
X = 2; Y = 1;
G3:
X = 2; Y = 3;

Si uno no desea alguno de estos resultados, debe agregar mecanismos de sincronización.

Condiciones de Bernstein.
Las condiciones de Bernstein se utilizan para ver si un programa es determinístico o no. Para determinar si dos conjuntos de instrucciones pueden ejecutarse concurrentemente o no.
Para una acción A, se consideran su conjunto de lectura y su conjunto de escritura.
Definición. Conjunto de lectura de una acción A: es el conjunto de variables que A lee, es decir, variables cuyos valores son referenciados durante la ejecución de A.
Definición. Conjunto de escritura de una acción A: es el conjunto de variables que A escribe, es decir, cuyos valores son actualizados durante la ejecución de A.
Estas definiciones se extienden a fragmentos de instrucciones.
Las condiciones de Bernstein dicen: dados S_i y S_j dos conjuntos de instrucciones. Éstas se pueden ejecutar concurrentemente sii se cumple:

- $R(S_i)$ intersección $W(S_j)$ = vacío;
- $W(S_i)$ intersección $R(S_j)$ = vacío;
- $W(S_i)$ intersección $W(S_j)$ = vacío;

En caso de cumplirse las condiciones de Bernstein, también se dice que la ejecución paralela de S_i y S_j es determinística.

Ejemplos.
S y T son los siguientes conjuntos de instrucciones:
(S_1) X = u + v;
(S_2) Y = X * w;

(T_1) a = x + y;
(T_2) b = z – 1,

Para S, se tienen los siguientes conjuntos:

R(S) = u, v, X, w
W(S) = Y, X
X pertenece a la intersección de R(S) y W(S), con lo cual, este fragmento es no determinístico.

$R(T_1)$ = x, y
$W(T_1)$ = a
$R(T_2)$ = z
$W(T_2)$ = b
Para estos conjuntos, las condiciones de Bernstein se cumplen, de manera que T_1 y T_2 pueden ejecutarse concurrentemente.

Maneras de expresar la concurrencia.

Puede ser mediante gráficos o de manera textual. Esta última ya se vio cuando se habló de cobegin/coend
La manera gráfica se realiza mediante la utilización del grafo de precedencias.

Definición. Grafo de precedencia es un grafo dirigido acíclico cuyos nodos son las actividades secuenciales y los arcos son los pares (v_i,v_j) de vértices tales que requieren que la actividad i se complete antes que la actividad j pueda comenzar.
Un nodo de un grafo de precedencia puede ser una tarea tal como la reconoce el Sistema Operativo (SO) o bien, pueden ser sentencias en un programa, las que deben ser ejecutadas concurrentemente con otras sentencias del mismo programa.
Se puede hablar de diferente granularidad de la concurrencia, tales como granularidad gruesa (coarse granularity) que es la presente entre las tareas del SO; granularidad fina (fine granularity) correspondiente a la concurrencia de únicas instrucciones o conjuntos de instrucciones en los procesadores superescalares; granularidad muy fina (very fine granularity) que se observa en el hardware durante la ejecución de una única instrucción.

Ejemplo.

Sea el siguiente fragmento de programa secuencial:
S_1: a = x + y;
S_2: b = z + 1;
S_3: c = a – b;
S_4: w = c + 1;
S_5: d = a + e;
S_6: w = w*d;

El grafo de precedencias correspondiente es:

(hacerlo o completarlo en la teórica)

Observando el grafo de precedencias, se puede inferir que el código podría ser ejecutado en dos procesadores. Uno podría ejecutar S_1, S_5 y S_6, mientras que el otro podría ejecutar S_2, S_3 y S_4.

Ejemplo.
Se tiene el siguiente fragmento de código
S_1: q = x*x;
S_2: m = a*q;
S_3: n = b*x;
S_4: z = m+n;
S_5: y = z + c;

El grafo de precedencia es:

(hacerlo o completarlo en la teórica)

Ejemplo. Utilizando la forma textual de indicar la concurrencia, se puede mostrar:

```
begin
        cobegin
                a = x + y;
                b = z – 1;
        coend;
        c = a-b;
        w = c+1;
end;
```

El grafo de precedencia, en este caso, sería:

(hacerlo o completarlo en la teórica)

Definición. Propiedad de un programa es cualquier afirmación que debe ser cierta para toda posible ejecución del programa.

