INTRODUCCIÓN A LA PROGRAMACIÓN CONCURRENTE

2013-2017-2018-2020

Historia

Edsger Wybe Dijkstra

- Premio ACM Turing 1972
  - Por sus contribuciones a la "ciencia y arte" de los lenguajes de programación.
- Contribuciones científicas fundamentales
  - diseño de algoritmo
  - lenguajes de programación
  - diseño de programa
  - sistemas operativos
  - procesamiento distribuido
  - especificación formal y verificación
  - diseño de argumentos matemáticos

Entrevista al Profesor Dr. Edsger W. Dijkstra, Austin, 03-04-1985

- Periodista
- En la práctica, la presión de la producción parece recompensar a la programación inteligente por sobre la buena programación: ¿Qué tanto estamos progresando hacia el caso en el que la buena programación también sea efectiva en lo que respecta al costo?
- Dijkstra
- Bueno, se ha dicho una y otra vez que el tremendo costo de programación es consecuencia de la mano de obra barata, lo que la vuelve muy costosa, y porque la gente se apresura a codificar. Una de las cosas que la gente aprende en las universidades hoy en día es a pensar primero; lo que vuelve al desarrollo más efectivo en término de costos. Conozco al menos una empresa de software en Francia, y debe de haber otras por lo que esta historia tiene un par de años, en la que existe una regla estricta por la que no se permite comenzar la codificación en tanto no haya transcurrido un setenta por ciento del tiempo pactado, cualquiera sea el software que deban entregar. Así que si después de nueve meses un equipo de proyecto reporta a su jefe que quieren comenzar a codificar, él les preguntará: “¿Están seguros que no queda otra cosa por hacer?” Y si ellos dicen que sí, se les dirá que el producto será despachado en tres meses. Esa compañía es muy exitosa.
- Periodista
- ¿Hay, en la producción de software…?
- Dijkstra
- Yo prefiero usar el término diseño de software, ya que es un producto abstracto.

Hitos

- ALGOL 60 fue el segundo lenguaje de programación de alto nivel, escribió el primer compilador
- En un documento de una página de 1965 introdujo el "problema de la exclusión mutua" para *n* procesos y discutió una solución al mismo. También se introdujo en este documento la noción, ya estándar, de una "sección crítica“
- Dijkstra y sus colegas en Eindhoven también diseñaron e implementaron el sistema operativo THE, que estaba organizado en capas claramente identificadas.
- En 1968 Dijkstra publicó su famoso artículo "Procesos secuenciales de cooperación", un ensayo de 70 páginas que originó el campo de la programación concurrente.
- Propuso el primer mecanismo de sincronización para procesos concurrentes, el semáforo
- Identificó el "problema del dead-lock" y propuso un elegante "algoritmo de Banker"
- Ilustró el problema del punto muerto por medio del "problema de los filósofos comensales“. . El trabajo también condujo a una intensa investigación de los mecanismos de sincronización de alto nivel, que finalmente condujo al concepto de un monitor
- Dio origen a uno de los principales enfoques de la computación tolerante a las fallas "Sistemas autoestabilizadores a pesar del control distribuido
- En 1976 publicó A Discipline of Programming (Una disciplina de programación), que presentaba su método de desarrollo sistemático de programas junto con sus pruebas de corrección

Estilo de trabajo

- Nunca escribió sus artículos usando una computadora. Prefería confiar en su máquina de escribir y más tarde en su pluma estilográfica
- Copias de sus artículos se enviaban a algunos amigos y asociados que luego servían como nodos de origen de los centros de distribución, rara vez superan las 15 páginas y están numerados consecutivamente
- están disponibles en el sitio web http://www.cs.utexas.edu/users/EWD/
- Dijkstra aplicó a su trabajo un riguroso procedimiento de autoevaluación y sólo una pequeña fracción de sus trabajos fueron finalmente sometidas a revistas arbitradas.
- Escribió sus artículos en un estilo único caracterizado por la concisión, la economía del argumento y la claridad de la exposición. Cada frase fue cuidadosamente cincelada. Cada párrafo era impactante.

Vida

- Estilo
  - Dijkstra fue un modelo de honestidad e integridad
  - La mayoría de sus publicaciones fueron escritas sólo por él.
  - Nunca tuvo una secretaria y se ocupó de toda su correspondencia solo.
  - Nunca buscó fondos en forma de subvenciones o consultoría y nunca prestó su nombre a las iniciativas a las que no contribuiría de manera sustancial.
  - Su suprema confianza en sí mismo se unió a un estilo de vida notablemente modesto, hasta el punto de ser espartano.
- Legado
  - El inmenso coraje intelectual y la audacia de Dijkstra, y las profundas, pero sorprendentemente simples y elegantes ideas, cambiaron el curso de la informática.
  - Su integridad como científico y como persona en la vida privada no puede ser igualada.

Introducción

El problema de la corrección del software

- Poder garantizar la corrección del software que construimos es una tarea deseable.
  - En algunas aplicaciones, es, sin duda, crucial:
    - Software para equipamiento médico
    - Software para el control de vehículos
    - Software para el control de procesos (algunas aplicaciones financieras)

- Estos sistemas, cuyas fallas pueden ocasionar daños de gran importancia, incluyendo pérdida de vidas humanas, catástrofes ecológicas, grandes pérdidas financieras, etc.

Se denominan sistemas críticos.

Las técnicas que se introducirán en esta asignatura son apropiadas para sistemas críticos que responden a comportamientos reactivos, concurrentes y/o de tiempo real.

Como responde el sistema?

Algunos Ejemplos de fallas de Software

- Pentium, FDIV
- Ariane 5:
  - 1996. Error de conversión de punto flotante de 64 bits a entero de 16 bits, overflow aritmético
- Therac-25 , one man job
  - 1985-1987. Software de control de equipamiento médico para radioterapia. Seis personas sobre-expuestas a radiación por error en el software.
- Mars Climate Orbiter
  - 1999. Problemas de representación (un módulo usaba sistema imperial y otro sistema decimal)
- Patriot Missile
  - 1991. Error de redondeo en el software de control.
- HMS Sheffield
  - Exocet amigo
- Voto electrónico
  - Integridad/Confidencialidad en la Florida
- Airbus A320
  - Problemas en el software de control. El avión se estrella.

