UF1288: Desarrollo de componentes software para servicios de comunicaciones

3.7.2023

Enrique Krause Buedo

1 Programación concurrente

3.7.2023

Enrique Krause Buedo

La **programación concurrente** trata sobre el paralelismo potencial entre tareas y cómo resolver los problemas de comunicación y sincronización entre procesos.

La principal razón para la investigación de la programación concurrente es que ofrece una manera diferente de afrontar la solución de un problema. La segunda razón es la de aprovechar el paralelismo del hardware existente para lograr una mayor rapidez.

La concurrencia tiene que ser funcionalmente independiente de la máquina o entorno(E.) en la que se utilice:

E.**monoprocesador** con multiprogramación (concurrencia virtual o simulada)

E.**multicomputador** con memoria compartida (Plena; varios procesadores comunicados por un bus, memoria...)

E.**distribuido** computadores distribuidos geográficamente conforman los nudos de una red. No comparten memoria, y la comunicación y sincronización es a través de mensajería.

En la **programación concurrente** el elemento principal es el **proceso** (división de programa en módulos) y dentro de este; **los hilos** o *hebras*

Existe una relación muy fuerte entre hilos, pues comparten memoria y recursos, y es fundamental la sincronización para que no se produzcan bloqueos ni esperas innecesarias.

Gestión de procesos

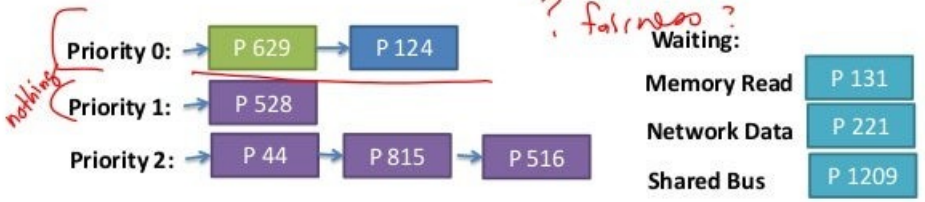
Los procesos tienen su propio estado. Van acompañados de recursos como archivos, memoria, etc. Es necesaria una planificación de sus estados (listo en ejecución, bloqueado…)

¿quién es el encargado de decidir qué procesos y en qué momento se van a ejecutar? Esa función recae sobre el **RTSS** (Sistema de Soporte de Tiempo Real), que es un componente que forma parte del OS. La política que se sigue para elegir el proceso a ejecutar se denomina planificación de procesos (*scheduling*). Utiliza información única de cada proceso, que se almacena en lo que se llama bloque de control de proceso (**PCB**).

Pre-emptive Priority Scheduling

Always run the **highest priority process** that is ready to run

Round-robin schedule among equally high, ready to run, highest-priority processes

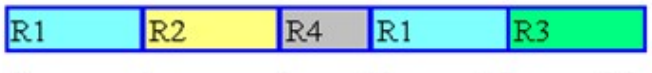


17

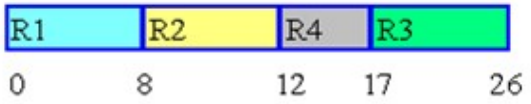
RAFAGA	TIEMPO LLEGADA	REQUERIMIENTOS DE CPU(ms)
R1	0	8
R2	1	4
R3	2	9
R4	3	5

