Programación de Servicios y Procesos

Tema 1 Programación Multiproceso

José Luis González Sánchez

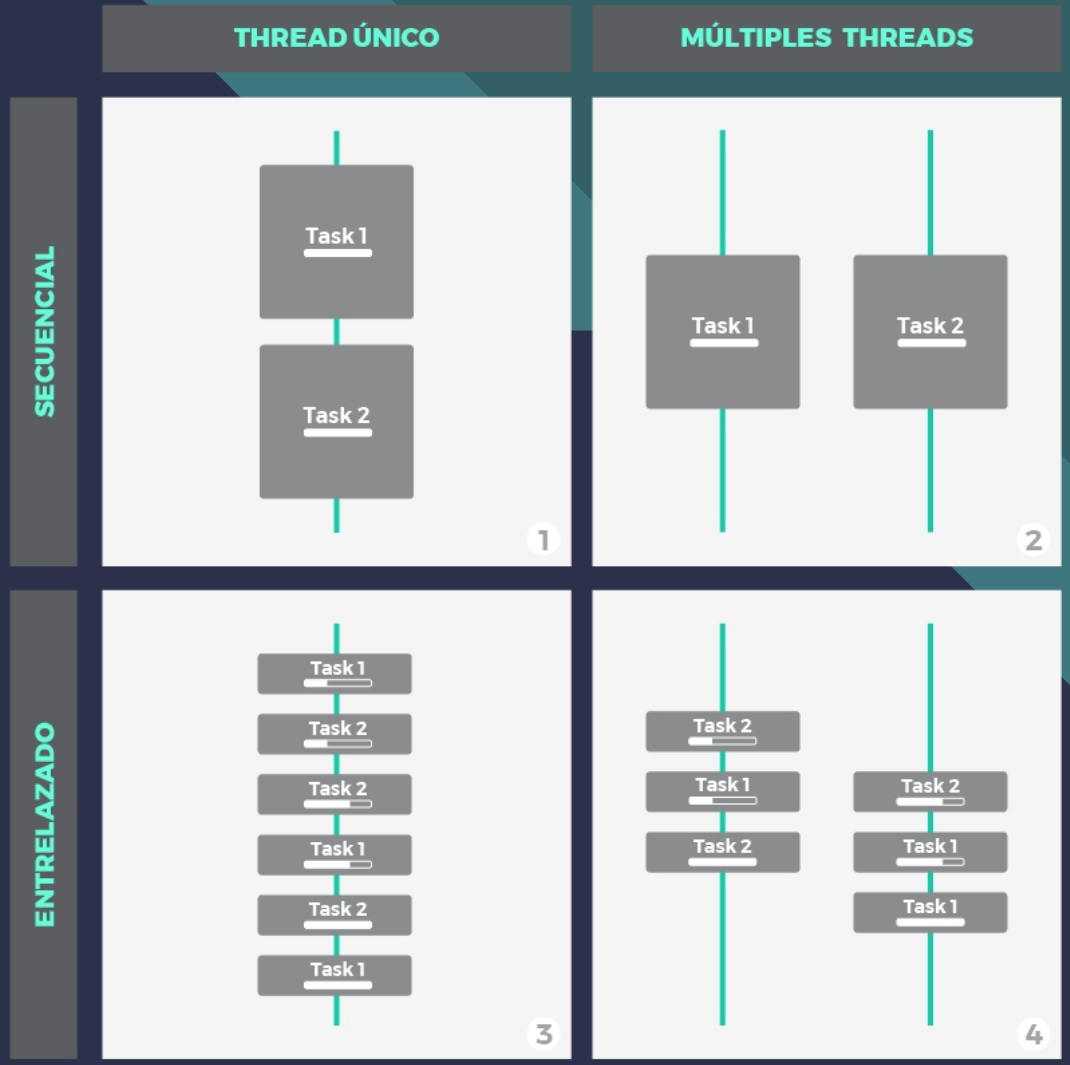
Contenidos

Concurrencia y Paralelismo

Cuando se ejecutan varias cosas

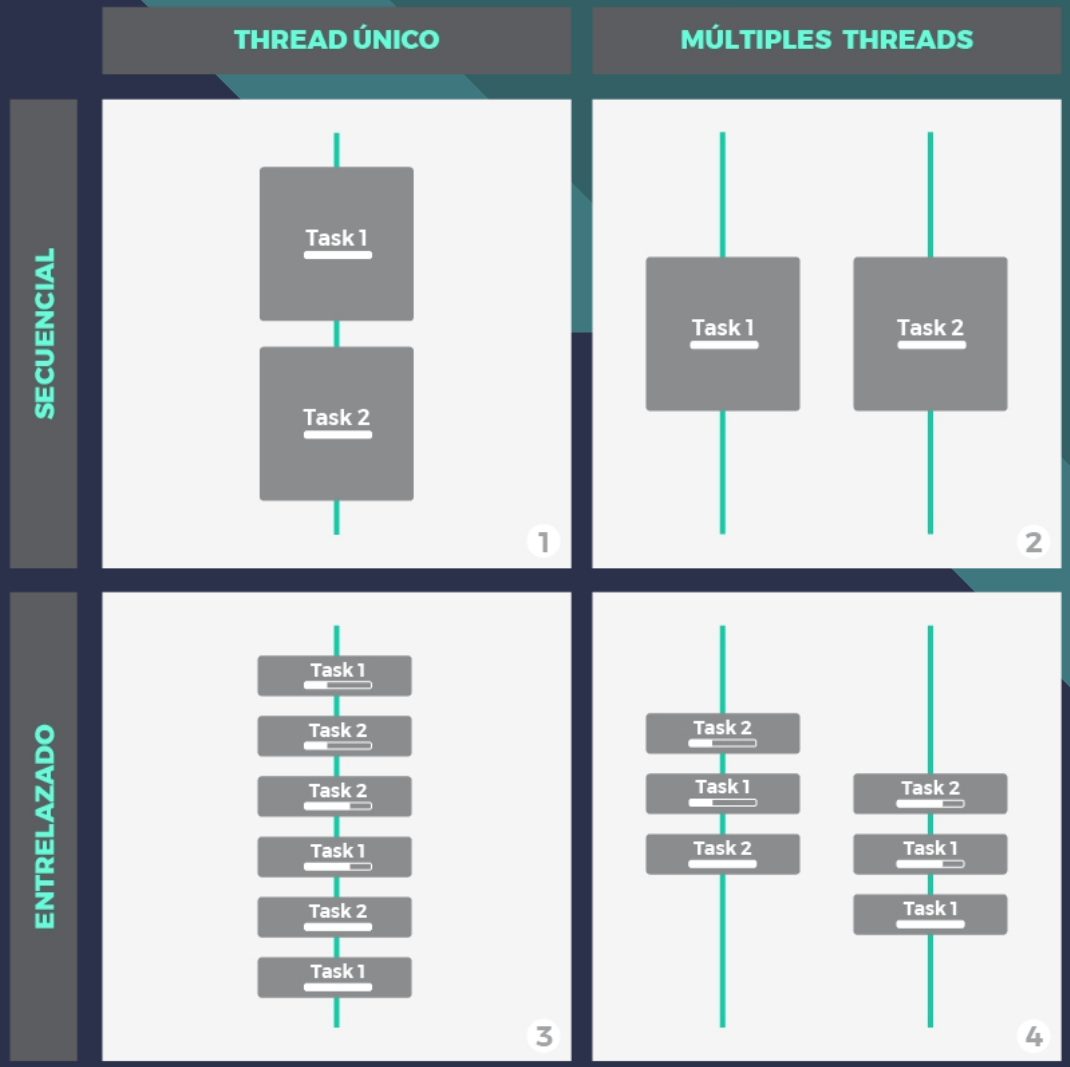
Concurrencia y Paralelismo

- Concurrencia y paralelismo son conceptos relacionados pero con un importante matiz de diferencia entre ellos. Es por esto que muy a menudo se confunden y se utilizan erróneamente. Vayamos al grano:
  - **Concurrencia: cuando dos o mas tareas progresan simultáneamente.**
  - **Paralelismo: cuando dos o mas tareas se ejecutan, literalmente, a la vez, en el mismo instante de tiempo.**
- Nótese la diferencia: que varias tareas progresen simultáneamente no tiene porque significar que sucedan al mismo tiempo. Mientras que la concurrencia aborda un problema más general, el paralelismo es un sub-caso de la concurrencia donde las cosas suceden exactamente al mismo tiempo.
- Mucha gente aún sigue creyendo que la concurrencia implica necesariamente más de un thread. Esto no es cierto. El entrelazado (o multiplexado), por ejemplo, es un mecanismo común para implementar concurrencia en escenarios donde los recursos son limitados. Piensa en cualquier sistema operativo moderno haciendo multitarea con un único core. Simplemente trocea las tareas en tareas más pequeñas y las entrelaza, de modo que cada una de ellas se ejecutará durante un breve instante. Sin embargo, a largo plazo, la impresión es que todas progresan a la vez.



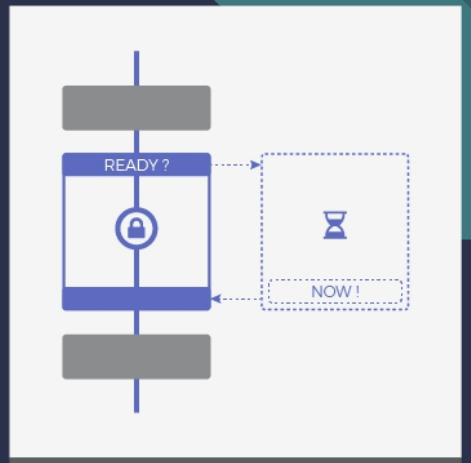
Concurrencia y Paralelismo

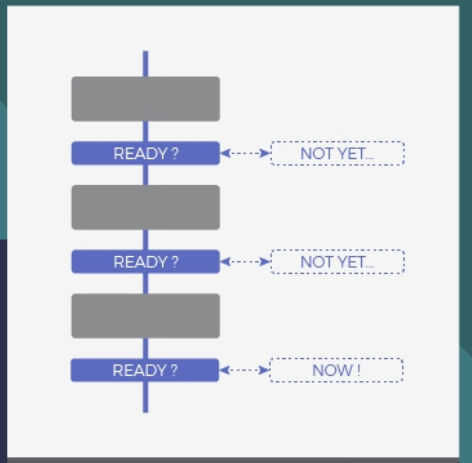