Limitaciones del testing y la simulación

- Tanto el testing como la simulación involucran experimentos previos al lanzamiento o uso masivo del software.
- En general, ambos métodos proveen una serie de entradas al software, y estudian el comportamiento del mismo en **esos** casos.
- El testing y la simulación raramente permiten garantizar la ausencia de errores, una frase famosa al respecto:

"El testing puede confirmar la presencia de errores pero nunca garantizar su ausencia"

- Con lo que se pretende afirmar que la **verificación** sólo puede hacerlo en **teoría**.

Verificación (semi)automática de software

- Existen serias limitaciones en lo que respecta a la verificación automática de software.
  - Por ejemplo, el problema de decidir si un programa dado termina o no o no es computable.
- Sin embargo, si imponemos algunas restricciones sobre las propiedades que queremos verificar, algunas tareas podrán verificarse automáticamente sobre el sistema.
  - Model checking es un ejemplo de esto (que estudiaremos en más detalle más adelante).

El enfoque de modelar

- Aquí presentamos el enfoque de modelo para el diseño de programas concurrentes.
- Nuestros modelos representan comportamiento verdadero de programas concurrentes escritos en Java.
- Los modelos abstractos de programas se centran en los detalles reales relacionados con la representación de datos, asignación de recursos y la interacción con el usuario.
- Aquí nos centramos en la concurrencia.
- Los alumnos serán motivados para investigar el comportamiento concurrente de diferentes programas.

El enfoque de modelar

- Introduciremos mecanismo para verificar que el modelo satisface condiciones particulares de seguridad y las propiedades de progreso, que son requeridas por las aplicaciones.

- Estos mecanismos de verificación o algorítmica son posible gracias a una herramienta de modelado, con la que comprobamos exhaustivamente el modelo utilizando, lo que permite comprobar propiedades deseables como indeseables para todas las posibles secuencias de eventos y acciones.

El enfoque de modelar

- Los modelos presentados se basan en máquinas de estados finitos y Redes de Petri.
  - Máquinas de estados finitos son familiares para muchos programadores e ingenieros.
  - Se utilizan para especificar el comportamiento dinámico de los objetos en métodos bien conocidos de diseño orientado a objetos.

- También se utiliza ampliamente en el diseño de circuitos digitales, usados en técnicas digitales.
  - Las máquinas de estado, tienen una semántica intuitivos, fáciles de entender y una representación gráfica sencilla.
  - Las máquinas de estados, tienen propiedades matemáticas bien definidas, que facilitan el análisis formal y la comprobación mecánica de sistemas, evitando el tedio y la introducción de errores (error) inherentes al trabajo manual e intelectual de los métodos formales.

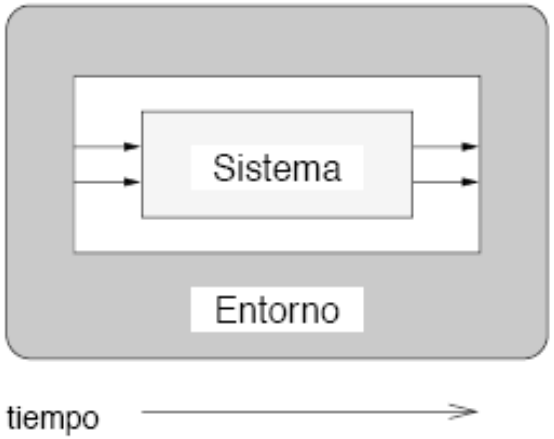
El enfoque de modelar

- Hay que notar que la representación gráfica de las máquinas de estado, limita seriamente la complejidad de los problemas que se pueden abordar.

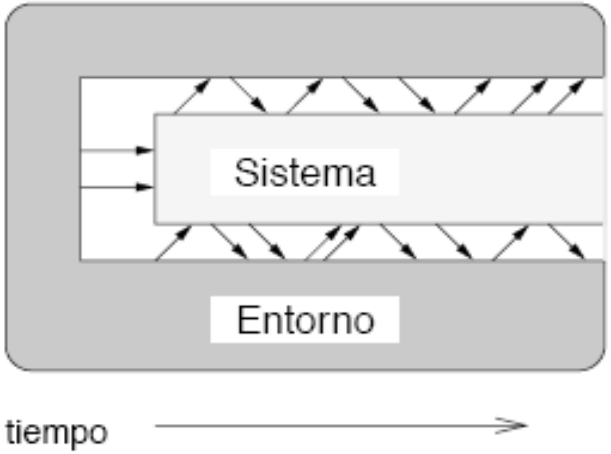
- Por lo que usaremos Redes de Petri.

Características Reactiva de los Sistemas Concurrentes

- Muchos programas concurrentes suelen ser reactivos, es decir, su funcionalidad involucra la interacción permanente con el ambiente (y otros procesos).



Sistema funcional o sistemas transformacionales



Sistemas reactivos

- Los sistemas reactivos tienen características diferentes a las de los programas transformacionales. En muchos casos, éstos no computan resultados, y suele no requerirse que terminen.
- Ejemplos de sistemas reactivos son: sistemas operativos, software de control, hardware, etc.

Interacción de programas concurrentes

- Los programas concurrentes están compuestos por procesos (o threads, o componentes) que necesitan interactuar.
  - Existen varios mecanismos de interacción entre procesos.
  - Entre éstos se encuentran la memoria compartida y el pasaje de mensajes.

- Además, los programas concurrentes deben, en general, colaborar para llegar a un objetivo común, para lo cual la sincronización entre procesos es crucial.