Ocupación de la CPU

SRTF



SJF

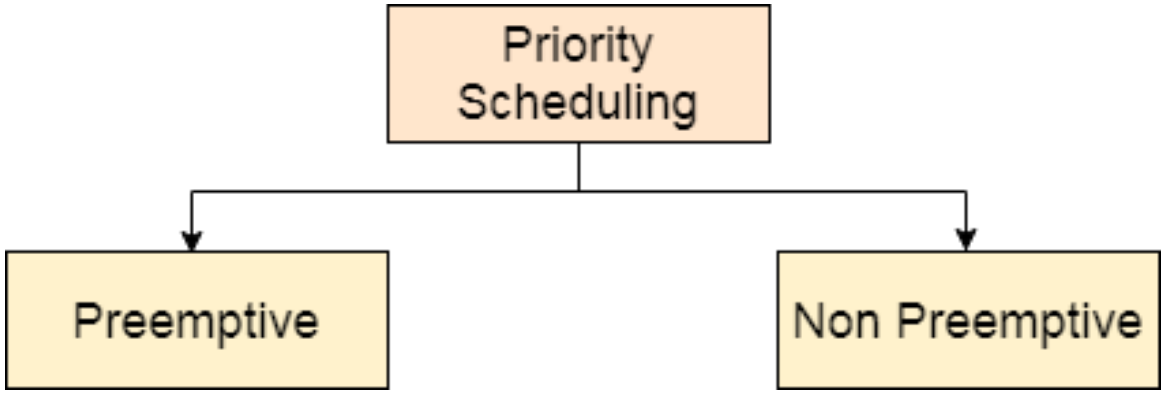


TIEMPO MEDIO DE FINALIZACIÓN

SRTF = (17 + 4 + 24 + 7) / 4 = 13 ms

SJF = (8 + 11 + 24 + 14) / 4 = 14'5 ms

Figura **6.5** Ejemplo de SRTF Y SJF



El **RTSS** utiliza información única de cada proceso en el (**PCB**):
Identificador, Status, Entorno, Enlaces
Existe un principio fundamental de la programación concurrente: **“El comportamiento funcional de un programa concurrente no debe depender de la política de planificación de los procesos que impone el RTSS”**.
El resultado del programa, tiene que ser el mismo con una planificación u otra
Las políticas del *scheduling* del RTSS pueden ser **desalojantes** (retira otro proceso de la CPU, si tienen prioridad) o **no desalojantes**.

Otro aspecto es el criterio de orden: **FIFO** (primero en entrar, primero en salir) **LIFO** (lo contrario) **Round Robin** (Asigna un tiempo a cada uno: quantum) **Prioridad Estática** (Cuando es claro el orden, se da acceso al de más prioridad) **Prioridad Dinámica** (cambia la prioridad según tiempo de espera)