- Escenario 1: no es ni concurrente ni paralelo. Es simplemente una ejecución secuencial, primero una tarea, después la siguiente.
- Escenario 2, 3 y 4: son escenarios donde se ilustra la concurrencia bajo distintas técnicas:
- Escenario 3: muestra como la concurrencia puede conseguirse con un único thread. Pequeñas porciones de cada tarea se entrelazan para que ambas mantengan un progreso constante. Esto es posible siempre y cuando las tareas puedan descompuestas en subtareas mas simples.
- Escenario 2 y 4: ilustran paralelismo, utilizando multiples threads donde las tareas o subtareas corren en paralelo exactamente al mismo tiempo. A nivel de thread, el escenario 2 es secuencial, mientras que 4 aplica entrelazado.



Concurrencia y Paralelismo

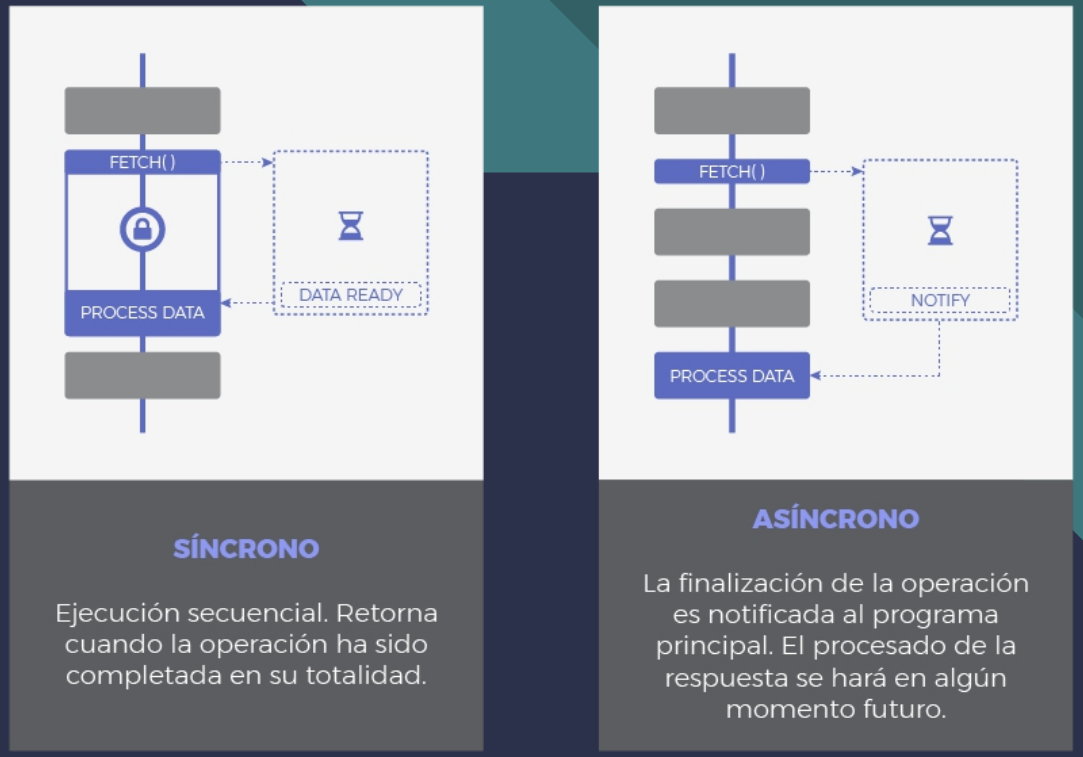
- Bloqueante vs No Bloqueante
  - **Bloqueante:** Una llamada u operación bloqueante no devuelve el control a nuestra aplicación hasta que se ha completado. Por tanto el thread queda bloqueado en estado de espera.
  - **No Bloqueante:** Una llamada no bloqueante devuelve inmediatamente con independencia del resultado. En caso de que se haya completado, devolverá los datos solicitados. En caso contrario (si la operación no ha podido ser satisfecha) podría devolver un código de error indicando algo así como 'Temporalmente no disponible', 'No estoy listo' o 'En este momento la llamada sería bloqueante. Por favor, posponga la llamada'. En este caso se sobreentiende que algún tipo de polling debería hacerse para completar el trabajo o para lanzar una nueva petición más tarde, en un mejor momento.