Problemas comunes de los programas concurrentes

- **Violación de propiedades universales** (invariantes)
- **Starvation** (inanición): Uno o más procesos quedan esperando indefinidamente un mensaje o la liberación de un recurso
- **Deadlock**: dos o más procesos esperan mutuamente el avance del otro
- Problemas de **uso no exclusivo** de recursos compartidos
- **Livelock**: Dos o más procesos no pueden avanzar en su ejecución porque continuamente responden a los cambios en el estado de otros procesos

Ejemplo

```
1  package default_package;
2
3  public class Main {
4
5      /**
6       * @param args
7       */
8      public static void main(String[] args) {
9          System.out.println("Hello World!");
10
11         Variable variable = new Variable();
12
13         Process1 p1 = new Process1(variable);
14         Process2 p2 = new Process2(variable);
15
16         Thread t1 = new Thread(p1);
17         Thread t2 = new Thread(p2);
18
19         t1.start();
20         t2.start();
21
22         System.out.println("Bye World!");
23     }
24
25
26 }
```

Ejemplo

```java
package default_package;

public class Process1 implements Runnable {

    private Variable v;

    public Process1(Variable variable) {
        this.v = variable;
    }

    public void run() {
        while (true) {
            v.y1 = v.y2 + 1;

            while (!(v.y2 == 0 || v.y1 < v.y2)) {}

            v.inCritical ++;
            v.inCritical --;
            v.y1 = 0;

            System.out.print("Prcess 1 - inCritical    ");
            System.out.println(v.inCritical);

        }
    }
}
```

```java
package default_package;

public class Process2 implements Runnable {

    private Variable v;
    private String s;

    public Process2(Variable v) {
        this.v = v;
    }


    public void run() {
        while (true) {
            v.y2 = v.y1 + 1;

            while (!(v.y1 == 0 || v.y2 < v.y1)) {}

            v.inCritical ++;
            v.inCritical --;
            v.y2 = 0;
            System.out.print("Prcess 2 - inCritical    ");
            System.out.println(v.inCritical);
        }
    }

}
```

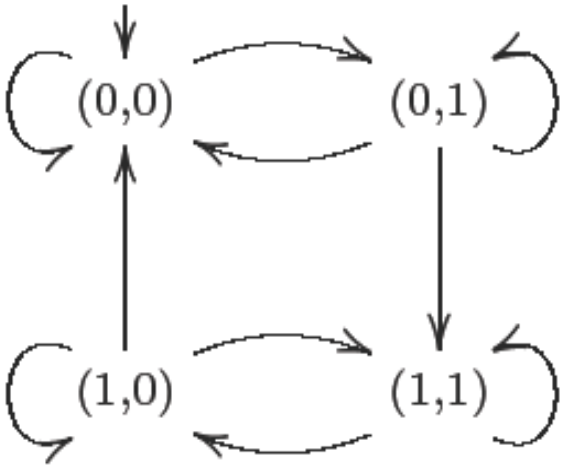
```java
package default_package;

public class Variable {

    int y1 = 0;
    int y2 = 0;
    int inCritical = 0;
}
```

Semántica de programas concurrentes

- Una semántica típica para programas concurrentes está basada en sistemas de transición de estados.
- Un sistema de transición de estados es un grafo dirigido en el cual:
  - los nodos son los estados del sistema (posiblemente infinitos estados)
  - las aristas son las transiciones atómicas de estados en estados, dadas por las sentencias del sistema.
  - Hay un nodo distinguido que reconoceremos como el estado inicial.

Semántica de programas concurrentes

- Una semántica típica para programas concurrentes está basada en sistemas de transición de estados.
- Un sistema de transición de estados es un grafo dirigido en el cual:
- $(S, s_0, \longrightarrow)$
- $S\ es\ el\ conjunto\ de\ estados$
- $s_0\ es\ el\ estado\ inicial$
- $Relacion\ de\ Transición$
  - $S = \{0,1\} x\ \{0,1\}$
  - $s_0 = (0,0)$
  - $(x,y) \longrightarrow (x,0)$
  - $(x,y) \longrightarrow (x,1)$
  - $(x,y) \longrightarrow (y,y)$



Ejecuciones de un programa

- El conjunto de todas las ejecuciones determina el comportamiento de un programa concurrente modelado con

Si	(0, 0)(0, 0)(0, 1)(1, 1)(1, 0)(1, 1)(1, 0)(1, 0)(1, 1)(1, 0)(0, 0)(0, 1)(0, 1)(1, 1) · · ·
No	(0, 1)(0, 0)(0, 1)(1, 1)(1, 0)(1, 1)(1, 0)(1, 0)(1, 1)(1, 0)(0, 0)(0, 1)(0, 1)(1, 1) · · ·
No	(0, 0)(0, 1)(1, 0)(1, 1)(0, 0)(1, 1)(1, 0)(1, 0)(1, 1)(1, 0)(0, 0)(1, 1)(0, 1)(1, 1) · · ·

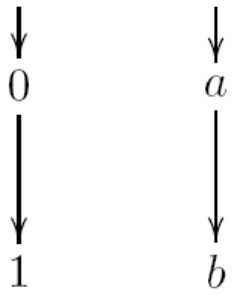
Porque la primera secuencia de estados es posible y la segunda y tercer a NO?

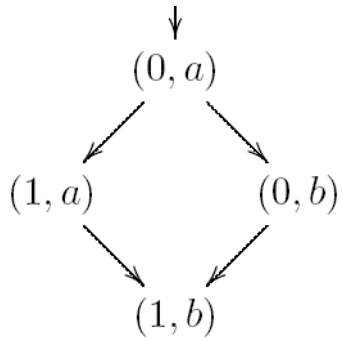
¿Cómo se ejecutan los procesos oncurrentes?

- De acuerdo al modelo computacional descrito

Los procesos concurrentes se ejecutan intercalando las acciones atómicas que los componen.

- Llamamos a esto, interleaving.
- El orden en que se ejecutan las acciones atómicas no puede decidirse en general, y un mismo par de procesos puede tener diferentes ejecuciones debido al no determinismo en la elección de las acciones atómicas a ejecutar.





¿Cómo se ejecutan los procesos concurrentes?

- El conjunto de estados está dado por la combinación de todos los valores posibles de las variables globales y1, y2 e in_critical, además de dos variables implícitas: los program counters (pc) de los dos procesos.
- El estado inicial es aquel en el cual las tres variables valen 0 y cada pc se sitúa al inicio de cada proceso.
- Por cada sentencia atómica tenemos una transición.
- Por ejemplo, la sentencia in_critical++ define un conjunto de transiciones donde cada una relaciona todos los estados con aquellos en los cuales y1 e y2 no cambian su valor, e in_critical se incrementa en uno (y el pc del proceso correspondiente también se incrementa).