Ejemplo. En el siguiente segmento de código

```
……
S1:  x = 1;
S2: y = x + 1;
S3: x = y + 2;
……
```

Una propiedad de este código puede expresarse como

$\forall$ e ● S_1 -> S_2 -> S_3

Esta propiedad dice que para toda ejecución válida del programa, S_1 precede a S_2 y S_2 precede a S_3. Por lo tanto, antes que comience a ejecutarse la siguiente instrucción, debe haber concluido la ejecución de la anterior. No se permiten los solapamientos entre las instrucciones. Esta propiedad debe satisfacerse hasta el nivel de instrucciones en código de máquina.

Razones para sostener la programación concurrente.

- Mediante la arquitectura de multiprocesadores se aumenta la rapidez en la ejecución de los programas (algoritmos paralelos),
- Un programa, en general, no provoca que el procesador esté ocupado todo el tiempo mientras éste se ejecuta,
- Se puede reducir el tiempo muerto de procesador disponiendo que ejecute varios programas de manera concurrente,
- Existen algunos problemas cuya solución no resulta adecuada si se realiza vía el modelo secuencial de programación. En cambio, son de naturaleza concurrente.

Un ejemplo de mejora en la utilización del procesador es la multiprogramación en sistemas operativos.
Otro ejemplo son los sistemas de control. Éstos son tales que capturan los datos mediante sensores, se analizan los datos y en términos del resultado del análisis, actúa el sistema. Los datos se recolectan en diferentes dispositivos y en diferentes lugares. Por ejemplo, las luces del tránsito o semáforos.
Otro ejemplo son las aplicaciones con tecnología Web. Los servidores Web atienden concurrentemente diferentes conexiones de usuarios. Programas para chatear entre varios usuarios; servidores de correo electrónico con envíos y recepciones de diferentes mensajes; navegadores que posibilitan la descarga de archivos al mismo tiempo que recorren otras páginas; aplicaciones basadas en interfaces de usuarios con varios procesos en front-end y back-end; simuladores de fenómenos físicos; sistemas de gestión de bases de datos.

Un ejemplo de concurrencia para lograr mayor rapidez en la ejecución es el siguiente: *obtener la suma de los elementos de un arreglo de enteros así como la suma de los cuadrados de los elementos.*
El programa podría componerse de dos procesos que se ejecuten en paralelo; uno para calcular la suma de los elementos del arreglo mientras el otro proceso calcula la suma de los cuadrados de los mismos elementos.

```
int LARGO 300;

Proceso sumaElementos (int arreglo[])
        int i, sumita;
        begin
                sumita = 0;
                for i = 0; i < LARGO; i++
                sumita = sumita + arreglo[i];
        return sumita;
end;

Proceso sumaCuadraElem (int arreglo[])
        int i, sumaCuadra;
        begin
                sumaCuadra = 0;
                for i = 0; i < LARGO; i++
                sumaCuadra = sumaCuadra + (arreglo[i])^2;
```

```
        return sumaCuadra;
end;


Programa SumaYsumaCuadrados;
        int suma, sumaCuadrados;
        int elementos[LARGO];
begin
        cobegin
                suma = sumaElementos(elementos);
                sumaCuadrados = sumaCuadraElem(elementos);
        coend
end.
```

Dos temas básicos de la concurrencia

Dos temas principales relativos a los programas concurrentes son la **sincronización** y la **comunicación** de procesos.
Los lenguajes específicos para escribir programas concurrentes deberán proveer una forma de restringir la ejecución del programa de manera que sólo órdenes aceptables puedan ser utilizadas, es decir, los lenguajes concurrentes deben proveer facilidades para la sincronización y comunicación de procesos.
Por comunicación se entiende el paso de información entre procesos.
Se puede definir la sincronización entre procesos como la satisfacción de restricciones temporales en la interacción de los procesos.
Por condición de sincronización se entiende un requisito por el cual, un proceso no puede llevar a cabo una acción hasta que otro proceso realice otra acción o se encuentre en un estado predefinido.