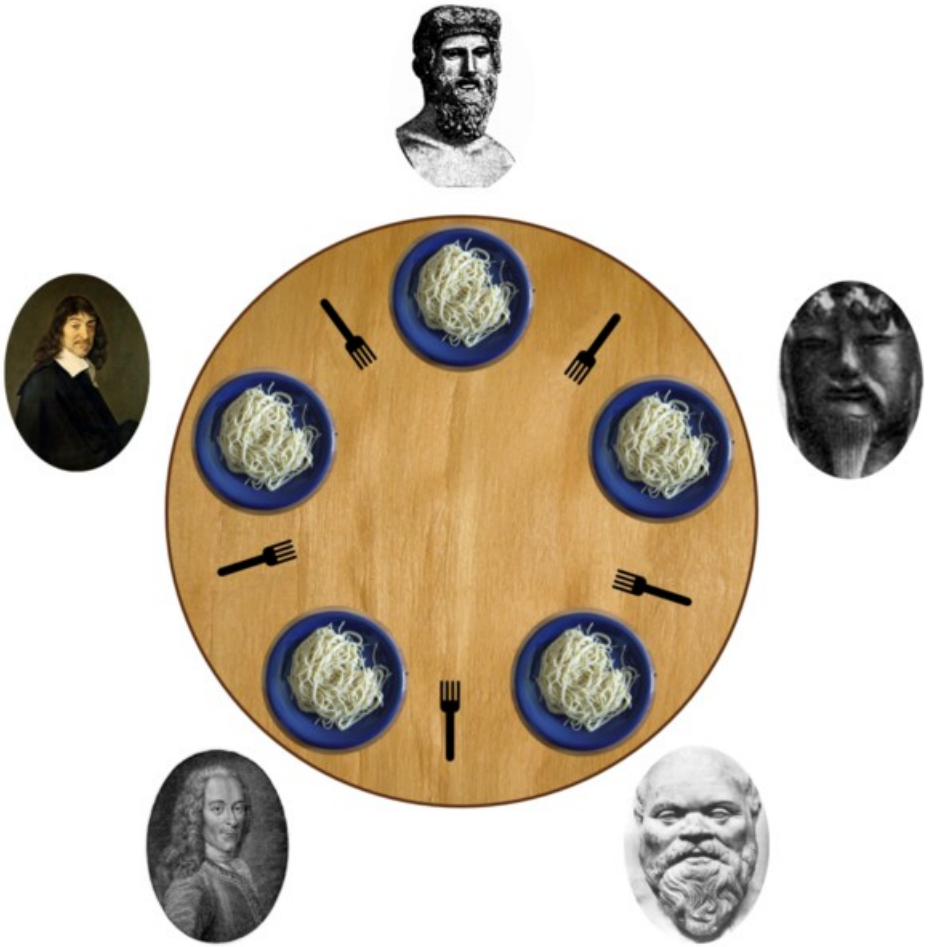
Hilos y sincronización

Todos los hilos que conforman un proceso comparten un entorno igual de ejecución pero, cada hilo sí tiene un juego de registros de CPU, pila y variables locale que permitirán que cuando vuelva a tomar el control siga en su línea.

Como se comparten recursos como la pila, un hilo podría estropear la pila del otro. Deben existir mecanismos que eviten este interbloqueo.

Debe existir una sincronización.

El concepto de sincronización se entiende mejor con el ejemplo propuesto por Edsger Dijkstra en 1965, de la cena de filósofos



Programación de eventos asíncronos

El elemento principal de la programación concurrente es el proceso y la comunicación entre ellos (interprocess communication o IPC). Los procesos siguen un protocolo (conjunto de reglas a observar) de comunicación (rol de emisor y de receptor). En la comunicación entre procesos se tienen las siguientes operaciones: Enviar, recibir, conectar (solicitar/aceptar conexión) y desconectar. Cada vez que se invoca a una de estas operaciones es un **evento**: Los modos de sincronizar eventos es por medio de peticiones:

Peticiones bloqueantes (síncronas)

Peticiones no bloqueantes (asíncronas)

Programación de eventos asíncronos

Eventos

La forma más sencilla para que haya ***sincronización*** de eventos, y no se ejecute un proceso hasta que no haya acabado el anterior, es por medio de **peticiones bloqueantes (síncronas),** que es la supresión de la ejecución del proceso hasta que la operación que se solicitó finalice. Sin embargo, este tipo de comunicaciones en algunas situaciones no son las más eficaces, pues si se produce un error en un proceso provocará que se detenga, dando lugar a lo que se conoce como un bloqueo.

Como alternativa a este tipo de comunicaciones están las **peticiones no bloqueantes (asíncronas)**. En este caso no causa bloqueo, porque el proceso gue invoca la operación sigue con su tarea. Posteriormente, se le informará al proceso si lo que solicitó se ha completado.

Programación de eventos asíncronos

Señales

Las señales permiten a los procesos comunicarse para conseguir la sincronización asíncrona. Cuando se recibe dicha señal, es el proceso quien decide el tratamiento que le va a dar: **ignorar** la señal, invocar a la **rutina de tratamiento** (kernel) o invocar a **una rutina** que se encarga de tratar la señal, y que ha sido **creada** por el programador. En base al lenguaje C, las acciones asociadas a señales :