Concurrencia y Paralelismo

- Síncrono vs Asíncrono
  - **Síncrono:** es frecuente emplear 'bloqueante' y 'síncrono' como sinónimos, dando a entender que toda la operación de entrada/salida se ejecuta de forma secuencial y, por tanto, debemos esperar a que se complete para procesar el resultado.
  - **Asíncrono:** la finalización de la operación I/O se señaliza más tarde, mediante un mecanismo específico como por ejemplo un callback, una promesa o un evento (se explicarán después), lo que hace posible que la respuesta sea procesada en diferido. Como se puede adivinar, su comportamiento es no bloqueante ya que la llamda I/O devuelve inmediatamente.

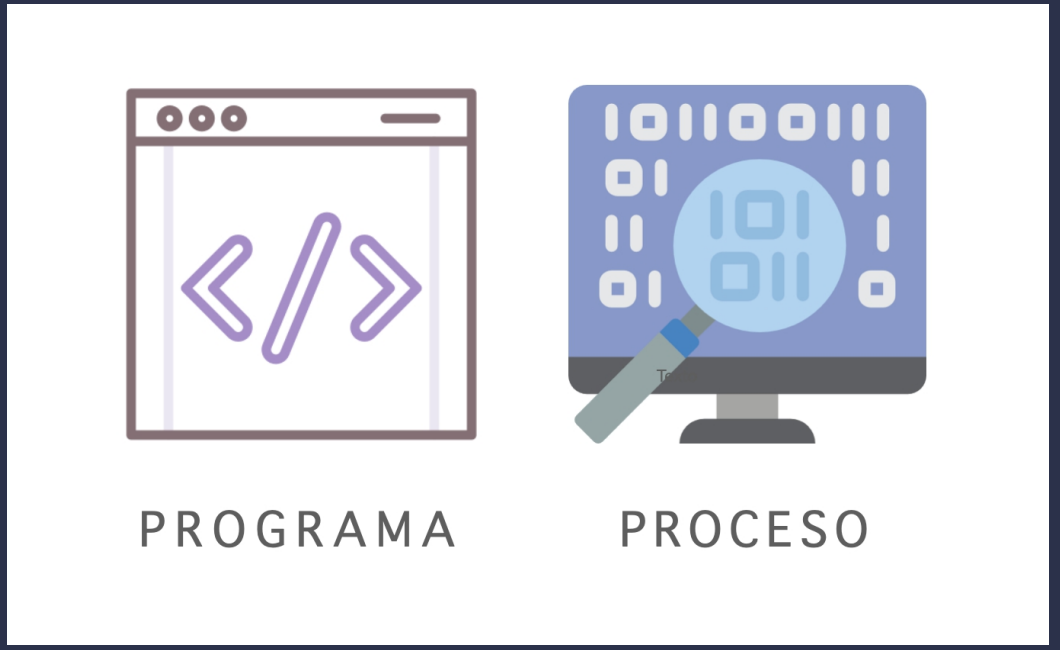


Apliaciones y Procesos

¿Qué es qué?

Apliaciones y Procesos

- Una **aplicación** es un tipo de programa informático, diseñado como herramienta para resolver de manera automática un problema específico del usuario.
- Un **ejecutable** es un fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.
- Un **proceso** es un programa en ejecución. Pero, es más que eso, un proceso en el sistema operativo (SO), es una unidad de trabajo completa; y, el SO gestiona los distintos procesos que se encuentren en ejecución en el equipo. Un proceso existe mientras que se esté ejecutando una aplicación. Es más, la ejecución de una aplicación, puede implicar que se arranquen varios procesos en nuestro equipo; y puede estar formada por varios ejecutables y librerías.



Gestión de Prceosos

Tipos de procesos y cómo tratarlos

Gestion de Procesos. Tipos de procesos

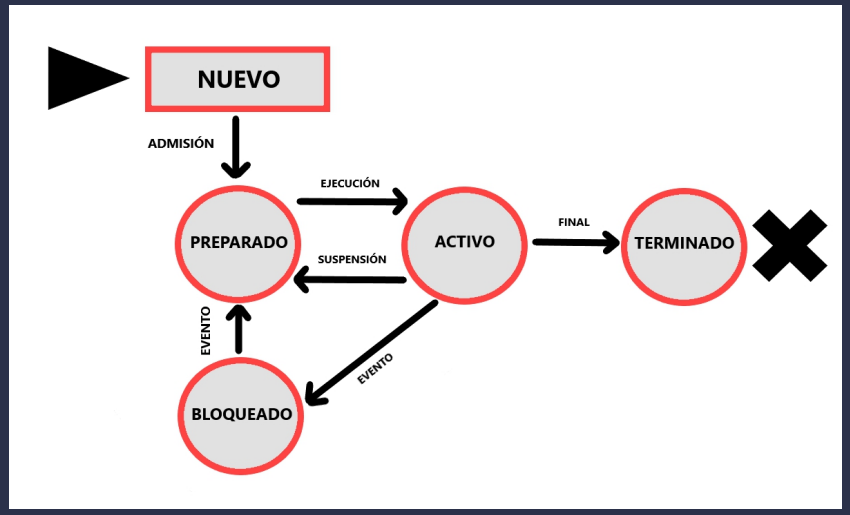
- **Por lotes**. Están formados por una serie de tareas, de las que el usuario sólo está interesado en el resultado final. El usuario, sólo introduce las tareas y los datos iniciales, deja que se realice todo el proceso y luego recoge los resultados. Por ejemplo: enviar a imprimir varios documentos, escanear nuestro equipo en busca de virus,...
- **Interactivos**. Aquellas tareas en las que el proceso interactúa continuamente con el usuario y actúa de acuerdo a las acciones que éste realiza, o a los datos que suministra. Por ejemplo: un procesador de textos; una aplicación formada por formularios que permiten introducir datos en una base de datos; ...
- **Tiempo real**. Tareas en las que es crítico el tiempo de respuesta del sistema. Por ejemplo: el ordenador de a borde de un automóvil, reaccionará ante los eventos del vehículo en un tiempo máximo que consideramos correcto y aceptable. Otro ejemplo, son los equiposque controlan los brazos mecánicos en los procesos industriales de fabricación.