Definición. *Sincronización entre procesos.*
Es el hecho de que un proceso debe esperar hasta que se complete cierta operación o evento que está siendo realizado por otro proceso, antes de proceder. Por ejemplo, un proceso escritor puede estar escribiendo datos en algún área de memoria, mientras otro proceso lector puede estar leyendo en esa misma área. El lector y el escritor deben sincronizarse de manera que el escritor no sobre escriba lo que está siendo leído por el lector.
Definición. *Comunicación entre procesos.*
Se produce comunicación cuando dos procesos intercambian información entre ellos.

Al hablar de sincronización surge el concepto de acciones atómicas.

Definición. Acción atómica.
Acciones atómicas son aquellas que aparecen como una operación indivisible.
Es necesario poder construir acciones atómicas arbitrarias que marquen una secuencia de instrucciones, de manera de considerarlas una unidad atómica de computación.
Se trata de poder declarar como ilegal todas las órdenes de ejecución que no son aceptables mientras se da completa libertad de elección entre las órdenes de ejecución

aceptadas. Éste es el rol de las facilidades de sincronización de los lenguajes concurrentes.

Definición. Sección crítica (CS) es una pieza de código que debe aparecer como una acción atómica.
Se cumple lo siguiente:

$\forall$ e ● (CS_iA -> CS_jB) o (CS_jB -> CS_iA) para todo par de procesos A, B y repeticiones de sus secciones críticas i, j.

Definición. Operaciones serializables. Dos operaciones son serializables cuando pueden ser ejecutadas en cualquier orden pero sin solapamientos.

Arquitecturas de hardware que soportan programación concurrente.

Existen dos tipos de sistemas:
- uniprocesadores,
- multiprocesadores.

Un sistema uniprocesador ejecuta a lo más una instrucción a la vez.

Ejercicio: elaborar un apunte acerca de los modelos de arquitectura de Flynn.

Razones para ejecutar programas concurrentes en sistemas uniprocesadores

- necesidad o deseo de mejorar la utilización de la UCP cuando se corren procesos acotados por I/O.
- proveer un servicio interactivo a distintos y múltiples usuarios,
- aprovechar la ventaja del orden parcial del paradigma concurrente cuando el dominio del problema no está totalmente ordenado.

Multiprocesadores.

A grandes rasgos, se distinguen dos tipos de multiprocesadores:
- en un sistema altamente acoplado se tiene una RAM compartida por varias CPUs quienes a su vez tienen memoria privada,
- en un sistema débilmente acoplado o distribuido, tal como una red de computadoras con algunos sitios que son multiprocesadores. No existe memoria compartida.

En un sistema altamente acoplado, la comunicación entre procesos puede hacerse con memoria compartida. ¿Qué pasa cuando dos procesadores intentan acceder a la misma área de memoria?
Una forma de evitar problemas es realizar arbitraje de bus, esto es, si dos procesadores intentan utilizar el bus del sistema al mismo tiempo, el hardware serializa estos accesos, es decir que son acciones atómicas.
Un sistema distribuido tiene enlaces de comunicación entre los procesadores: son los mensajes que intercambian.

Nota: un lenguaje concurrente es el Pascal F-C, desarrollado por Davies & Burns en 1990. F-C significa functionally concurrent.

1972 aparece el lenguaje Concurrent Pascal, de Brinch – Hansen.

Definición. Programa concurrente es un conjunto de procesos secuenciales ejecutándose simultáneamente. Cada proceso es un programa secuencial en el sentido que tiene su propio código y sus propios datos y se considera que es ejecutado en una computadora con cpu, memoria y canales de entrada/salida.