Razonamiento sobre programas concurrentes

- En general, es muy difícil razonar sobre programas concurrentes.
  - Luego, garantizar que un programa concurrente es correcto es también muy difícil.
- Una de las razones tiene que ver con que diferentes interleavings de acciones atómicas pueden llevar a diferentes resultados o comportamientos de los sistemas concurrentes.
- El número de interleavings posibles, por su parte, es en general muy grande, lo que hace que el testing difícilmente pueda brindarnos confianza de que nuestros programas concurrentes funcionan correctamente.
- En el ejemplo anterior es prácticamente imposible realizar el test que lleve al overflow.
- El espacio de estados crece exponencialmente con el número de componentes y variables.

Modelos de programas concurrentes

- **Abstracción**

  "Acto mental en el que se aísla conceptualmente una propiedad o función concreta de un objeto, y se piensa qué es, ignorando otras propiedades del objeto en cuestión"

  Manera de ocultar los detalles de implementación de ciertas funcionalidades

  - Una forma de aliviar, en parte, el problema de razonar sobre programas concurrentes es considerar representaciones abstractas de éstos.
  - Estas representaciones abstractas, llamadas modelos, nos permiten concentrarnos en las características particulares que queremos analizar.
  - Las álgebras de procesos (Redes de Petri, CSP, CCS, ACP, LOTOS, etc.) permiten construir estos modelos, concentrándose en las propiedades funcionales de sistemas concurrentes.
  - Para esto, es importante considerar los eventos en los cuales cada proceso puede estar involucrado, y los patrones de ocurrencia que éstos siguen.

Proceso

- Formalmente un proceso es

"Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados“

  - Sistemas operativos
    - tiempo compartido
    - SMP
  - El O.S proporciona los servicios que son necesarios para que el usuario pueda ejecutar procesos
  - Al comienzo de la ejecución del programa se inicia la ejecución de un proceso
    - proceso podría crear nuevos procesos (**proceso padre**, **proceso hijo)**
    - Una vez creado un proceso hijo, la ejecución de padre e hijo transcurre de manera concurrente

Definición de proceso

- Un proceso es un programa en ejecución
  - Un proceso simple tiene un hilo de ejecución (concepto de hilo)
  - Diferencia básicas entre un programa y un proceso, un proceso es una actividad de cierto tipo que contiene:
    - programa,
    - entradas salidas
    - y estados
  - Procesos cooperantes se entiende que los procesos interactúan entre sí
  - Proceso independientes no requiere información de otros.

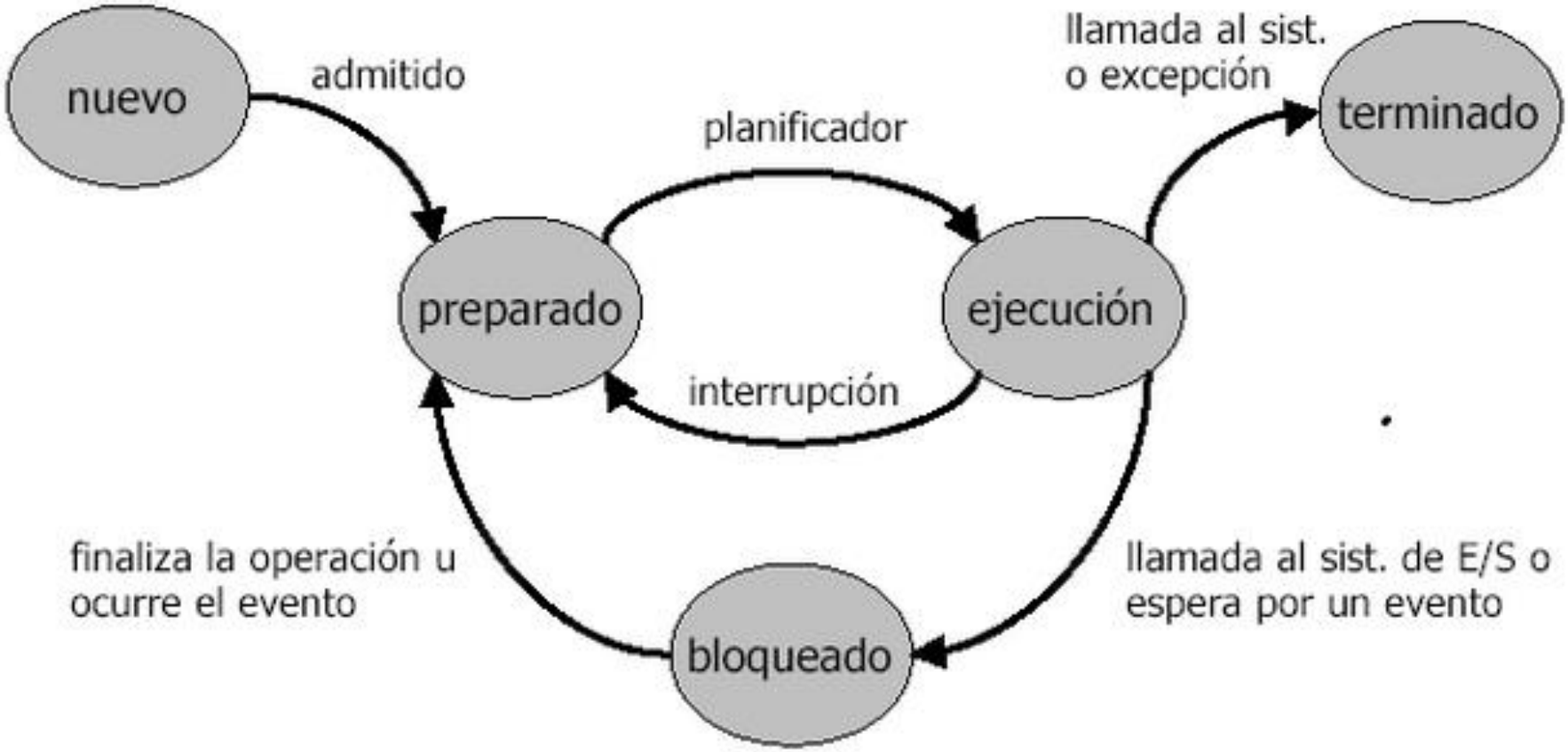
Estados de los procesos

- **3 estados**
  - ***Listo***
    - son los que pueden pasar a estado de ejecución
  - ***En ejecución***
    - son los que se están ejecutando en el procesador
  - ***Bloqueado.***
    - están esperando la respuesta de algún otro proceso para poder continuar con su ejecución
- **5 estados**
  - ***Activo***
    - está ejecutándose
  - ***Preparado***
    - todas las tareas están listas para ejecutarse pero se espera a que un/el procesador quede libre(hay otros procesos más prioritarios en ejecución)
  - ***Bloqueado o suspendido***
    - que se termine una operación de E/S o que se reciba una señal de sincronización…
  - ***Nonato***
    - indica que el programa realmente existe pero todavía no es conocido por el OS
  - ***Muerto***
    - cuando ha terminado su ejecución o el sistema operativo a detectado un error fatal