Gestion de Procesos. Estados de un proceso

- 1. Los procesos nuevos, entran en la cola de procesos activos en el sistema.
- 2. Los procesos van avanzando posiciones en la cola de procesos activos, hasta que les toca el turno para que el SO les conceda el uso de la CPU.
- 3. El SO concede el uso de la CPU, a cada proceso durante un tiempo determinado y equitativo, que llamaremos quantum. Un proceso que consume su quantum, es pausado y enviado al final de la cola.
- 4. Si un proceso finaliza, sale del sistema de gestión de procesos. Cuando un proceso, necesita datos de un archivo o una entrada de datos que deba suministrar el usuario; o, tiene que imprimir o grabar datos; cosa que llamamos 'el proceso está en una operación de entrada/salida' (E/S para abreviar). El proceso, queda bloqueado hasta que haya finalizado esa E/S. El proceso es bloqueado, porque, los dispositivos son mucho más lentos que la CPU, por lo que, mientras que uno de ellos está esperando una E/S, otros procesos pueden pasar a la CPU y ejecutar sus instrucciones. Cuando termina la E/S que tenga un proceso bloqueado, el SO, volverá a pasar al proceso a la cola de procesos activos, para que recoja los datos y continúe con su tarea (dentro de sus correspondientes turnos).
- Sólo mencionar (o recordar), que cuando la memoria RAM del equipo está llena, algunos procesos deben pasar a discoalmacenamiento secundario) para dejar espacio en RAM que permita la ejecución de otros procesos.

Gestion de Procesos. Estados de un proceso

- Con todo lo anterior, podemos quedarnos con los siguientes estados en el ciclo de vida de un proceso:
  - 1. Nuevo. Proceso nuevo, creado.
  - 2. Listo. Proceso que está esperando la CPU para ejecutar sus instrucciones.
  - 3. En ejecución. Proceso que actualmente, está en turno de ejecución en la CPU.
  - 4. Bloqueado. Proceso que está a la espera de que finalice una E/S.
  - 5. Suspendido. Proceso que se ha llevado a la memoria virtual para liberar, un poco, la RAM del sistema.
  - 6. Terminado. Proceso que ha finalizado y ya no necesitará más la CPU.



Gestion de Procesos. Planificación de SO

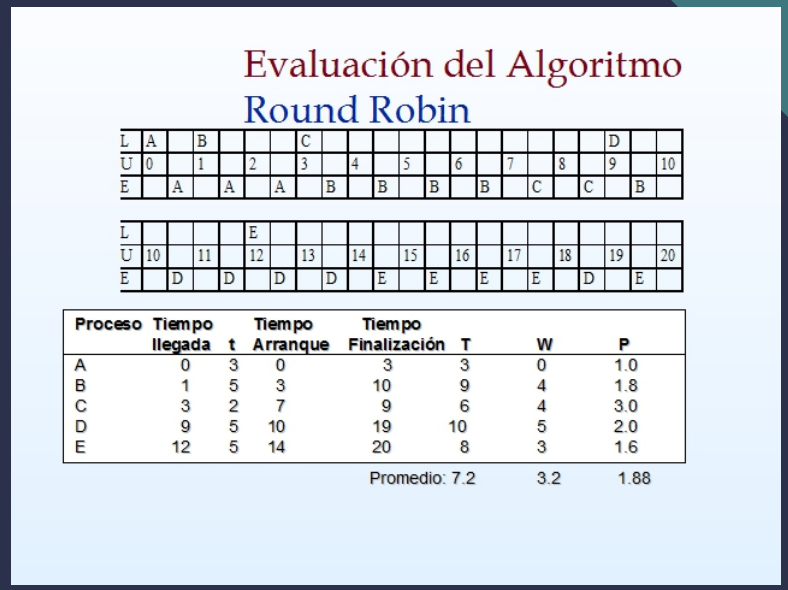
- El cargador es el encargado de crear los procesos. Cuando se inicia un proceso (para cada proceso), el cargador, realiza las siguientes tareas:
  - Carga el proceso en memoria principal. Reserva un espacio en la RAM para el proceso. En ese espacio, copia las instrucciones del fichero ejecutable de la aplicación, las constantes y, deja un espacio para los datos (variables) y la pila (llamadas a funciones). Un proceso, durante su ejecución, no podrá hacer referencia a direcciones que se encuentren fuera de su espacio de memoria; si lo intentara, el SO lo detectará y generará una excepción (produciendo, por ejemplo, los típicos pantallazos azules de Windows).
  - Crea una estructura de información llamada PCB (Bloque de Control de Proceso). La información del PCB, es única para cada proceso y permite controlarlo. Esta información, también la utilizará el planificador. Entre otros datos, el PCB estará formado por:
    - Identificador del proceso o PID. Es un número único para cada proceso, como un DNI de proceso.
    - Estado actual del proceso: en ejecución, listo, bloqueado, suspendido, finalizando.
    - Espacio de direcciones de memoria donde comienza la zona de memoria reservada al proceso y su tamaño.
    - Información para la planificación: prioridad, quamtum, estadísticas, ...
    - Información para el cambio de contexto: valor de los registros de la CPU, entre ellos el contador de programa y el puntero a pila. Esta información es necesaria para poder cambiar de la ejecución de un proceso a otro.
    - Recursos utilizados. Ficheros abiertos, conexiones, …

Gestion de Procesos. Planificación de SO

- Una vez que el proceso ya está cargado en memoria, será el planificador el encargado de tomar las decisiones relacionadas con la ejecución de los procesos. Se encarga de decidir qué proceso se ejecuta y cuánto tiempo se ejecuta. El planificador es otro proceso que, en este caso, es parte del SO. La política en la toma de decisiones del planificador se denominan: algoritmo de planificación. Los más importantes son:
  - Round-Robin. Este algoritmo de planificación favorece la ejecución de procesos interactivos. Es aquél en el que cada proceso puede ejecutar sus instrucciones en la CPU durante un quamtum. Si no le ha dado tiempo a finalizar en ese quamtum, se coloca al final de la cola de procesos listos, y espera a que vuelva su turno de procesamiento. Así, todos los procesos listos en el sistema van ejecutándose poco a poco.
  - Por prioridad. En el caso de Round-Robin, todos los procesos son tratados por igual. Pero existen procesos importantes, que no deberían esperar a recibir su tiempo de procesamiento a que finalicen otros procesos de menor importancia. En este algoritmo, se asignan prioridades a los distintos procesos y la ejecución de estos, se hace de acuerdo a esa prioridad asignada. Por ejemplo: el propio planificador tiene mayor prioridad en ejecución que los procesos de usuario, ¿no crees?
  - Múltiples colas. Es una combinación de los dos anteriores y el implementado en los sistemas operativos actuales. Todos los procesos de una misma prioridad, estarán en la misma cola. Cada cola será gestionada con el algoritmo Round-Robin. Los procesos de colas de inferior prioridad no pueden ejecutarse hasta que no se hayan vaciado las colas de procesos de mayor prioridad.

Gestion de Procesos. Planificación de SO

- En la planificación (scheduling) de procesos se busca conciliar los siguientes objetivos:
  - Equidad. Todos los procesos deben poder ejecutarse.
  - Eficacia. Mantener ocupada la CPU un 100 % del tiempo.
  - Tiempo de respuesta. Minimizar el tiempo de respuesta al usuario.
  - Tiempo de regreso. Minimizar el tiempo que deben esperar los usuarios de procesos por lotes para obtener sus resultados.
  - Rendimiento. Maximizar el número de tareas procesadas por hora.