Definición. Proceso.
El concepto tradicional de proceso, en el ámbito de los sistemas operativos que se define como secuencia de instrucciones que ejecutan una única actividad, de los años ochenta, no es adecuado en ambientes distribuidos o para ambientes mono procesadores con alta concurrencia.
Definición: Un proceso es un entorno de ejecución formado por uno o más hilos.
Definición: Un hilo es una abstracción del sistema operativo asociado a una actividad.
Definición: Un entorno de ejecución equivale a una unidad de gestión de recursos, una colección de recursos locales gestionados por el núcleo sobre los que tienen acceso los hilos.
Un entorno de ejecución es:

- un espacio de direcciones (unidad de gestión de memoria virtual, con varias regiones. Puede haber varias regiones de memoria compartida; esto se da cuando se asocia la misma memoria física a una o más regiones pertenecientes a otros espacios de direcciones. Pueden ser regiones compartidas tales como bibliotecas, kernel (código y datos del núcleo),
- recursos de comunicación y sincronización de hilos, como semáforos e interfaces de comunicación,
- recursos de alto nivel como ventanas y archivos abiertos.

Varios hilos pueden compartir un mismo entorno de ejecución, sin embargo, no comparten pila.
Existen algunos sistemas operativos que admiten un único hilo por proceso.
Definición: proceso pesado es un entorno de ejecución.
Definición: proceso ligero es un hilo.

En el caso de un proceso UNIX, éste tiene tres regiones: la región de texto de tamaño fijo que incluye al código del programa; el heap o cúmulo y la pila o stack.
Estudiando el diagrama de transición de estados de los procesos UNIX, se quiere destacar lo siguiente:

La diferencia entre los estados “ready to run” y “preempted” es que preempted es ready to run en memoria, sin embargo, se muestran como distintos para destacar que un proceso que corre en modo kernel sólo puede ser “preempted” cuando va a retornar al modo usuario.
Un proceso UNIX está en estado “creado” cuando el padre ejecuta la llamada a fork(). Eventualmente, de este estado se pasa a “ready to run”.
“Ready to run in memory”: es un estado en el cual el planificador del sistema operativo, eventualmente lo elegirá para ejectuarlo. Entonces, el proceso comenzará corriendo en modo kernel lo que completa la llamada a fork().

Corriendo en modo usuario es el estado en el cual luego de completar fork() ya cuenta con el recurso de procesador. Luego de un tiempo, el proceso podría ser interrumpido. Para correr en modo kernel otra vez, cuando el manejador de interrupciones del reloj termina de completar la interrupción, el kernel puede decidir de planificar otro proceso a ejecutar, de manera que nuestro proceso puede pasar a estado preempted.
Estado "dormido" o "asleep in memory": cuando el proceso debe esperar por una operación de I/O, el manejador de interrupciones de I/O hará que pase a listo para correr en memoria.
El estado "ready to run swapped" resulta cuando el proceso "swapper", intercambia el proceso para dar lugar a otro.
El estado "exit" de un proceso es cuando se completa el mismo, invoca la llamada exit() y así entra en modo kernel y por último pasa a estado "zombie".
El proceso no tiene control sobre sus estados y transiciones. Depende del kernel.
Un proceso puede hacer llamadas al sistema para pasar de corriendo en modo usuario a corriendo en modo kernel y entrar a modo kernel por su propia voluntad. Pero no controla sobre cuándo volver del modo kernel al modo usuario.
Un proceso puede terminar por su propia voluntad (exit()) pero también lo pueden terminar de otras maneras (kernel).
Nota: el swapper tiene pid == 0
Regla: ningún proceso puede hacer que otro pase a estado "preempted", cuando el otro proceso se esté ejecutando en modo kernel.

Definición. *Hilo.* Secuencia de control dentro de un proceso, el cual ejecuta sus instrucciones de manera independiente. Consta de un espacio de usuario y comparte la información, el código, los datos con el proceso.
Si un hilo modifica una variable del proceso, los demás hilos ven el cambio.
Los cambios de contexto entre hilos consumen menos tiempo.