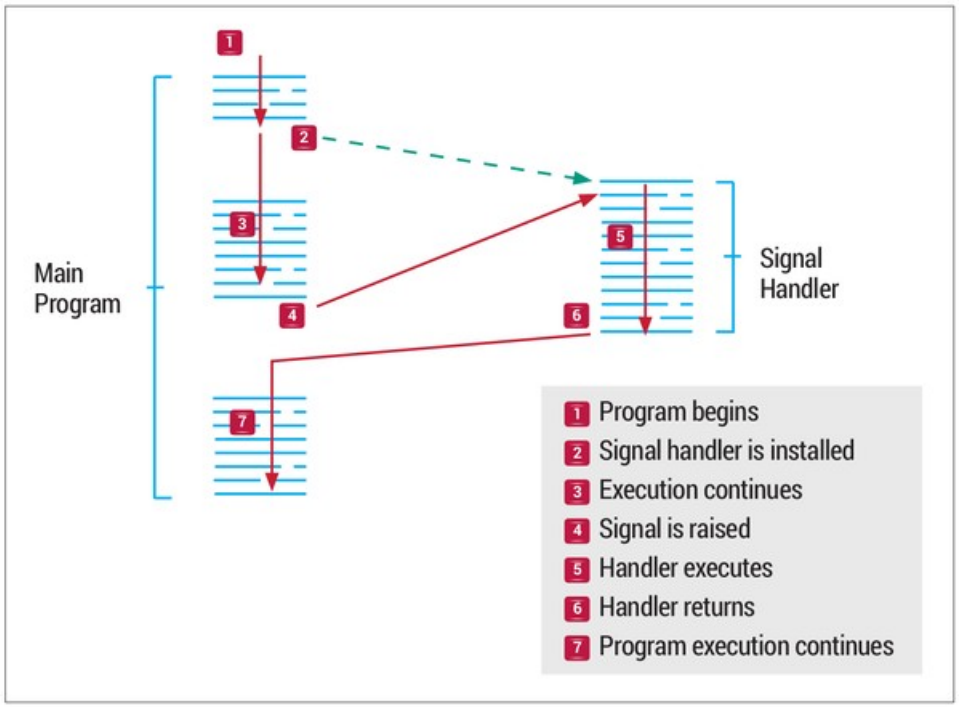
Enviar una señal: kill (pid, sig) (pid: process ID)

Recibir una señal: signal (sig, action)

Esperar una señal: pause ()

Temporizadores

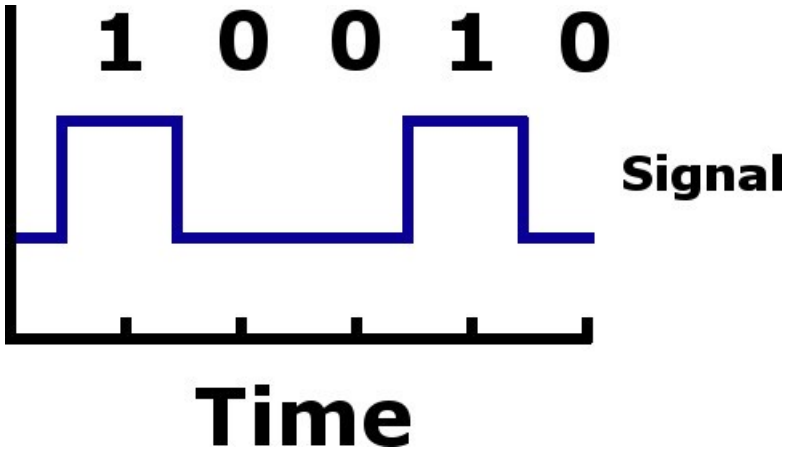
Los temporizadores permiten crear periodos temporales que una vez finalizados pueden generar señales que se envíen a otros procesos o al OS. Permiten sincronizar procesos, generar tiempos de espera. Recordemos que la comunicación entre procesos es asíncrona