Proceso	Tiempo llegada	t	Tiempo Arranque	Tiempo Finalización	T	W	P
A	0	3	0	3	3	0	1.0
B	1	5	3	10	9	4	1.8
C	3	2	7	9	6	4	3.0
D	9	5	10	19	10	5	2.0
E	12	5	14	20	8	3	1.6
				Promedio: 7.2		3.2	1.88

Gestion de Procesos. Cambio de Contexto

- **La CPU realiza un cambio de contexto cada vez que cambia la ejecución de un proceso a otro distinto. En un cambio de contexto, hay que guardar el estado actual de la CPU y restaurar el estado de CPU del proceso que va a pasar a ejecutar.**
- Una CPU, además de circuitos encargados de realizar las operaciones con los datos (llamados circuitos operacionales), tiene unas pequeños espacios de memoria (llamados registros), en los que se almacenan temporalmente la información que, en cada instante, necesita la instrucción que esté procesando la CPU. El conjunto de registros de la CPU es su estado. Entre los registros, destacamos el Registro Contador de Programa y el puntero a la pila.
  - El Contador de Programa, en cada instante almacena la dirección de la siguiente instrucción a ejecutar. Recordemos, que cada instrucción a ejecutar, junto con los datos que necesite, es llevada desde la memoria principal a un registro de la CPU para que sea procesada; y, el resultado de la ejecución, dependiendo del caso, se vuelve a llevar a memoria (a la dirección que ocupe la correspondiente variable). Pues el Contador de Programa, apunta a la dirección de la siguiente instrucción que habrá que traer de la memoria, cuando se termine de procesar la instrucción en curso. Este Contador de Programa nos permitirá continuar en cada proceso por la instrucción en dónde lo hubiéramos dejado todo.
  - El Puntero a Pila, en cada instante apunta a la parte superior de la pila del proceso en ejecución. En la pila de cada proceso es donde será almacenado el contexto de la CPU. Y de donde se recuperará cuando ese proceso vuelva a ejecutarse.

Gestion de Procesos. Servicios e Hilos

- Un proceso, estará formado por, al menos, un hilo de ejecución.
- Un proceso es una unidad pesada de ejecución. Si el proceso tiene varios hilos, cada hilo, es una unidad de ejecución ligera.
- El ejemplo más claro de hilo o thread, es un juego. El juego, es la aplicación y, mientras que nosotros controlamos uno de los personajes, los 'malos' también se mueven, interactúan por el escenario y quitan vida. Cada uno de los personajes del juego es controlado por un hilo.
- Un servicio es un proceso que, normalmente, es cargado durante el arranque del sistema operativo. Recibe el nombre de servicio, ya que es un proceso que queda a la espera de que otro le pida que realice una tarea.

Gestion de Procesos. Comandos

- Los comandos que nos interesa conocer para la gestión de procesos son:
  - Windows. Este sistema operativo es conocido por sus interfaces gráficas, el intérprete de comandos conocido como Símbolo del sistema, no ofrece muchos comandos para la gestión de procesos. Tendremos:
    - tasklist. Lista los procesos presentes en el sistema. Mostrará el nombre del ejecutable; su correspondiente Identificador de proceso; y, el porcentaje de uso de memoria; entre otros datos.
    - taskkill. Mata procesos. Con la opción /PID especificaremos el Identificador del proceso que queremos matar.
  - GNU/Linux. En este sistema operativo, todo se puede realizar cualquier tarea en modo texto, además de que los desarrolladores y desarrolladoras respetan en la implementación de las aplicaciones, que sus configuraciones se guarden en archivos de texto plano. Esto es muy útil para las administradoras y administradores de sistemas.
    - ps. Lista los procesos presentes en el sistema. Con la opción "aux" muestra todos los procesos del sistema independientemente del usuario que los haya lanzado.
    - pstree. Muestra un listado de procesos en forma de árbol, mostrando qué procesos han creado otros. Con la opción "AGu" construirá el árbol utilizando líneas guía y mostrará el nombre de usuario propietario del proceso.
    - kill. Manda señales a los procesos. La señal -9, matará al proceso. Se utiliza "kill -9 <PID>".
    - killall. Mata procesos por su nombre. Se utiliza como "killall nombreDeAplicacion".
    - nice. Cambia la prioridad de un proceso. "nice -n 5 comando" ejecutará el comando con una prioridad 5. Por defecto la prioridad es 0. Las prioridades están entre -20 (más alta) y 19 (más baja).

Programación Coincurrente

Comunicando procesos

Programación Concurrente

- La **programación concurrente** proporciona mecanismos de comunicación y sincronización entre procesos que se ejecutan de forma simultanea en un sistema informático. La programación concurrente nos permitirá definir qué instrucciones de nuestros procesos se pueden ejecutar de forma simultánea con las de otros procesos, sin que se produzcan errores; y cuáles deben ser sincronizadas con las de otros procesos para que los resultados de sean correctos.
- Las principales razones por las que se utiliza una estructura concurrente son:
  - Optimizar la utilización de los recursos. Podremos simultanear las operaciones de E/S en los procesos. La CPU estará menos tiempo ociosa. Un equipo informático es como una cadena de producción, obtenemos más productividad realizando las tareas concurrentemente.
  - Proporcionar interactividad a los usuarios (y animación gráfica). Todos nos hemos desesperado esperando que nuestro equipo finalizara una tarea. Esto se agravaría sino existiera el multiprocesamiento, sólo podríamos ejecutar procesos por lotes.
  - Mejorar la disponibilidad. Servidor que no realice tareas de forma concurrente, no podrá atender peticiones de clientes simultáneamente.
  - Conseguir un diseño conceptualmente más comprensible y mantenible. El diseño concurrente de un programa nos llevará a una mayor modularidad y claridad. Se diseña una solución para cada tarea que tenga que realizar la aplicación (no todo mezclado en el mismo algoritmo). Cada proceso se activará cuando sea necesario realizar cada tarea.
  - Aumentar la protección. Tener cada tarea aislada en un proceso permitirá depurar la seguridad de cada proceso y, poder finalizarlo en caso de mal funcionamiento sin que suponga la caída del sistema.

Programación Concurrente

- Donde hacerla:
  - Microprocesadores con múltiples núcleos que comparten la memoria principal del sistema.
  - Entornos multiprocesador con memoria compartida. Todos los procesadores utilizan un mismo espacio de direcciones a memoria, sin tener conciencia de dónde están instalados físicamente los módulos de memoria.
  - Entornos distribuidos. Conjunto de equipos heterogéneos o no, conectados por red y/o Internet.
- Los beneficios que obtendremos al adoptar un modelo de programa concurrente son:
  - Estructurar un programa como conjunto de procesos concurrentes que interactúan, aporta gran claridad sobre lo que cada proceso debe hacer y cuando debe hacerlo.
  - Puede conducir a una reducción del tiempo de ejecución. Cuando se trata de un entorno monoprocesador, permite solapar los tiempos de E/S o de acceso al disco de unos procesos con los tiempos de ejecución de CPU de otros procesos. Cuando el entorno es multiprocesador, la ejecución de los procesos es realmente simultánea en el tiempo (paralela), y esto reduce el tiempo de ejecución del programa.
  - Permite una mayor flexibilidad de planificación. Procesos de alta prioridad pueden ser ejecutados antes de otros procesos menos urgentes.
  - La concepción concurrente del software permite un mejor modelado previo del comportamiento del programa, y en consecuencia un análisis más fiable de las diferentes opciones que requiera su diseño.