Existen tres estándares de implementación de hilos:
Propietarios: WIN 32 y OS/2
IEEE 1003.1c o POSIX (o pthreads), disponible para la mayoría de las implementaciones.
Los hilos de Java se implementan en la MVJ construida sobre los hilos del sistema operativo nativo.
Sistemas operativos como Solaris definen el concepto de procesador lógico y LWP o "light weight process". Esto hace que existan planificaciones a dos niveles, nivel 1 para asignar hilos de usuarios a procesadores lógicos y nivel 2 para asignar procesadores lógicos a procesadores físicos.
Existen tres técnicas para planificar hilos sobre los recursos del kernel:

i) muchos hilos en un procesador lógico,
ii) un hilo por procesador lógico. Se hace una llamada al sistema que toma recursos del kernel. Esto es Win 32 y OS/2 y algunos POSIX
iii) muchos hilos en muchos procesadores lógicos. Una cantidad de hilos es multiplexada en una cantidad de procesadores lógicos igual o menor. Muchos hilos pueden correr en paralelo en distintas cpus. Las llamadas al sistema del tipo bloqueante no bloquean al proceso entero.

Comportamiento de un proceso.
El comportamiento de un proceso queda definido por la secuencia de sentencias que ejecuta.

Sea “a” el primer evento del proceso P. P se puede expresar como:
P = (a -> P´), donde P´ es un proceso.
P´= (b -> P”) con P” un proceso. Entonces,
P = (a –> b-> P”)
Esta secuencia puede terminar o repetirse, es decir, ejecutarse siempre.
Si la secuencia termina, se deberá distinguir entre terminación correcta (SUCCESS) o terminación con error (FAILURE):

P = (a-> b-> ……-> SUCCESS) ó
P = (a-> b-> ……-> FAILURE)

En programas concurrentes, no siempre es un error que un programa no termine, tal como sucede en los sistemas embebidos de control.
En un programa concurrente, cada lazo queda representado por un proceso cíclico:
P = (a -> P´)
P´= (b -> P´). En este ejemplo, b se repite indefinidamente.

Otro proceso a tener en cuenta es el que no tiene eventos significativos, siempre termina y termina bien. Es el proceso nulo, a veces conocido como SKIP
SKIP = (null -> SUCCESS)

Definición. Alfabeto de un proceso es el conjunto de acciones que puede contener el proceso. Notación: α(P)

Ejemplo. Si el proceso P está definido como P = (a -> P´); P´= (b -> P´), entonces α(P) = {a, b}.
Ejemplo. Un reloj puede ser modelado como un proceso el cual, una vez inicializado, repetidamente se compone del evento tic, es decir:
α(Reloj) = {inicializar, tic}.
Reloj = (inicializar ->P´)
P´= (tic -> P´).

Ejemplo. Sistema para la reservación de asientos en un teatro. El programa convierte una colección de manejadores de terminales mediante las cuales, los futuros espectadores reservan sus asientos, H1 || H2 || ……|| Hn (manejador n-simo de terminal del sistema de reservas), con α(Hi) = { mostrarAsientos, leer, reservar, emitirTicket }

Definición. Dos procesos se dicen concurrentes entre sí cuando sus respectivas instrucciones se ejecutan de manera intercalada, compiten por el mismo recurso (mínimamente el procesador) o acceden a una particular región de memoria o canal. Además, se hace necesaria una comunicación entre ellos para el pasaje de información.

Definición. Procesos paralelos son dos o más procesos que se ejecutan al mismo tiempo.

En sistemas monoprocesadores, la concurrencia no es paralelismo real. Por ejemplo, la multiprogramación, es decir, la ejecución concurrente de varios programas independientes en un solo procesador. En este caso, lo que se intercala son las operaciones de entrada / salida de los programas con procesamiento de los mismos.

Esto posibilita el servicio multiusuario.
Se utilizan variables compartidas para la sincronización y comunicación.
En sistemas multiprocesadores existe paralelismo real. Sin embargo, en general, siempre hay más procesos que procesadores.

Factores que inciden en el diseño de lenguajes concurrentes.
Existen varios factores que inciden en el diseño de lenguajes orientados a soportar naturalmente la concurrencia de procesos. En particular se destacan los siguientes:

a) estructura (del proceso),
b) nivel (del proceso),
c) inicialización (del proceso),
d) terminación (del proceso),
e) representación (del proceso).

Estructura. Las estructuras de un proceso se clasifican en

- Estática,
- Dinámica.