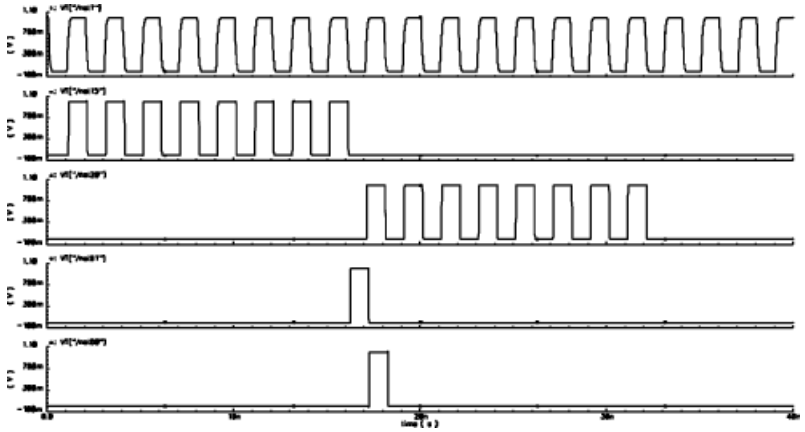


Open *raise.c Save

```c
#include<stdio.h>
#include <unistd.h>
#include<signal.h>
void sig_handler(int signum){
  printf("Inside handler function\n");
}

int main(){
  pid_t pid;
  signal(SIGUSR1,sig_handler); // Register signal handler
  printf("Inside main function\n");
  pid=getpid();      //Process ID of itself
  kill(pid,SIGUSR1);        // Send SIGUSR1 to itself
  printf("Inside main function\n");
  return 0;
}
```





Mecanismos de comunicación entre procesos

En la comunicación entre procesos dos son los problemas fundamentales que aparecen: **la exclusión mutua** persigue es que un solo proceso pueda excluir temporalmente a todos los demás para usar un recurso compartido de forma que garantice la integridad del sistema. A la parte del programa donde se usa el recurso compartido se le denomina sección crítica. **La condición de sincronización** entre procesos consiste en que cuando entre procesos se usa un recurso compartido, este debe estar en un determinado estado para que el proceso pueda hacer uso de él.

Tuberías (pipes)

La comunicación por medio de tuberías se basa en la interacción productor / consumidor. Cuando ya se ha leído un determinado elemento es eliminado de la tubería. De esta manera el planificador de corto plazo va a dar el uso de la CPU a cada proceso a medida que pueda ejecutarse, minimizando los tiempos muertos.
Sin nombre (más temporales) y con nombre.
El proceso de escritura se realiza en exclusión mutua. Si dos procesos lo intentan al mismo tiempo, uno se bloquea.

Mecanismos de comunicación entre procesos

Semáforos

Los semáforos son componentes desarrollados a bajo nivel y sirven para asegurar el acceso correcto a un recurso compartido, es decir, soluciona el problema de la exclusión mutua. Valores enteros o cero (cerrado, está en sección crítica).
Del cap. Pasado, operaciones : wait() y signal()
son estructuras simples, eficientes, resuelven la concurrencia pero son de muy bajo nivel y fácilmente pueden generar bloqueos con una mala programación. Además la depuración de los errores en su gestión es muy difícil.

Compartición de memoria

Este mecanismo de comunicación entre procesos se basa en compartir variables entre las tareas concurrentes. Una variable compartida únicamente puede ser modificada dentro de las regiones críticas.
Es el compilador el encargado de incluir los mecanismos necesarios para garantizar que la variable compartida sea modificada por un único proceso al mismo tiempo.