5 Estados

- El sistema operativo gestione los recursos disponibles (memoria, CPU, etc)
- estados en los que se puede encontrar un proceso en este tipo de sistemas:
  - **New**: cuando el proceso esta siendo creado.
  - **Running**: cuando el proceso se esta ejecutando.
  - **Waiting**: cuando el proceso esta esperando que se cumpla algún otro evento.
  - **Ready**: cuando el proceso esta pronto para ejecutar, esperando por la CPU.
  - **Terminated**: cuando el proceso esta terminado.

Estados


nuevo
admitido
planificador
llamada al sist.
o excepción
terminado
preparado
ejecución
interrupción
finaliza la operación u
ocurre el evento
llamada al sist. de E/S o
espera por un evento
bloqueado

Estados

- Son aquellos que compiten con el procesador o están en condiciones de hacerlo. Se dividen en:
- **Activos**
  - **Ejecución**:Estado en el que se encuentra un proceso cuando tiene el control del procesador. En un sistema monoprocesador este estado sólo lo puede tener un proceso.
  - **Preparado**: Aquellos procesos que están dispuestos para ser ejecutados, pero no están en ejecución por alguna causa (Interrupción, haber entrado en cola estando otro proceso en ejecución, etc.).
  - **Bloqueado**: Son los procesos que no pueden ejecutarse de momento por necesitar algún recurso no disponible (generalmente recursos de entrada/salida).
- **inactivos**
  - **Suspendido bloqueado:** Es el proceso que fue suspendido en espera de un evento, sin que hayan desaparecido las causas de su bloqueo.
  - **Suspendido programado**: Es el proceso que han sido suspendido, pero no tiene causa parta estar bloqueado.

Control de procesos.

- Es un registro especial BCP donde el sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular
  - **Identificador** del proceso (Process Identificator -PID-, de sus siglas en Inglés).
  - **Estado** del proceso. Por ej. listo, en espera, bloqueado.
  - **Contador de Programa**: Dirección de la próxima instrucción a ejecutar.
  - **Valores de registro de CPU**. Se utilizan también en el cambio de contexto.
  - **Espacio de direcciones de memoria**.
  - **Prioridad** en caso de utilizarse dicho algoritmo para planificación de CPU.
  - **Lista de recursos asignados** (incluyendo descriptores de archivos y sockets abiertos).
  - **Estadísticas** del proceso.
  - **Datos del propietario** (owner).
  - **Permisos asignados.**
  - **Signals pendientes** de ser servidos. (Almacenados en un mapa de bits)

Esta lista es indicativa

hilos.

- Un hilo
  - Permite a una aplicación realizar varias tareas a la vez(concurrentemente).
  - Los distintos hilos de ejecución comparten recursos
    - el espacio de memoria,
    - los archivos abiertos,
    - autenticación, etc.
  - Esta técnica simplificada permite llevar a cabo distintas funciones simultáneamente.

- Los **hilos** de ejecución que **comparten los mismos recursos**, y **estos recursos**, son en conjunto conocidos como un **proceso**

- Si un mismo proceso compartan los recursos hace que cualquiera de sus hilos pueda modificarlos
- Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente
  - Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros)
  - El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo.