Programación Concurrente. Condiciones de Competencia

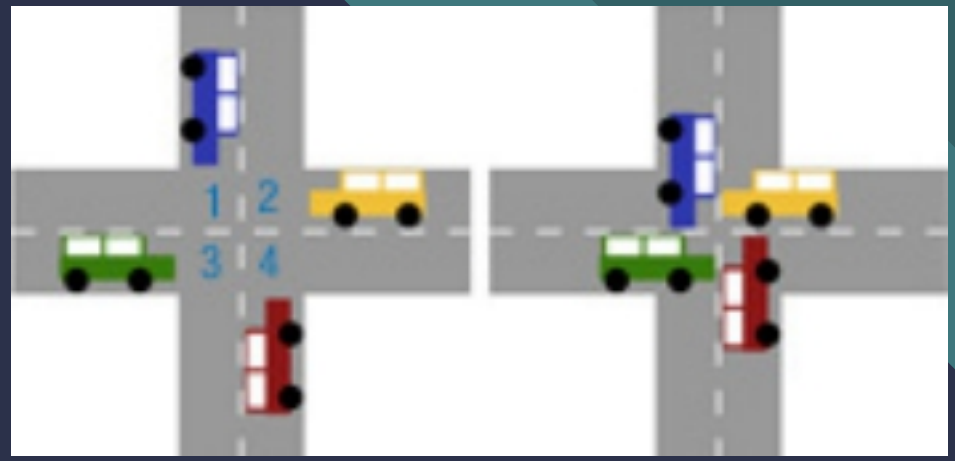
- Un proceso entra en condición de competencia con otro, cuando ambos necesitan el mismo recurso, ya sea forma exclusiva o no; por lo que será necesario utilizar mecanismos de sincronización y comunicación entre ellos.
- Cuando un proceso **necesita un recurso de forma exclusiva**, es porque mientras que lo esté utilizando él, ningún otro puede utilizarlo. Se llama **región de exclusión mutua o región crítica** al conjunto de instrucciones en las que el proceso utiliza un recurso y que se deben ejecutar de forma exclusiva con respecto a otros procesos competidores por ese mismo recurso.
- Cuando más de un proceso necesitan el mismo recurso, antes de utilizarlo tienen que pedir su uso, una vez que lo obtienen, el resto de procesos quedarán bloqueados al pedir ese mismo recurso. Se dice que un proceso hace un **lock (bloqueo)** sobre un recurso cuando ha obtenido su uso en exclusión mutua.
- Por ejemplo dos procesos, compiten por dos recursos distintos, y ambos necesitan ambos recursos para continuar. Se puede dar la situación en la que cada uno de los procesos bloquee uno de los recursos, lo que hará que el otro proceso no pueda obtener el recurso que le falta; quedando bloqueados un proceso por el otro sin poder finalizar. **Deadlock o interbloqueo**, se produce cuando los procesos no pueden obtener, nunca, los recursos necesarios para continuar su tarea. El interbloqueo es una situación muy peligrosa, ya que puede llevar al sistema a su caída o cuelgue.

Programación Concurrente. Condiciones de Competencia

- ¿Crees que no es usual que pueda darse una situación de interbloqueo? Veamos un ejemplo sencillo: un cruce de caminos y cuatro coches.

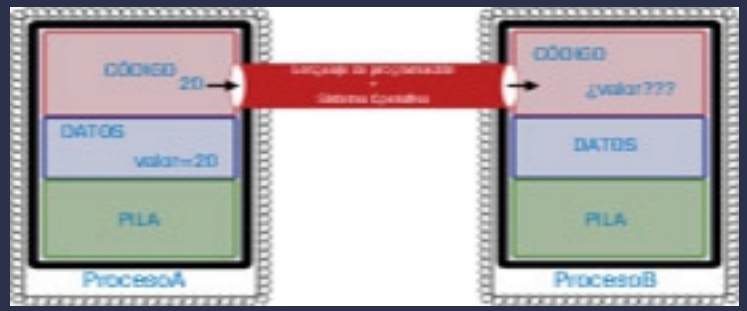
- El coche azul necesita las regiones 1 y 3 para continuar, el amarillo: 2 y 1, el rojo: 4 y 2, y el verde: 3 y 4.

- Obviamente, no siempre quedarán bloqueados, pero se puede dar la situación en la que ninguno ceda. Entonces, quedarán interbloqueados.



Programación Concurrente. Comunicación

- Comunicación entre procesos: un proceso da o deja información; recibe o recoge información.
- Los lenguajes de programación y los sistemas operativos, nos proporcionan primitivas de sincronización que facilitan la interacción entre procesos de forma sencilla y eficiente.
- Una primitiva, hace referencia a una operación de la cual conocemos sus restricciones y efectos, pero no su implementación exacta. Veremos que usar esas primitivas se traduce en utilizar objetos y sus métodos, teniendo muy en cuenta sus repercusiones reales en el comportamiento de nuestros procesos.
- Clasificaremos las interacciones entre los procesos y el resto del sistema (recursos y otros procesos), como estas tres:
  - Sincronización: Un proceso puede conocer el punto de ejecución en el que se encuentra otro en ese determinado instante.
  - Exclusión mutua: Mientras que un proceso accede a un recurso, ningún otro proceso accede al mismo recurso o variable compartida.
  - Sincronización condicional: Sólo se accede a un recurso cuando se encuentra en un determinado estado interno.



Programación Concurrente. Comunicación

- Si pensamos en la forma en la que un proceso puede comunicarse con otro. Se nos ocurrirán estas dos:
  - Intercambio de mensajes. Tendremos las primitivas enviar (send) y recibir (receive o wait) información.
  - Recursos (o memoria) compartidos. Las primitivas serán escribir (write) y leer (read) datos en o de un recurso.
- En el caso de comunicar procesos dentro de una misma máquina, el intercambio de mensajes, se puede realizar de dos formas:
  - Utilizar un buffer de memoria.
  - Utilizar un socket.
- La diferencia entre ambos, está en que un socket se utiliza para intercambiar información entre procesos en distintas máquinas a través de la red; y un buffer de memoria, crea un canal de comunicación entre dos procesos utilizando la memoria principal del sistema. Actualmente, es más común el uso de sockets que buffers para comunicar procesos. Trataremos en profundidad los sockets en posteriores unidades. Pero veremos un par de ejemplos muy sencillos de ambos.
- En java, utilizaremos sockets y buffers como si utilizáramos cualquier otro stream o flujo de datos. Utilizaremos los métodos read-write en lugar de send-receive.
- Con respecto a las lecturas y escrituras, debemos recordar, que serán bloqueantes. Es decir, un proceso quedará bloqueado hasta que los datos estén listos para poder ser leídos. Una escritura, bloqueará al proceso que intenta escribir, hasta que el recurso no esté preparado para poder escribir. Aunque, esto está relacionado con el acceso a recursos compartidos, cosa que estudiaremos en profundidad, en el apartado **Regiones críticas**.