Mecanismos de comunicación entre procesos

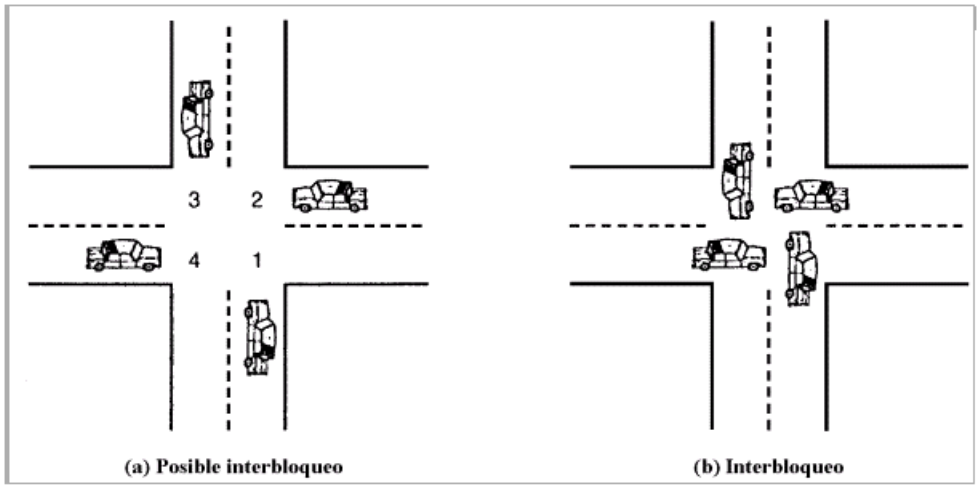
Compartición de memoria

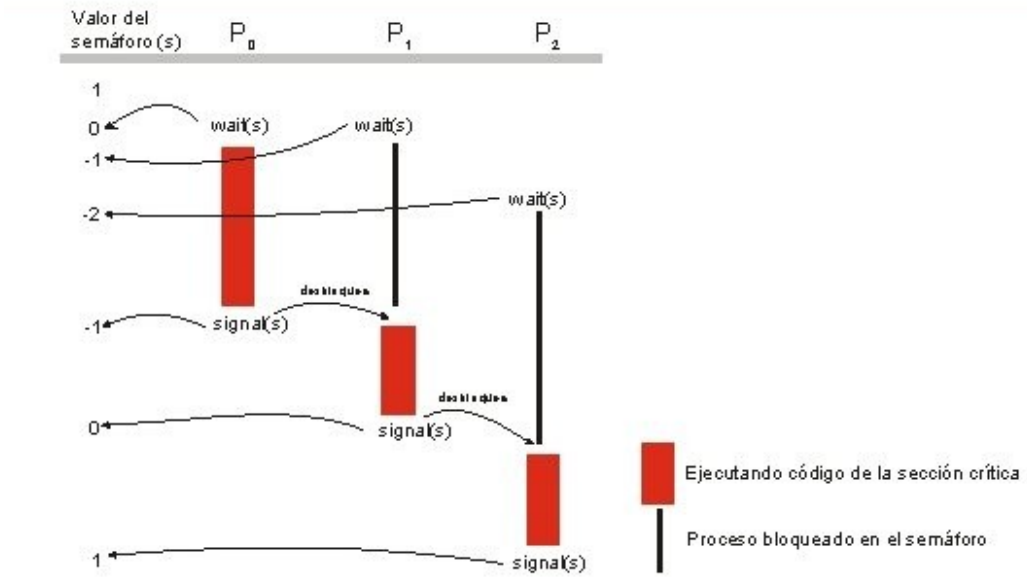
La variable se declara como **shared**. Las regiones críticas tal como se han definido son útiles y simples, pero no proporcionan solución, sobre todo en mecanismos de sincronización. Para solucionar esto se extiende el concepto a **regiones críticas condicionales (CCR).** Al bloque de programa se le asigna una condición para que pueda ser ejecutado. De tal forma que se ejecutará esa región crítica solo si la condición se cumple.

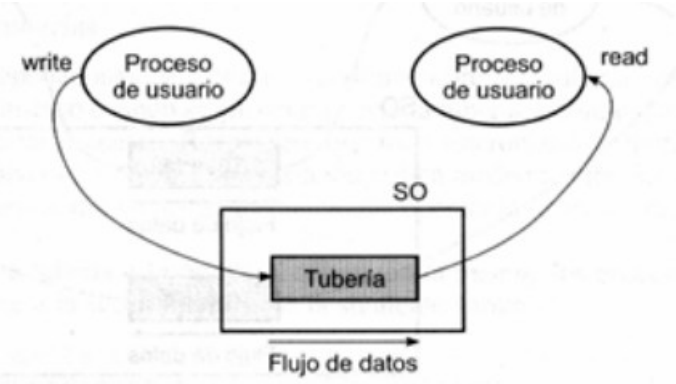
Es un nivel de abstracción superior al de los semáforos, pero tiene inconvenientes: 1 Sentencias dispersas por todo el código del programa. 2 La integridad de la variable compartida puede ser fácilmente dañada. 3 Estructura más compleja