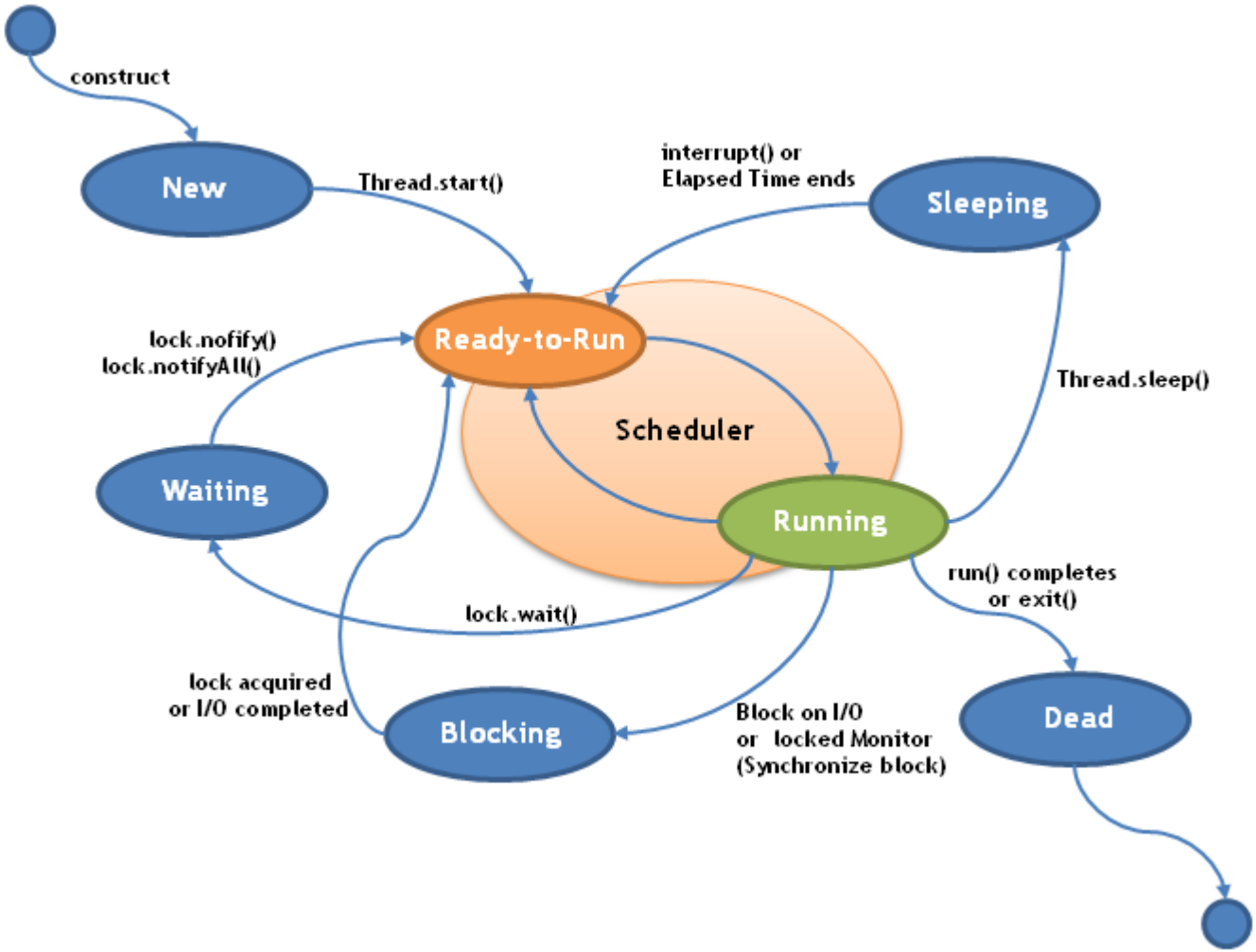
Diferencias entre hilos y procesos

- Los procesos son –generalmente– independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema.
- Los hilos generalmente comparten otros recursos de forma directa.
- Es mas simpley rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro.
  - Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen.
- Al cambiar de un proceso a otro, el sistema operativo, (mediante el dispatcher) genera lo que se conoce como overhead, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (context switch), en este caso pasar del estado de ejecución (running) al estado de espera (waiting) y colocar el nuevo proceso en ejecución.
- En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.
- Sistemas operativos como Windows y Linux dicen tener hilos "baratos", y procesos "costosos.

Estados de un hilo

- Ejecución, Listo y Bloqueado
- **Cambio de estados**
  - *Creación*
    - Cuando se crea un proceso se crea un hilo para ese proceso.
    - Luego, este hilo puede crear otros hilos dentro del mismo proceso.
    - El hilo tendrá su propio contador, registros, espacio de pila, y pasara a la cola de listos.
  - *Bloqueo*
    - Cuando un hilo necesita esperar por un suceso, se bloquea (salvando sus registros de usuario, contador de programa y punteros de pila).
    - Ahora el procesador podrá pasar a ejecutar otro hilo que esté en la cola de Listos mientras el anterior permanece bloqueado.
  - *Desbloqueo*
    - Cuando el suceso por el que el hilo se bloqueó se produce, el mismo pasa a la cola de Listos.
  - *Terminación*
    - Cuando un hilo finaliza se liberan tanto su contexto como sus pilas.

Estados de un hilo



Ventajas de los hilos contra procesos

- Si bien los hilos son generados a partir de la creación de un proceso, podemos decir que un proceso es un hilo de ejecución, conocido como Monohilo.
  - Pero las ventajas de los hilos se dan cuando hablamos de Multihilos, que es cuando un proceso tiene múltiples hilos de ejecución los cuales realizan actividades distintas, que pueden o no ser cooperativas entre sí.
  - Los beneficios de los hilos se derivan de las implicaciones de rendimiento.

Ventajas de los hilos contra procesos

1. Se tarda mucho menos tiempo en crear un hilo nuevo en un proceso existente que en crear un proceso.
   - Algunas investigaciones llevan al resultado que esto es así en un factor de 10.
2. Se tarda mucho menos en terminar un hilo que un proceso,
   - cuando se elimina un proceso se debe eliminar el BCP del mismo, mientras que un hilo se elimina su contexto y pila.
3. Se tarda mucho menos tiempo en cambiar entre dos hilos de un mismo proceso

- Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución.
  - La comunicación entre procesos debe intervenir el núcleo para ofrecer protección de los recursos y realizar la comunicación misma.
- La comunicación entre hilos no requiere la invocación al núcleo.

Por lo tanto, si hay una aplicación que debe implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de hilos que con una colección de procesos separados.

Cuando un hilo está en ejecución, posee el acceso a todos los recursos que tiene asignados la tarea.

Que tiene un hilo

- Estado.
- Contexto del procesador.
  - Punto en el que estamos ejecutando, la instrucción.
  - Se usa para reanudar un hilo que fue interrumpido con, para poder continuar la ejecución del hilo.
- Pila de ejecución donde se irá metiendo y sacando instrucciones. (Lugar donde almacenaremos las instrucciones que van a ser ejecutadas).
- Espacio de almacenamiento estático donde almacenará las variables.
- Acceso a los recursos de la tarea, que son compartidos por todos los hilos de la tarea.

Concurrencia
exclusión mutua y sincronización.

- Aclaremos algunis conceptos
  - **Multiprogramación**:
    - consiste en la gestión de varios procesos dentro de un sistema mono-procesador.
  - **Multiprocesamiento**:
    - consiste en la gestión de varios procesos, dentro de un sistema multiprocesador
  - **Procesamiento distribuido:**
    - consiste en la gestión de varios procesos, ejecutándose en sistemas de computadores múltiples y distribuidos.

- La concurrencia es fundamental en todas estas áreas y para el diseño
- La concurrencia comprende un gran número de cuestiones de diseño, incluida:
  - la comunicación entre procesos
  - compartición y competencia por los recursos
  - sincronización de la ejecución de varios procesos
  - asignación del tiempo.

- Estas cuestiones no solo surgen en entornos de multiprocesadores y proceso distribuido, sino incluso en sistemas multiprogramados con un solo procesador.

Concurrencia

- La concurrencia puede presentarse en tres contextos diferentes:
  - Múltiples aplicaciones:
    - la multiprogramación se creó para permitir que el tiempo de procesador de la máquina fuese compartido dinámicamente entre varias aplicaciones activas.
  - Aplicaciones estructuradas:
    - como ampliación de los principios del diseño modular y la programación objetos algunas aplicaciones pueden implementarse eficazmente como un conjunto de hilos concurrentes.
  - Estructura del sistema operativo
    - las mismas ventajas de estructuración son aplicables a los programadores de sistemas y se ha comprobado que algunos sistemas operativos están implementados como un conjunto de procesos o hilos.