Programación Concurrente. Comunicación

- En cualquier comunicación, vamos a tener los siguientes elementos:
  - Mensaje. Información que es el objeto de la comunicación.
  - Emisor. Entidad que emite, genera o es origen del mensaje.
  - Receptor. Entidad que recibe, recoge o es destinataria del mensaje.
  - Canal. Medio por el que viaja o es enviado y recibido el mensaje.
- Podemos clasificar el canal de comunicación según su capacidad, y los sentidos en los que puede viajar la información, como:
  - Símplex. La comunicación se produce en un sólo sentido. El emisor es origen del mensaje y el receptor escucha el mensaje al final del canal. Ejemplo: reproducción de una película en una sala de cine.
  - Dúplex (Full Duplex). Pueden viajar mensajes en ambos sentidos simultáneamente entre emisor y receptor. El emisor es también receptor y el receptor es también emisor. Ejemplo: telefonía.
  - Semidúplex (Half Duplex). El mensaje puede viajar en ambos sentidos, pero no al mismo tiempo. Ejemplo: comunicación con walkie-talkies.
- Otra clasificación dependiendo de la sincronía que mantengan el emisor y el receptor durante la comunicación, será:
  - Síncrona. El emisor queda bloqueado hasta que el receptor recibe el mensaje. Ambos se sincronizan en el momento de la recepción del mensaje.
  - Asíncrona. El emisor continúa con su ejecución inmediatamente después de emitir el mensaje, sin quedar bloqueado.
  - Invocación remota. El proceso emisor queda suspendido hasta que recibe la confirmación de que el receptor recibido correctamente el mensaje, después emisor y receptor ejecutarán síncronamente un segmento de código común.
- Dependiendo del comportamiento que tengan los interlocutores que intervienen en la comunicación, tendremos comunicación:
  - Simétrica. Todos los procesos pueden enviar y recibir información.
  - Asimétrica. Sólo un proceso actúa de emisor, el resto sólo escucharán el o los mensajes.

Programación Concurrente. Sincronización

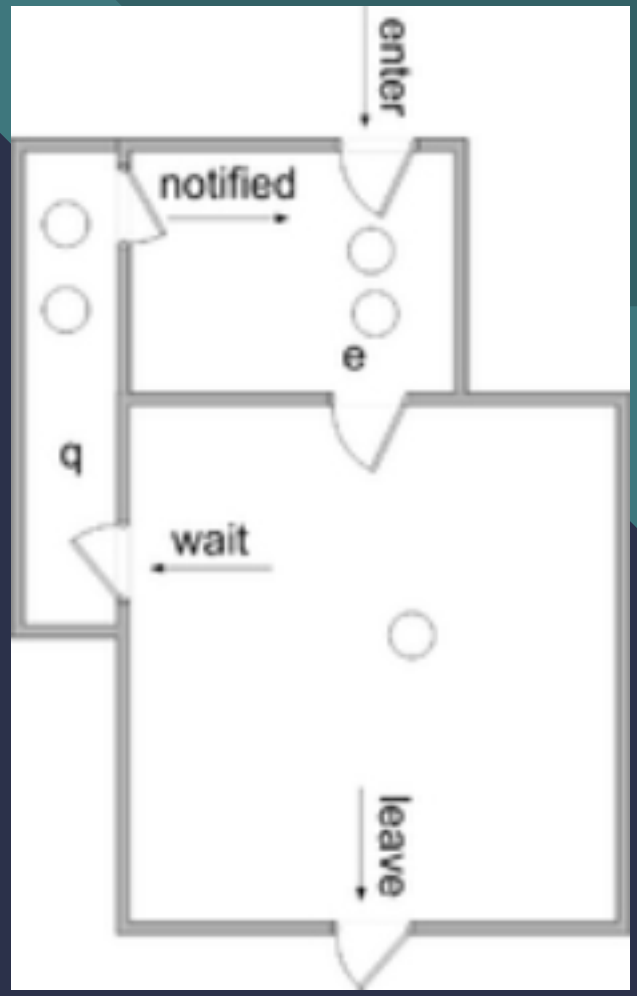
- En programación concurrente, siempre que accedamos a algún recurso compartido (eso incluye a los ficheros), deberemos tener en cuenta las condiciones en las que nuestro proceso debe hacer uso de ese recurso: ¿será de forma exclusiva o no? Lo que ya definimos anteriormente como condiciones de competencia. ¿Cómo nos sincronizamos para acceder a la secciónb crítica?
- La definición común, y que habíamos visto anteriormente, de una región o **sección crítica**, es, el conjunto de instrucciones en las que un proceso accede a un recurso compartido. Para que la definición sea correcta, añadiremos que, las instrucciones que forman esa región crítica, se ejecutarán de forma indivisible o atómica y de forma exclusiva con respecto a otros procesos que accedan al mismo recurso compartido al que se está accediendo.
- Al identificar y definir nuestras regiones críticas en el código, tendremos en cuenta:
  - Se protegerán con secciones críticas sólo aquellas instrucciones que acceden a un recurso compartido.
  - Las instrucciones que forman una sección crítica, serán las mínimas. Incluirán sólo las instrucciones imprescindibles que deban ser ejecutadas de forma atómica.
  - Se pueden definir tantas secciones críticas como sean necesarias.
  - Un único proceso entra en su sección crítica. El resto de procesos esperan a que éste salga de su sección crítica. El resto de procesos esperan, porque encontrarán el recurso bloqueado. El proceso que está en su sección crítica, es el que ha bloqueado el recurso.
  - Al final de cada sección crítica, el recurso debe ser liberado para que puedan utilizarlo otros procesos.

Programación Concurrente. Sincronización. Semáforos

- Un semáforo, es un componente de bajo nivel de abstracción que permite arbitrar los accesos a un recurso compartido en un entorno de programación concurrente.
- Al utilizar un semáforo, lo veremos como un tipo dato, que podremos instanciar. Ese objeto semáforo podrá tomar un determinado conjunto de valores y se podrá realizar con él un conjunto determinado de operaciones. Un semáforo, tendrá también asociada una lista de procesos que suspendidos que se encuentran a la espera de entrar en el mismo.
- Dependiendo del conjunto de datos que pueda tomar un semáforo, tendremos:
  - Semáforos binarios. Aquellos que pueden tomar sólo valores 0 ó 1. Como nuestras luces verde y roja.
  - Semáforos generales. Pueden tomar cualquier valor Natural (entero no negativo).
- Para utilizar semáforos, seguiremos los siguientes pasos:
- Un proceso padre creará e inicializará tanto semáforo.
- El proceso padre creará el resto de procesos hijo pasándoles el semáforo que ha creado. Esos procesos hijos acceden al mismo recurso compartido.
- Cada proceso hijo, hará uso de las operaciones seguras wait y signal respetando este esquema:
  - objSemaforo.wait(); Para consultar si puede acceder a la sección crítica.
  - Sección crítica; Instrucciones que acceden al recurso protegido por el semáforo objSemaforo.
  - objSemaforo.signal(); Indicar que abandona su sección y otro proceso podrá entrar.
- El proceso padre habrá creado tantos semáforos como tipos secciones críticas distintas se puedan distinguir en el funcionamiento de los procesos hijos (puede ocurrir, uno por cada recurso compartido).