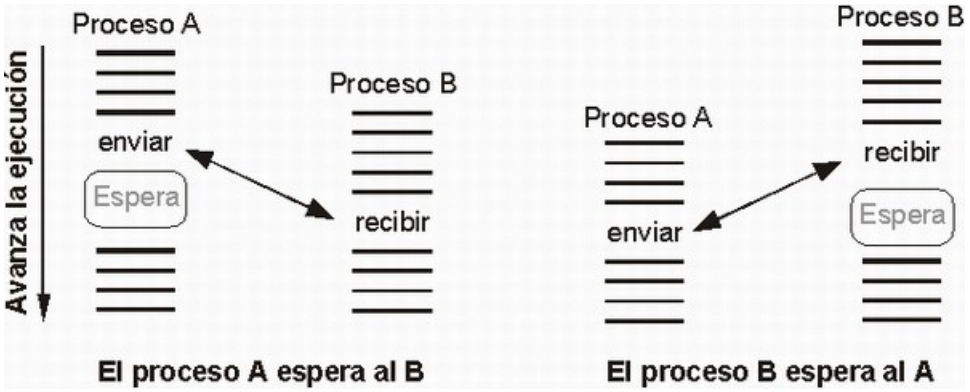
Mensajes

En este caso, la sincronización y comunicación entre procesos es a base de envío y recepción de mensajes. Con este sistema se pretende evitar compartir variables como se vio anteriormente, y principalmente se utilizan en entornos distribuidos, es decir, en máquinas distribuidas en redes.


3
2
4
1
(a) Posible interbloqueo
(b) Interbloqueo


Valor del semáforo (s)
P0
P1
P2
1
0
-1
-2
wait(s)
wait(s)
wait(s)
signal(s)
signal(s)
signal(s)
Ejecutando código de la sección crítica
Proceso bloqueado en el semáforo


write
Proceso de usuario
Proceso de usuario
read
SO
Tubería
Flujo de datos


Avanza la ejecución
Proceso A
Proceso B
enviar
Espera
recibir
El proceso A espera al B
Proceso A
Proceso B
recibir
enviar
Espera
El proceso B espera al A

Mecanismos de comunicación entre procesos

Sincronización de mensajes

En el método anterior, al haber un emisor y un receptor. El proceso receptor siempre espera si el mensaje no ha llegado todavía. Por lo tanto, este proceso tiene que tener implementado un mecanismo de escucha permanente.

entonces hay tres métodos básicos e comunicación: **asíncrona** o **síncrona** (se espera que el receptor envíe ”recibido” en la asíncrona no), **invocación remota** (solicita respuesta)

Identificación proceso Emisor Receptor

Directa (emisor identifica) **indirecta** (intermediario; buzón, pipe…) **simétrica** (se identifican ambos)

Estructura de los mensajes

Un lenguaje debería permitir que cualquier estructura de datos sea transmitida en un mensaje, pero se puede encontrar una representación diferente en el emisor y en el receptor. Por eso se restringe a objetos fijos definidos por el sistema. El uso de mensajería entre procesos es la base de la programación en red.

Sincronización

La sincronización es uno de los problemas a solucionar en la programación concurrente, junto con la exclusión mútua. Intercala la ejecución de las diferentes tareas si se trata de un entorno monoprocesador, o en a uno de los procesadores si el programa se está ejecutando en una arquitectura multiprocesador.