PRINCIPIOS GENERALES DE LA CONCURRENCIA

- Multiprogramado con un único procesador, los procesos se intercalan en el tiempo aparentando una ejecución simultánea.
  - Aunque no se logra un procesamiento paralelo y produce una sobrecarga en los intercambios de procesos, la ejecución intercalada produce beneficios en la eficiencia del procesamiento y en la estructuración de los programas.
- los problemas son consecuencia de la velocidad de ejecución de los procesos que no pueden predecirse y depende de las actividades de otros procesos, de la forma en que el sistema operativo trata las interrupciones surgen dificultades

PRINCIPIOS GENERALES DE LA CONCURRENCIA

- Compartir recursos globales es riesgoso
- El hecho de compartir recursos ocasiona problemas, por esto es necesario proteger a dichos recursos.
  - seguir a los distintos procesos activos
  - asignar y retirar los distintos recursos a cada proceso activo
  - proteger los datos y los recursos físicos
  - los resultados de un proceso deben ser independientes de la velocidad a la que se realiza la ejecución de otros procesos concurrentes

INTERACCIÓN ENTRE PROCESOS

- interactúan los procesos en función del nivel de conocimiento que cada proceso tiene de la existencia de los demás. Existen tres niveles de conocimiento:
  - 1) Los procesos no tienen conocimiento de los demás
  - 2) Los procesos tienen un conocimiento indirecto de los otros: los procesos no conocen a los otros por sus identificadores de proceso, pero muestran cooperación el objeto común.
  - 3) Los procesos tienen conocimiento directo de los otros: los procesos se comunican por el identificador de proceso y pueden trabajar conjuntamente.

Competencia entre procesos por los recursos

- **Los procesos concurrentes entran en conflicto cuando compiten por el uso del mismo recurso**; dos o más procesos necesitan acceder a un recurso durante su ejecución .Cada proceso debe dejar tal y como esté el estado del recurso que utilice.
- **La ejecución de un proceso puede influir en el comportamiento de los procesos que compiten**. Por Ej. Si dos procesos desean acceder a un recurso, el sistema operativo le asignará el recurso a uno y el otro tendrá que esperar.
- **Cuando hay procesos en competencia, se deben solucionar tres problemas de control**:
  - **la necesidad de exclusión mutua**.
  - Suponiendo que dos procesos quieren acceder a un recurso no compartible.
  - A estos recursos se les llama “recursos críticos” y la parte del programa que los utiliza es la “sección crítica” del programa.
  - Es importante que sólo un programa pueda acceder a su sección crítica en un momento dado
  - **Hacer que se cumpla la exclusión mutua provoca un interbloqueo**

Competencia entre procesos por los recursos

- Otro problema es la **inanición**
  - si tres procesos necesitan acceder a un recurso, P1 posee al recurso, luego lo abandona y le concede el acceso al siguiente proceso P2, P1 solicita acceso de nuevo y el sistema operativo concede el acceso a P1 YP2 alternativamente, se puede negar indefinidamente a P3 el acceso al recurso.
- Control de competencia a los recursos.
- Cooperación entre procesos por compartimiento Comprende los procesos que interactúan con otros sin tener conocimiento explícito de ellos. Ej. : Varios procesos pueden tener acceso a variables compartidas.
- Los procesos deben cooperar para asegurar que los datos que se comparten se gestionan correctamente.
- Los mecanismos de control deben garantizar la integridad de los datos compartidos.
  - La comunicación sincroniza o coordina las distintas actividades, está formada por mensajes de algún tipo. Las primitivas para enviar y recibir mensajes, vienen dadas como parte del lenguaje de programación o por el núcleo del sistema operativo

REQUISITOS PARA LA EXCLUSIÓN MUTUA

- Sólo un proceso, de todos los que poseen secciones críticas por el mismo recurso compartido, debe tener permiso para entrar en ella en un momento dado
- Un proceso que se interrumpe en una sección no crítica debe hacerlo sin interferir con los otros procesos.
- Un proceso no debe poder solicitar acceso a una sección crítica para después ser demorado indefinidamente, no puede permitirse el interbloqueo o la inanición .
- Si ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin demora.
- No se debe suponer sobre la velocidad relativa de los procesos o el número de procesadores.
- Un proceso permanece en su sección crítica por un tiempo finito.

SINCRONIZACION

- La comunicación de un mensaje entre 2 procesos implica cierto nivel de sincronización entre ambos.
- El receptor no puede recibir un mensaje hasta que sea enviado por otro proceso.
- Además hace falta especificar que le sucede a un proceso después de ejecutar una primitiva SEND o RECEIVE.
- Considérese en primer lugar la primitiva send.
- Cuando se ejecuta una primitiva send en un proceso, hay 2 posibilidades:
  - o bien el proceso emisor se bloquea hasta que recibe el mensaje o no se bloquea.

SINCRONIZACION

- Igualmente cuando un proceso ejecuta una primitiva RECEIVE, existen 2 opciones:
  - 1) Si previamente se ha enviado algún mensaje, este es recibido y continua la ejecución.
  - 2) Si no hay ningún mensaje esperando entonces:
    - a) el proceso se bloquea hasta que llega un mensaje o,
    - b) el proceso continúa ejecutando, abandonando el intento de recepción.

Sincronización

- El emisor y el receptor pueden ser bloqueantes o no bloqueantes
- Existen 3 tipos de combinaciones pero un sistema solo implementa uno o dos.
  - I) Envío bloqueante, recepción bloqueante: tanto el emisor como el receptor se bloquean hasta que llega el mensaje; esta técnica se conoce como rendezvous.
  - II) Envío no bloqueante, recepción bloqueante: aunque el emisor puede continuar, el receptor se bloquea hasta que llega el mensaje solicitado. Es la combinación más útil.
  - III) Envío no bloqueante, recepción no bloqueante: nadie debe esperar.
- El send no bloqueante es la forma más natural para muchas tareas de programación concurrente. Un posible riesgo del send no bloquente es que por error puede llevar a una situación en la que el proceso genere mensajes repetidamente.
- Para el receive, la versión bloqueante es la mas natural para muchas tareas de programación concurrente. En general, un proceso que solicita un mensaje necesitara la información esperada antes de continuar

Principios generales de la concurrencia

- En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo aparentando una ejecución simultánea.
- Aunque no se logra un procesamiento paralelo y produce una sobrecarga en los intercambios de procesos, la ejecución intercalada produce beneficios en la eficiencia del procesamiento y en la estructuración de los programas.
- La intercalación y la superposición pueden contemplarse como ejemplos de procesamiento concurrente en un sistema monoprocesador, los problemas son consecuencia de la velocidad de ejecución de los procesos que no pueden predecirse y depende de las actividades de otros procesos.