Estructura estática es cuando la cantidad de procesos en un programa queda fijada y conocida antes de la ejecución del programa.
Estructura dinámica es cuando los procesos pueden ser creados dinámicamente. La cantidad de procesos sólo pueden determinarse en tiempo de ejecución.

Nivel. El nivel de un proceso se refiere a la posibilidad de utilizar jerarquías de procesos. Tales jerarquías pueden construirse anidando declaraciones de procesos dentro de otras.
Ada admite anidado dinámico de procesos y Pascal concurrente sólo soporta procesos planos estáticos.
Cuando hay anidamiento de niveles, surgen las relaciones padre/ hijo entre procesos.
Relación "hijo de": un proceso es hijo del proceso en el cual está declarado.
La relación hijo de tiene una propiedad que es: "el padre no puede terminar hasta que los hijos hayan terminado".
Para los lenguajes con estructura de bloques, es posible para un bloque interior (interno) o procedimiento o función, que sea el padre de un proceso.
Inicialización. Se refiere a la habilidad, dentro del lenguaje, de pasar datos de inicialización a un proceso cuando éste es creado.
Esta capacidad es útil en el caso de un arreglo de procesos porque los datos de inicialización pueden ser utilizados para indicar a cada proceso cuál miembro del arreglo es.
Teminación. Las siguientes son diferentes maneras de terminar un proceso:

1. completar la ejecución con buen éxito, es decir que la última acción es SUCCESS.
2. ocurrencia de una condición de error no tratada, es decir que la última acción es FAILURE,
3. suicidio por ejecución de una sentencia de auto-terminación, es decir una acción FAILURE forzada internamente,
4. aborto forzado por otro proceso, es decir FAILURE forzado externamente,
5. finalizar la ejecución porque ya no es más necesario,
6. nunca finalizar, como resultado de una condición de error, es decir, la ocurrencia de un conjunto de acciones recursivas no intencionales,

7. nunca finalizar como resultado de la ejecución de un lazo que no termina, es decir un conjunto de acciones recursivas intencionales.

La manera 5 significa que si un proceso desea terminar y su proceso padre desea que termine y todos los demás procesos hijos del proceso padre están o bien terminados o desean terminar, entonces todos los procesos hijos y el proceso padre, terminan.

Observación: en lenguaje Ada, el padre de un proceso puede no ser el responsable de la terminación del mismo.

Algo más acerca de la representación de procesos

Se trata de describir cómo se representan en un lenguaje y qué reglas gobiernan su creación.
Algunos lenguajes concurrentes no tienen una construcción explícita para proceso. En cambio, definen una forma de denotar la ejecución concurrente de sentencias ordinarias introduciendo la estructura siguiente:
cobegin / coend (concurrent begin /end).
Ejemplo:

```
cobegin
        s1;
        s2;
        .
        .
        .
        sn;
coend
```

En el fragmento anterior, s_i, i = 1,…n, representan sentencias en un lenguaje de programación.

Estas construcciones pueden estar anidadas.
Otra manera de representar los procesos es mediante una definición explícita de los mismos.
Puede ser que exista el tipo proceso y que se definan objetos proceso. En este caso, pueden generarse más de un proceso del mismo tipo.
Cuando se utiliza la construcción "proceso" explícitamente, existen dos formas en las cuales la ejecución inicial puede ser controlada. Puede ser utilizada la construcción cobegin / coend y el segundo método involucra el uso de reglas de alcance para controlar el ejecución.
Este método se utiliza en Ada. El proceso comienza su ejecución cuando el procedimiento está listo vía su primer "begin" y luego correrá concurrentemente con el cuerpo del "procedure". Vgr:

```
procedure PROC;
        process P;
        begin
                ….
        end;
```

```
        process Q;
        begin
                ….
        end;

begin
        (* PROC, P y Q se ejecutan concurrentemente *)
end; (*PROC*)
```

Definición. *Acciones compartidas*. Si diferentes procesos en una composición tienen acciones en común, se dice que son acciones compartidas. Las acciones compartidas constituyen la manera en que la interacción de procesos puede ser modelada.