Programación Concurrente. Sincronización. Monitores

- Un monitor, es un componente de alto nivel de abstracción destinado a gestionar recursos que van a ser accedidos de forma concurrente.
- Un grupo de personas esperando, probablemente, a un medio de transporte público.
- Los monitores encierran en su interior los recursos o variables compartidas como componentes privadas y garantizan el acceso a ellas en exclusión mutua.
- La declaración de un monitor incluye:
  - Declaración de las constantes, variables, procedimientos y funciones que son privados del monitor (solo el monitor tiene visibilidad sobre ellos).
  - Declaración de los procedimientos y funciones que el monitor expone (públicos) y que constituyen la interfaz a través de las que los procesos acceden al monitor.
  - Cuerpo del monitor, constituido por un bloque de código que se ejecuta al ser instanciado o inicializado el monitor. Su finalidad es inicializar las variables y estructuras internas del monitor.
  - El monitor garantiza el acceso al código interno en régimen de exclusión mutua.
  - Tiene asociada una lista en la que se incluyen los procesos que al tratar de acceder al monitor son suspendidos.



Programación Concurrente. Sincronización. Mensajes

- El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, etc. Su principal característica es que no precisa de memoria compartida.
- Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje.
- Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono:
  - En el paso de **mensajes asíncrono**, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear buzones o colas, en los que se almacenan los mensajes a espera de que un proceso los reciba. Generalmente empleando este sistema, el proceso que envía mensajes sólo se bloquea o para, cuando finaliza su ejecución, o si el buzón está lleno. Para conseguir esto, estableceremos una serie de reglas de comunicación (o protocolo) entre emisor y receptor, de forma que el receptor pueda indicar al emisor qué capacidad restante queda en su cola de mensajes y si está lleno o no.
  - En el paso de **mensajes síncrono**, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. Por esto se suele llamar a esta técnica encuentro, o rendezvous. Dentro del paso de mensajes síncrono se engloba a la llamada a procedimiento remoto (RPC), muy popular en las arquitecturas cliente/servidor.

Programación Concurrente. Requisitos

- Todo programa concurrente debe satisfacer dos tipos de propiedades:
  - **Propiedades de seguridad** ("safety"): estas propiedades son relativas a que en cada instante de la ejecución no debe haberse producido algo que haga entrar al programa en un estado erróneo:
    - Dos procesos no deben entrar simultáneamente en una sección crítica.
    - Se respetan las condiciones de sincronismo, como: el consumidor no debe consumir el dato antes de que el productor los haya producido; y, el productor no debe producir un dato mientras que el buffer de comunicación esté lleno.
- **Propiedades de vivacidad ("liveness"):** cada sentencia que se ejecute conduce en algún modo a un avance constructivo para alcanzar el objetivo funcional del programa. Son, en general, muy dependientes de la política de planificación que se utilice. Ejemplos de propiedades de vivacidad son:
  - No deben producirse bloqueos activos (livelock). Conjuntos de procesos que ejecutan de forma continuada sentencias que no conducen a un progreso constructivo.
  - Aplazamiento indefinido (starvation): consiste en el estado al que puede llegar un programa que aunque potencialmente puede avanzar de forma constructiva. Esto puede suceder, como consecuencia de que no se le asigna tiempo de procesador en la política de planificación; o, porque en las condiciones de sincronización hemos establecido criterios de prioridad que perjudican siempre al mismo proceso.
  - Interbloqueo (deadlock): se produce cuando los procesos no pueden obtener, nunca, los recursos necesarios para finalizar su tarea.

Programación Concurrente. Requisitos

- Es evidentemente, que también nos preocuparemos por diseñar nuestras aplicaciones para que sean eficientes:
  - No utilizarán más recursos de los necesarios.
  - Buscaremos la rigurosidad en su implementación: toda la funcionalidad esperada de forma correcta y concreta.
- Y, en cuanto a la reusabilidad, debemos tenerlo ya, muy bien aprendido:
  - Implementar el código de forma modular: definiendo clases, métodos, funciones, ...
  - Documentar correctamente el código y el proyecto.
  - Para conseguir todos lo anterior contaremos con los patrones de diseño; y pondremos especial cuidado en documentar y depurar convenientemente.

Programación Paralela y Distribuida

Un poco más allá

Programación Paralela y Distribuida

- Dos procesos se ejecutan de forma paralela, si las instrucciones de ambos se están ejecutando realmente de forma simultánea. Esto sucede en la actualidad en sistemas que poseen más de un núcleo de procesamiento.
- La programación paralela y distribuida consideran los aspectos conceptuales y físicos de la computación paralela; siempre con el objetivo de mejorar las prestaciones aprovechado la ejecución simultánea de tareas.
- Tanto en la programación paralela como distribuida, existe ejecución simultánea de tareas que resuelven un problema común. La diferencia entre ambas es:
  - La **programación paralela** se centra en microprocesadores multinúcleo (en nuestros PC y servidores); o ,sobre los llamados supercomputadores, fabricados con arquitecturas específicas, compuestos por gran cantidad de equipos idénticos interconectados entre sí, y que cuentan son sistemas operativos propios.
  - La **programación para distribuida**, en sistemas formados por un conjunto de ordenadores heterogéneos interconectados entre sí, por redes de comunicaciones de propósito general: redes de área local, metropolitana; incluso, a través de Internet. Su gestión se realiza utilizando componentes, protocolos estándar y sistemas operativos de red.
- En la computación paralela y distribuida:
  - Varios cables de red con su correspondiente conector RJ45, conectado a un switch o dispositivo de interconexión de redes.
  - Cada procesador tiene asignada la tarea de resolver una porción del problema.
  - En programación paralela, los procesos pueden intercambiar datos, a través de direcciones de memoria compartidas o mediante una red de interconexión propia.
  - En programación distribuida, el intercambio de datos y la sincronización se realizará mediante intercambio de mensajes.
  - El sistema se presenta como una unidad ocultando la realidad de las partes que lo forman.ones de diseño; y pondremos especial cuidado en documentar y depurar convenientemente.

"Si automatizas un procedimiento desastroso, obtienes un procedimiento desastroso automatizado"

- Rod Michae

Recursos

- Twitter: https://twitter.com/joseluisgonsan
- GitHub: https://github.com/joseluisgs
- Web: https://joseluisgs.github.io
- Discord: https://discord.gg/kyhx7kGN
- Aula Virtual: https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=248

Gracias

José Luis González Sánchez