Principios generales de la concurrencia

- Según como el sistema operativo trata las interrupciones surgen las siguientes dificultades:
  - Compartir recursos globales es riesgoso
  - Para el sistema operativo es difícil gestionar la asignación óptima de recursos.
  - Las dificultades anteriores también se presentan en los sistemas multiprocesador.
  - El hecho de compartir recursos ocasiona problemas, por esto es necesario proteger a dichos recursos.
  - Los problemas de concurrencia se producen incluso cuando hay un único procesado

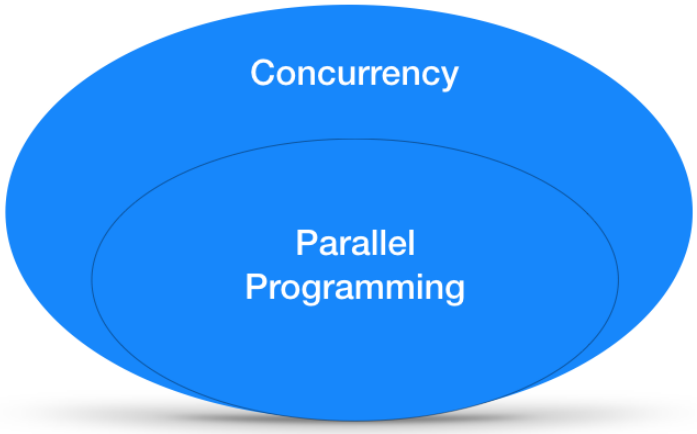
Computación vs manejo de eventos

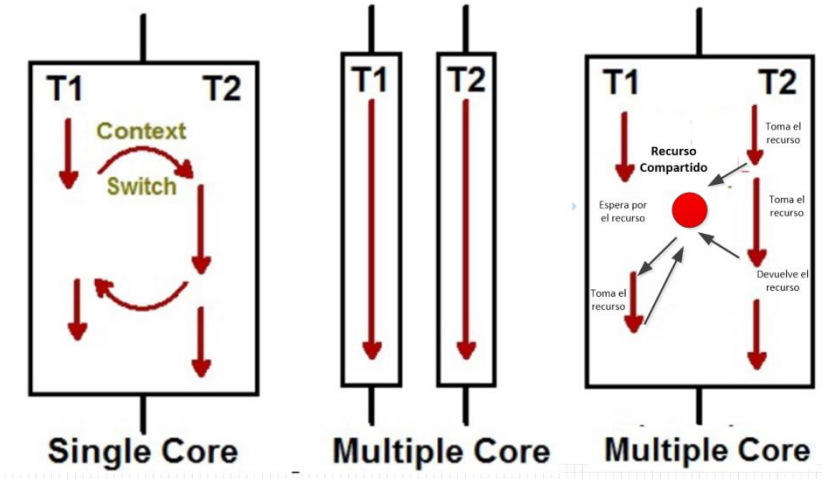
- Con sistemas de manejo de eventos como máquinas expendedoras, telefonía, servidores web y bancos, la **concurrencia es inherente al problema** :
  - debe resolver conflictos inevitables entre solicitudes impredecibles. El paralelismo es una parte de la *solución* : acelera las cosas, pero la raíz del *problema* es la concurrencia.

- Con sistemas computacionales como cajas de regalo, gráficos, visión por computadora y computación científica, la **concurrencia *no* es parte del problema** :
  - se calcula una salida de entradas conocidas de antemano, sin eventos externos. El paralelismo es donde *comienzan* los problemas : acelera las cosas, pero puede introducir errores.

Programación

- **Concurrente**
  - Un sistema concurrente correcto es aquel en el que un conjunto de cómputos avanzar colaborativamente para lo cual esta garantizada y coordinar la secuencia de las interacciones o comunicaciones entre diferentes cómputos como así también el acceso a los recursos que se comparten
- **Paralela**
  - Es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo)





Historia

Carl Adam Petri

- 12 de julio de 1926-2 de julio de 2010
- Matemático e informático alemán
- Hijo del un doctor en matemático y jugador de ajedrez Max Petri
- A los 13 años, Petri inventó las redes de Petri con sus gráficos y reglas para describir los procesos químicos.
- En 1941 se enteró de Konrad Zuse a través de su padre.y su trabajo con máquinas calculadoras, con lo que se ocupó de las leyes físicas con el propósito de la informática y los autómatas e incluso construyó una pequeña computadora analógica .
- Se doctoro, y su disertación fue "Comunicación con autómatas" y trata, entre otras cosas, con modelos simultáneos (Petri-Netze)
- uno de los padres fundadores del maravilloso campo de la concurrencia

Carl Adam Petri

- Discurso de Robin Milner al recibir el premio Tuning
  - Carl Adam Petri fue pionero en el modelado científico de la concurrencia de sistemas discretos
  - El trabajo de Petri tiene un lugar seguro en la raíz de la teoría de la concurrencia!
  - Las redes de Petri se han convertido en una herramienta estándar para modelar y analizar procesos donde la concurrencia juega un papel prominente.
  - Usando unos pocos conceptos básicos (lugares, transiciones, arcos y fichas) y la simpleza de un disparo, uno entra en un nuevo "mundo" donde es posible modelar una amplia gama de comportamientos y estudiar fenómenos no triviales (conflicto, concurrencia, confusión, etc.).
  - Los principios rectores de mi actual área de investigación:
    - - GP1: La concurrencia debe ser un punto de partida para el diseño y análisis de sistemas y no se añadió como un pensamiento posterior (localidad de las acciones).
    - - GP2: Un formalismo debe ser consistente con las leyes de la física y no tomar ninguna atajos en el nivel fundamental.