Funciones de sincronización entre hilos

Las aplicaciones de múltiples hilos permiten al programador dividir un eje- cutable en varias subtareas más pequeñas que se ejecutarán. WIN: CreateThread() ExitThread(). UNIX: pthread_create y pthread_detach. Tienen funcionalidad de retorno. Recordar que un thread pertenece a un proceso y solo tiene alcance allí.

Problemas de sincronización (deadlocks)

No existe una solución general para los interbloqueos. Es un problema crítico en la programación concurrente.

El bloqueo surge fácilmente en el momento en el que dos procesos hagan uso de recursos que se necesitan al mismo tiempo. Como los filósofos.

Sincronización

Problemas de sincronización (deadlocks)

Un sistema informático está compuesto por un número limitado de recursos, que son solicitados por los procesos que están en competición en un programa concurrente. El espacio de memoria, los ciclos de CPU, los archivos y os dispositivos de E/S constituyen ejemplos de tipos de recursos.

Cuando un proceso quiere emplear un recurso debe seguir esta secuencia:

1 Solicitud. 2 Uso. 3 Liberación.

Los pasos 1 y 3, o sea, la solicitud y liberación de los recursos, son llamadas al sistema. El problema crítico se da cuando dos procesos están en la secuencia de uso y necesitan utilizar otro recurso que al mismo tiempo está siendo utilizado.

Sincronización

Problemas de sincronización (deadlocks) INTERBLOQUEO

se puede abordar: 1 Empleando un **protocolo** para impedir o evitar los interbloqueos. 2 Permitir que el sistema entre en estado interbloqueo, detectarlo y realizar una **recuperación**. 3 **Ignorar** el problema

La tercera solución es la que utilizan la mayoría de los sistemas operativos, y es conocida como el algoritmo del avestruz. Se fundamenta en que raramente se producen bloqueos, y el coste de montar un mecanismo de evitación es muy elevado para las veces que ocurre.

Prevención de interbloqueos

Un bloqueo se produce cuando se dan estas cuatro condiciones: Exclusión mutua. Retención y espera. Sin desalojo. Espera circular

Si se evita **al menos una de ellas**, se evaden los interbloqueos.

Otra posibilidad es la **recuperación, informando al usuario de modo que trate el proceso de manera manual**

Por ejemplo, obligar a tener los
dos tenedores para comer
y si no se tienen después de
un tiempo intentando
obtener el segundo tenedor, soltarlo
De modo que otro filósofo comiera.

3.7.2023

Enrique Krause Buedo

Acceso a dispositivos

En un ordenador el elemento principal de computación es la CPU, que es la encargada de procesar las instrucciones y los datos de un programa. En comunicación con ella están los dispositivos de E/S. (periféricos, de almacenamiento o de comunicaciones) el acceso es mediante controladores, y la comunicación mediante los registros de ellos: *recordamos libro anterior* (de **datos**, de **control** y de **estado**). Así vimos que se puede dar una E/S programada, por interrupciones o mediante DMA

Puertos de E/S

Los puertos lógicos de entrada y salida son zonas o localizaciones de la memoria de un ordenador que se asocian con un puerto físico o con un canal de comunicación, y que proporcionan un espacio para el almacenamiento temporal de la información que se va a transferir entre la localización de memoria y el canal de comunicación.

Acceso a dispositivos

Puertos de E/S

Los puertos se identifican por números desde 1 hasta 65000, o más, siendo **reservados** los puertos de 1 a 1024. Por ejemplo HTTP: puerto 80 o FTP: puerto 20.

A partir del 1024 son no **estándar**, hasta el 49.151, luego los **efímeros**, elegidos aleatoriamente, en cada caso. Se detallan algunos muy conocidos:

21-FTP | 22-SSH | 23-Telnet | 53-DNS | 80-http | 135-RPC | 139-NetBIOS | 5000-UpnP | 8080-WebProxy

Fin del capítulo 1

Leeros al menos el resumen!!

Pág. 58 del manual